Algorithmique Distribuée
Diffussion, Parcours, Construction d'arbre

Janna Burman
janna.burman@lisn.fr

Diffusion d'information

Hypothèses : réseau asynchrone avec graphe de communication (quelconque) **connexe** et **bidirectionnel**

Problème : un processus distingué (initiateur) a une pièce d'information (notée *Info*) venant de l'extérieur et le but est **de diffuser/propager cette pièce à tous les processus**

Diffusion d'information
par l'inondation/en rush (contrôlée)

Les règles de l'algorithme sont les suivantes :

- Un processus (distingué/l'initiateur), sur réception d'une information (Info) venant de l'application, l'envoie à tous ses voisins.

- Un processus i, sur réception de nouvelle pièce d'information Info (s'il n'en a pas eu avant) d'un voisin p, enregistre l'information et l'envoie à tous ses voisins.

Diffusion d'information par l'inondation/en rush (contrôlée). Pseudo-code pour processus i

Variables :

déjà_vu : booleen ; déjà_vu := *faux*

lors de réception d'une pièce d'information Info de l'app. extérieur (par le processus distingué) →

déjà_vu := vrai
envoyer(Info) à tout voisin

lors de réception d'un message Info →

si (¬ déjà_vu) alors
    daja_vu := *vrai*
    envoyer(Info) à tout voisin
fin si

Diffusion d'information par l'inondation/en rush (contrôlée). Amélioration de complexité en messages

Comment on peut améliorer, à combien ?

Les règles de l'algorithme sont les suivantes :

- Un processus (distingué/l'initiateur), sur réception d'une information (Info) venant de l'application, l'envoie à tous ses voisins.

- Un processus i, sur réception de nouvelle pièce d'information Info (s'il n'en a pas eu avant) d'un voisin p, enregistre l'information et l'envoie à tous **ses autres voisins (à part de p)**.

Diffusion d'information par l'inondation/en rush (contrôlée). Pseudo-code pour processus i (meilleur complexité)

Variables :

déjà_vu : booleen ; déjà_vu := *faux*

lors de réception d'une pièce d'information Info de l'app. extérieur (par le processus distingué) →

déjà_vu := vrai
envoyer(Info) à tout voisin

lors de réception de message Info de processus p →

si (¬ déjà_vu) alors
    daja_vu := *vrai*
    envoyer(Info) à tout voisin **sauf p**
fin si

Diffusion d'information avec terminaison

(explicite ; sur le processus distingué)

Dans ce dernier algorithme, le processus distingué finit-il par savoir que la propagation est terminée ?

Sinon, peut-on modifier le protocole pour que ce soit le cas ?

Diffusion d'information avec terminaison (feedback)

Les règles de l'algorithme sont les suivantes :

- Un processus (distingué/l'initiateur), sur réception d'une information (Info) venant de l'application, l'envoie à tous ses voisins.
  S'il reçoit le même message Info **de tous ses voisins** (en retour), alors il termine.
- Un processus i, sur réception de nouvelle pièce d'information (s'il n'en a pas eu avant) d'un voisin p, enregistre l'information et l'envoie à tous ses autres voisins (à part de p).
  Si processus i reçoit le même message Info **de tous ses voisins (incluant p)**, alors il envoie Info à p, et termine.

Diffusion d'information avec feedback.
Pseudo-code pour processus i

Variables :

déjà_vu : booleen ; déjà_vu := *faux*

attendus : ensemble de canaux/processus ; **attendus := ensemble de voisins (∅, si il est vide)**

père : canal/processus ou *null* ; pere := *null*

lors de réception d'une pièce d'information Info de l'app. extérieur (par le processus distingué) →

déjà_vu := vrai

si (attendu = ∅) alors <termine>

envoyer(Info) à tout voisin

lors de réception de message Info de processus p →

si (¬ déjà_vu) alors

daja_vu := *vrai*

envoyer(Info) à tout voisin **sauf p**

père := p

fin si

attandus := attandus – {p}

si attandus = ∅ alors

si père ≠ *null* alors envoyer(Info) à père

<termine>

fin si

Autres problèmes distribués résolus

- **Parcours de réseau** : chaque processus est visité par un message
  - dans notre cas, par le message Info, en démarrant et terminant sur le processus distingué
- **Constructions d'arbre couvrant** : chaque processus p calcule (un pointeur vers) un voisin q tel que tous les liens pq représentent un arbre qui couvre tous les processus
  - dans notre cas, avec les pointeurs *père*, enraciné dans le processus distingué
  - les pointers vers les enfants peuvent être utiles (e.g., pour les diffusions efficaces en temps).
    Comment peut-on calculer les enfants ?

Diffusion d'information avec feedback, parcours et construction d'arbre couvrant enraciné **avec enfants.** Pseudo-code pour processus i

Variables :

déjà_vu : booleen ; déjà_vu := *faux*

attendus : ensemble de canaux/processus ; attendus := ensemble de voisins (∅, si il est vide)

père : canal/processus ou *null* ; pere := *null*

enfants : ensemble de canaux/processus ; enfants = ∅

lors de réception d'une pièce d'information Info de l'app. extérieur (par le processus distingué) →

déjà_vu := vrai

si (attendu = ∅) alors <termine>

envoyer(Info) à tout voisin

lors de réception de message Info ou Info||enfant de processus p →

si (¬ déjà_vu) alors

déja_vu := *vrai*

envoyer(Info) à tout voisin **sauf p**

père := p

fin si

attandus := attandus – {p}

si (attandus = ∅) alors

si (père ≠ *null)* alors envoyer(Info||enfant) à père

<termine>

fin si

Si (le message reçu est Info||enfant) alors enfants := enfants ∪ {p}

Correction
(Diffusion d'information avec feedback)

- Tout processus a reçu l'Info (et était exploré) : Par absurde, il existe un p pas exploré avec un voisin q exploré (car le graph est connexe et au moins l'initiateur est exploré). Donc, q envoie Info à p → absurde. Peut être prouvé aussi par récurrence sur la distance de l'initiateur.

- Construction d'arbre : Chaque p reçoit l'Info pour la premier fois de son père (événement unique pour chaque p, lors de quel le père est désigné). Cela garantie qu'il n'y pas de cycles. De plus, p se désigne comme enfant une fois « à sa terminaison », en envoyant un message Info||enfant sur le même lien vers père → |V| -1 arêtes père-enfant dans l'arbre, couvrant tous les processus.

- La terminaison : Prouvons que chaque processus p exécute <termine> et puis ni reçoit ni envoie des messages ainsi que tout processus dans sous arbre enraciné dans p; par récurrence sur la distance d'un processus à l'initiateur sur l'arbre construit.
  Pour la base, on considère les feuilles. Une feuille p n'a pas d'enfant, alors tout voisin q de p reçoit le 1er Info de processus différent de p; chaque q envoie alors Info à p. p les reçoit tous et termine.
  Pour le pas de récurrence, considérons un processus interne dans l'arbre (incluant le nœud distingué), et supposons que ces enfants ont terminé (ainsi que leurs sous-arbres), et donc envoyé Info||enfant. Tout autre voisin q de p a envoyé Info à p (au moment de premier visite d'Info chez q). Quand tous ces messages sont réceptionnés chez p, p termine, ainsi que son sous-arbre.

Analyse de complexité

(Diffusion d'information avec feedback)

- Complexité en messages : **2|E|**

- Complexité en temps :
  O(profondeur d'un arbre BFS enraciné à l'initiateur + |V|)

  * par récurrence sur la distance à l'initiateur, chaque nœud p est exploré après au plus dist(p, l'initiateur) unités de temps (UTs), mais le feedback peut être plus longue, borné par |V|-1 UTs …
  = **O(diamètre + |V|)**
  = **O(|V|)** UTs

  - Est-ce que un arbre BFS est construit ?
  - Et si le réseau est parfaitement synchrone ? Quelle est la complexité en round ?

Construction d'arbre BFS Algorithme Bellman-Ford

- Hypothèses : **réseau asynchrone** avec graphe de communication (quelconque) **connexe** et **bidirectionnel**, et un nœud distingué (racine)
- **Le problème :** construction d'arbre couvrant de type BFS, i.e., l'arbre de plus courts chemins (tout chemin, dans l'arbre obtenu, allant de la racine à tout processus est le plus court)
- **Idée :**
  - Utilise l'idée de l'inondation, mais en enregistrant la distance estimée (notée ***d***) à la racine (l'initiateur).
  - Si une distance plus courte est découverte, le processus l'adopte, met à jour son pointeur ***père*** et informe ces voisins

Algorithme Bellman-Ford pour un processus i

Variables :

d : entiers non-négatifs ou ∞ ; d := ∞

père : canal/processus ou *null* ; pere := *null*

lors de réveil de la racine →

d := 0

envoyer(d) à tout voisin

lors de réception de message d_p de processus p →

si (d_p + 1 < d) alors

père := p

d := d_p + 1

envoyer(d) à tout voisin (sauf p)

fin si

Correction et complexité en temps (algo. Bellman-Ford)

Lemme : Pour tout entier *dist* ≥ 1, après *dist* unités de temps, tout processus à distance *dist* de la racine, a déjà reçu un message *dist – 1*, son d = *dist* et son père est un processus avec d = *dist – 1*.

Preuve - par une récurrence simple, sur *dist*.

Corolaire : L'algorithme calcul un arbre BFS en **O(diameter) unités de temps**.

Complexité en message (algo. Bellman-Ford)

- La valeur maximum attribuée à d dans un pas, après le réveil, est n-1 (modification de ∞ à n-1).
- Alors après, d peut être modifié au plus n-2 fois (chaque fois en décrémentant la valeur).
- A chaque modification, le processus p envoie un message à tous ses voisins
- Au plus (n-1)·dégrée(p) messages en tout, par processus p

→ **O(n · |E|) messages** en tout

Algorithme Bellman-Ford - questions

- Est ce que cet algorithme termine ?
- Est ce qu'il termine explicitement ?

Construction d'arbre BFS Algorithme de Dijkstra

- Mêmes **hypothèses** : **réseau asynchrone** avec graphe de communication G(V,E) (quelconque) **connexe** et **bidirectionnel**, et un nœud distingué r0

- **Idée :** une construction itérative (par vagues synchronisantes), à partir d'un nœud distingué (racine) r0, en ajoutant à l'arbre les nœuds et les arêtes de niveau k+1 en étape k+1, après avoir terminer la construction de l'étape k (et les précédentes), avec une synchronisation forte et terminaison (explicite) à la racine.

Algorithme de Dijkstra - hypothèse d'étape p

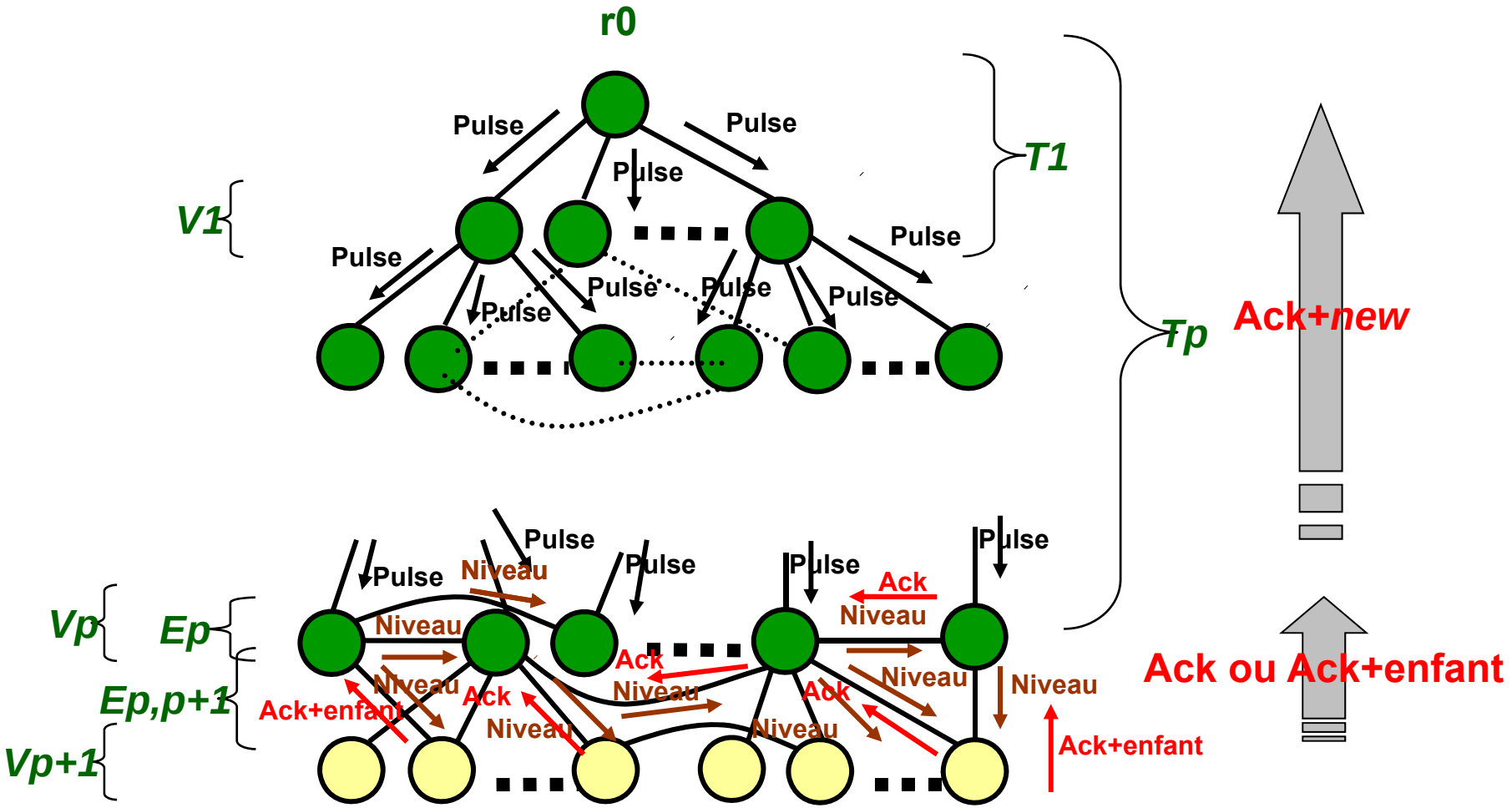
Supposons que les p premières étapes ont terminées et un arbre BFS Tp enraciné en r0 est construit et tout nœud dans Tp connais :

- son père dans Tp
- ses enfants dans Tp
- sa distance à la racine

Algorithme de Dijkstra - itération - étape p+1

1. La racine r0 génère et diffuse un message **Pulse** sur Tp.
2. Toute feuille de Tp recevant **Pulse**, envoie un message d'exploration **Niveau** à tout voisin sauf le père.
3. Un processus v, sur réception de **Niveau** pour la première fois, possiblement de plusieurs voisins :
   i. choisi un de ces voisins w pour être son ***père***
   ii. envoie un acquittement **Ack** pour chaque Niveau et avec une notification comme **enfant** au père w (w note v dans le ***tableau de ces enfants***, sur la réception d'un tel acquittement)
4. Un processus w∈Tp, sur réception **Ack + notification enfant** de processus v:
   i. ajoute v dans le tableau de ces enfants
   ii. attribue ***new*** := Vrai (le booléen *new* est réinitialisée à Faux à chaque réception de **Pulse**)
5. Un processus w∈Tp, sur réception de **Niveau**, retourne un acquittement **Ack**.
6. Sur réception de **Ack** pour tout message **Niveau** envoyé, un processus v∈Tp envoie **Ack**, en ajoutant la valeur de booléen ***new,*** à son père
7. Tout nœud dans Tp, sur réception de **Ack+*new*$_e$** de chaque enfant e, calcul le *ou* logique (∨) sur les bits ***new*** de tous ses enfants et relaye **Ack+** ∨***new***$_e$ à son père.
8. Dès que r0 reçoit **Ack+*new*$_e$** messages de chaque enfants e (et voisins), avec au moins un ***new***$_e$ **= Vrai** il démarre l'itération suivante (p+2). Sinon, **r0 termine**.

Algorithme de Dijkstra - itération - étape p+1



Correction (algo. Dijkstra)

Lemme : Après une étape p, les variables père et enfants forment un arbre BFS Tp enraciné en r0 et couvrant tout les nœuds dans le p-voisinage de r0 (nœud à distance au plus p de r0).

Preuve : par récurrence simple sur p

Complexité en temps (algo. Dijkstra)

Complexité en unités de temps :

Σ_p temps par étape p =

Σ_p(temps de diffusion de Pulse sur Tp +
temps de collecte Ack+new sur Tp +
temps d'exploration par messages Niveau + Ack) =
Σ_p (2p+2) = 2Σ_p p + O(diamètre(G)) =
2O(diamètre^2(G)) + O(diamètre(G)) =
O(diamètre^2(G))

Complexité en messages (algo. Dijkstra)

Σ_p Messages(étape p) =

Σ_p(messages de diffusion Pulse sur Tp + messages de collecte Ack+new sur Tp + messages d'exploration Niveau + Ack sur Ep et Ep,p+1) =

Σ_p (O(n) + O(|Ep| + |Ep,p+1|)) =

O(n·diamètre(G) + |E|) messages

Construction d'arbre DFS

- Mêmes **hypothèses** : **réseau asynchrone** avec graphe de communication G(V,E) (quelconque) **connexe** et **bidirectionnel**
- **Le problème :** construction d'arbre DFS, diffusion, parcours en profondeur (séquentiel)
  - dans un arbre DFS, une arête qui n'est pas dans l'arbre connecte *toujours* un nœud avec son ancêtre (ou son descendant).

- **Idée :**
  - une construction basé sur un parcours séquentiel, débutant en r0 et visitant nœud après nœud - pas de parallélisme
  - parcours exhaustif des nœuds : en revenant sur ces pas seulement quand il n'y a plus de nœuds à découvrir (dans une région de G).

Construction d'arbre DFS (description)

1. Le parcours débute en r0.
2. Quand un nœud v est visité, par un message unique « jeton » venant de voisin p :
   i. Si v est visité pour la 1ere fois, p est noté **père**.
   ii. Si v a des voisins non-visités, le « jeton » est envoyé pour visité un de ces voisin w.
      i. Il est possible que w est déjà visité. Dans ce cas, le « jeton » est retourné avec en acquittement particulier.
   iii. Sinon (tous les voisins sont visités), le « jeton » est renvoyé à son père, si v ≠ r0, et sinon r0 termine.

Complexité en temps et en messages est **O(|E|)**.

Peut-on améliorer les complexités ? (Construction d'arbre DFS)

- **Pour le complexité en message, non.**
  Il est au moins $\Omega(|E|)$ – il faut parcourir tout les arêtes.

- **Pour le temps, oui !**
  - l'idée est d'économiser le temps pour traverser chaque arête vers un nœud déjà visité (arête pas dans l'arbre)
  - Pour ca, chaque nœud visité pour la 1ere fois fonctionne comme suite:

  1. arrête temporairement le parcours,
  2. informe tous les voisins qu'il était visité, par un message V envoyé à chacun,
  3. puis, attend de recevoir un acquittement Ack pour chaque tel message V
  4. redémarre le parcours

  ➔ De cette façon à chaque visite, un nœud sais exactement qui sont les voisins qui étaient déjà visités est qui non. Le « jeton » est envoyé seulement à des voisins non-parcourus, et donc seulement sur les liens de l'arbre (O(V) unités de temps). Le surcoût de messages V et Ack à chaque 1ere visite est de O(1) unités de temps, donc O(|V|) en tout.

  ➔ **O(|V|) unités de temps**

Questions ?