Chapitre 4

Algorithmes d’élection

Chap 4 Algorithmes d'élection

- Protocoles à contrôle centralisé

  => panne du processus coordinateur

- Protocoles à contrôle distribué

  => tâches spécifiques

  => initialisation exécutée par un seul processus (ex: jeton)

- Algorithmes d'élection :
  - détermination dynamique du site jouant un rôle particulier
  - basés sur l'identité des processus

    => choix du plus grand / plus petit numéro

1. Election sur un anneau unidirectionnel

- ## 1.1. Algorithme de Chang et Roberts (variante 1)

1.1.1 Hypothèses et principes

Hypothèses

- le nombre de processus n'est pas connu a priori
- chaque processus P_i est identifié par un n° unique (qu'il connaît)
- les processus sont rangés de façon quelconque sur l'anneau

Principes : il faut *trouver le processus ayant le numéro maximum*.

⇒ Pour cela, chaque processus P_i transmet son n° à son voisin de gauche P_j

⇒ celui-ci, à la réception d'un tel message, compare son propre numéro j au numéro reçu i ;

- si ce dernier est plus grand, il le transmet à son voisin de gauche ;
- sinon il transmet son propre numéro

⇒ principe *d'extinction sélective* des messages.

1.1. Algorithme de Chang et Roberts (variante 1)

- Principes (suite)
  - Initialement, un (ou plusieurs) processus commence(nt) une élection en envoyant un message ;
  - Un tel processus se marque comme participant à l'élection.
  - L'arrivée d'un message à un processus non marqué provoque sa participation à l'élection.
  - Le processus participant à l'élection (marqué) qui reçoit son propre numéro est élu et doit diffuser son identité aux autres *
    - si cela est nécessaire au fonctionnement de l'algorithme qui a déclenché l'élection
  - Remarque : la progression des autres messages a été arrêtée grâce au principe de l'extinction sélective.

1.1. Algorithme de Chang et Roberts (variante 1)

1.1.2. Algorithme

- Chaque processus P_i est doté des déclarations suivantes :
  const mon_numéro ;
  var participant : booléen initialisé à faux ;
  coordinateur : entier ;

- La variable « coordinateur » n'est utile que si le vainqueur de l'élection doit être connu de tous les processus

- Les messages circulant sur l'anneau sont de deux types :
  - type *élection* : la valeur qui l'accompagne est candidate à l'élection
  - type *élu* : la valeur associée a gagné l'élection

- La primitive de communication *env_vg* permet d'envoyer au voisin gauche du processus émetteur sur l'anneau.

1.1. Algorithme de Chang et Roberts (variante 1)

Début

Lors de

Décision de provoquer une élection faire
    début
    participant ← vrai ;
    env_vg (élection, mon_numéro) ;
    fin ;

Réception de (élection, j) faire
    début
    si j > mon_numéro alors
            env_vg (élection, j) ;
            participant ← vrai ;
    fsi ;

1.1. Algorithme de Chang et Roberts (variante 1)

```
/ * Suite : Réception de (élection, j) */
      si j < mon_numéro et ¬participant alors
                  env_vg (élection, mon_numéro) ;
                  participant ← vrai ;
      fsi ;
      si j = mon_numéro alors env_vg (élu, j)  fsi ;
      fin;

Réception de (élu, j) faire
      début
      coordinateur ← j ;
      participant ← faux ;
      si j ≠ mon_numéro alors env_vg (élu, j) fsi ;
      fin ;
Fin
```

1.1. Algorithme de Chang et Roberts (variante 1)

Exercice : exemple de scénario

1.1.3. Preuve [à faire en exercice]

- Montrer que cet algorithme détecte un et un seul plus grand numéro

- Comportement temporel : évaluer le temps requis pour que le plus grand numéro fasse le tour de l'anneau en fonction de n :
  - Cas le plus favorable : proportionnel à n
  - Cas le plus défavorable : proportionnel à 2n - 1

- Nombre de messages de type « élection » :
  - Cas le plus favorable : O (n)
  - Cas le plus défavorable : O (n^2)

1.2. Algorithme de Chang et Roberts (variante 2)

- Dans la version 1, tous les processus de l'anneau participent à l'élection

  $\Rightarrow$ pour une configuration donnée de l'anneau, le gagnant sera toujours le même

- Dans la version 2, seuls *les processus candidats à l'élection rentrent en concurrence*

Variables du processus i :

- *état* : état du service (*repos, en_cours, terminé*). Cette variable est initialisée à repos.
- *chef* : identité du site élu

1.2. Algorithme de Chang et Roberts (variante 2)

Algorithme du site i :

- Si aucun processus d'élection n'a atteint le site i, le service initialise un tel processus.
- Ensuite, il attend que le processus d'élection soit terminé pour renvoyer l'identité du site élu à la couche supérieure.

Candidature ()

```
début
si (etat = repos) alors
        etat ← en_cours ;
        chef ← i ;
        env_vg (élection, i)
fsi ;
attendre (etat = terminé) ;
renvoyer (chef) /* envoi à la couche supérieure : primitive du service */
fin
```

1.2. Algorithme de Chang et Roberts (variante 2)

Algorithme du site i :

- à la réception d'un message de type « élection », si le site est au repos, alors son état devient «en_cours ».
- dans le cas où il reçoit une meilleure requête (numéro plus grand), il en tient compte et la fait suivre.
- dans le cas contraire (numéro plus petit), la requête est avortée (principe d'extinction sélective des messages).
- Dans le cas où la requête revient à un processus candidat, il est l'élu. Il envoie alors un message « élu » sur l'anneau pour annoncer la fin du processus d'élection

1.2. Algorithme de Chang et Roberts (variante 2)

sur_reception_de (élection,initiateur)

```
Début
si ((etat = repos) ou (initiateur > chef) alors
        etat ← en_cours ;
        chef ← initiateur ;
        envoyer_vg (élection, initiateur);
sinon
        si (i = initiateur) alors
                etat ← terminé ;
                envoyer_vg (élu, i) ;
        fsi
fsi
Fin
```

1.2. Algorithme de Chang et Roberts (variante 2)

sur_reception_de (élu, initiateur)

Début
si (i ≠ initiateur) alors
    envoyer_vg (élu, initiateur) ;
    etat ← terminé ;
fsi
Fin

- Preuve : faire en exercice

- Complexité de l'algorithme : idem variante 1
    - Cas le plus défavorable : $O(n^2)$
    - Cas le plus favorable : $O(n)$

- Exemple : reprendre l'exemple du scénario de la variante 1

2. Election sur un graphe quelconque

=> Généralisation de l'algorithme de Chang et Roberts

2.1. Algorithme de parcours du graphe de communication

Principe et exemple

- Cas d'un graphe de communication quelconque :
  => la connaissance du graphe par un site se limite à ses voisins.

- Le parcours du graphe vérifie les propriétés suivantes :
  - ce parcours se termine chez l'initiateur
  - lorsqu'il se termine, ce parcours a visité au moins une fois tous les sites et traversé exactement une fois chaque canal de communication dans chaque sens.

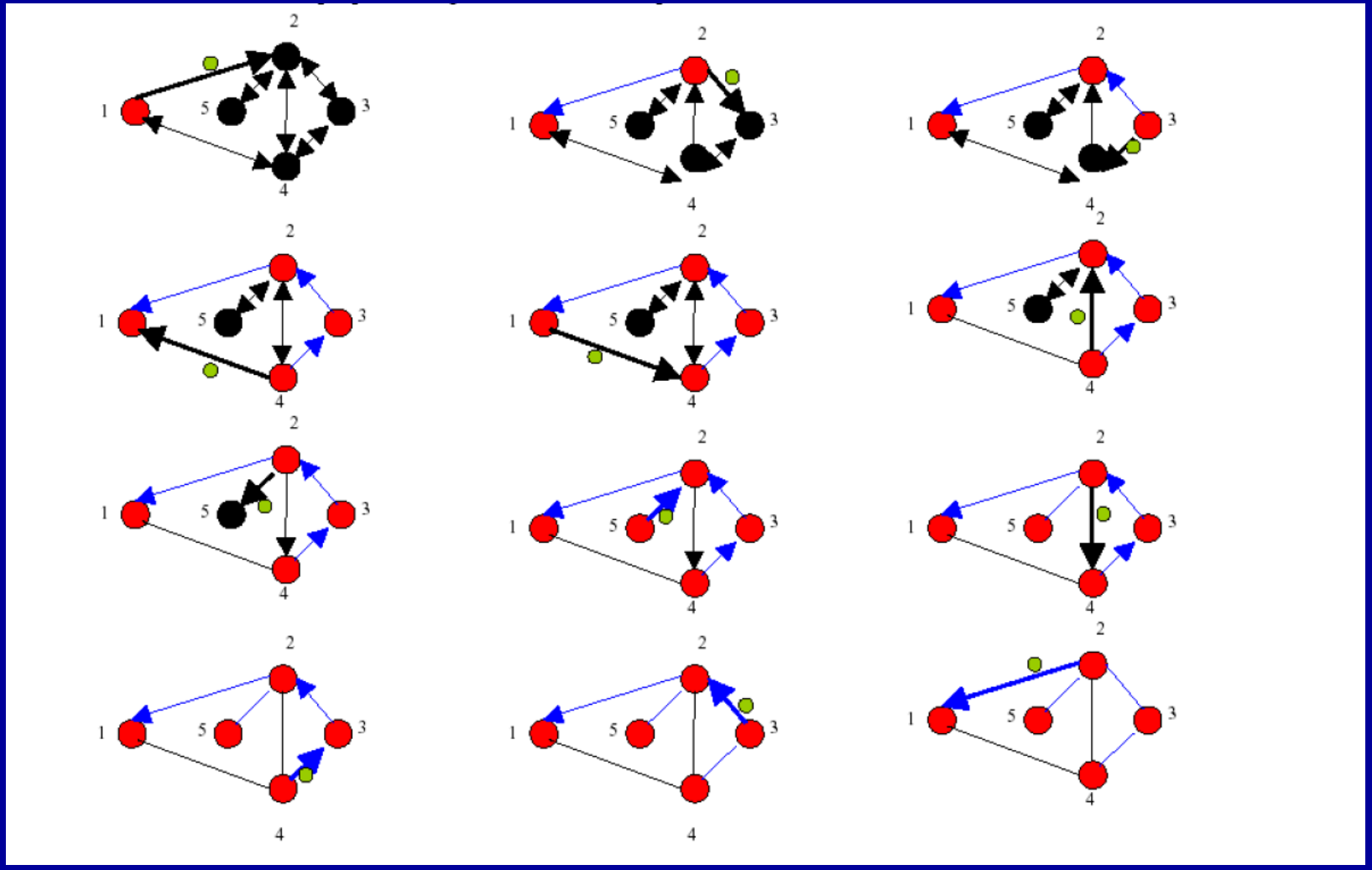
2.1. Algorithme de parcours du graphe de communication

Principe de parcours du graphe

- Chaque canal sortant est considéré comme initialement « ouvert » (représenté par une flèche dans le sens concerné).
- Lorsque le message associé au parcours est envoyé sur un canal ouvert, celui-ci se ferme et ne pourra plus être utilisé par ce parcours.
- Un site déjà visité par ce message est indiqué en rouge sur la figure et dans le cas contraire en noir.
- De plus, lors de la première visite par le message (excepté pour l'initiateur) le site mémorise le canal comme étant celui de son père, indiqué en bleu sur la figure. Il n'utilisera ce canal qu'après avoir utilisé tous les autres canaux.

2.1. Algorithme de parcours du graphe de communication

Exemple :



2.1. Algorithme de parcours du graphe de communication

Algorithme

Fonctions :

- *initier (type)* : initie un parcours avec un certain type de message,
- *faire_suivre (j, type, k)* : fait suivre un type de message *venant de j* dans un parcours *initié par k.*
- *extraire (ensemble) :* renvoie un élément de l'ensemble (s'il est non vide) et le retire de celui-ci.

2.1. Algorithme de parcours du graphe de communication

Variables du site i :

- *voisins* : constante contenant l'ensemble des voisins de i dans le graphe de communication
- T [1..N] : tableau dont la cellule j contribue au parcours initié par j.
- Chaque cellule à deux champs :
  - T [j].voisins
  - T [j].père : initialisé à i qui signifie dans le cas où i ≠ j que le parcours de j n'est pas encore parvenu à i
- *prochain* : variable contenant l'identité d'un voisin de i.

2.1. Algorithme de parcours du graphe de communication

Algorithme du site i :

Pour initier un parcours de graphe, on initialise le champ « voisins » aux voisins du site et on extrait un voisin de cet ensemble pour débuter le parcours.

initier(type)

début
T [i].voisins ← voisins ;
prochain ← extraire (T [i].voisins) ;
envoyer_à (prochain , (type, i)) ;
fin

2.1. Algorithme de parcours du graphe de communication

- S'il s'agit de la première visite de ce parcours, on initialise les champs « père » et « voisins » de la cellule concernée en extrayant le « père » des voisins.
- S'il reste des voisins on extrait l'un d'entre eux pour continuer le parcours.
- Sinon, si i n'est pas l'initiateur du parcours, on emprunte le canal de son père. Le parcours étant terminé sur ce site, on réinitialise le champ « père » pour un éventuel nouveau parcours.
- Si i est l'initiateur, la fonction renvoie l'indication que le parcours est terminé.

2.1. Algorithme de parcours du graphe de communication

- S'il s'agit de la première visite de ce parcours, on initialise les champs « père » et « voisins » de la cellule concernée en extrayant le « père » des voisins.
- S'il reste des voisins on extrait l'un d'entre eux pour continuer le parcours.
- Sinon, si i n'est pas l'initiateur du parcours, on emprunte le canal de son père. Le parcours étant terminé sur ce site, on réinitialise le champ « père » pour un éventuel nouveau parcours.
- Si i est l'initiateur, la fonction renvoie l'indication que le parcours est terminé.

2.1. Algorithme de parcours du graphe de communication

```
sur_reception_de (j, (type,k))   /* réception d'un type de message
                                    venant de j dans un parcours initié par k */

début
si (¬ faire_suivre (j, type, k)) alors
        afficher (« parcours terminé ») ;
fsi
Fin

faire_suivre (j, type, k) /* faire suivre un type de message venant de j
                             dans un parcours initié par k */
début
si (k ≠ i) et (T [k].père = i) alors
        T [k].père ← j ;
        T[k].voisins ← voisins \ {j} ;
fsi
```

2.1. Algorithme de parcours du graphe de communication

```
faire_suivre (j, type, k) /* suite */

si T [k].voisins ≠ ∅ alors
        prochain ← extraire (T [k].voisins) ;
        envoyer_à (prochaini , (type, k)) ;
        renvoyer (VRAI) ;
sinon
        si (k ≠ i) alors
            envoyer_à (T [k].père, (type, k)) ;
            T [k].père = i ;
            renvoyer (VRAI) ;
        sinon
            renvoyer (FAUX)
        fsi
fsi
fin
```

2.2. Algorithme d'élection

Variables du site i :

- ***état*** **: état du service, qui prend une valeur parmi (repos, en_cours, terminé). Cette variable est initialisée à repos.**
- ***chef*** **: identité du site élu.**

Algorithme du site i :

- **Si aucun processus d'élection n'a atteint le site i, le service initialise un tel processus par un parcours du graphe avec une requête de type « élection ».**
- **Ensuite il attend que le processus d'élection soit terminé pour renvoyer l'identité du site élu au niveau supérieur.**

2.2. Algorithme d'élection

candidature ()

```
début
si (etat = repos) alors
        etat ← en_cours ;
        chef ← i ;
        initier(élection) ;
fsi ;
attendre (etat = terminé) ;
renvoyer (chef) /* envoi à la couche supérieure */
Fin
```

2.2. Algorithme d'élection

- Si le site est au repos, la réception d'une requête provoque le changement d'état.
- Dans le cas où le processus reçoit une « meilleure » requête, il la propage.
- Dans le cas contraire, le parcours est avorté et la requête disparaît (extinction sélective des messages).
- Si le parcours n'est pas avorté, alors à l'aide de la fonction ***faire_suivre*** *() :*

  - soit on continue le parcours,
  - soit le parcours est terminé (i est alors le site élu)

  et i initie alors un parcours de confirmation (« élu »).

2.2. Algorithme d'élection

sur_réception_de (j, (élection, k))

```
Début
si ((etat = repos) ou (initiateur > chef) alors
      etat ← en_cours ;
      chef ← initiateur ;
      si (¬ faire_suivre (j, type, k)) alors
         etat ← terminé ;
         initier(élu, i);
      fsi
fsi
Fin
```

2.2. Algorithme d'élection

sur_reception_de (j, (élu,k))

Début
si (i ≠ k) **alors**
    etat ← terminé ;
    faire_suivre (j, élu, k) ;
fsi
Fin

Preuve : faire sous forme d'exercice

Exemple : reprendre le scénario de la section 2.1