# Autour des algorithmes distribués

Arnaud Casteigts, Jérémie Chalopin, Emmanuel Godard,
Yves Métivier, Mohamed Mosbah, John Michael Robson et Akka Zemmari

27 novembre 2017

# Table des matières

# Chapitre 1

# Introduction

Un système distribué est un ensemble d'entités autonomes pouvant échanger des informations à travers un réseau de communication. Ces entités autonomes doivent pouvoir traiter et stocker des informations : elles peuvent être des processus, des processeurs, des capteurs ou bien encore des ordinateurs. Le réseau peut être local ou bien à l'échelle d'un campus ou de la terre. Suivant sa nature des problèmes différents peuvent se poser : fiabilité des communications, temps de communication, hétérogénéité des machines. Un système distribué peut être motivé par le partage de ressources (physiques ou logicielles), la fiabilité du système obtenu, l'échange d'informations ou bien encore l'efficacité.

On suppose que le système est totalement distribué : il n'y a pas d'horloge globale et aucun sommet n'a une connaissance (en temps réel) de l'état global du réseau. Chaque entité du réseau a sa propre vie et sa vie en tant qu'élément du réseau ; l'algorithmique distribuée traite de cette deuxième partie. On peut citer en particulier :

- la communication en général, le routage ou la diffusion d'informations en particulier ;
- le contrôle de l'exécution d'algorithmes, avec en particulier le problème de l'élection, la détection de propriétés concernant l'état du réseau comme la terminaison d'une tâche ou la détection d'un inter-blocage ou bien encore le calcul de points de reprise ;
- la gestion du temps ou la synchronisation entre différents éléments du réseau ;
- la résistance ou la tolérance aux pannes.

En général, le réseau est représenté par un graphe connexe $G = (V, E)$ où les sommets représentent les entités de base (processeurs, processus...) et les arêtes les liens de communication ou d'interaction.

Un algorithme distribué est un ensemble d'algorithmes chacun étant exécuté par une "machine".

On se propose, à travers ce texte, de fournir des outils permettant de coder, d'étudier et d'enseigner différents aspects de l'algorithmique distribuée en s'intéressant plus particulièrement aux problèmes cités ci-dessus. Ces études sont faites d'une part en fonction des règles de fonctionnement ou de calcul autorisées sur le réseau et d'autre part de la connaissance initiale ou de la configuration initiale du réseau. La non existence d'algorithmes distribués déterministes pour résoudre certains problèmes nous conduit à proposer et à étudier des algorithmes distribués probabilistes dont l'expression est souvent simple mais l'analyse difficile. Un autre aspect est l'étude des liens entre différents modèles soit à base de calculs locaux soit à base de communication par messages et la comparaison de leurs puissances.

Les questions relatives aux systèmes distribués constituent un axe majeur de l'informatique ; citons par exemple la conception et le développement d'architectures distribuées, la définition et

le développement d'environnements de programmation distribués, la description et la validation de programmes distribués ou bien encore l'étude des communications dans les systèmes distribués. Un moyen décisif est la maîtrise des mécanismes fondamentaux qui y interviennent ; cela passe par :

— la définition et l'étude de différents modèles rendant compte de phénomènes qui s'y produisent,
— le développement d'outils permettant de visualiser, de tester et, plus généralement, d'aider à la mise au point et aux preuves de systèmes distribués,
— la mise en œuvre de plate-formes distribuées afin de vérifier et d'expérimenter.

Un système distribué est un ensemble de processeurs (processus) qui peuvent interagir. Il existe principalement 3 modèles d'interactions :

— le modèle avec communication par messages,
— le modèle avec communication par mémoires partagées,
— le modèle des calculs locaux.

Dans ces trois modèles, les processus sont représentés par les sommets d'un graphe et les interactions par des arêtes. Pour les modèles à base de communications, les processus interagissent à travers des primitives de communication : des messages peuvent être envoyés le long des liens, des opérations de lecture/écriture peuvent être exécutées dans des registres associés aux arêtes ou bien des ondes sont émises. Dans le modèle des calculs locaux les interactions sont définies par des réécritures de graphes. Ces différents modèles (et sous-modèles) codent différentes architectures de systèmes, différents niveaux de synchronisation et différents niveaux d'abstraction.

L'existence et l'expression des algorithmes distribués dépendent d'hypothèses techniques sur les réseaux concernant le mode de communication, la synchronisation partielle de processeurs voisins, l'anonymat des processeurs, l'état initial des processeurs ou bien encore de la connaissance initiale que l'on a sur le réseau. Les cadres classiques (comme par exemple CSP) rendent difficile la lisibilité de certains algorithmes, ils ne permettent pas toujours de comprendre leur puissance et leurs limites et de faire des preuves formelles de leurs propriétés.

Dans le modèle des calculs locaux, un système distribué est codé par un graphe étiqueté : les sommets correspondent aux processeurs, les arêtes aux liens d'interactions et les étiquettes associées codant l'état du processeur ou du lien. Une règle de calcul est définie par la donnée d'un graphe connexe et de deux étiquetages de ce graphe, donc elle est appliquée localement.

Les principaux types de règle utilisés et étudiés sont décrits sur la figure 1.1.

Un système de réécriture est défini par la donnée d'un ensemble dénombrable de telles règles. On considère des exécutions asynchrones : il n'y a pas d'horloge globale et deux pas de réécriture peuvent avoir lieu en parallèle s'ils modifient des parties disjointes de leurs supports. Le comportement du réseau est défini par un étiquetage initial et par une suite de pas de calcul. Le caractère abstrait des calculs locaux facilite la conception d'algorithme et des bonnes structures combinatoires afférentes ; par ailleurs, une fois les algorithmes, les preuves et les analyses réalisés, le travail de traduction (ou d'adaptation) dans des modèles plus réalistes est relativement plus simple. Ce modèle permet de comprendre où passe la frontière entre les tâches réalisables et celles qui ne le sont pas.

Ce texte ne parle ni des machines supportant les réseaux, ni des architectures du réseau et des différentes "couches" qui le composent et ni des langages de programmation correspondant.

Les livres suivants ont largement contribué à l'écriture de ce document : [AW04, Ber83, Dol00, KS08, Lav95, Lyn96, Mas91, TvS02, Tel00, Ray85, Ray87, Ray91, San06].

De nombreux chapitres sont issus de ces articles ou thèses : [BGM$^+$01, CGMO06, CM07, LMS99, MSZ03, MSZ02, God02, Zem00, Sel04, Cha06, CGM12, CM10, MRSDZ10, MRSDZ11].

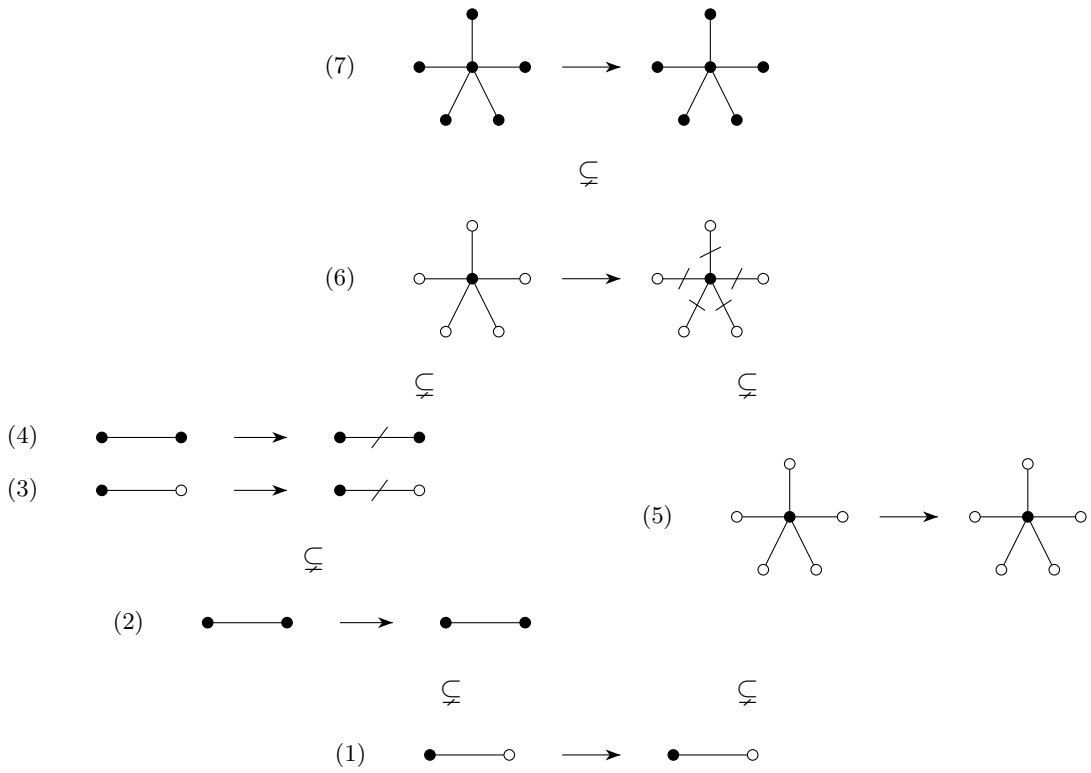



FIGURE 1.1 – Étant donnée une règle, les sommets (blancs ou noirs) et les arêtes formant le membre gauche de cette règle constituent le support de la règle. Les sommets noirs sont actifs : leurs étiquettes peuvent changer quand la règle est appliquée. Les sommets blancs sont passifs : leurs étiquettes conditionnent l'application de la règle mais ne peuvent être modifiées par la règle. Dans les modèles où les arêtes sont marquées, les arêtes sont étiquetées et l'application de la règle peut modifier leurs étiquettes. Cette figure montre la hiérarchie entre ces différents types de règles pour le problème de l'élection dans un réseau.

# Chapitre 2

# Graphes

Un système distribué est défini par un ensemble d'entités qui interagissent entre elles et donc très naturellement peut être codé par un graphe (dirigé ou non dirigé). Par ailleurs de nombreux algorithmes sur les graphes constituent des briques de base pour réaliser des algorithmes sur les systèmes distribués.

De nombreuses propriétés de tels systèmes et des algorithmes associés ainsi que leurs preuves utilisent également des notions de graphes.

Ce chapitre présente ces notions de base que l'on utilisera dans ce texte.

## 2.1 Graphes non-dirigés

### Quelques définitions

**Définition 2.1** *Un* graphe non-dirigé sans boucle *est défini par un ensemble de sommets $V(G)$, un ensemble d'arêtes $E(G)$ et par une fonction ext qui associe à chaque arête deux éléments distincts de $V(G)$, appelés ses* extrémités.

*Pour toute arête $e \in E(G)$ et tous sommets $u, v \in E(G)$ tels que $ext(e) = \{u, v\}$, on dit que l'arête $e$ est* incidente *à $u$ et à $v$. Les sommets $u$ et $v$ sont dits* adjacents *ou* voisins.

Un tel graphe $G$ sera noté $G = (V(G), E(G))$ ou $G = (V, E)$.

Par la suite, sauf précision contraire, le terme *graphe* désignera un graphe non-dirigé sans boucle.

**Définition 2.2** *Un* graphe simple non-dirigé *est un graphe non-dirigé sans boucle tel qu'il y ait au plus une arête entre deux sommets, i.e., pour tous sommets $u, v \in V(G)$, $|\{e \in E(G) \mid ext(e) = \{u, v\}\}| \leq 1$.*

*Chaque arête $e \in E(G)$ peut alors être vue comme une paire de sommets distincts qui sont ses extrémités.*

Par la suite, sauf précision contraire, le terme *graphe simple* désignera un graphe simple non-dirigé.

**Définition 2.3** *Un graphe $H$ est un sous-graphe partiel d'un graphe $G$ si $V(H) = V(G)$ et $E(H) \subseteq E(G)$.*

*Un graphe $H$ est un sous-graphe de $G$ si $V(H) \subseteq V(G)$ et $E(H) \subseteq E(G)$.*

*Un graphe $H$ est un sous-graphe induit de $G$ si $V(H) \subseteq V(G)$ et $E(H)$ est l'ensemble des arêtes de $G$ dont les extrémités sont dans $V(H)$, i.e., $\forall e \in E(G), ext(e) \subseteq V(H) \iff e \in E(H)$.*

*Pour tout graphe $G$ et tout ensemble $S \subseteq V(G)$, on note $G[S]$ le sous-graphe induit de $G$ dont les sommets sont les éléments de $S$.*

**Définition 2.4** *Dans un graphe $G$, le* voisinage *d'un sommet $u$, noté $N_G(u)$, est l'ensemble des voisins de $u$, i.e., $N_G(u) = \{v \in V(G) \mid \exists e \in E(G)$ telle que $ext(e) = \{u, v\}\}$. On note $I_G(u)$ l'ensemble des arêtes incidentes au sommet $u$, i.e., $I_G(u) = \{e \in E(G) \mid u \in ext(e)\}$.*

*Dans un graphe $G$, le* degré *d'un sommet $u$, noté $\deg_G(u)$, est le nombre d'arêtes incidentes à $u$, i.e., $\deg_G(u) = |I_G(u)|$. En particulier, si $G$ est un graphe simple, alors le degré de $u$ est son nombre de voisins, i.e., $\deg_G(u) = |N_G(u)|$.*

Un graphe $G$ est *$d$-régulier* si tous les sommets de $G$ sont de degré $d$ ; on dit aussi que $G$ est *régulier*. Dans le cas particulier où tous les sommets ont un degré 0 (le graphe ne contient pas d'arête), on dit que $G$ est un *stable*. Un graphe $G$ est *biparti* si on peut partitionner les sommets de $V(G)$ en deux ensembles $V_1$ et $V_2$ tels que $G[V_1]$ et $G[V_2]$ sont des stables. Un graphe biparti $G$ est *$(k, l)$-semi-régulier* si tous les sommets de $V_1$ sont de degré $k$ et tous les sommets de $V_2$ sont de degré $l$ ; on dit aussi que $G$ est *biparti semi-régulier*. Un graphe $G$ admet un *couplage parfait* s'il existe un ensemble d'arêtes $E' \subseteq E(G)$ tel que chaque sommet $v \in V(G)$ soit incident à exactement une arête de $E'$.

**Définition 2.5** *Étant donnés un graphe simple $G$ et un sommet $u \in V(G)$, l'*étoile *de centre $u$ est le sous-graphe de $G$ défini par $V(B_G(u)) = N_G(u) \cup \{u\}$ et $E(B_G(u)) = I_G(u)$.*

**Définition 2.6** *Dans un graphe $G$, un* chemin $\Gamma$ *entre deux sommets $u$ et $v$ de $G$ est une suite alternée $(u_0, e_1, u_1, e_2, \ldots, e_n, u_n)$ telle que :*

- *pour tout $i \in [0, n]$, $u_i \in V(G)$,*
- *pour tout $i \in [1, n]$, $e_i \in E(G)$,*
- *pour tout $i \in [1, n]$, $ext(e_i) = \{u_{i-1}, u_i\}$,*
- *$u_0 = u$ et $u_n = v$.*

*Les sommets $u_0$ et $u_n$ sont les* extrémités *du chemin $\Gamma$ et les sommets $u_1, \ldots, u_{n-1}$ sont les* sommets internes *du chemin ; la* longueur *$n$ du chemin est son nombre d'arêtes.*

*Un chemin dont toutes les arêtes sont distinctes est dit* simple *et un chemin qui ne contient pas deux fois le même sommet est dit* élémentaire.

*Lorsque les extrémités d'un chemin simple $\Gamma$ sont confondues, on dit que $\Gamma$ est un* cycle. *Un* cycle élémentaire *est un cycle dont tous les sommets internes sont distincts.*

Dans un graphe simple, un chemin sera généralement décrit par la suite de ses sommets $(u_0, u_1, \ldots, u_n)$, puisque pour tout $i \in [1, n]$, l'arête $e_i$ est nécessairement l'arête $\{u_{i-1}, u_i\}$. Ainsi, dans un graphe simple $G$, une suite de sommets $(u_0, u_1, \ldots, u_n)$ est un chemin si pour tout $i \in [1, n]$, $\{u_{i-1}, u_i\} \in E(G)$.

**Définition 2.7** *Un graphe $G$ est* connexe *si pour tous sommets $u, v \in V(G)$, il existe un chemin entre $u$ et $v$ dans $G$.*

Tous les graphes considérés dans ce mémoire seront connexes.

Un *arbre* est un graphe connexe sans cycle. Un *anneau* est un graphe constitué d'un cycle simple. Un arbre couvrant $A$ d'un graphe $G$ est un arbre qui est un graphe partiel de $G$.

**Définition 2.8** *Étant donnés un graphe connexe $G$ et deux sommets $u, v \in V(G)$, la* distance *de $u$ à $v$ dans $G$, notée $\text{dist}_G(u, v)$, est la longueur du plus court chemin de $u$ à $v$. Le* diamètre *de $G$, noté $D(G)$, est la plus grande distance entre deux sommets de $G$, i.e., $D(G) = \max\{\text{dist}_G(u, v) \mid u, v \in V(G)\}$.*

**Définition 2.9** *Un* homomorphisme $\varphi$ *d'un graphe $G$ dans un graphe $H$ est une application de $V(G)$ dans $V(H)$ et de $E(G)$ dans $E(H)$ qui préserve les relations d'incidence, i.e., pour toute arête $e \in E(G)$, $ext(\varphi(e)) = \varphi(ext(e))$.*

**Définition 2.10** *Un homomorphisme $\varphi$ d'un graphe $G$ dans un graphe $H$ est un* isomorphisme *si $\varphi$ est bijective.*

*On dit alors que $G$ et $H$ sont* isomorphes *et on note $G \simeq H$.*

Une *famille* (ou *classe*) de graphes est un ensemble de graphes clos par isomorphisme.

Dans les définitions suivantes, on considère les homomorphismes entre graphes simples. Il est facile de voir que dans ce cas particulier, ces définitions sont équivalentes avec les précédentes.

**Définition 2.11** *Un* homomorphisme $\varphi$ *d'un graphe simple $G$ dans un graphe simple $H$ est une application de $V(G)$ dans $V(H)$ qui préserve les relations d'adjacence entre sommets, i.e., pour toute arête $\{v, w\} \in E(G)$, $\{\varphi(v), \varphi(w)\} \in E(G)$.*

**Définition 2.12** *Un homomorphisme $\varphi$ d'un graphe simple $G$ dans un graphe simple $H$ est un* isomorphisme *si $\varphi$ est bijective et si $\varphi^{-1}$ est aussi un homomorphisme.*

## 2.2 Graphes dirigés

### Définitions

**Définition 2.13** *Un* graphe dirigé *$D$ est défini par un ensemble de* sommets *$V(D)$, un ensemble d'*arcs *$A(D)$ et deux fonctions $s_D$ et $t_D$ de $A(D)$ dans $V(D)$.*

*Pour chaque arc $a \in A(D)$, $s_D(a)$ est la* source *de l'arc $a$ et $t_D(a)$ est la* cible *de $a$. On dit que l'arc $a$ est* incident *à $s_D(a)$ et à $t_D(a)$. Si $s_D(a) = t_D(a)$, l'arc $a$ est une* boucle.

*Pour tous sommets $u, v \in V(D)$, s'il existe un arc $a \in A(D)$ tel que $s_D(a) = u$ et $t_D(a) = v$, $u$ est un* prédecesseur *de $v$ et $v$ est un* successeur *de $u$.*

Lorsqu'il n'y aura pas d'ambiguité, les fonctions $s_D$ et $t_D$ seront respectivement notées $s$ et $t$.

**Définition 2.14** *Dans un graphe dirigé $D$, le* voisinage sortant *d'un sommet $u$, noté $N_D^+(u)$ est l'ensemble des successeurs de $u$, i.e., $N_D^+(u) = \{v \in V(D) \mid \exists a \in A(D)$ tel que $s(a) = u$ et $t(a) = v\}$. On note $I_D^+(u)$ l'ensemble des arcs sortants de $u$, i.e., $I_D^+(u) = \{a \in A(D) \mid s(a) = u\}$. Le* degré sortant *de $u$, noté $d_D^+(u)$ est le nombre d'arcs sortants de $u$, i.e., $d_D^+(u) = |I_D^+(u)|$.*

*Le* voisinage entrant *d'un sommet $u$ dans un graphe dirigé $D$, noté $N_D^-(u)$, est l'ensemble des prédecesseurs de $u$, i.e., $N_D^-(u) = \{v \in V(D) \mid \exists a \in A(D)$ tel que $s(a) = v$ et $t(a) = u\}$. On note $I_D^-(u)$ l'ensemble des arcs entrants de $u$, i.e., $I_D^-(u) = \{a \in A(D) \mid t(a) = u\}$. Le* degré entrant *de $u$, noté $d_D^-(u)$ est le nombre d'arcs entrant de $u$, i.e., $d_D^-(u) = |I_D^-(u)|$.*

**Définition 2.15** *Dans un graphe dirigé $D$, un* chemin *de $u$ à $v$ est une suite d'arcs $(a_0, a_2, \ldots, a_n)$ telle que $s(a_0) = u$, $t(a_n) = v$ et pour tout $i \in [1, n]$, $t(a_{i-1}) = s(a_i)$.*

**Définition 2.16** *Un graphe dirigé $D$ est fortement connexe si pour tous sommets $u$ et $v$, il existe un chemin de $u$ à $v$ dans $D$.*

**Définition 2.17** *Un* graphe dirigé symétrique *est un graphe dirigé $D$ muni d'une involution $Sym : A(D) \to A(D)$ qui à chaque arc $a \in A(D)$ associe son arc symétrique $Sym(a)$ tel que $s_D(Sym(a)) = t_D(a)$.*

**Définition 2.18** *Un* homomorphisme $\varphi$ *d'un graphe dirigé* $D$ *dans un graphe dirigé* $D'$ *est une application de* $V(D)$ *dans* $V(D')$ *et de* $A(D)$ *dans* $A(D)$ *qui commute avec les fonctions source et cible, i.e., pour tout arc* $a \in A(D)$, $s_{D'}(\varphi(a)) = \varphi(s_D(a))$ *et* $s_{D'}(\varphi(a)) = \varphi(s_D(a))$.

**Définition 2.19** *Un homomorphisme* $\varphi$ *d'un graphe dirigé* $D$ *dans un graphe dirigé* $D'$ *est un* isomorphisme *si* $\varphi$ *est bijective.*

*On dit alors que* $D$ *et* $D'$ *sont* isomorphes.

## 2.3 Graphes étiquetés

On travaille sur des graphes dont les sommets et les arêtes sont étiquetés par un ensemble d'étiquettes $L$, qui peut être fini ou infini. On supposera l'ensemble $L$ muni d'un ordre total, noté $<_L$.

Un graphe étiqueté, noté $(G, \lambda)$, est un graphe $G$ muni d'une fonction d'étiquetage $\lambda : V(G) \cup E(G) \to L$ qui associe une étiquette à chaque sommet $v \in V(G)$ et à chaque arête $e \in E(G)$. Lorsque la fonction d'étiquetage n'aura pas à être explicitée, les graphes $(G, \lambda_G), (H, \lambda_H), \ldots$ seront dénotés $\mathbf{G}, \mathbf{H}, \ldots$ ; on dit que le graphe $G$ est le graphe *sous-jacent* de $\mathbf{G}$.

On dit que l'étiquetage d'un graphe $(G, \lambda)$ est *uniforme* s'il existe deux étiquettes $\alpha, \beta \in L$ telles que pour tout sommet $v \in V(G)$, $\lambda(v) = \alpha$ et pour toute arête $e \in E(G)$, $\lambda(v) = \beta$.

Les notions de sous-graphes, sous-graphes induits, sous-graphes partiels s'étendent aux graphes étiquetés en demandant que l'étiquetage soit préservé.

**Définition 2.20** *Un graphe* $\mathbf{H} = (H, \eta)$ *est un graphe partiel (resp. sous-graphe, sous-graphe induit) d'un graphe* $\mathbf{G} = (G, \lambda)$ *si* $H$ *est un graphe partiel (resp. sous-graphe, sous-graphe induit)de* $G$ *et pour tout* $x \in V(H) \cup E(H)$, $\lambda(x) = \eta(x)$.

Un homomorphisme entre deux graphes étiquetés est un homomorphisme entre les graphes sous-jacents qui préserve l'étiquetage.

**Définition 2.21** *Un* homomorphisme $\varphi$ *d'un graphe étiqueté* $\mathbf{G} = (G, \lambda)$ *dans un graphe* $\mathbf{H} = (H, \eta)$ *est un homomorphisme de* $G$ *dans* $H$ *tel que pour tout* $x \in V(G) \cup E(G)$, $\lambda(x) = \eta(\varphi(x))$.

*L'homomorphisme* $\varphi$ *est un isomorphisme entre* $\mathbf{G}$ *et* $\mathbf{H}$ *si* $\varphi$ *définit un isomorphisme entre* $G$ *et* $H$.

Une *occurrence* de $\mathbf{G} = (G, \lambda)$ dans $\mathbf{G'} = (G', \lambda')$ est un isomorphisme entre $(G, \lambda)$ et un sous-graphe $(H, \eta)$ de $(G', \lambda')$.

**Remarque 2.22** *On peut assimiler tout graphe non-étiqueté* $G$ *au graphe étiqueté* $(G, \lambda_\epsilon)$ *où* $\lambda_\epsilon$ *est un étiquetage tel que pour tout* $x \in V(G) \cup E(G)$, $\lambda_\epsilon(x) = \epsilon$, *où* $\epsilon$ *est le mot vide. Par conséquent, le terme « graphe » désignera indistinctement un graphe étiqueté et un graphe non-étiqueté.*

### 2.3.1 Étiquetages particuliers

Les fonctions d'étiquetage considérées dans cette partie n'attribuent des étiquettes qu'aux sommets (les arêtes ne sont pas étiquetées). Une coloration d'un graphe simple $G$ est un étiquetage tel que deux sommets adjacents dans $G$ ont deux couleurs distinctes.

**Définition 2.23** *Une* coloration $\ell$ *d'un graphe simple* $G$ *est un étiquetage de* $G$ *tel que pour toute arête* $\{u, v\} \in V(G)$, $\ell(u) \neq \ell(v)$.

Les colorations et étiquetages qu'on introduit dans les chapitres suivants sont définies par des propriétés que doivent vérifier les graphes induits par les sommets de chaque couleur. On considère un graphe simple $G$ et un étiquetage $\ell$ de $G$. Pour toute couleur $i \in \ell(V(G))$, on note $G[i]$ le graphe $G[\ell^{-1}(i)]$ qui est le graphe induit par tous les sommets étiquetés $i$. De plus, pour toutes couleurs distinctes $i, j \in \ell(V(G))$, on note $G[i,j]$ le graphe biparti obtenu à partir du graphe induit $G[\ell^{-1}(i) \cup \ell^{-1}(j)]$ dans lequel on a supprimé toutes les arêtes $\{u,v\}$ telles que $\ell(u) = \ell(v)$.

Par exemple, une coloration d'un graphe $G$ est un étiquetage $\ell$ d'un graphe $G$ tel que pour toute couleur $i \in \ell(V(G))$, $G[i]$ est un stable, i.e., $G[i]$ ne contient pas d'arêtes. Un graphe $G$ est un couplage parfait s'il admet une coloration $\ell$ telle que $\ell(V(G)) = \{1,2\}$ et telle que $G[1,2]$ soit un graphe 1-régulier.

Un étiquetage est dit propre si au moins deux sommets du graphe ont la même couleur.

**Définition 2.24** *Un étiquetage $\ell$ d'un graphe $G$ est un étiquetage* propre *si $|\ell(V(G))| < |V(G)|$.*

## 2.4 Graphes dirigés étiquetés

Comme les graphes non-dirigés, les graphes dirigés que l'on considère sont étiquetés par un ensemble d'étiquettes $L$, qui peut être fini ou infini. On supposera l'ensemble $L$ muni d'un ordre total, noté $<_L$.

Comme pour les graphes non-dirigés, un graphe dirigé étiqueté, noté $(D, \lambda)$ est un graphe dirigé $D$ muni d'une fonction d'étiquetage $\lambda : V(D) \cup A(D) \to L$ qui associe une étiquette à chaque sommet $v \in V(D)$ et à chaque arc $a \in A(D)$. Comme dans le cas des graphes non-dirigés, lorsque les fonctions d'étiquetages n'auront pas à être explicitées, le graphe $(D, \lambda)$ sera dénoté $\mathbf{D}$ ; on dira que le graphe dirigé $D$ est le graphe dirigé *sous-jacent* de $\mathbf{D}$.

Un homomorphisme entre deux graphes dirigés étiquetés est un homomorphisme entre les graphes dirigés sous-jacents qui préserve l'étiquetage.

**Définition 2.25** *Un* homomorphisme *$\varphi$ d'un graphe dirigé étiqueté $\mathbf{D} = (D, \lambda)$ dans un graphe dirigé $\mathbf{D}' = (D', \lambda')$ est un homomorphisme de $D$ dans $D'$ tel que pour tout $x \in V(D) \cup A(D)$, $\lambda(x) = \lambda'(\varphi(x))$.*

*L'homomorphisme $\varphi$ est un isomorphisme entre $\mathbf{D}$ et $\mathbf{D}'$ si $\varphi$ définit un isomorphisme entre $D$ et $D'$.*

**Remarque 2.26** *Comme pour les graphes non-dirigés, on peut assimiler tout graphe dirigé non-étiqueté $D$ au graphe étiqueté $(D, \lambda_\epsilon)$ où $\lambda_\epsilon$ est la fonction d'étiquetage qui associe à chaque sommet et à chaque arc de $D$ l'étiquette $\epsilon$ qui est le mot vide.*

On présente maintenant quelques propriétés que peut avoir l'étiquetage des arcs d'un graphe dirigé. Les termes employés sont similaires à ceux utilisés en théorie des automates.

**Définition 2.27** *Étant donné un graphe dirigé $D$, on dit qu'une fonction d'étiquetage $\lambda$ est* déterministe *(resp.* codéterministe*), si pour tous arcs $a \neq a' \in A(D)$ tels que $s(a) = s(a')$ (resp. $t(a) = t(a')$), $\lambda(a) \neq \lambda(a')$.*

Lorsqu'un graphe dirigé est symétrique, l'étiquetage des arcs peut refléter cette propriété.

**Définition 2.28** *Étant donné un graphe dirigé symétrique $D$, on dit qu'une fonction d'étiquetage $\lambda$ est* symétrique*, s'il existe une fonction $Sym : L \to L$ telle que pour tout arc $a$, $Sym(\lambda(a)) = \lambda(Sym(a))$.*

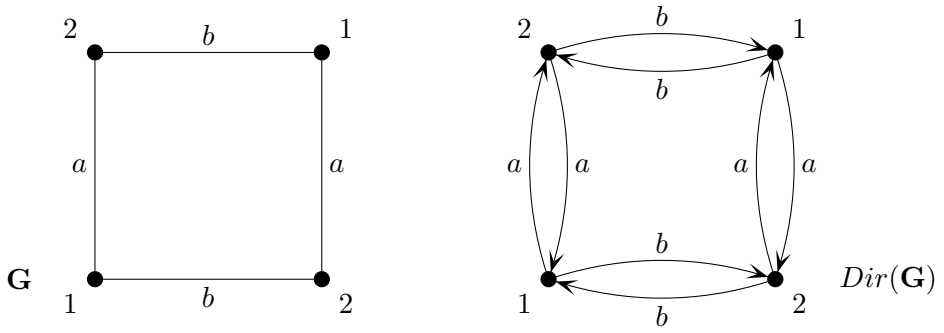



FIGURE 2.1 – Un graphe simple $\mathbf{G}$ et le graphe dirigé symétrique $Dir(\mathbf{G})$ correspondant.

Par la suite, lorsqu'on considérera des graphes dirigés symétriques ayant un étiquetage symétrique, on supposera que tout arc $a$ a une étiquette de la forme $(p, q)$ telle que $Sym(a) = (q, p)$. Étant donnés un graphe dirigé symétrique $D$ et un étiquetage symétrique $\lambda$ de $D$, il est toujours possible d'obtenir un étiquetage $\lambda'$ de cette forme, en posant pour tout arc $a \in A(D)$, $\lambda'(a) = (\lambda(a), Sym(\lambda(a)))$.

## 2.5 Des Graphes non-dirigés aux Graphes Dirigés

À partir d'un graphe $\mathbf{G} = (G, \lambda)$, on peut construire un graphe dirigé symétrique, noté $Dir(\mathbf{G}) = (Dir(G), \lambda')$, défini comme suit. L'ensemble des sommets de $V(Dir(G))$ est l'ensemble $V(G)$ et pour chaque arête $e \in E(G)$ dont les extrémités sont $u$ et $v$, il existe deux arcs $a_{e,u,v}$ et $a_{e,v,u}$ dans $A(Dir(G))$ tels que $s(a_{e,u,v}) = t(a_{e,v,u}) = u$, $s(a_{e,v,u}) = t(a_{e,u,v}) = v$ et $Sym(a_{e,u,v}) = a_{e,v,u}$. Pour tout sommet $u \in V(G) = V(Dir(G))$, $\lambda'(u) = \lambda(u)$ et pour toute arête $e \in E(G)$ dont les extrémités sont $u$ et $v$, $\lambda'(a_{e,u,v}) = \lambda'(a_{e,v,u}) = \lambda(e)$.

En général, on manipulera les graphes dirigés symétriques obtenus à partir de graphes simples. Dans ce cas là, pour tout graphe simple, $\mathbf{G} = (G, \lambda)$, on peut définir $Dir(\mathbf{G}) = (Dir(G), \lambda')$ de la manière suivante. L'ensemble des sommets de $V(Dir(G))$ est l'ensemble $V(G)$ et pour chaque arête $\{u, v\} \in E(G)$, il existe deux arcs $a_{u,v}$ et $a_{v,u}$ dans $A(Dir(G))$ tels que $s(a_{u,v}) = t(a_{v,u}) = u$, $s(a_{v,u}) = t(a_{u,v}) = v$ et $Sym(a_{u,v}) = a_{v,u}$. Pour tout sommet $u \in V(G)$, $\lambda'(u) = \lambda(u)$ et pour toute arête $\{u, v\}$, $\lambda'(a_{u,v}) = \lambda'(a_{v,u}) = \lambda(\{u, v\})$. Il est facile de voir que dans le cas des graphes simples, cette construction permet d'obtenir le même graphe dirigé symétrique que la précédente.

Un exemple de cette construction est présenté sur la Figure 2.1.

# Chapitre 3

# Système distribué

Un système distribué est constitué d'un ensemble d'entités pouvant mémoriser, calculer et communiquer. Ces entités sont des processus, des processeurs, des stations de travail, des sites ou bien encore des capteurs. Ce système est appelé indifféremment un système distribué, un réseau ou bien un graphe. Pour le définir il faut également spécifier le modèle.

Si on ne peut pas garantir que les sommets sont caractérisés par des identificateurs uniques alors on dit que le réseau est anonyme ou partiellement anonyme. L'anonymat peut avoir pour origine la confidentialité de certaines informations pour des raisons de vie privée ou de sécurité. L'anonymat peut avoir également pour origine la fabrication de capteurs avec une petite mémoire et en nombre très important.

On peut ne pas connaître la taille du réseau ; on peut ne connaître qu'une borne supérieure et/ou inférieure de sa taille.

Dans le cas où les sommets sont identifiés on suppose, en général, que la taille des identificateurs est en $O(\log n)$ bits (où $n$ est le nombre de sommets du graphe).

## 3.1 Communication

La communication et plus particulièrement le mode de communication est un élément clé d'un système distribué. Elle peut se faire de différentes façons :

— en mode point-à-point, il y a dans ce cas un seul émetteur et seul récepteur,
— en mode radio, le même message est adressé simultanément à plusieurs destinataires à travers une émission radio,
— par mémoires partagées.

En général on suppose dans les 2 premiers modes que la communication est symétrique, i.e., elle peut avoir lieu dans les 2 directions (canaux bidirectionnels).

Sauf précision contraire, les communications sont supposées fiables, i.e., un message arrive à son destinataire sans être altéré ; on suppose également que la transmission se fait sans déséquencement : les messages arrivent dans l'ordre où ils ont été émis.

Le mode point-à-point peut-être du type asynchrone : l'émetteur envoie son message et n'a pas d'information sur le moment où le destinataire le reçoit effectivement (penser à l'envoi d'une lettre papier, d'un courrier via internet ou bien encore d'un sms). Il peut être également du type synchrone : l'échange d'un message est une opération atomique (l'émission et la réception ne constituent qu'une seule opération comme, par exemple, lors d'une communication téléphonique), cette opération nécessite une synchronisation entre l'émetteur et le destinataire. Parfois on suppose que l'on connaît une borne sur le délai d'acheminement des messages.

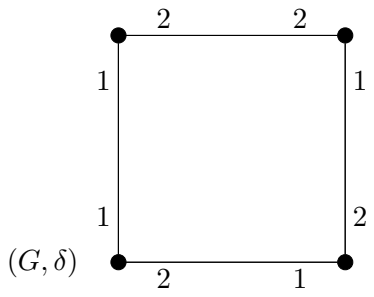



FIGURE 3.1 – Un graphe simple $G$ avec un étiquetage de ports $\delta$.

Le mode radio présente de nombreuses variantes. A priori, dans un réseau radio un processus ne peut entendre un message que si ce message n'a été émis que par exactement un voisin et qu'aucun autre voisin n'a émis de message en même temps. Si aucun message n'est envoyé à un sommet $v$ alors $v$ entend un bruit de fond. Si un sommet $v$ reçoit au moins 2 messages simultanément alors une collision se produit et $v$ perçoit un bruit d'interférence. Si un noeud fait la différence entre un bruit de fond et un bruit d'interférence alors le réseau détecte les collisions sinon il est sans détection de collisions. Finalement, détecter une collision dans un réseau radio c'est être capable pour un sommet de faire la diférence entre recevoir 0 message et recevoir au moins 2 messages.

Par ailleurs, deux autres hypothèses peuvent être formulées :

— lorsqu'un processus émet un message, une couche réseau assure qu'au bout d'un certain temps tous les voisins vont recevoir le message ; on obtient ainsi un mode radio asynchrone ;

— des slots ont été définis de telle sorte qu'un message émis n'entre pas en collision avec un autre message et par conséquent tous les destinataires reçoivent instantanément le message : mode radio synchrone.

Enfin on parle de communication radio single-hop (un seul bond ou saut) dans le cas où l'émetteur peut atteindre l'ensemble des éléments du réseau ou de communication radio multi-hop (plusieurs bonds ou sauts) dans le cas où un message émis ne peut pas atteindre l'ensemble des éléments du réseau. Si on souhaite diffuser le message à tout le réseau cela nécessite des relais.

La communication par mémoire partagée nécessite des méchanismes de synchronisation pour assurer en particulier qu'une opération de lecture dans une mémoire ne puisse pas s'effectuer pendant une opération d'écriture dans cette mémoire.

Dans les chapitres suivants, un système distribué va être représenté par un graphe étiqueté dont les sommets correspondent aux processus et les arêtes aux liens de communication. Dans certains modèles, chaque processus peut distinguer ses voisins, i.e., il peut distinguer les canaux de communication par lesquels il reçoit ou envoie des messages. Pour cela, on suppose que le graphe correspondant au système distribué a un étiquetage des ports défini de la manière suivante. Dans un graphe simple, un étiquetage des ports est un ensemble de fonctions locales qui permettent à chaque sommet de distinguer ses voisins.

**Définition 3.1** *Étant donné un graphe simple* $\mathbf{G}$*, un étiquetage des ports* $\delta$ *est un ensemble de fonctions* $\{\delta_u \mid u \in V(G)\}$ *tel que pour tout sommet* $u$*,* $\delta_u$ *est une bijection entre* $N_G(u)$ *et* $[1, \deg_G(u)]$.

On représente un graphe $\mathbf{G}$ avec un étiquetage $\delta$ en représentant le numéro $\delta_u(v)$ sur l'incidence entre le sommet $u$ et l'arête $\{u, v\}$ (voir Figure 3.1).

## 3.2 Systèmes Distribués avec Communication par échange de messages en mode point-à-point

Un système distribué avec communication par échange de message $(P, C)$ est défini par un ensemble $P$ de processus et un sous-système de communication $C$ qui est représenté par un graphe simple connexe $G$ où les sommets représentent les processus et les arêtes représentent les liens de communication bidirectionnels. Le système est asynchrone : il n'y a pas d'horloge globale et les processus exécutent les instructions à des vitesses arbitraires (on parle également de système décentralisé). Les processus communiquent en échangeant des messages en mode asynchrone, les messages parviennent à leurs destinataires en un temps fini mais arbitraire et chaque processus distingue les canaux par lesquels il reçoit ou envoie chaque message, i.e., il existe une fonction $\delta$ d'étiquetage des ports du graphe $G$. Ainsi, le système de communication $C$ est représenté par un graphe simple $G$ avec un étiquetage des ports $\delta$. Chaque processus a un état initial défini par une fonction d'étiquetage des sommets $\lambda$. Le système distribué $(P, C)$ est alors représenté par un graphe étiqueté $\mathbf{G} = (G, \lambda)$ avec un étiquetage des ports $\delta$. On appellera parfois ce système distribué un *réseau* on le notera $(\mathbf{G}, \delta)$.

**Remarque 3.2** *Un réseau est anonyme si au moins deux sommets ont des étiquettes identiques. Si les processus du réseau ont tous des identités distinctes, l'étiquetage $\lambda$ est tel que pour tout $u \neq v \in V(G)$, $\lambda(u) \neq \lambda(v)$. S'il existe un sommet distingué dans le réseau, alors il existe un sommet $v \in V(G)$ tel que pour tout $u \in V(G)$ différent de $v$, $\lambda(u) \neq \lambda(v)$.*

**Remarque 3.3** *Chaque noeud du réseau est actif ou passif. Initialement, un ou plusieurs noeud peuvent être actifs. Sauf mention contraire, on suppose qu'un noeud actif qui ne reçoit pas de message finit par devenir passif. Un noeud passif qui reçoit une message devient actif. Un noeud actif qui reçoit un message reste actif.*

#### Le modèle $\mathcal{LOCAL}$

Ce modèle autorise des messages de taille quelconque et des calculs locaux sans limite. On suppose également, en général, que le fonctionnement du réseau est synchrone et que tous les processeurs démarrent en même temps. On peut remarquer que, si les sommets sont identifiés, alors tout sommet peut obtenir une image du réseau (ou une carte du réseau en sachant se situer) en un temps proportionnel au diamètre du réseau.

#### Le modèle $\mathcal{CONGEST}$

Ce modèle suppose que chaque message a une taille $O(\log n)$ où $n$ est la taille du réseau.

#### Le modèle $\mathcal{ASYNC}$

La particularité de ce modèle est de considérer des communications asynchrones.

### 3.2.1 Le modèle à base de bips

Ce modèle autorise les noeuds du réseau à communiquer par bips. Le temps est divisé en intervalles synchrones appelés slots. À chaque slot tous les processeurs agissent en parallèle, chaque processeur :

— bipe ou écoute,

— puis effectue un calcul interne.

Soit $v$ un processeur, deux types de collisions peuvent se produire :

— $v$ bipe et simultanément au moins un voisin bipe, cette collision est appelée collision interne ;

— deux voisins de $v$ bipent simultanément, cette collision est appelée collision périphérique.

Plusieurs variantes du modèle à base de bips sont utilisées :

— si un processeur bipe, il y a deux possibilités :

1. il ne peut pas savoir si un autre processeur bipe simultanément, ce cas est noté $B$ ;
2. il peut savoir s'il bipe tout seul ou si au moins un autre processeur bipe en même temps, ce cas est noté $B_{cd}$ ;

— si un processeur écoute, il y a deux possibilités :

1. il peut distinguer entre le silence et au moins un processeur bipe, ce cas est noté $L$ ;
2. il peut distinguer entre le silence, la présence d'un bip et la présence d'au moins deux bips, ce cas est noté $L_{cd}$.

Finalement, un modèle à base de bips est défini en choisissant entre $B$ et $B_{cd}$ d'une part et d'autre part entre $L$ et $L_{cd}$, on obtient ainsi quatre possibilités : $BL$, $B_{cd}L$, $BL_{cd}$, et $B_{cd}L_{cd}$.

**Remarque 3.4** *Finalement, détecter une collision dans un réseau radio c'est faire la différence entre* 0 *message et au moins* 2 *messages. Détecter une collision dans un modèle à base de bips c'est faire la différence entre* 1 *message et au moins* 2 *messages.*

### 3.2.2 Algorithmes utilisant des messages et systèmes de transition

Un système de transition est associé à chaque processus, et ce système de transition interagit avec le système de communication. Les différents types de transitions (ou d'événements) associées à chaque processus sont des transitions *internes*, des transitions d'*envoi* et des transitions de *réception*. Si une transition d'envoi (resp. de réception) est effectuée, un message est créé (resp. consommé).

Étant donné un processus $p$ représenté par le sommet $v$, l'algorithme local du processus $p$, dénoté $\mathcal{D}_p$, est défini par :

— l'ensemble récursif $Q$ des états possibles de $p$,

— l'état initial $\lambda(v)$ de $p$,

— une relation récursive $\vdash_p$ de transitions (transitions internes, transitions d'envoi et transitions de réception).

Pour toute arête $\{v, v'\} \in E(G)$, on note $\mathtt{M}(v, v')$ (resp. $\mathtt{M}(v', v)$) le multi-ensemble des messages en transit entre le processus $p$ (resp. $p'$) correspondant au sommet $v$ (resp. $v'$) et le processus $p'$ (resp. $p$) correspondant au sommet $v'$ (resp. $v$). Initialement, tous les ensembles $\mathtt{M}(v, v')$ sont vides.

Pour tout processus $p$ représenté par un sommet $v$, l'état de $p$ est noté $\mathtt{state}(p)$ et une transition associée au processus $p$ est notée sous la forme :

$$(c, in, m) \vdash_p (d, out, m'),$$

où $c$ et $d$ sont des états, $in$ et $out$ sont des entiers, et $m$ et $m'$ sont des messages. Une telle transition a la signification suivante.

— Si $in = out = 0$, alors $m = m' = \perp$ ($m$ et $m'$ ne sont pas définis) et cela représente une transition interne. Cette transition ne peut être effectuée que si l'état de $p$ est $c$. Lorsque cette transition est effectuée, l'état de $p$ devient $d$.

— Si $in \neq 0$, alors $out = 0$ et $m' = \perp$ ($m'$ n'est pas défini) et cela représente une transition de réception. Cette transition ne peut être appliquée que si l'état de $p$ est $c$ et qu'il existe un message $m \in \mathtt{M}(v', v)$ tel que $\delta_v(v') = in$. Lorsque cette transition est effectuée, l'état du processus devient $d$ et une occurrence de $m$ est effacée de $\mathtt{M}(v', v)$, où $v'$ est le sommet tel que $\delta_v(v') = in$.

— Si $out \neq 0$, alors $in = 0$ et $m = \perp$ ($m$ n'est pas défini) et cela représente une transition d'envoi. Cette transition ne peut être effectuée que si l'état du processus $p$ est $c$. Lorsque cette transition est effectuée, l'état de $p$ devient $d$ et un message $m'$ est ajouté à $\mathtt{M}(v, v')$, où $v'$ est le sommet tel que $\delta_v(v') = out$.

Un algorithme distribué $\mathcal{D}$ utilisant des échanges de messages pour le réseau $(\mathbf{G}, \delta)$ est une collection d'algorithmes locaux $\mathcal{D}_p$ (un pour chaque processus $p$ associé à chaque sommet de $\mathbf{G}$). Un tel algorithme est noté $\mathcal{D} = (\mathcal{D}_p)_{p \in P}$. Une *transition* de l'algorithme distribué est une transition effectuée par un processus du système.

**Remarque 3.5** *On considère généralement des algorithmes distribués tels que deux processus $p, p'$ dans le même état puissent effectuer les mêmes transitions. Cependant, si $p$ peut communiquer avec plus de processus que $p'$, $p'$ ne pourra pas envoyer ou recevoir de messages par le port* $\deg(v)$ *où $v$ est le sommet correspondant à $p$.*

*Ainsi, pour tous processus $p, p'$ correspondant respectivement à des sommets $v, v'$ de même degré, $\vdash_p = \vdash_{p'}$. Si on ne veut pas que deux processus correspondant à des sommets de même degré exécutent les mêmes transitions, on utilise l'étiquetage initial des sommets pour que les processus aient des comportements différents.*

*De cette manière, un algorithme n'a pas besoin d'être défini pour un réseau particulier, mais il suffit de décrire pour tout entier $d$, les transitions que peuvent effectuer les processus de degré $d$.*

*Par la suite, sauf si c'est explicitement indiqué, on ne considère que des algorithmes respectant cette propriété.*

**Remarque 3.6** *Si on considère des systèmes distribués dont les canaux de communication préservent l'ordre des messages (canaux FIFO), il faut voir chaque ensemble $\mathtt{M}(v, v')$ comme une file.*

*Lorsqu'une transition d'envoi $(c, in, m) \vdash_p (d, out, m')$ est effectuée par un processus $p$ correspondant à un sommet $v$, une occurrence de $m'$ est ajoutée en queue de la file correspondant à $\mathtt{M}(v, v')$ où $v'$ est le sommet tel que $\delta_v(v') = out$. Une transition de réception $(c, in, m) \vdash_p (d, out, m')$ peut être effectuée si $m$ est le message en tête de la file $\mathtt{M}(v', v)$ où $v'$ est le sommet tel que $\delta_v(v') = in$ ; auquel cas, la tête de la file $\mathtt{M}(v', v)$ est supprimée.*

*Dans ce chapitre, sauf si c'est explicitement indiqué, on ne considère que des systèmes distribués dont les canaux de communication préservent l'ordre des messages.*

### 3.2.3 Exécution d'un algorithme utilisant des échanges de messages

Une exécution $\rho$ d'un algorithme utilisant des messages est définie par une suite $(\mathtt{state}_0, \mathtt{M}_0)$, $(\mathtt{state}_1, \mathtt{M}_1), \ldots, (\mathtt{state}_i, \mathtt{M}_i), \ldots$ telle que :

— pour tout $i$, $\mathtt{M}_i$ est l'ensemble des ensembles $\mathtt{M}_i(v, v')$ de messages en transit entre les sommets $v$ et $v'$,

— pour toute $\{v, v'\} \in E(G)$, $\mathtt{M}_0(v, v') = \mathtt{M}_0(v', v) = \emptyset$,

— pour tout $i$ et tout processus $p$, $\mathtt{state}_i(p)$ est l'état du processus $p$,

— pour tout processus $p$, $\mathtt{state}_0(p) = \lambda(p)$ est l'état initial de $p$,

— pour toute étape $i$, il existe un unique processus $p$ tel que :

— si $p' \neq p$, alors $\texttt{state}_{i+1}(p') = \texttt{state}_i(p')$,
— $\texttt{state}_{i+1}(p)$ et $\texttt{M}_{i+1}$ sont obtenus à partir de $\texttt{state}_i(p)$ et $\texttt{M}_i$ par une transition effectuée par le processus $p$.

Par définition, $(\texttt{state}_i, \texttt{M}_i)$ est une *configuration* du réseau. L'exécution $\rho$ de l'algorithme est définie par $\mathcal{E} = (\texttt{state}_i, \texttt{M}_i)_{i \geq 0}$.

Une configuration *finale* est une configuration dans laquelle aucune transition n'est applicable. On suppose que dans une configuration finale, aucun message n'est en transit. Autrement dit, pour tout processus $p$ correspondant à un sommet $v$, pour tout état $c$, s'il n'existe aucune transition de la forme $(c, 0, \perp) \vdash_p (d, out, m)$ (i.e., lorsque $p$ est dans l'état $c$, il ne peut ni envoyer de message, ni effectuer de transition interne), alors pour tout message $m$ et pour tout entier $in \in [1, \deg(v)]$, il existe une transition $(c, in, m) \vdash_p (d, 0, \perp)$.

Étant donnée une exécution qui atteint une configuration finale, la *longueur* de l'exécution est la longueur de la séquence $(\texttt{state}_0, \texttt{M}_0), (\texttt{state}_1, \texttt{M}_1), \ldots, (\texttt{state}_i, \texttt{M}_i), \ldots$

**Remarque 3.7** *Les systèmes de transition sont des modèles théoriques de systèmes distribués. Cependant, les algorithmes qu'on va étudier ne seront pas décrits par l'énumération de leurs états et de leurs transitions, mais en utilisant le pseudo-code habituel. La mise à jour de l'état d'un sommet se fera à l'aide de variables et les transitions d'envoi et de réception de messages seront exprimées respectivement à l'aide des primitives* **envoyer** *et* **recevoir**.

### 3.2.4 Trois premiers exemples : diffusion d'une information et calcul d'un arbre recouvrant

Dans cette section, on suppose que l'on dispose d'un système distribué avec une communication par échange de messages en mode point à point. La communication est asynchrone, et chaque noeud peut distinguer ses différents voisins à l'aide de ports d'entrée/sortie. On présente trois algorithmes distribués de diffusion d'une information ou de calcul d'un arbre recouvrant. Cette présentation permet d'introduire des notations, des concepts et des descriptions d'algorithmes distribués que nous utiliseront dans ce texte.

L'algorithme 1 assure la diffusion d'un message dans un réseau sans aucune structuration initiale. Initialement, un unique sommet source dispose du message. Il l'envoie à tous ses voisins. Un sommet qui reçoit pour la première fois le message l'envoie à son tour à tous ses voisins différents de celui qui vient de lui envoyer. Un sommet qui reçoit le message après l'avoir déjà reçu l'ignore.

L'algorithme contient deux actions. Chaque action est consituée d'une expression booléenne -la garde de l'action, et d'un corps (ensemble) d'instructions qui peuvent être exécutées si l'expression booléenne est vraie. Plusieurs gardes peuvent être simultanément vraies auquel cas une est sélectionnée et le corps correspondant est exécutée ; ce qui induit du non déterminisme. Les variables utilisées dans l'algorithme peuvent être déclarées explicitement avant l'algorithme ou bien implicitement lors de leur utilisation dans l'algorithme. Il en est de même pour l'initialisation. L'algorithme 1 contient 2 gardes nommées I et R.

Une telle description d'algorithme s'appelle une description dirigée par les événements.

L'algorithme 2 diffuse un message dans un réseau en utilisant un arbre recouvrant calculé au préalable. Ainsi le message n'est diffusé qu'à travers les arêtes de l'arbre. On suppose que chaque sommet connaît les numéros de ports qui sont dans l'arbre. Le nombre de messages générés par l'algorithme 1 peut être de l'ordre de $n^2$ alors que l'algorithme 2 engendre exactement $n - 1$ messages où $n$ est le nombre de sommets du réseau. Ainsi cet algorithme est optimal pour le nombre de messages générés.

**Algorithme 1:** Un algorithme de diffusion d'une information.

**I** : {Le sommet source $r$ initialise la diffusion du message $M$}
**début**
  **envoyer**$< M >$ par chaque port de sortie de $r$
**fin**
**R** : {Le message $M$ arrive sur le sommet $v$ par le port $q$}
**début**
  **si** $< M >$ *arrive sur* $v$ *pour la première fois* **alors**
    **envoyer**$< M >$ par tous les ports de sortie de $v$ différents de $q$
**fin**

**Algorithme 2:** Un algorithme de diffusion d'un message le long d'un arbre.

**I** : {Le sommet source $r$ initialise la diffusion du message $M$}
**début**
  **envoyer**$< M >$ par chaque port de sortie de $r$ dans l'arbre
**fin**
**R** : {Le message $M$ arrive sur le sommet $v$ par le port $q$}
**début**
  **envoyer**$< M >$ par chacun des ports de sortie de $v$ dans l'arbre différent de $q$
**fin**

**Calcul d'un arbre couvrant en parallèle sans détection de la terminaison.**

Dans l'algorithme 3, chaque processus $v$ dispose des variables suivantes :
— $fils$, $autres$ sont du type ensemble d'entiers ; elles sont initialisées à vide ;
— $pere$ est un entier.

Un sommet distingué $r$ initialise le calcul de l'arbre en diffusant le message **Calcul**. Lorsqu'un sommet reçoit pour la première fois ce message il prend pour père le sommet émetteur en notant le port d'entrée par lequel il a reçu ce message ; il lui envoit le message **Père** pour lui dire qu'il est son fils et diffuse à son tour le message **Calcul** par ses ports de sortie (par convention, le port du sommet racine lui permettant d'accéder à son père est le port 0). Si un sommet est dans l'arbre et qu'il reçoit de nouveau le message **Calcul** alors il indique au processus émetteur que son statut est autre (ni fils ni père) en lui envoyant le message **Déjà-dans-l'arbre**. Ainsi à la fin de l'exécution de l'algorithme, tout sommet connaît le statut de chacun de ses voisins : père, fils ou autre. Il connaît en particulier les liens qui sont dans l'arbre recouvrant.

**Remarque 3.8** *Les trois algorithmes présentés dans cette section se terminent : au bout d'un temps fini tous les processus sont passifs et tous les canaux de communication sont vides. La terminaison de chacun est implicite : aucun sommet ne peut la détecter, c'est à dire être dans un état caractéristique de la terminaison.*

**Remarque 3.9** *On peut également constater avec l'algorithme* 3 *que plusieurs exécutions d'un même algorithme sur un réseau donné peuvent produire des arbres différents. Ceci étant, on obtient toujours un arbre couvrant mais le résultat dépend de la vitesse de progression des messages et de la vitesse d'exécution des instructions sur les processus.*

**Algorithme 3:** Un algorithme de calcul d'un arbre couvrant en parallèle sans détection de la terminaison.

**I** : {Le sommet source $r$ initialise le calcul d'un arbre couvrant}
**début**
  **envoyer**<**Calcul**> par chaque port de sortie ;
  $pere := 0$
**fin**
$\mathbf{R}_1$ : {Un message <**Calcul**> arrive sur le sommet $v$ par le port $q$}
**début**
  **si** <***Calcul***> *arrive sur v pour la première fois* **alors**
    $pere := q$ ;
    **envoyer**<**Père**> par le port $q$ ;
    **envoyer**<**Calcul**> par tous les ports de sortie de $v$ différents de $q$
  **sinon**
    **envoyer**<**Déjà-dans-l'arbre**> par le port $q$ ;
    $autre := autre \cup \{q\}$
**fin**
$\mathbf{R}_2$ : {Un message <**Déjà-dans-l'arbre**> arrive sur le sommet $v$ par le port $q$}
**début**
  $autre := autre \cup \{q\}$
**fin**
$\mathbf{R}_3$ : {Un message <**Père**> arrive sur le sommet $v$ par le port $q$}
**début**
  $fils := fils \cup \{q\}$
**fin**

### 3.2.5 Communication synchrone

Dans un système asynchrone (sans horloge globale), une communication par message est dite synchrone si l'émission d'un message et sa réception sont simultanées : c'est une opération atomique. Autrement dit l'émetteur et le récepteur doivent se synchroniser avant de communiquer (penser au téléphone). Une telle communication a été définie par Hoare [Hoa78]. Des exemples de communications synchrones sont donnés par le Rendez-vous en ADA, par des RPC (Remote Procedure Calls) et des RMI (Remote Method Invocation).

On verra plus loin un exemple de description de ces méchanismes à l'aide de réécritures d'arêtes.

## 3.3 Complexités

Étant donné un algorithme distribué, on peut distinguer plusieurs formes de complexité : complexité en temps, complexité en espace, complexité en messages et complexité en bits.

Pour la complexité en temps, on suppose très souvent que le temps de traitement d'un événement par un processus est égal à 0 et que l'acheminement d'un message nécessite une unité de temps. On calcule alors le temps maximum nécessaire à l'exécution de l'algorithme.

La complexité en espace mesure la quantité de mémoire nécessaire à un processus pour l'exécution de l'algorithme.

La complexité en messages mesure le nombre de messages émis lors de l'exécution de l'algorithme. On prend également le pire des cas.

La complexité en bits compte le nombre de bits qui transitent dans les canaux de communication. Clairement cette complexité est plus fine qu'un simple comptage du nombre de messages, sachant qu'un message peut contenir un bit ou l'encyclopédie universalis.

Dans le cas d'une exécution synchrone d'un algorithme, une ronde (ou bien un round) pour un processus est composée des trois étapes suivantes :

1. envoyer un message aux (à un sous-ensemble des) voisins ;
2. recevoir un message des (d'un sous-ensemble des) voisins ;
3. exécuter un traitement local.

La complexité en temps est alors le nombre maximum de rondes nécessaires pour que tous les processus aient terminé.

**Sur la complexité en bits**

Une étude plus fine de la complexité des messages pour résoudre un problème peut conduire à l'étude du nombre de bits qui transitent à travers un canal de communication.

Par définition, dans un bit-round chaque sommet peut envoyer/recevoir au plus 1 bit à/de chaque voisin. La complexité en bit-rounds d'un algorithme $\mathcal{A}$ est le nombre de bit-rounds nécessaires pour l'exécuter.

Un round d'un algorithme se décompose en 1 ou plusieurs bit-rounds. La complexité en bit-rounds d'un algorithme distribué est une borne supérieure du nombre total de bits qui transitent par chaque canal de communication pendant son exécution. C'est également une borne supérieure de sa complexité en temps. Si on considère un algorithme distribué dont les messages ont une taille en $O(1)$ alors la complexité en temps et la complexité en bit-rounds sont égales modulo une constante multiplicative.

## 3.4 Sur la terminaison dans un système distribué

Contrairement au cadre séquentiel, le mot terminaison dans un cadre distribué peut avoir différentes significations. Si l'on se place d'un point de vue globale la terminaison d'un algorithme est effective quand le système est passif : les processus sont passifs et les canaux de communications sont vides. Si par contre on adopte le point de vue d'un processus alors différentes situations peuvent intervenir et le processus peut se trouver dans l'impossibilité de les détecter même au prix de calculs supplémentaires. De plus, si on analyse plus précisément le but d'un algorithme distribué, on doit distinguer la terminaison de la tâche ou de l'objectif de l'algorithme (états ou valeurs calculées) et la terminaison effective de ce qui permet ces calculs : l'algorithme.

En particulier un sommet peut ne pas avoir besoin de savoir quand l'ensemble des sommets du réseau a fini ; il veut seulement savoir quand lui est dans son état terminal. Finalement on peut décliner le mot terminaison suivant différents niveaux :

— *terminaison implicite* : l'algorithme est globalement terminé mais aucun sommet ne le sait ou ne peut le savoir ;
— *terminaison locale faible* : chaque sommet sait quand il a sa valeur finale (ou bien il a atteint son état terminal), mais doit continuer pour éventuellement transmettre des valeurs à ses voisins (cela peut, par exemple, lui permettre de récupérer de la mémoire ou tout autre moyen mobilisé par l'exécution de l'algorithme) ;
— *terminaison locale* : chaque noeud sait quand il a sa valeur finale et peut arrêter de participer à l'exécution de l'algorithme (cela peut, par exemple, lui permettre de se mettre en mode économie d'énergie) ;
— *terminaison globale* : au moins un noeud sait que tous les noeuds ont calculé leur valeur finale (cela peut permettre de composer certains algorithmes) ;

— *terminaison globale forte* : un noeud sait quand plus rien ne se passe dans le réseau.

## 3.5 Sur l'élection

Un algorithme distribué permet de résoudre le problème de l'élection sur un graphe $\mathbf{G}$ si toute exécution de l'algorithme sur $\mathbf{G}$ termine et si dans la configuration finale, il existe exactement un sommet $v \in V(G)$ qui est dans l'état ÉLU, alors que tous les autres sommets de $\mathbf{G}$ sont dans l'état NON-ÉLU. On suppose par ailleurs, que les états ÉLU et NON-ÉLU sont *finaux*, i.e., une fois qu'un sommet est dans l'un de ces états alors son état n'est plus modifié jusqu'à la fin de l'exécution de l'algorithme.

Le problème de l'élection est fondamental en algorithmique distribuée, en effet un sommet élu peut être utilisé pour centraliser ou diffuser de l'information, pour initialiser l'exécution d'un autre algorithme qui nécessite un sommet distingué (comme, par exemple, l'algorithme de calcul d'un arbre couvrant présenté dans la section 3.6.1, ou bien encore de réinitialiser un réseau), pour prendre une décision de manière centralisée, etc.

## 3.6 Systèmes distribués et réétiquetages

Les algorithmes distribués peuvent être modélisés par des relations de réétiquetage de graphes. On présente ici les modèles les plus généraux que l'on considère.

Les systèmes de réécritures peuvent faciliter la conception d'algorithmes distribués et des structures combinatoires afférentes qui peuvent ensuite être traduits et appliqués dans des modèles plus réalistes. Ils permettent également de définir précisément ce qu'il est possible de résoudre (ou de ne pas résoudre) à l'aide d'un algorithme distribué en fonction des hypothèses que l'on fait en utilisant des outils sur les graphes. Les résultats d'impossibilité sur certains modèles induisent automatiquement des résultats d'impossibilité sur des modèles moins puissants (ce qui est le cas, par exemple, pour les réécritures d'arêtes et les échanges de messages).

De fait, comme on le verra plus loin, ces modèles permettent de décrire des modèles plus classiques comme par exemple le modèle à base de communications synchrones (modèle de Hoare), le modèle d'Angluin ou bien le modèle à état de Dijkstra [Dij74] (un noeud du réseau peut en un pas atomique consulter les états de ses voisins puis changer son propre état, ce qui est désigné ici comme une réécriture d'étoiles ouvertes).

### 3.6.1 Relations de Réétiquetage sur les Arêtes.

**Remarque 3.10** *Ce type de réécritures peut être implémenté automatiquement et d'une manière totalement décentralisée à l'aide de la procédure probabiliste présentée et étudiée dans le chapitre 18 (voir ViSiDiA section 3.11).*

#### Définitions

Une relation de réétiquetage est localement engendrée sur les arêtes si sa restriction aux arêtes détermine son comportement sur tout le graphe. Un pas élémentaire de calcul est représenté par une règle de réétiquetage de la forme décrite par la figure 3.2.

Soit $\mathcal{R}$ un ensemble de règles de réétiquetage comme définies sur la figure 3.2. Soient $\mathbf{G}_1$ et $\mathbf{G}_2$ deux graphes étiquetés. On dit que $\mathbf{G}_2$ est obtenu à partir de $\mathbf{G}_1$ en un pas de calcul, ceci est


FIGURE 3.2 – Représentation graphique d'une règle de réétiquetage $R$ sur des arêtes étiquetées où $X' = f_1(X,Y,Z)$, $Y' = f_2(X,Y,Z) = f_2(Z,Y,X)$, $Z' = f_3(Z,Y,X)$, $f_1$, $f_2$ et $f_3$ sont des fonctions de transition sur des triplets d'états ; parfois $R$ est notée : $R = ((X,Y,Z),(X',Y',Z'))$.

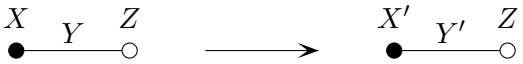



FIGURE 3.3 – Représentation graphique d'une règle de réétiquetage $R$ sur des arêtes ouvertes où $X' = f_1(X,Y,Z)$, $Y' = f_2(X,Y,Z)$, et $f_1$, et $f_2$ sont des fonctions de transition sur des triplets d'états ; parfois $R$ est notée : $R = ((X,Y,Z),(X',Y',Z))$.

noté $\mathbf{G}_1 \mathcal{R} \mathbf{G}_2$, s'il existe une arête $e = \{v_1, v_2\}$ de $\mathbf{G}_1$ et une règle $R = ((X,Y,Z),(X',Y',Z'))$ de $\mathcal{R}$ telles que :

— dans $\mathbf{G}_1$, $X$ est l'étiquette de $v_1$, $Y$ est l'étiquette de $e$, et $Z$ est l'étiquette de $v_2$ ;
— $\mathbf{G}_2$ est obtenu à partir de $\mathbf{G}_1$ en remplaçant l'étiquette $X$ de $v_1$ par $X'$, l'étiquette $Y$ de $e$ par $Y'$, et l'étiquette $Z$ de $v_2$ par $Z'$.

Les étiquettes des autres sommets n'interviennent pas dans ce pas de calcul et restent inchangées. Les sommets de $G$ qui interviennent seront dits *actifs* (et noircis dans les figures). Les calculs faisant intervenir ce type de réétiquetage seront appelés *calculs locaux sur les arêtes (étiquetées)*.

La clôture réflexive et transitive de $\mathcal{R}$ est notée $\mathcal{R}^*$ . Les calculs locaux sur les arêtes étiquetées sont des calculs sur les graphes étiquetés correspondant aux relations de réétiquetage obtenues de cette façon. Une suite $(\mathbf{G}_i)_{0 \leq i \leq n}$ est appelée une suite de réétiquetage modulo (ou un calcul modulo) $\mathcal{R}$ si $\mathbf{G}_i \mathcal{R} \mathbf{G}_{i+1}$ pour tout $0 \leq i < n$ ($n$ est la longueur de la suite). Une suite de longueur 1 est un pas de réétiquetage (ou de calcul).

Soit $\mathcal{R}$ un système de réétiquetage, par définition $\mathcal{R}$ a la propriété de terminaison s'il n'y a pas de suite de réécriture infinie modulo $\mathcal{R}$ (on dit également que $\mathcal{R}$ est noethérien).

**Remarque 3.11** *Par abus de langage, on confond $\mathcal{R}$ ensemble de règles et la relation engendrée par $\mathcal{R}$.*

**Notation.** Soit $\mathcal{R}$ une relation de réétiquetage. Soient $\mathbf{G}$ et $\mathbf{G}'$ deux graphes étiquetés ; $\mathbf{G} \mathcal{R} \mathbf{G}'$ sera également noté $\mathbf{G} \underset{\mathcal{R}}{\Longrightarrow} \mathbf{G}'$.

Ainsi un algorithme distribué dans ce modèle peut être donné par un ensemble (éventuellement infini mais toujours récursif) de règles du type présenté sur la figure 3.2. Une exécution de l'algorithme consiste à appliquer les règles de réétiquetage jusqu'à l'obtention d'un graphe irréductible, c'est à dire un graphe sur lequel aucune règle ne s'applique. Les règles sont appliquées d'une manière asynchrone et dans n'importe quel ordre. Ceci implique en particulier que de nombreuses exécutions différentes peuvent correspondre à une même situation initiale. Ce "non-déterminisme" correspond au non-déterminisme induit par les vitesses non constantes de progression des messages dans les canaux de communications d'un réseau.

On considère également une restriction de ce modèle présentée sur la figure 3.3. Dans cette restriction une étiquette d'un sommet qui apparaît dans la règle ne change pas. Le sommet correspondant est dit *passif* (et marqué blanc sur les figures). Les calculs utilisant ce type de règle sont appelés *calculs locaux sur des arêtes ouvertes*.


FIGURE 3.4 – Règle pour le calcul d'un arbre sans détection de la terminaison.

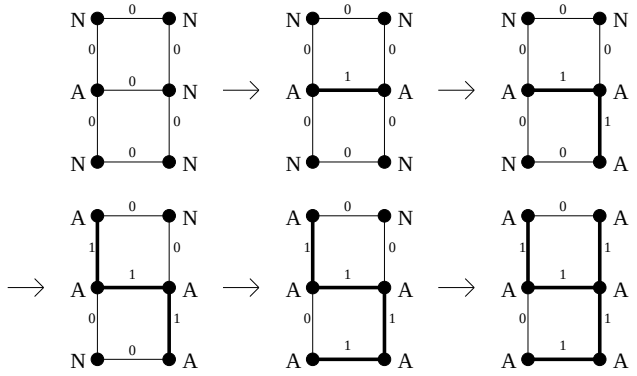


FIGURE 3.5 – Calcul distribué d'un arbre couvrant réalisé par $R$.

Dans les deux types de transformations, les sommets de $G$ ne participant pas à la transformation sont dits *oisifs*.

**Calcul d'un arbre recouvrant en parallèle par réécriture d'arêtes sans détection de la terminaison**

On suppose qu'initialement un seul sommet est étiqueté $A$, tous les autres sont étiquetés $N$ et les arêtes sont étiquetées 0.

À chaque pas de calcul, un sommet $u$ étiqueté $A$ peut activer un voisin $v$ étiqueté $N$. Dans ce cas $u$ garde l'étiquette $A$, $v$ est étiqueté $A$ et l'arête $\{u, v\}$ est étiquetée 1. Plusieurs sommets peuvent être actifs en même temps, des pas en parallèle sont autorisés dès lors qu'ils concernent des arêtes disjointes. Le calcul s'arrête dès qu'il est impossible d'effectuer un pas de calcul.

L'arbre couvrant est défini par les arêtes étiquetées 1.

L'algorithme est codé par le système de réécriture $\mathcal{R}_1 = (L_1, I_1, P_1)$ défini par $L_1 = \{N, A, 0, 1\}$, $I_1 = \{N, A, 0\}$, et $P_1 = \{R\}$ où $R$ est la règle décrite par la figure 3.4.

La figure 3.5 présente un exemple de calcul développé par ce système.

**Un outil utile pour prouver la terminaison d'un système de réécriture.**

**Définition 3.12** *Soit $\mathcal{R}$ une relation binaire sur $X$, soit $\mathcal{R}'$ une relation binaire sur $X'$ et $\varphi$ une application de $X$ sur $X'$. La relation $\mathcal{R}'$ est* compatible *avec la relation $\mathcal{R}$ via $\varphi$ si :*

$$\forall\ a, b \in X,\ a\ \mathcal{R}\ b \quad \Longrightarrow \quad \varphi(a)\ \mathcal{R}'\ \varphi(b).$$

On utilisera cette notion à travers le lemme suivant :

**Lemme 3.13** *Soit $\mathcal{R}$ une relation binaire sur $X$, soit $\mathcal{R}'$ une relation binaire sur $X'$ et $\varphi$ une application de $X$ sur $X'$ telles que $\mathcal{R}'$ est compatible avec $\mathcal{R}$ via $\varphi$. Si $\mathcal{R}'$ a la propriété de terminaison alors $\mathcal{R}$ vérifie également cette propriété.*

En général $\mathcal{R}'$ est acyclique et transitive, on dit dans ce cas que $\mathcal{R}'$ est un ordre compatible. L'ensemble $X'$ sera en général l'ensemble $\mathbb{N}^p$ , $p > 0$, de p-tuples d'entiers positifs. Dans ce cas on utilise l'ordre lexicographique :

**Définition 3.14** *Soit $p$ un entier positif. La relation d'ordre $>_p$ sur $\mathbb{N}^p$ est définie par : $(x_1, x_2, \ldots, x_p) >_p (y_1, y_2, \ldots, y_p)$ si il existe $i$, $1 \leq i \leq p$, tel que $x_1 = y_1$, $\ldots$, $x_{i-1} = y_{i-1}$ et $x_i > y_i$.*

La relation $>_p$ a la propriété de terminaison pour tout entier $p$.

**Propriétés de $\mathcal{R}_1$ .**

**Théorème 3.15** *1. Le système $\mathcal{R}_1$ a la propriété de terminaison.*

*2. Soit $(G, \lambda)$ un graphe étiqueté tel qu'exactement un sommet est étiqueté $A$, les autres étant étiquetés $N$ et les arêtes sont étiquetées 0. Soit $(G, \lambda')$ un graphe irréductible obtenu à partir de $(G, \lambda)$. L'ensemble des arêtes étiquetées 1 dans $(G, \lambda')$ induit un arbre recouvrant de $G$.*

***Preuve :*** *Soit $\varphi : \mathcal{G}_{L_1} \to \mathbb{N}$ l'application qui associe à un graphe son nombre de sommets étiquetés $N$. L'ordre $>$ sur $\mathbb{N}$ est clairement compatible avec $\mathcal{R}_1$ puisque l'application d'une transformation fait décroître strictement le nombre de sommets étiquetés $N$. Donc $\mathcal{R}_1$ a la propriété de terminaison.*

*Pour prouver la deuxième partie du théorème on utilise les invariants suivants :*

*$I_1$. Une arête incidente à un sommet $N$ est étiquetée 0.*

*$I_2$. Excepté dans la situation initiale, chaque sommet étiqueté $A$ a une arête incidente étiquetée 1.*

*$I_3$. Le sous graphe induit par les arêtes étiquetées 1 est un arbre.*

□

La connexité du graphe initial et l'existence d'un sommet $A$ impliquent que le graphe irréductible obtenu ne contient pas de sommet $N$. Ainsi le graphe induit pas les arêtes marquées 1 est un arbre recouvrant de $G$.

**Remarque 3.16** *On peut noter que le calcul peut s'exécuter suivant différents scénarii donnant différents arbres ; ceci étant, on obtient à chaque fois un arbre recouvrant du graphe initial.*

**Remarque 3.17** *Si le graphe a initialement plusieurs sommets étiquetés $A$ alors $\mathcal{R}_1$ construit une forêt couvrante.*

**Remarque 3.18** *On sait que toute exécution de l'algorithme défini par la règle de la figure 3.4 termine sur tout graphe* **G**. *Cependant, on remarque que dans la configuration finale, aucun sommet ne peut déduire à partir de son état que l'exécution est terminée : la terminaison de l'algorithme est dite* implicite.

*Lorsqu'un sommet peut détecter à partir de son état que le résultat recherché a été obtenu, on parle de terminaison* explicite *ou de détection locale de la terminaison globale.*

**Communication Synchrone et Réécriture d'arêtes**

Si on considère un système asynchrone avec une numérotation des ports et une communication par passage de messages en mode synchrone les actions de communication peuvent être codées par des règles ayant la forme donnée par la figure 3.6.

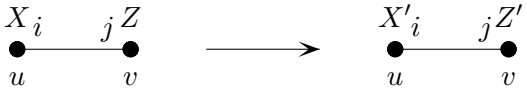



FIGURE 3.6 – La règle $R_{Synch}$ où $X' = f_1(X, i)$, $Z' = f_2(Z, X, j)$, $i$ et $j$ sont les numéros des ports et $f_1$, $f_2$ sont des fonctions de transition.

### 3.6.2 Relations de Réétiquetage sur des étoiles ou sur des étoiles ouvertes.

**Remarque 3.19** *Une implémentation totalement décentralisée de ce type de règles peut être obtenue à l'aide de procédures probabilistes décrites et étudiées dans le chapitre 19 ; de fait ces procédures sont utilisées dans ViSiDiA (section 3.11).*

#### Définitions

On dit qu'un calcul est local sur des étoiles si un pas de calcul permet à un sommet $v$ de modifier son étiquette, les étiquettes de ses voisins et les étiquettes des arêtes incidentes à $v$. L'application d'un tel pas de calcul ne dépend que des étiquettes de $v$, des étiquettes des voisins de $v$ et des étiquettes des arêtes incidentes à $v$.

Une relation de réétiquetage $\mathcal{R}$ est localement engendrée sur des étoiles si sa restriction aux étoiles détermine son comportement sur tout le graphe.

Étant donnée une relation de réétiquetage $\mathcal{R}$, un *pas de calcul* sur le graphe $\mathbf{G}$ est la modification de l'étiquetage d'une étoile de $\mathbf{G}$ pour obtenir un graphe $\mathbf{G}'$ tel que $\mathbf{G}\ \mathcal{R}\ \mathbf{G}'$. Les notions, d'*exécution*, de *configuration* et de *configuration finale* sont définies de la même manière que pour les relations de réétiquetage localement engendrées sur des arêtes.

On définit de la même façon les calculs sur les étoiles ouvertes : un pas de calcul permet à un sommet $v$ de modifier son étiquette et les étiquettes des arêtes incidentes à $v$. L'application d'un tel pas de calcul ne dépend que des étiquettes de $v$, des étiquettes des voisins de $v$ et des étiquettes des arêtes incidentes à $v$.

Une relation de réétiquetage localement engendrée sur des étoiles peut être décrite par un ensemble récursif de règles de réétiquetage où chaque règle permet de modifier les étiquettes d'une étoile du graphe en fonction seulement des étiquettes apparaissant dans l'étoile. Réciproquement, un tel ensemble de règles induit une relation de réétiquetage localement engendrée sur les étoiles. Ainsi, on notera $\mathcal{R}$ l'ensemble des règles de réétiquetage aussi bien que la relation de réétiquetage correspondante.

Une relation de réétiquetage localement engendrée sur les étoiles sera généralement décrite par un ensemble de règles "génériques". Une règle générique permet de décrire la règle de réétiquetage d'une étoile, quel que soit le degré du centre de l'étoile. En général, on considère une étoile générique $(B(v_0), \lambda)$ de centre $v_0$ et la règle est décrite par une précondition portant sur les étiquettes présentes dans $(B(v_0), \lambda)$ et un réétiquetage $(B(v_0), \lambda')$. La règle peut être appliquée dans un graphe $\mathbf{G}$ sur une étoile $B_G(u)$ de centre $u$ si la précondition est vérifiée par $B_G(u)$ et les étiquettes des sommets de $B_G(u)$ sont alors modifiées en fonction du réétiquetage $\lambda'$. En général, on ne mentionne pas dans le réétiquetage les étiquettes des sommets qui ne sont pas modifiées.

#### Un exemple à propos de l'élection dans les arbres

On considère l'algorithme décrit par les deux règles $\mathcal{E}_1$ et $\mathcal{E}_2$ présentées ci-dessous. Cet algorithme est un algorithme d'élection pour les arbres où tous les sommet sont initialement étiquetés $A$. Il est décrit par des réécritures sur des étoiles ouvertes. L'algorithme supprime des feuilles de l'arbre jusqu'à ce qu'il ne reste plus qu'un seul sommet qui est, de fait, le sommet élu.

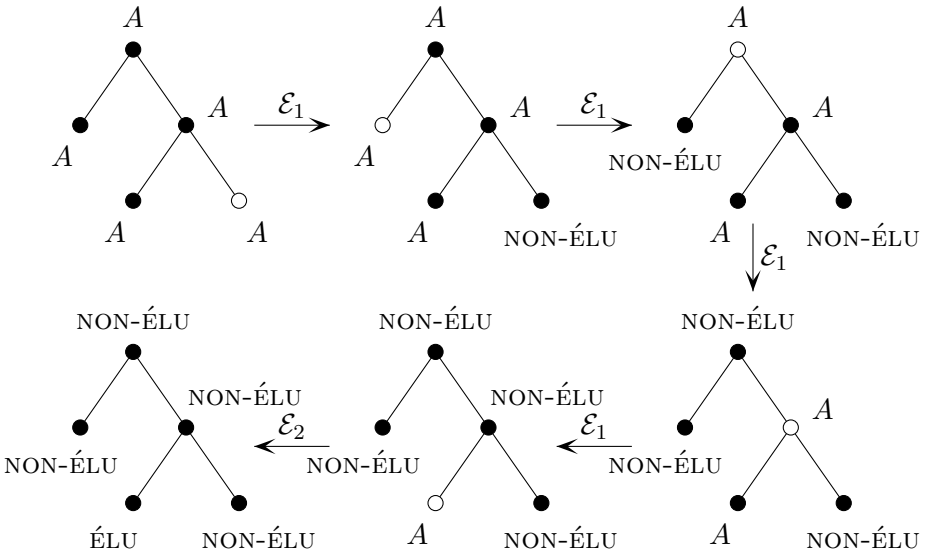



FIGURE 3.7 – Une exécution de l'algorithme décrit par les règles $\mathcal{E}_1$ et $\mathcal{E}_2$. Pour chaque configuration, le centre de l'étoile sur laquelle est appliquée la règle $\mathcal{E}_i$ pour passer à la configuration suivante est le sommet blanc.

$\mathcal{E}_1$: **Règle d'Élagage**

*Précondition :*
— $\lambda(v_0) = A$,
— $\exists! v \in V(B(v_0)) \setminus \{v_0\}$ tel que $\lambda(v) = A$.
*Réétiquetage :*
— $\lambda'(v_0) :=$ NON-ÉLU.

$\mathcal{E}_2$: **Règle d'Élection**

*Précondition :*
— $\lambda(v_0) = A$,
— $\nexists v \in V(B(v_0)) \setminus \{v_0\}$ tel que $\lambda(v) = A$.
*Réétiquetage :*
— $\lambda'(v_0) :=$ ÉLU.

La règle $\mathcal{E}_1$ peut être appliquée dans un graphe $\mathbf{G}$ sur une étoile $B_G(u)$ de centre $u$ si l'étiquette de $u$ est $A$ et si $u$ a un unique voisin étiqueté $A$. Lorsque cette règle est appliquée, seule l'étiquette de $u$ est modifiée et devient NON-ÉLU.

La règle $\mathcal{E}_2$ peut être appliquée dans un graphe $\mathbf{G}$ sur une étoile $B_G(u)$ de centre $u$ si l'étiquette de $u$ est $A$ et si $u$ n'a aucun voisin étiqueté $A$. Lorsque cette règle est appliquée, seule l'étiquette de $u$ est modifiée et devient ÉLU.

Une exécution de cet algorithme est présentée sur la figure 3.7. On va montrer que pour tout arbre $\mathbf{T} = (T, \lambda)$ où tous les sommets sont étiquetés $A$ (i.e., $\forall v \in V(T), \lambda(v) = A$), toute exécution de l'algorithme décrit par les règles $\mathcal{E}_1$ et $\mathcal{E}_2$ permet de résoudre l'élection dans $\mathbf{T}$.

On considère un arbre $\mathbf{T}$ et une exécution de l'algorithme décrit précédemment sur $\mathbf{T}$. Pour chaque étape $i$ de l'exécution, on note $\lambda(v)$ l'étiquette du sommet $v$ après le $i$ème pas de réétiquetage.

On observe qu'à chaque application d'une des deux règles, le nombre de sommets qui n'ont pas

d'étiquettes finales diminue strictement. On est donc assuré que toute exécution de l'algorithme termine. On montre dans le lemme suivant un invariant qui permet de prouver la correction de l'algorithme.

**Lemme 3.20** *Pour toute étape $i$, le graphe induit par l'ensemble des sommets étiquetés $A$ est un arbre et s'il existe un sommet étiqueté $A$ alors aucun sommet n'a l'étiquette* ÉLU.

**Preuve :** On montre ce lemme par récurrence sur $i$. Initialement, tous les sommets sont étiquetés $A$ et puisque $\mathbf{T}$ est un arbre, la propriété est bien vérifiée. On suppose que la propriété est vérifiée à l'étape $i$.

Si la règle $\mathcal{E}_1$ est appliquée à l'étape $i+1$ sur l'étoile $B_T(v)$ de centre $v$, alors $v$ est une feuille de l'arbre induit par les sommets étiquetés $A$ à l'étape $i$. Par conséquent, à l'étape $i+1$, le graphe induit par les sommets étiquetés $A$ est toujours un arbre et aucun sommet n'a l'étiquette ÉLU.

Si la règle $\mathcal{E}_2$ est appliquée à l'étape $i+1$ sur l'étoile $B_T(v)$ de centre $v$, cela signifie que $v$ n'a aucun voisin étiqueté $A$ et par conséquent, l'arbre induit par les sommets étiquetés $A$ à l'étape $i$ ne contient que le sommet $v$. À l'étape $i+1$, il n'y a aucun sommet étiqueté $A$ et la propriété est vraie. □

On considère la configuration finale d'une exécution de l'algorithme sur $\mathbf{T}$. S'il existe encore des sommets étiquetés $A$, alors d'après le lemme 3.20, le graphe induit par les sommets étiquetés $A$ est un arbre et ou bien, cet arbre est réduit à un sommet $v$ et la règle $\mathcal{E}_2$ peut être appliquée sur l'étoile $B_T(v)$ de centre $v$, ou bien il existe un sommet $v$ qui est une feuille dans cet arbre et la règle $\mathcal{E}_1$ peut être appliquée sur l'étoile $B_T(v)$ de centre $v$. Par conséquent, dans la configuration finale, tous les sommets ont l'étiquette ÉLU ou NON-ÉLU. De plus, puisqu'à chaque étape, l'étiquette d'un seul sommet est modifiée, le dernier sommet qui a changé d'étiquette a nécessairement pris l'étiquette ÉLU, et d'après le lemme 3.20, les autres sommets ont l'étiquette NON-ÉLU.

L'algorithme décrit par les règles $\mathcal{E}_1$ et $\mathcal{E}_2$ permet donc de résoudre le problème de l'élection dans la famille des arbres.

### 3.6.3 Sérialisation

Les notions d'exécution introduites ci-dessus correspondent à des exécutions *séquentielles* des algorithmes. Cependant, il faut noter que si des règles de réétiquetage peuvent être appliquées sur deux arêtes (ou étoiles) disjointes, i.e., qui n'ont aucun sommet en commun, alors ces règles peuvent être appliquées simultanément. Ainsi, on peut définir une exécution distribuée en disant que deux pas de réétiquetage consécutifs appliqués à des arêtes (ou des étoiles) disjointes peuvent être appliqués dans n'importe quel ordre. On dit que de tels pas commutent et peuvent être appliqués de manière concurrente.

Plus généralement, deux suites de réétiquetage partant du même graphe étiqueté et telles qu'on peut obtenir l'une à partir de l'autre par ce type de commutation aboutiront au même résultat, i.e., au même graphe étiqueté final. Ainsi, la notion d'exécution définie par une suite de réétiquetages peut être vue comme une *sérialisation* d'une exécution distribuée donnée.

Pour l'analyse de nos algorithmes, on considérera des exécutions séquentielles, mais il est important de se rappeler qu'elles peuvent être distribuées.

Une présentation du cadre des monoïdes de commutation pour étudier certains aspects de ces exécutions distribuées est faite dans [DM97].


FIGURE 3.8 – Forme d'une règle de calcul dans le modèle d'Angluin, où $X' = f(X, Z, i, 0)$, $Z' = f(Z, X, j, 1)$, $i$ et $j$ sont les numéros des ports et $f$ est la fonction de transition.


FIGURE 3.9 –

## 3.7 Sur les liens entre le modèle d'Angluin et les réécritures d'arêtes

Les résultats présentés dans cette section sont issus de [CM10].

Le modèle d'Angluin [Ang80] est défini de la façon suivante. La communication se fait par passage de messages du type point-à-point et chaque processus peut distinguer ses voisins, ainsi on suppose qu'il y a une numérotation des ports. Il n'y a pas d'horloge globale. Un pas de calcul consiste en un échange de messages entre 2 sommets voisins. Cet échange autorise les 2 sommets à changer d'état en fonction des états de ces 2 sommets sans affecter les états des autres sommets.

Pour casser la symétrie entre 2 sommets voisins un "pile-ou-face" non-déterministe, qui de fait rompt la symétrie, peut être effectué entre ces sommets. Soit $A$ l'ensemble des messages, le comportement d'un processus de degré $d$ est défini par un couple $(Q, M)$ tel que $Q$ est un ensemble non vide (éventuellement infini) d'états et $M$ est une application de $Q \times [1, d]$ dans les sous-ensembles de $A \times A \times Q$, appelée fonction de communication.

Finalement un pas de calcul a la forme donnée sur la figure 3.8.

Les numéros des ports des arêtes incidentes à un noeud peuvent être utilisés par le processus logé sur ce noeud pour mémoriser les étiquettes associées aux arêtes correspondantes. Cette remarque implique que les réécritures d'arêtes munies d'une numérotation des ports peuvent simuler les réécritures d'arêtes, i.e., les règles ayant la forme donnée sur la figure 3.9 simulent les règles ayant la forme donnée sur la figure 3.10 avec $\alpha(i) = \alpha(j) = Y$ et $\alpha'(i) = \alpha'(j) = Y'$.

Finalement :

**Lemme 3.21** *Les calculs locaux définis dans le modèle d'Angluin peuvent simuler les calculs locaux sur des arêtes étiquetées.*

## 3.8 Communications synchrones et réécritures d'arêtes

On rappelle qu'une communication est dite synchrone si l'émission d'un message et sa réception sont coordonnées de telle sorte qu'elles ne forment qu'une seule transition du système (opération atomique). Cette hypothèse implique que la symétrie entre 2 sommets voisins peut être cassée d'une manière non-déterministe : si 2 sommets voisins veulent se parler simultanément, le mécanisme de communication fera en sorte que l'un sera émetteur et l'autre récepteur. Finalement dans un tel système les événements peuvent être codés par des règles du type décrit sur la figure 3.11. Ce


FIGURE 3.10 –

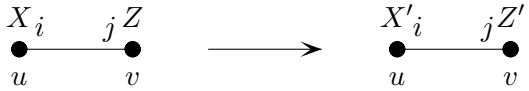



FIGURE 3.11 – La règle $R_{Synch}$ où $X' = f_1(X, i)$, $Z' = f_2(Z, X, j)$, $i$ et $j$ sont les numéros de ports correspondant et $f_1$, $f_2$ sont des fonctions de transitions.

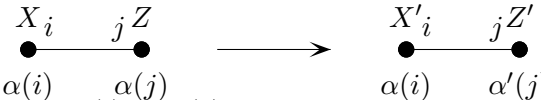



FIGURE 3.12 – On suppose que $\alpha(i)$ et $\alpha(j)$ sont les états associés à l'arête à travers les ports $i$ et $j$ ; l'état actuel de l'arête $e$ est $\alpha'(j)$.

modèle a été défini par Hoare (C. A. R. Hoare, Communicating sequential processes, Communications of the ACM, vol. 21, no 8, 1978, p. 666 à 677).

Par définition, le modèle d'Angluin est plus puissant que le modèle à base de communications synchrones.

Dans le cas de passage de messages en mode synchrone avec une numérotation des ports, on ne peut pas mémoriser l'étiquette de l'arête $e = \{u, v\}$ sur ses 2 extrémités simultanément : seulement une extrémité peut être réétiquetée (avec les notations de la règle $R_{Synch}$ cela correspond à $Z' = f(Z, X, j)$ sur le sommet $v$). On mémorise le nouvel état de l'arête sur le sommet $v$ associé au port $j$. On est alors amené lors de la synchronisation suivante sur cette arête à déterminer sur quel sommet se trouve l'état actuel de l'arête $e$ en examinant les états de $u$ et $v$.

Une solution naturelle à ce problème est obtenue en associant des compteurs $c(i)$ et $c(j)$ à l'état de l'arête mémorisé sur les extrémités de l'arête. La valeur initiale de ces compteurs est 0, et à chaque fois qu'une transformation est réalisée sur l'arête $e$ l'état de l'arête est l'état associé aux extrémités avec la plus grande valeur de compteur. Quand une transformation est appliquée à l'arête, on calcule le nouvel état, on ajoute 1 au compteur de valeur maximale et on associe cette nouvelle valeur au nouvel état de $e$ sur $v$.

Pour la règle $R_{Synch}$, soit $k$ tel que $c(k) = Max\{c(i), c(j)\}$, avant l'application de $R_{Synch}$ l'état de l'arête est $\alpha(k)$. Après l'application de $R_{Synch}$ le nouvel état de $e$ est $\alpha'(j)$ et son compteur est $c(k) + 1$ (voir Figure 3.13).

En fait, on peut décider sur quel sommet se trouve l'état de l'arête avec seulement 3 valeurs $\{0, 1, 2\}$ en appliquant l'algorithme suivant : si à une extrémité la valeur du compteur est 0 et à l'autre extrémité la valeur est 1 alors la plus grande valeur est 1 ; si c'est 1 et 2 alors le plus grand est 2 ; enfin si c'est 0 et 2 alors le plus grand est 0. Si l'état à jour de l'arête (l'état associé au compteur de valeur maximale) avant l'application de la transformation est sur le sommet receveur alors la nouvelle valeur du compteur associé au nouvel état de l'arête est égale à l'ancienne valeur. Si non, pour obtenir la nouvelle valeur du compteur on incrémente modulo 3 la valeur maximale avant la transformation.

Ceci prouve que l'on peut simuler des calculs locaux sur des arêtes ouvertes par des calculs associés à des passages de messages en mode synchrone.

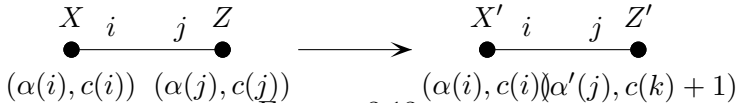



FIGURE 3.13 –

## 3.9 Connaissances Initiales

Dans un système distribué un sommet n'a, a priori, qu'une connaissance locale (donc très limitée) de son environnement. L'existence d'algorithmes distribués ou bien leur efficacité va dépendre de la connaissance globale dont on disposera pour les concevoir. Par ailleurs l'utilisation ou non d'une connaissance globale fera qu'un algorithme pourra être utilisé ou non sur des réseaux dynamiques.

**Exemple 3.22** *Si on sait qu'un graphe* $\mathbf{G}$ *est un arbre, alors il existe un algorithme codé par une relation de réétiquetage localement engendrée sur les étoiles qui permet de résoudre l'élection dans* $\mathbf{G}$ *: c'est l'algorithme présenté dans la section 3.6.2.*

Par la suite, on va distinguer deux types de connaissances initiales : les connaissances *globales* et les connaissances *locales*. Une connaissance est *globale* si c'est une connaissance qui porte sur le graphe $\mathbf{G}$. Par exemple, savoir si le graphe $\mathbf{G}$ sur lequel est exécuté un algorithme est un arbre est une connaissance globale, la taille du graphe ou une borne de sa taille sont d'autres exemples de connaissance globale. Un algorithme $\mathcal{A}$ permet de résoudre un problème $\mathcal{P}$ sur une famille avec une connaissance globale $\mathcal{C}$ si pour tout graphe $\mathbf{G}$ tel que $\mathcal{C}$ est une propriété de $\mathbf{G}$ ($\mathbf{G}$ est un arbre, par exemple), l'algorithme $\mathcal{A}$ permet de résoudre le problème $\mathcal{P}$ sur $\mathbf{G}$.

On va plus particulièrement étudier les connaissances globales suivantes et étudier ce qu'elles permettent de calculer.

- la topologie du graphe : l'algorithme dépend de la topologie du graphe ;
- la taille du graphe : l'algorithme dépend de la taille du graphe ;
- une borne serrée $B$ sur la taille du graphe : l'algorithme doit fonctionner pour tout graphe $\mathbf{G}$ tel que $|V(G)| \leq B < 2|V(G)|$ ;
- une borne $B$ sur la taille (ou le diamètre) du graphe : l'algorithme doit fonctionner pour tout graphe $\mathbf{G}$ tels que $|V(G)| \leq B$ (ou $D(G) \leq B$) ;
- aucune connaissance globale, i.e., l'algorithme doit fonctionner sur tous les graphes.

Une connaissance est *locale* si chaque sommet $v$ d'un graphe $\mathbf{G}$ dispose d'une information qui dépend de $v$ ou plus généralement il existe un entier $k$ tel que pour tout sommet $v$ cette information peut être calculée en fonction de la connaissance intiale des sommets à distance au plus $k$ de $v$. Par exemple, si initialement chaque sommet connaît son degré, alors on parle de connaissance initiale *locale* : un sommet $v$ ne peut pas déduire quelles sont les connaissances initiales des autres sommets à partir de l'information dont il dispose initialement. Généralement, les connaissances initiales locales sont stockées dans les étiquettes initiales de chaque sommet. Par exemple, si dans un graphe $\mathbf{G} = (G, \lambda)$, les sommets connaissent initialement leurs degrés, on suppose que pour tout sommet $v \in V(G)$, $\lambda(v) = (d(v), \lambda'(v))$ où $d(v)$ est le degré de $v$ dans $\mathbf{G}$. Par la suite, la seule connaissance initiale locale dont on va étudier l'importance est la connaissance du degré.

Étant donné un problème $\mathcal{P}$, on remarque qu'un algorithme $\mathcal{A}$ permet de résoudre $\mathcal{P}$ sans connaissance initiale si et seulement si $\mathcal{A}$ permet de résoudre $\mathcal{P}$ sur la famille de tous les graphes. De même, si $\mathcal{A}$ permet de résoudre $\mathcal{P}$ avec la connaissance initiale de la taille, alors pour tout entier $n$, il existe un algorithme $\mathcal{A}_n$ qui permet de résoudre le problème $\mathcal{P}$ sur la famille des graphes de taille $n$. Par ailleurs, si $\mathcal{A}$ permet de résoudre $\mathcal{P}$ avec la connaissance initiale de la topologie, alors pour tout graphe $\mathbf{G}$, il existe un algorithme (qui dépend de $\mathbf{G}$) qui permet de résoudre $\mathcal{P}$ sur $\mathbf{G}$. Par la suite, on va donc dire indistinctement qu'on peut résoudre un problème $\mathcal{P}$ avec une connaissance initiale ou qu'on peut résoudre $\mathcal{P}$ sur la famille des graphes disposant de la même connaissance initiale (les graphes de taille donnée, par exemple).

Un algorithme $\mathcal{A}$ qui permet de résoudre l'élection (ou le nommage) sur tous les graphes de la famille $\mathcal{F}$ est un algorithme *universel* d'élection (ou de nommage) pour $\mathcal{F}$. Cependant, dans les différents modèles qu'on considère, il existe des graphes qui n'admettent pas d'algorithme d'élection (ou de nommage). Parfois, on souhaite donc pouvoir détecter si un graphe admet un algorithme d'élection (ou de nommage). Ainsi, un algorithme $\mathcal{A}$ est un algorithme *effectif* d'élection (ou de nommage) pour une famille $\mathcal{F}$ si pour toute exécution de $\mathcal{A}$ sur tout graphe $\mathbf{G} \in \mathcal{F}$, $\mathcal{A}$ résout l'élection (ou le nommage) sur $\mathbf{G}$, ou alors $\mathcal{A}$ détecte qu'il n'existe pas d'algorithme d'élection ou de nommage pour $\mathbf{G}$. La définition suivante rappelle les propriétés souhaitées pour ce type d'algorithme.

**Définition 3.23** *Un algorithme $\mathcal{A}$ est un algorithme* universel *d'élection (resp. de nommage) pour une famille $\mathcal{F}$ si pour tout graphe $\mathbf{G} \in \mathcal{F}$, toute exécution de $\mathcal{A}$ sur $\mathbf{G}$ permet de résoudre l'élection (resp. le nommage) sur $\mathbf{G}$.*

*Un algorithme $\mathcal{A}$ est un algorithme* effectif *d'élection (resp. de nommage) pour une famille $\mathcal{F}$ si pour tout graphe $\mathbf{G} \in \mathcal{F}$, toute exécution de $\mathcal{A}$ sur $\mathbf{G}$ permet ou bien de résoudre l'élection (resp. le nommage) sur $\mathbf{G}$, ou bien de détecter qu'il n'existe pas d'algorithme d'élection (resp. de nommage) pour $\mathbf{G}$.*

**Remarque 3.24** *Étant donné un algorithme effectif d'élection (ou de nommage) pour une famille de graphes $\mathcal{F}$, si lors d'une exécution de $\mathcal{A}$ sur un graphe $\mathbf{G} \in \mathcal{F}$, un sommet $v \in V(G)$ est dans un état indiquant qu'il n'existe pas d'algorithme d'élection (ou de nommage) pour $\mathbf{G}$, alors $v$ ne modifiera pas son état par la suite, i.e., les états indiquant qu'on ne peut pas résoudre l'élection (ou le nommage) sont finaux.*

## 3.10 JBotSim

Une plateforme est développée au LaBRI par Arnaud Casteigts : il s'agit de JBotSim. C'est une plateforme de simulation, permettant de valider des algorithmes distribués, aussi bien dans des réseaux statiques que dynamiques (tels que des réseaux de capteurs sans fil, des réseaux de drones ou de robots). Avec cet outil, il est possible de prototyper rapidement une solution algorithmique et de la tester en interagissant avec l'environnement, par exemple en ajoutant, en déplaçant ou en supprimant des noeuds ou des liens de communication alors même que l'algorithme est en train de s'exécuter. Il permet ainsi de valider à moindre coût une solution avant de la réaliser matériellement, ce qui peut s'avérer très coûteux. Au delà du test, il a vocation à permettre la réalisation de démonstrations ou de vidéos pour échanger des idées, et au delà, à communiquer sur un résultat. Cet outil est principalement développé au LaBRI et commence à être utilisé pour la recherche et pour l'enseignement dans un nombre croissant de sites : le site bordelais (plusieurs UE de Master 2 et d'IUT, ainsi que les recherches sur les réseaux mobiles au LaBRI). Il est aussi utilisé en enseignement et en recherche à Marseille (LIF), à Paris (LIP6), à Ottawa (SEECS) et à San Sebastian (BaiLab).

## 3.11 ViSiDiA

Au LaBRI, sous la direction de Mohamed Mosbah, un projet logiciel pour la simulation et la visualisation d'algorithmes distribués est en cours depuis quelques années. Ce projet s'appelle ViSiDiA (Visualization and Simulation of Distributed Algorithms) et est disponible sur le site `http://www.labri.fr/projet/visidia`.

Cet outil permet d'implémenter des algorithmes distribués dans différents modèles. Un algorithme est généralement implémenté dans un système asynchrone où les processus communiquent en échangeant des messages. Cependant, il est aussi possible d'implémenter des algorithmes codés par des relations de réétiquetage de graphes ; pour cela, des méthodes de synchronisation locale probabilistes sont utilisées (chapitre 18 et chapitre 19).

# Chapitre 4

# Le temps dans les systèmes distribués

Étant donné un processeur, les événements qui s'y déroulent sont naturellement ordonnés par son horloge (ou compteur d'événements). En général, un réseau ne dispose pas d'une telle horloge (globale) à laquelle n'importe quel élément peut accéder instantanément. Par ailleurs, pouvoir déterminer la causalité entre événements dans un réseau est fondamental, en particulier pour pouvoir analyser ou mettre au point des algorithmes ou des systèmes distribués.

Pour déterminer cette causalité on définit des horloges logiques dont les valeurs sont associées aux événements et sont appelées estampilles.

## 4.1 Horloges de Lamport

Ce que l'on peut dire à propos d'un ordre éventuel entre 2 événements sur 2 noeuds distincts c'est :

- l'émission d'un message par un noeud $u$ est antérieure à la réception de ce message par le noeud destinataire $v$,
- tout événement sur $u$ antérieur à l'émission d'un message par $u$ est antérieur à tout événement sur $v$ postérieur à la réception de ce message par $v$, $v$ étant noeud destinataire de ce message.

On considère la clôture transitive de cette relation "être antérieur" que l'on appelle relation de causalité et que l'on note $\mathcal{C}$.

Si on souhaite ordonner des événements quelconques dans le réseau, le "mieux" que l'on puisse faire est de considérer l'ordre partiel engendré par les 2 relations ci-dessus.

Supposons que chaque noeud $u$ dispose d'une horloge logique (compteur d'événements) dont la valeur est notée $h_u$. Si on associe à tout événement sur un noeud $u$ la valeur $h_u$ de cette horloge et que l'on joint $h_u$ à l'émission d'un message alors on peut "recaler" les horloges en appliquant la procédure suivante.

L'horloge progresse naturellement sur chaque noeud par incrémentation à chaque événement interne sur ce noeud par exemple. On joint la valeur de l'horloge $h_u$ à tout message émis par $u$ et lors de la réception de ce message par $v$, la nouvelle valeur de l'horloge de $h_v$ est $1 + Max(h_v, h_u)$.

Initialement pour tout noeud $u$ : $h_u := 0$.

Ces horloges logiques ont été introduites par Lamport. Soit $e$ un événement sur un noeud du réseau. Soit $h(e)$ la valeur de l'horloge sur le noeud où a lieu l'événement $e$.

Soient $e_1$ et $e_2$ deux événements sur le réseau. on a :

$$e_1 \mathcal{C} e_2 \Rightarrow h(e_1) < h(e_2).$$

**Algorithme 4:** Horloges logiques de Lamport.

{Un événement interne à $u$ se produit}
**début**
  $h_u := h_u + 1$ ;
  $u$ change d'état
**fin**
{Le noeud $u$ envoie un message $M$ }
**début**
  $h_u := h_u + 1$ ;
  **envoyer**$< M, h_u >$
**fin**
{Le noeud $u$ reçoit un message $M$ }
**début**
  **recevoir**$< M, h >$ ;
  $h_u := max(h_u, h) + 1$
**fin**

La réciproque est fausse.

## 4.2 Vecteur d'horloges

L'ordre associé aux horloges de Lamport ne permet pas, en comparant les valeurs des horloges, de déduire une causalité entre 2 événements. Pour y remédier on va associer aux événements un vecteur d'horloges qui permettra de décider s'il y a une relation de causalité entre 2 événements.

On considère un réseau avec $n$ processeurs numérotés de 1 à $n$. Chaque processeur $i$ dispose d'une horloge locale $h_i(i)$. À chaque événement sur le processeur $i$ va être associé un vecteur d'horloges (horloge de Mattern) $(h_i(k))_{1\leq k\leq n}$ de la façon suivante.

- Comme pour les horloges de Lamport, un processeur qui exécute une action incrémente son horloge.
- Un processeur $i$ mémorise pour chaque processeur $k$ du réseau la valeur maximale, $h_i(k)$, de l'horloge de $k$ dont il a connaissance.
- Lorsque le processeur $i$ émet un message il lui joint comme estampille le vecteur d'horloges $(h_i(k))_{1\leq k\leq n}$.
- Lorsque le processeur $i$ reçoit un message avec l'estampille $(v_k)_{1\leq k\leq n}$, il met à jour son vecteur comme suit : pour chaque $1 \leq k \leq n$ $k \neq i$ $h_i(k) := max(h_i(k), v_k)$, et $h_i(i) := max(v_i, h_i(i)) + 1$, i.e. $h_i(i) := v_i + 1$ puisque $i$ est le seul processeur a pouvoir faire progresser sa propre horloge $h_i(i)$.

On considère les relations définies sur les $n$-uplets par :
étant donnés 2 $n$-uplets $v = (v_k)_{1\leq k\leq n}$ et $v' = (v'_k)_{1\leq k\leq n}$
$v \leq v'$ si $\forall k$ $1 \leq k \leq n$ $v_k \leq v'_k$ ;
de plus : $v < v'$ si $v \leq v'$ et $\exists\, k \quad 1 \leq k \leq n$ tel que : $v_k < v'_k$.
On a la propriété fondamentale suivante :

**Proposition 4.1** *Soient $e_1$ et $e_2$ 2 événements se produisant dans le réseau ; soient $v_1$ et $v_2$ les estampilles associées. Il y a une relation de causalité entre $e_1$ et $e_2$, i.e., $e_1 \mathcal{C} e_2$, si et seulement si $v_1 < v_2$.*

# Chapitre 5

# Diffusion, centralisation et routage d'une information

Les opérations de diffusion, de centralisation et de routage d'informations dans un réseau sont fondamentales.

## 5.1 Diffusion

On suppose qu'un sommet détient une information particulière qu'il doit envoyer à tous les autres sommets du réseau. les 2 algorithmes présentés dans la section 3.2.4 réalisent cette tâche.

## 5.2 Centralisation

On suppose que l'on dispose d'un arbre recouvrant avec une racine $r$. Pour qu'un réseau puisse centraliser une information sur le sommet $r$ il suffit que d'une manière inductive dès qu'un sommet détient une information, cette information remonte à travers les pères des sommets jusqu'au sommet $r$.

## 5.3 Routage

Le routage d'un message consiste en la donnée d'un algorithme disposé sur chaque sommet du réseau et permettant à un message émis d'atteindre son destinataire à travers une série de noeuds du réseau. Des tables de routage peuvent être disposées sur chaque sommet. Un exemple classique est le routage par le plus court chemin.

# Chapitre 6

# À propos du calcul d'un arbre recouvrant

Disposer d'un arbre recouvrant dans un réseau permet de synchroniser et plus généralement de contrôler l'exécution d'un algorithme distribué, ou bien encore de faire du routage de messages. Un tel arbre permet de diffuser ou de centraliser des informations d'une manière optimale du point de vue du nombre de messages générés. Lorsque l'on dispose d'un arbre recouvrant on peut également organiser l'élection d'un processus.

Les calculs locaux présentés dans ce chapitre sont extraits de [LM92, LMS99, LMS95].

## 6.1 Calcul d'un arbre recouvrant en parallèle sans détection de la terminaison

Soit $G$ un graphe. L'idée de l'algorithme est simple. On suppose qu'il existe un sommet distingué $r$ qui lance le calcul d'un arbre réduit initialement à $r$. Un sommet qui est dans l'arbre envoie un message à chacun de ses voisins. Un sommet qui reçoit pour la première fois un message devient une feuille de l'arbre ayant pour père le sommet qui a envoyé le message. L'algorithme se termine : au bout d'un temps fini il n'y a plus de message dans les canaux et tous les sommets sont passifs. Lorsque l'algorithme est terminé chaque sommet connaît le statut de ses arêtes incidentes : dans l'arbre recouvrant ou non.

### 6.1.1 Algorithme décrit par passage de messages

Soit $G$ un graphe et soit $r$ un sommet de $G$ appelé sommet source. On suppose que le sommet $r$ initialise le calcul de l'arbre recouvrant en envoyant le message **Calcul** à travers chacun de ses ports de sortie. Quand un sommet reçoit pour la première fois le message **Calcul** il note que son père se situe de l'autre côté du port par lequel il a reçu ce message et il lui dit en envoyant le message **Père** (par convention le port du père du sommet source est 0). À la réception du message **Père** un sommet note qu'un fils se trouve de l'autre côté du port correspondant. Si un sommet, qui a reçu le message **Calcul**, reçoit de nouveau le message **Calcul** alors il indique à l'expéditeur qu'il est déjà dans l'arbre ce qui permet à celui-ci de noter que ce sommet est "autre" (ni père ni fils).

Chaque sommet $v$ dispose des variables suivantes : *père* qui indique au sommet le port qui pointe vers son père, $fils$ qui est du type ensemble de numéros de ports et qui contient les numéros

des ports menant aux fils de $v$ (initialement $fils$ est vide) et $autre$ qui indique au sommet $v$ ses voisins qui sont ni fils ni père (initialement $autre$ est vide).

**Algorithme 5:** Un algorithme de calcul d'un arbre couvrant sans détection de la terminaison.

**I** : {Le sommet source $r$ initialise le calcul d'un arbre couvrant}
**début**
  **envoyer**<**Calcul**> par chaque port de sortie ;
  $père := 0$
**fin**
$\mathbf{R}_1$ : {Un message <**Calcul**> arrive sur le sommet $v$ par le port $q$}
**début**
  **si** <***Calcul***> *arrive sur* $v$ *pour la première fois* **alors**
    $père := q$ ;
    **envoyer**<**Père**> par le port $q$ ;
    **envoyer**<**Calcul**> par tous les ports de sortie de $v$ différents de $q$
  **sinon**
    **envoyer**<**Déjà-dans-l'arbre**> par le port $q$
    $autre := autre \cup \{q\}$
**fin**
$\mathbf{R}_2$ : {Un message <**Déjà-dans-l'arbre**> arrive sur le sommet $v$ par le port $q$}
**début**
  $autre := autre \cup \{q\}$
**fin**
$\mathbf{R}_3$ : {Un message <**Père**> arrive sur le sommet $v$ par le port $q$}
**début**
  $fils := fils \cup \{q\}$
**fin**

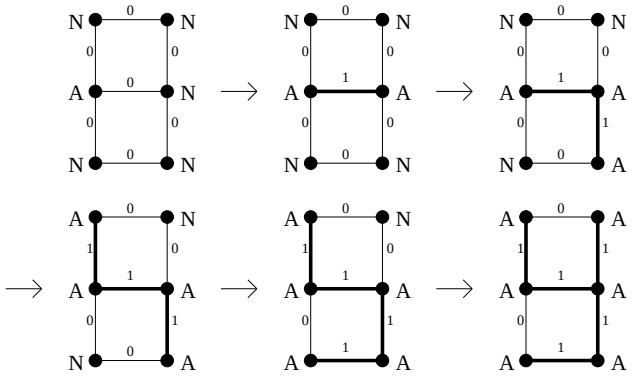


FIGURE 6.1 – Calcul distribué d'un arbre couvrant.

### 6.1.2 Algorithme décrit par réécriture d'arêtes

On suppose qu'initialement un seul sommet de $G$ est étiqueté $A$, tous les autres sont étiquetés $N$ et les arêtes sont étiquetées 0.

À chaque pas de calcul, un sommet $u$ étiqueté $A$ peut activer un voisin $v$ étiqueté $N$. Dans ce cas $u$ garde l'étiquette $A$, $v$ est étiqueté $A$ et l'arête $\{u, v\}$ est noircie et étiquetée 1. Plusieurs sommets peuvent être actifs en même temps, des pas en parallèle sont autorisés dés lors qu'ils concernent des arêtes disjointes. Le calcul s'arrête dès qu'aucun sommet ne peut participer à un pas de calcul.

L'arbre couvrant est défini par les arêtes noircies étiquetées 1.

L'algorithme est codé par le système de réécriture $\mathcal{R}_1 = (L_1, I_1, P_1)$ défini par $L_1 = \{N, A, 0, 1\}$, $I_1 = \{N, A, 0\}$, et $P_1 = \{R\}$ où $R$ est la règle suivante :

La figure 6.1 décrit un exemple de calcul développé par ce système.

**Proposition 6.1** *1. Le système $\mathcal{R}_1$ a la propriété de terminaison.*

*2. Soit $(G, \lambda)$ un graphe étiqueté tel qu'exactement un sommet est étiqueté $A$, les autres étant étiquetés $N$ et les arêtes sont étiquetées 0. Soit $(G, \lambda')$ un graphe irréductible obtenu à partir de $(G, \lambda)$. L'ensemble des arêtes étiquetées 1 dans $(G, \lambda')$ induit un arbre recouvrant de $G$.*

***Preuve :*** *Soit $\varphi : \mathcal{G}_{L_1} \to \mathbb{N}$ l'application qui associe à un graphe son nombre de sommets étiquetés $N$. L'ordre $>$ sur $\mathbb{N}$ est clairement compatible avec $\mathcal{R}_1$ puisque l'application d'une transformation fait décroître strictement le nombre de sommets étiquetés $N$. Donc $\mathcal{R}_1$ a la propriété de terminaison.*

*Pour prouver la deuxième partie du théorème on utilise les invariants suivants :*

*$I_1$. Une arête incidente à un sommet $N$ est étiquetée 0.*

*$I_2$. Excepté dans la situation initiale, chaque sommet étiqueté $A$ a une arête incidente étiquetée 1.*

*$I_3$. Le sous graphe induit par les arêtes étiquetées 1 est un arbre.*

*De plus, $(G, \lambda')$ n'a plus de sommet étiqueté $N$ sinon, sachant que $G$ est connexe, on en déduirait qu'un sommet $N$ est voisin d'un sommet $A$ et donc une contradiction avec le fait que $(G, \lambda')$ est irréductible.* □

**Remarque 6.2** *Aucun sommet du réseau ne peut savoir si le calcul de l'arbre recouvrant par le système $\mathcal{R}_1$ est terminé.*

## 6.2 Calcul d'un arbre couvrant en profondeur avec détection de la terminaison

Cet algorithme calcule un arbre recouvrant d'un réseau $G$ en faisant une exploration séquentielle ; on suppose qu'il y a initialement un sommet distingué, noté $r$, ce sommet lance l'exécution de l'algorithme et peut détecter un instant où la tâche est terminée : détection locale de la terminaison globale de la tâche. Cette exploration se fait par un parcours en profondeur du réseau.

À tout moment un seul sommet est actif et peut donc agir. Un sommet actif, $v$, essaie en priorité d'ajouter dans l'arbre un voisin, $w$, en lui envoyant le message *explorer*. Lors de la réception de *explorer*, si $w$ est déjà dans l'arbre alors il envoie le message *rebrousser* à $v$. Sinon $w$ devient fils de $v$ et le sommet actif. Si $w$ n'a pas d'autre voisin non exploré alors il envoie le message *rebrousser* à son père. Sinon il active un voisin non exploré. Lors de la réception du message *rebrousser* par un sommet $v$ si un voisin de $v$ n'a pas été exploré $v$ lui envoie le message *explorer*. Sinon $v$ envoie le message *rebrousser* à son père si $v \neq r$ ; finalement si $v = r$ alors l'algorithme est terminé.

### 6.2.1 Algorithme décrit par messages

Chaque processus $v$ dispose des variables suivantes :
- *marque* est du type booléen et initialisée à $faux$ ;
- *voisins*, *aux* sont du type ensemble d'entiers ; *voisin* contient initialement les numéros des ports d'entrée/sortie de v et *aux* est initialisée à vide ;
- *père* est un entier qui indique au sommet le port qui pointe vers son père ;
- $fils$ qui est du type ensemble de numéros de ports et qui contient les numéros des ports menant aux fils de $v$ (initialement $fils$ est vide) et *autre* qui indique au sommet $v$ ses voisins qui sont ni fils ni père (initialement *autre* est vide).

**Algorithme 6:** Calcul d'un arbre recouvrant en profondeur.

**I** : {le sommet source $r$ initialise le parcours du graphe}
**début**
  $marque := vrai$ ;
  $père := 0$;
  *aux est l'ensemble des ports de sortie de r* ;
  **suivant** *La valeur de aux* **faire**
    **cas où** $aux = \emptyset$
      C'est fini
    **cas où** $aux \neq \emptyset$
      $x :=$ *un element de aux* ;
      $aux := aux \setminus \{x\}$;
      **envoyer** $<$ *explorer* $>$ par $x$
**fin**
**R**$_1$ : {Lors de la réception du message explorer par le port $q$}
**début**
  **suivant** *La valeur de marque* **faire**
    **cas où** $marque = vrai$
      $aux := aux \setminus \{q\}$;
      **envoyer**$<$**Déjà-dans-l'arbre** $>$ par le port $q$;
      **envoyer** $<$ *rebrousser* $>$ *par le port q*
    **cas où** $marque = faux$
      $marque := vrai$;
      $père := q$;
      **envoyer**$<$**Père**$>$ par le port $q$ ;
      $aux := voisins \setminus \{q\}$ ;
      **suivant** *La valeur de aux* **faire**
        **cas où** $aux = \emptyset$
          **envoyer** $<$ *rebrousser* $>$ *par le port q*
        **cas où** $aux \neq \emptyset$
          $x :=$ *un element de aux*;
          $aux := aux \setminus \{x\}$ ;
          **envoyer** $<$ *explorer* $>$ *par le port x*
**fin**
**R**$_2$ : {Lors de la réception du message rebrousser par le sommet $v$ à travers le port $q$}
**début**
  **suivant** *La valeur de aux et de v* **faire**
    **cas où** $aux = \emptyset$ *et* $v = r$
      C'est fini
    **cas où** $aux = \emptyset$ *et* $v \neq r$
      **envoyer** $<$ *rebrousser* $>$ par *père*
    **cas où** $aux \neq \emptyset$
      $x :=$ *un element de aux* ;
      $aux := aux \setminus \{x\}$ ;
      **envoyer**$<$ *explorer* $>$ par $x$
**fin**
**R**$_2$ : {Un message $<$**Déjà-dans-l'arbre**$>$ arrive sur le sommet $v$ par le port $q$}
**début**
  $autre := autre \cup \{q\}$
**fin**
**R**$_3$ : {Un message $<$**Père**$>$ arrive sur le sommet $v$ par le port $q$}
**début**
  $fils := fils \cup \{q\}$
**fin**

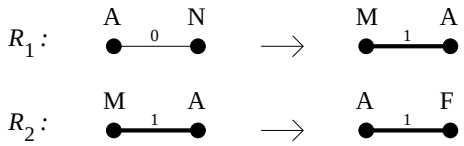



FIGURE 6.2 – Les règles de $\mathcal{R}_2$ avec la relation de priorité : $R_1 > R_2$.

### 6.2.2 Algorithme décrit par des réécritures d'arêtes avec priorités

On considère le calcul d'un arbre recouvrant à travers un parcours en profondeur du graphe. On en donne une formulation utilisant les calculs locaux dans les graphes.

On suppose qu'initialement un seul sommet est actif et étiqueté $A$, les autres étant étiquetés $N$ et les arêtes sont étiquetées 0.

Pendant l'exécution de l'algorithme, à tout moment un seul sommet est étiqueté $A$.

À chaque pas du calcul, le sommet étiqueté $A$, noté ici $u$, agit de la façon suivante :

1. si $u$ a un voisin $v$ étiqueté $N$, alors $u$ active ce voisin : $u$ est étiqueté $M$, $v$ devient actif en prenant l'étiquette $A$ et l'arête $\{u, v\}$ est étiquetée 1.
2. Si $u$ n'a pas de voisin étiqueté $N$ alors il a un unique voisin $w$ étiqueté $M$ auquel il est relié par une arête marquée 1, $u$ réactive $w$ en lui donnant l'étiquette $A$ et prend l'étiquette finale $F$.

L'exécution s'arrête dès que ces règles ne peuvent plus s'appliquer. Dans ce cas le sommet étiqueté $A$ a tous ses voisins étiquetés $F$ et les arêtes marquées 1 constituent un arbre recouvrant du graphe.

Pour augmenter le pouvoir d'expression des calculs sur les arêtes, on introduit une structure de contrôle locale : la priorité, notée $>$. Ce méchanisme permet de restreindre l'application de certaines règles (ou bien de favoriser l'application de certaines règles). Une règle $R$ peut être appliquée s'il n'existe pas de règle $R'$ avec une plus grande priorité ($R' > R$) qui peut s'appliquer sur une partie du graphe où $R$ s'applique. Finalement, l'algorithme peut être codé par le système avec priorité suivant. $\mathcal{R}_2 = (L_2, I_2, P_2, >)$ défini par : $L_2 = \{N, A, M, F, 0, 1\}$, $I_2 = \{N, A, 0\}$, $P_2 = \{R_1, R_2\}$ où $R_1$ et $R_2$ sont les règles décrites sur la figure 6.2. avec la relation de priorité : $R_1 > R_2$.

La relation de priorité donne la "priorité" à l'extension de l'arbre à partir d'un sommet donné (calcul en profondeur).

La figure 6.3 décrit un exemple de calcul correspondant à ce système (les arêtes étiquetées 1 sont représentées avec des traits épais).

On a :

**Théorème 6.3** *1. Le système $\mathcal{R}_2$ est noethérien.*

*2. Soit $(G, \lambda)$ un graphe étiqueté sur $I_2$ tel qu'exactement un sommet a l'étiquette $A$, les autres sont étiquetés $N$ et les arêtes ont l'étiquette 0. Soit $(G, \lambda')$ un graphe irréductible obtenu à partir de $(G, \lambda)$. Alors le graphe induit par les arêtes étiquetées 1 constitue un arbre recouvrant de $G$.*

*3. De plus $(G, \lambda)$ est irréductible si et seulement s'il contient un sommet étiqueté $A$ dont tous les voisins sont étiquetés $F$.*

**Preuve :** Soit $\varphi : \mathcal{G}_{L_2} \to \mathbb{N}^2$ l'application qui associe à un graphe $(H, \nu)$ le couple $(\nu_N, \nu_M)$, où pour toute étiquette $X$, $\nu_X$ désigne le cardinal de $\nu^{-1}(X)$. La relation d'ordre $>$ sur $\mathbb{N}^2$ est

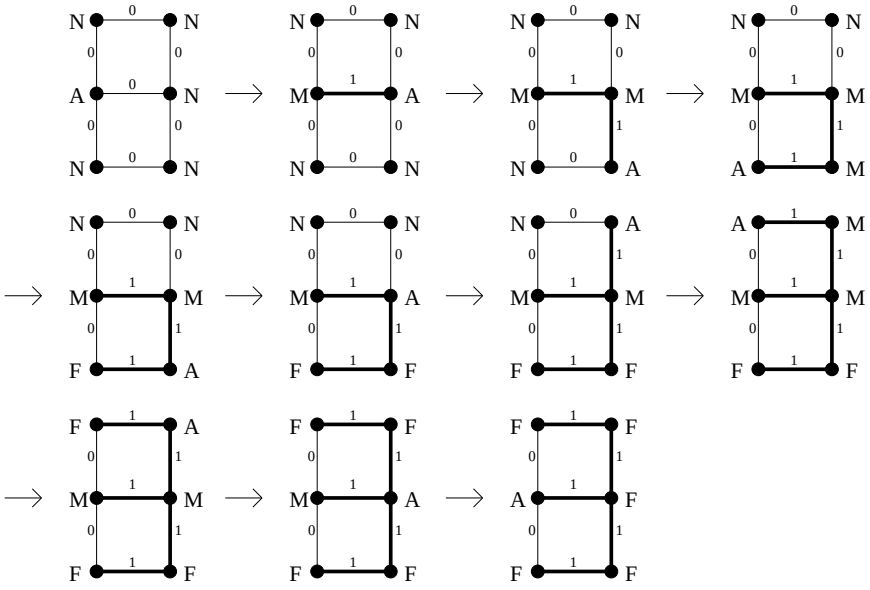


FIGURE 6.3 – Calcul séquentiel d'un arbre recouvrant.

compatible avec $\mathcal{R}_2$ puisque une application de $R_1$ ou de $R_2$ fait diminuer le couple $(\nu_N, \nu_M)$. Donc le système $\mathcal{R}_2$ est noethérien.

Pour prouver la seconde partie du théorème on prouve les invariants suivants :

$I_1$. Une arête incidente à un sommet étiqueté $N$ est étiquetée 0.

$I_2$. Sauf initialement, tout sommet étiqueté $A$, $M$ ou $F$ est incident à au moins une arête étiquetée 1.

$I_3$. Le sous-graphe induit par les arêtes étiquetées 1 est un arbre.

$I_4$. Il existe exactement un sommet étiqueté $A$.

$I_5$. Le sous graphe induit par le sommet $A$, les sommets $M$ et les arêtes étiquetées 1 qui les lient est un chemin simple dont une extrémité est marquée $A$.

$I_6$. Un sommet $F$ n'a pas de voisin $N$.

□

### 6.2.3 Algorithme décrit par des réécritures d'étoiles

Le système $\mathcal{R}_2$ peut également s'exprimer à l'aide de réécritures sur des étoiles.

$\mathcal{E}_1$: **Règle d'ajout d'un sommet**

*Précondition :*

— $\lambda(v_0) = A$,

— $\exists v_1 \in V(B(v_0)) \setminus \{v_0\}$ tel que $\lambda(v_1) = N$.

*Réétiquetage :*

— $\lambda'(v_0) := M$,

— $\lambda'(v_1) := A$,

— $\lambda'(\{v_0, v_1\}) := 1$.

$\mathcal{E}_2$: **Règle de retour en arrière**

*Précondition :*

— $\lambda(v_0) = A$,
— $\nexists v \in V(B(v_0)) \setminus \{v_0\}$ tel que $\lambda(v) = N$,
— $\exists v_1 \in V(B(v_0)) \setminus \{v_0\}$ tel que $\lambda(v_1) = M$,
— $\lambda(\{v_0, v_1\}) = 1$.

*Réétiquetage :*

— $\lambda'(v_0) := F$,
— $\lambda'(v_1) := A$.

## 6.3 Calcul d'un arbre couvrant en parallèle avec détection de la terminaison

L'algorithme suivant calcule un arbre recouvrant en faisant une exploration en parallèle du réseau. On suppose que le réseau a un sommet distingué noté $r$. Ce sommet lance l'exécution de l'algorithme et détecte un instant où le calcul est fini.

L'algorithme est initialisé par le sommet distingué $r$ qui est initialement le seul sommet actif. Un sommet actif essaie d'ajouter à l'arbre ses voisins qui ne sont pas encore dans l'arbre en envoyant le message *explorer*. Quand un sommet qui vient d'être activé ne peut pas ajouter de sommets à l'arbre il initialise un processus d'accusé de réception par l'envoi du message *acquitter* qui remonte de proche en proche jusqu'au sommet $r$ qui ainsi détecte la terminaison du calcul de l'arbre recouvrant.

### 6.3.1 Algorithme décrit par passage de messages

Cet algorithme est décrit par l'algorithme 7. Chaque processus $v$ du réseau dispose des variables suivantes :

— *marque* est du type booléen et initialisée à $faux$ ;
— *voisins* et *succ* sont du type ensemble d'entiers ; *voisins* contient initialement les numéros des ports d'entrée/sortie de v ;
— *père* et *nbacq* sont du type entier, *nbacq* est initialisée à 0. et *père* indique au sommet le port qui pointe vers son père ;
— $fils$ qui est du type ensemble de numéros de ports et qui contient les numéros des ports menant aux fils de $v$ (initialement $fils$ est vide) et *autre* qui indique au sommet $v$ ses voisins qui sont ni fils ni père (initialement *autre* est vide).

**Algorithme 7:** Calcul d'un arbre recouvrant en parallèle.

**I** : {le sommet source $r$ initialise le parcours du graphe}
**début**
  $marque := vrai$ ;
  $père := 0$;
  $succ := voisins$;
  **suivant** *La valeur de succ* **faire**
    **cas où** $succ = \emptyset$
      C'est fini
    **cas où** $succ \neq \emptyset$
      $nbacq := Card(succ)$;
      **envoyer**$<$ *explorer* $>$ par tous les ports de $succ$
**fin**

**R**$_1$ : {Lors de la réception du message explorer à travers le port $q$}
**début**
  **suivant** *La valeur de marque* **faire**
    **cas où** $marque = vrai$
      **envoyer** <**Déjà-dans-l'arbre**> *par le port* $q$ ;
      **envoyer**<acquitter> *par le port* $q$
    **cas où** $marque = faux$
      $marque := vrai$;
      $père := q$;
      **envoyer**<**Père**> par le port $q$ ;
      $succ := voisins \setminus \{q\}$ ;
      **suivant** *La valeur de succ* **faire**
        **cas où** $succ = \emptyset$
          **envoyer**$<$ *acquitter* $>$ *par le port* $q$
        **cas où** $succ \neq \emptyset$
          $nbacq := Card(succ)$;
          *envoyer* $<$ *explorer* $>$ *par les ports de succ*
**fin**

**R**$_2$ : {Lors de la réception du message acquitter par le sommet $v$ à travers le port $q$}
**début**
  $nbacq := nbacq - 1$;
  **si** $nbacq = 0$ **alors**
    **suivant** *La valeur de v* **faire**
      **cas où** $v = r$
        C'est fini
      **cas où** $v \neq r$
        *envoyer* $<$ *acquitter* $>$ à *père*
**fin**

**R**$_2$ : {Un message <**Déjà-dans-l'arbre**> arrive sur le sommet $v$ par le port $q$}
**début**
  $autre := autre \cup \{q\}$
**fin**

**R**$_3$ : {Un message <**Père**> arrive sur le sommet $v$ par le port $q$}
**début**
  $fils := fils \cup \{q\}$
**fin**

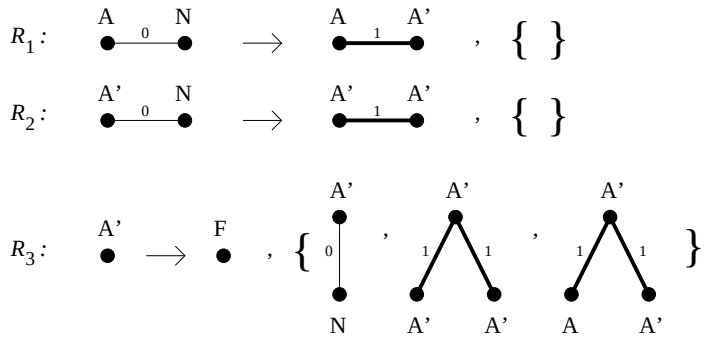



FIGURE 6.4 – Les règles de $\mathcal{R}_3$.

### 6.3.2 Algorithme décrit par des réécritures d'arêtes avec contextes interdits

De nouveau, on suppose qu'un unique sommet est initialement étiqueté $A$, les autres sont étiquetés $N$ et les arêtes sont étiquetées 0. L'unique sommet $A$ garde cette étiquette pendant l'exécution de l'algorithme alors que les autres sommets actifs seront étiquetés $A'$. Dès qu'un sommet $A'$ ne peut plus ajouter de sommet à l'arbre et qu'il est feuille de cet arbre il devient $F$.

À chaque pas de calcul, un sommet actif $u$ (i.e., ayant l'étiquette $A$ ou $A'$) agit de la façon suivante :

1. Si $u$ a un voisin $v$ étiqueté $N$, alors $u$ peut activer ce voisin : $u$ conserve son étiquette et $v$ devient actif en prenant l'étiquette $A'$ et l'étiquette de l'arête $\{u, v\}$ devient 1.
2. Si $u$ est étiqueté $A'$, n'a pas de voisin $N$ et a au plus un voisin auquel il est relié par 1 et différent de $F$ alors l'étiquette de $u$ devient $F$.

À tout moment, le sous-graphe induit par les arêtes étiquetées 1 contient les sommets $A$ et $A'$ et constitue un arbre. Intuitivement, la deuxième règle signifie que $u$ est une feuille de cet arbre.

Donc cet algorithme fonctionne en 2 phases (qui peuvent se chevaucher) : dans la première phase, l'arbre pousse jusqu'à ce qu'il contienne tous les sommets du réseau ; dans la seconde phase il perd ses feuilles jusqu'à ce qu'il soit réduit au sommet $A$. Ce sommet est alors en mesure de détecter la terminaison de l'algorithme.

Cet algorithme peut être décrit aussi bien avec des réécritures avec contextes interdits qu'avec des réécritures sur des étoiles ouvertes.

On peut décrire l'algorithme avec le système suivant $\mathcal{R}_3 = (L_3, I_3, P_3)$ défini par : $L_3 = \{N, A, A', F, 0, 1\}$, $I_3 = \{N, A, 0\}$, $P_3 = \{R_1, R_2, R_3\}$ où $R_1$, $R_2$ et $R_3$ sont les règles de réécriture avec contextes interdits présentées sur la figure 6.4.

Les règles $R_1$ et $R_2$ n'ont pas de contexte interdit. La règle $R_3$ a 3 contextes interdits. L'unique sommet du membre gauche de $R_3$ étiqueté $A'$ est associé au sommet "en position haute dans les contextes et marqué A'. " Un sommet $A'$ peut devenir $F$ s'il n'est pas relié à un sommet $N$, et s'il n'est pas relié à 2 sommets $A'$ par des arêtes marquées 1, et s'il n'est pas relié à 1 sommet $A$ et 1 sommet $A'$ par des arêtes marquées 1.

La figure 6.5 décrit un exemple de calcul.

### 6.3.3 Algorithme décrit par des réécritures sur des étoiles ouvertes.

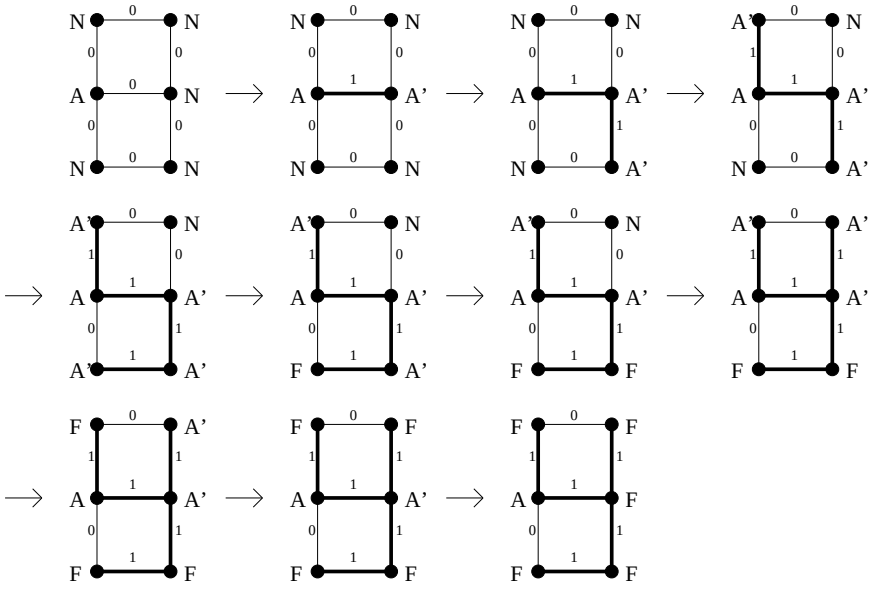


FIGURE 6.5 – Calcul distribué d'un arbre recouvrant par $\mathcal{R}_3$ avec détection locale de la terminaison globale.

$\mathcal{E}_1$: **Règle d'ajout d'un sommet à l'arbre**

*Précondition :*
— $\lambda(v_0) = A$,
— $\exists v_1 \in V(B(v_0)) \setminus \{v_0\}$ tel que $\lambda(v_1) = N$.
*Réétiquetage :*
— $\lambda'(v_0) := A$,
— $\lambda'(v_1) := A'$,
— $\lambda'(\{v_0, v_1\}) := 1$.

$\mathcal{E}_2$: **Règle d'ajout d'un sommet à l'arbre**

*Précondition :*
— $\lambda(v_0) = A'$,
— $\exists v_1 \in V(B(v_0)) \setminus \{v_0\}$ tel que $\lambda(v_1) = N$.
*Réétiquetage :*
— $\lambda'(v_0) := A'$,
— $\lambda'(v_1) := A'$,
— $\lambda'(\{v_0, v_1\}) := 1$.

$\mathcal{E}_3$: **Règle de remontée pour la détection de la fin**

*Précondition :*
— $\lambda(v_0) = A'$,
— $\nexists v \in V(B(v_0)) \setminus \{v_0\}$ tel que $\lambda(v) = N$,
— $\exists! v_1 \in V(B(v_0)) \setminus \{v_0\}$ tel que $((\lambda(v_1) = A' \quad ou \quad \lambda(v_1) = A) \quad et \quad \lambda(\{v_0, v_1\}) = 1)$
.
*Réétiquetage :*

— $\lambda'(v_0) := F$.

**Théorème 6.4** *1. Le système $\mathcal{R}_3$ est noethérien.*

2. *Soit $(G, \lambda)$ un graphe étiqueté sur $I_3$ tel qu'exactement un sommet est étiqueté $A$, les autres sont étiquetés $N$ et les arêtes sont étiquetées 0. Soit $(G, \lambda')$ un graphe irréductible modulo $\mathcal{R}_3$ obtenu à partir de $(G, \lambda)$. Le graphe induit par les arêtes étiquetées 1 dans le graphe $(G, \lambda')$ est un arbre recouvrant de $G$.*
3. *Pour chaque chaîne de réécriture obtenue modulo $\mathcal{R}_3$ partant de $(G, \lambda)$ et terminant sur le graphe $(G, \lambda'')$, le graphe $(G, \lambda'')$ est irréductible si et seulement s'il contient un sommet $A$ n'ayant que des voisins $F$.*

**Preuve :** Soit $\varphi : \mathcal{G}_{L_3} \to \mathbb{N}^2$ l'application qui associe au graphe $(H, \nu)$ le couple d'entiers $(\nu_N, \nu_{A'})$ (pour mémoire : $\nu_X$ désigne le cardinal de $\nu^{-1}(X)$). La relation d'ordre $>$ sur $\mathbb{N}^2$ est clairement compatible avec $\mathcal{R}_3$ . Donc $\mathcal{R}_3$ est noethérien.

Pour prouver la seconde partie du théorème, il suffit de vérifier les invariants suivants :

$I_1$. Une arête incidente à un sommet $N$ est étiquetée 0.

$I_2$. Sauf initialement, un sommet étiqueté $A$, $A'$ ou $F$ est incident à au moins une arête étiquetée 1.

$I_3$. Le sous-graphe induit par les arêtes étiquetées 1 est un arbre.

$I_4$. Il existe à tout moment exactement un sommet étiqueté $A$.

$I_5$. Le sous-graphe induit par les sommets $A$, $A'$ et les arêtes étiquetées 1 qui les lient est un arbre.

$I_6$. Un sommet $F$ n'a pas de voisin $N$.

□

## 6.4 Calcul d'un arbre couvrant chaque sommet ayant un numéro unique

Dans cette partie on suppose que chaque sommet a un identificateur propre. Pour simplifier on suppose que cet identificateur est un entier. On reprend les trois exemples de calcul d'arbre recouvrant vu précédemment dans ce cadre.

Chaque sommet peut initialiser le calcul d'un arbre recouvrant ; les arêtes de l'arbre calculé par le sommet $i$ sont marquées $i$. L'algorithme assure qu'un seul arbre survivra : l'arbre dont le calcul a été initialisé par le sommet doté du plus grand identificateur noté $max$. Il est caractérisé par les arêtes marquées $max$.

Initialement, chaque sommet $i$ est étiqueté $(A, i)$ et chaque arête est étiquetée 0.

**Calcul séquentiel d'un arbre recouvrant avec détection locale de la terminaison globale.**

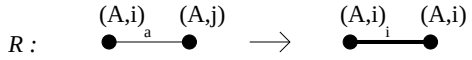
$R_1$ : $(A,i) \overset{\alpha}{\text{—}} (X,j) \longrightarrow (M,i) \overset{i}{\text{—}} (A,i)$ ; if $j < i$ ; $X \in \{A, M, N, F\}$

$R_2$ : $(A,i) \overset{\alpha}{\text{—}} (Y,k) \longrightarrow (N,i) \overset{\alpha}{\text{—}} (Y,k)$ ; if $i < k$ ; $Y \in \{A, M, N\}$

$R_3$ : $(M,i) \overset{i}{\text{—}} (A,i) \longrightarrow (A,i) \overset{i}{\text{—}} (F,i)$

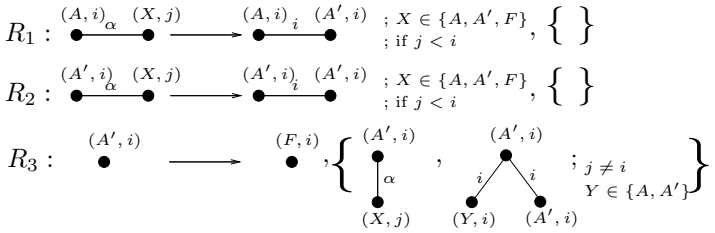
avec la relation de priorité : $R_1, R_2 > R_3$.

**Calcul parallèle d'un arbre recouvrant sans détection de la terminaison globale.**




où $a, i, j$ sont des entiers tels que $j < i$.

**Calcul parallèle d'un arbre recouvrant avec détection locale de la terminaison globale.**




**Remarque 6.5** *Dans le premier cas et le troisième cas on obtient également des algorithmes d'élection : le sommet élu étant le sommet ayant le plus grand identificateur. Ce sommet sait qu'il a le plus grand identificateur dès qu'il détecte la fin du calcul en n'ayant que des voisins F.*

**Remarque 6.6** *Certains systèmes de réécriture de ce chapitre utilisent deux structures de contrôle locale pour l'applicabilité de certaines règles : les priorités et les contextes interdits. L'article [LMS95] montre que ces deux strutures ont la même puissance d'expression : l'une peut être simulée par l'autre.*

# Chapitre 7

# Élection et Nommage

Un algorithme distribué permet de résoudre le problème de l'élection sur un graphe **G** si toute exécution de l'algorithme sur **G** termine et si dans la configuration finale, il existe exactement un sommet $v \in V(G)$ qui est dans l'état ÉLU, alors que tous les autres sommets de **G** sont dans l'état NON-ÉLU. On suppose par ailleurs, que les états ÉLU et NON-ÉLU sont *finaux*, i.e., une fois qu'un sommet est dans l'un de ces états alors son état n'est plus modifié jusqu'à la fin de l'exécution de l'algorithme.

Le problème de l'élection est fondamental en algorithmique distribuée, en effet un sommet élu peut être utilisé pour centraliser ou diffuser de l'information, pour initialiser l'exécution d'un autre algorithme qui nécessite un sommet distingué (comme, par exemple, l'algorithme de calcul d'un arbre couvrant présenté dans la Section 3.6.1), pour prendre une décision de manière centralisée, etc.

Le but d'un algorithme de *nommage* est d'arriver dans une configuration finale où un identifiant unique est associé à chaque sommet. Ce problème aussi est très important puisque de nombreux algorithmes distribués fonctionnent correctement sous l'hypothèse que tous les sommets ont des identifiants uniques. Le problème de l'*énumération* est une variante du problème de nommage. Le but d'un algorithme d'énumération pour un graphe **G** est d'attribuer à chaque sommet de **G** un entier de telle sorte que, dans la configuration finale, l'ensemble des numéros des sommets de **G** soit exactement $[1, |V(G)|]$.

Les problèmes du nommage et de l'élection ne sont pas toujours équivalents suivants les modèles.

Ce chapitre ne traite que de l'élection déterministe, c'est à dire ne faisant pas appel au hasard. Le problème de l'élection probabiliste sera traité plus loin.

## 7.1 Élection dans un anneau, chaque processus étant identifié par un nom

Cette section décrit l'algorithme de Chang et Roberts à l'aide de réécritures d'arcs (Figure 7.1). On considère un anneau unidirectionel de $n$ processus, chacun est identifié par un unique nom. On suppose que l'ensemble des noms est totalement ordonné. Chaque noeud a une identité et une variable mémorisant la plus grande identité vue par ce sommet ; cette variable est initialisée avec le nom du noeud lui-même ; par ailleurs on suppose que tous les noeuds sont candidats (le statut d'un sommet peut être : candidat, élu, non-élu). Finalement, l'étiquette de chaque sommet est : $(id, max\text{-}id, statut\text{-}du\text{-}sommet)$. Initialement l'étiquette d'un noeud est de la forme :

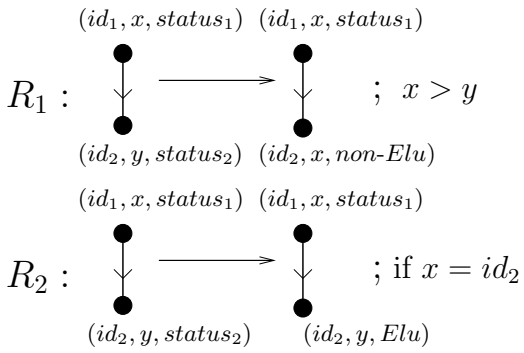



FIGURE 7.1 – L'algorithme de Chang et Roberts codé par des réécritures d'arcs.

$(id, id, candidat)$.

L'algorithme propage de proche en proche une identité enregistrée par un sommet qui est plus grande que celle vue par son successeur. Un sommet qui voit une identité plus grande que la sienne est non élu. Le sommet élu est le sommet qui voit revenir sa propre identité.

**Remarque 7.1** *En modifiant légèrement le système, on peut supposer qu'au moins un sommet est candidat à l'élection.*

## 7.2 Élection dans un réseau avec identités

Deux algorithmes possibles sont décrits dans la section 6.4.

## 7.3 Élection dans les arbres

Un exemple d'algorithme est décrit dans la section 3.6.2.

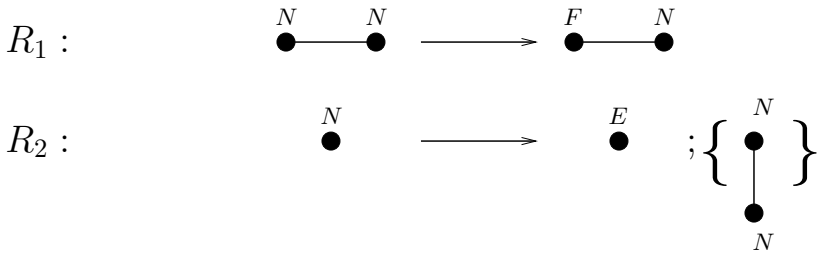



FIGURE 7.2 – Règles de réécriture avec contexte interdit permettant d'élire dans un graphe complet.

## 7.4 Élection dans un graphe complet anonyme

On décrit un algorithme d'élection dans les graphes complets à l'aide d'un système à contextes interdits. Soit $\mathcal{R}_5 = (\mathcal{L}_5, \mathcal{I}_5, P_5)$ le système défini par $\mathcal{L}_5 = \{N, F, E, 0\}$, $\mathcal{I}_5 = \{N, 0\}$ et $P_5 = \{R_1, R_2\}$ où $R_1$, $R_2$ sont les règles données par la figure 7.2.

Initialement, tous les sommets sont étiquetés $N$. La règle $R_1$ supprime un sommet $N$ s'il a un voisin $N$. La règle $R_2$ n'est appliquée qu'une fois : lorsqu'il ne reste plus qu'un seul sommet étiqueté $N$ qui, de fait, devient le sommet élu. La preuve de ce système se fait en montrant l'invariant suivant :

*à chaque étape le graphe induit par les sommets $N$ est un graphe complet.*

**Remarque 7.2** *Les algorithmes d'élection dans les arbres anonymes ou dans les graphes complets anonymes peuvent être implémentés très simplement à l'aide d'algorithmes probabilistes vus plus loin.*

# Chapitre 8

# Caractérisation des réseaux admettant un algorithme d'élection

La caractérisation des réseaux pour lesquels il existe un algorithme d'élection est fonction du modèle considéré. Dans cette section on considère le modèle point-à-point asynchrone avec passage de messages en mode asynchrone.

Ce chapitre présente tout d'abord les fibrations et les revêtements qui sont des outils combinatoires qui permettent de caractériser les graphes pour lesquels il existe un algorithme d'élection. La section suivante montre comment le modèle point-à-point asynchrone avec passage de messages en mode asynchrone peut être codé par des graphes dirigés et des réécritures d'arcs. On donne ensuite une condition nécessaire pour l'existence d'un algorithme d'élection. Enfin en présentant un algorithme d'élection on montre que cette condition nécessaire est également suffisante. (Ce chapitre est extrait de : [CM07].)

Une autre carctérisation a été donnée par Yamashita et Kameda [YK96] ; cette caractérisation utilise la notion de vue. Étant donné un graphe $G$, la vue d'un sommet $v$ est l'arbre infini dont les sommets sont les chemins issus de $v$ dans le graphe.

## 8.1 Fibrations et revêtements

Les notions de fibrations et de revêtements sont fondamentales dans cette étude. On considère des graphes dirigés pouvant avoir des arcs multiples et des boucles. Une fibration est un homomorphisme qui induit un isomorphisme entre les arcs entrants d'un sommet quelconque et les arcs entrants de son image.

**Définition 8.1** *Un graphe dirigé $D$ est fibré sur un graphe dirigé $D'$ via l'homomorphisme $\varphi$ si $\varphi$ est un homomorphisme de $D$ sur $D'$ tel que pour chaque arc $a' \in A(D')$ et pour chaque sommet $v \in \varphi^{-1}(t(a'))$, il existe un unique arc $a \in A(D)$ tel que $t(a) = v$ et $\varphi(a) = a'$ ; cet arc $a$ est appelé le relèvement de $a'$ en $v$.*

*On dit que l'homomorphisme $\varphi$ est une* fibration *de $D$ sur $D'$, le graphe dirigé $D$ est le graphe total de $\varphi$ et le graphe dirigé $D'$ est la base de $\varphi$.*

*La fibre sur un sommet $v'$ (resp. un arc $a'$) de $D'$ est par définition l'ensemble $\varphi^{-1}(v')$ de sommets de $D$ (resp. l'ensemble $\varphi^{-1}(a')$ d'arcs de $D$).*

Dans la suite les graphes dirigés sont toujours fortement connexes et un graphe total est non vide.

Un revêtement est une fibration qui induit un isomorphisme entre les arcs sortants de chaque sommet $v$ du graphe total et les arcs sortants de l'image de $v$.

**Définition 8.2** *Un graphe dirigé $D$ est un revêtement du graphe dirigé $D'$ via l'homomorphisme $\varphi$ si $\varphi$ est un homomorphisme de $D$ sur $D'$ tel que pour chaque arc $a' \in A(D')$ et pour chaque sommet $v \in \varphi^{-1}(t(a'))$ (resp. $v \in \varphi^{-1}(s(a'))$, il existe un unique arc $a \in A(D)$ tel que $t(a) = v$ (resp. $s(a) = v$) et $\varphi(a) = a'$.*

Dans la suite $\varphi$ sera également appelé le revêtement.

Si $D'$ n'a pas de boucle et d'arêtes multiples alors le revêtement sera dit simple.

Un revêtement symétrique est un revêtement entre deux graphes symétriques qui préserve la fonction $Sym$.

**Définition 8.3** *Un graphe dirigé symétrique $D$ est appelé un revêtement symétrique d'un graphe dirigé symétrique $D'$ via un homomorphisme $\varphi$ si $D$ est un revêtement de $D'$ via $\varphi$ et si pour chaque arc $a \in A(D)$, $\varphi(Sym(a)) = Sym(\varphi(a))$.*

*Un graphe dirigé symétrique $D$ est dit revêtement symétrique premier si pour chaque graphe dirigé symétrique $D'$ tel que $D$ est un revêtement symétrique de $D'$ alors $D$ est isomorphe à $D'$, $D \simeq D'$.*

Toutes ces définitions sont étendues d'une façon naturelle aux graphes dirigés étiquetés.

Une propriété intéressante des revêtements est que toutes les fibres ont le même cardinal, qui est appelé le nombre de feuillets du revêtement. On a :

**Proposition 8.4** *Un revêtement $\varphi : \mathbf{D} \to \mathbf{D}'$ ayant une base connexe est surjectif; de plus il existe $q \in \{1, 2, \cdots\}$ tel que pour chaque $x \in V(D') \cup A(D')$, $|\varphi^{-1}(x)| = q$.*

Étant donné un graphe dirigé étiqueté $\mathbf{D}$, il existe un graphe dirigé étiqueté minimal $\mathbf{D}_0$ tel que $\mathbf{D}$ est fibré sur $\mathbf{D}_0$.

**Proposition 8.5** *Pour tout graphe dirigé étiqueté $\mathbf{D}$, il existe un graphe dirigé étiqueté fortement connexe $\mathbf{D}_0$ tel que $\mathbf{D}$ est fibré sur $\mathbf{D}_0$ et tel que pour tout graphe dirigé étiqueté fortement connexe $\mathbf{D}'$, si $\mathbf{D}$ est fibré sur $\mathbf{D}'$ alors $\mathbf{D}'$ est fibré sur $\mathbf{D}_0$.*

*Le graphe $\mathbf{D}_0$ est appelé la base minimale de $\mathbf{D}$.*

Soient $D$ et $D'$ deux graphes dirigés tels que $D$ est un revêtement surjectif de $D'$ via $\varphi$. Si $D'$ n'a pas de boucles alors pour chaque arc $a \in A(D) : \varphi(s(a)) \neq \varphi(t(a))$. Finalement, la propriété suivante est une conséquence directe des définitions et est fondamentale pour l'étude de l'élection.

**Proposition 8.6** *Soient $\mathbf{D}$ et $\mathbf{D}'$ deux graphes dirigés étiquetés tels que $\mathbf{D}'$ n'a pas de boucles et $\mathbf{D}$ est un revêtement surjectif de $\mathbf{D}'$ via $\varphi$. Soient $a_1$ et $a_2$ deux arcs de $\mathbf{D}$. Si $a_1 \neq a_2$ et $a_1, a_2 \in \varphi^{-1}(a')$ $(a' \in A(D'))$ alors $\{s(a_1), t(a_1)\} \cap \{s(a_2), t(a_2)\} = \emptyset$.*

## 8.2 Revêtements et calculs locaux sur les arcs

Nous présentons un résultat fondamental établissant un lien entre les revêtements et les calculs locaux sur les arcs.

**Lemme 8.7** *Soit $\mathcal{R}$ une relation de réétiquetage localement engendrée sur les arcs et soit $\mathbf{D}_1$ un revêtement d'un graphe dirigé étiqueté $\mathbf{D}'_1$ via le morphisme $\gamma$; on suppose que $\mathbf{D}'_1$ n'a pas de boucle. Si $\mathbf{D}'_1 \; \mathcal{R}^* \; \mathbf{D}'_2$ alors il existe $\mathbf{D}_2$ tel que $\mathbf{D}_1 \; \mathcal{R}^* \; \mathbf{D}_2$ et $\mathbf{D}_2$ est un revêtement de $\mathbf{D}'_2$ via $\gamma$.*

**Preuve :** Soient $\mathbf{D}_1 = (D_1, \lambda)$ et $\mathbf{D}'_1 = (D'_1, \eta)$ deux graphes dirigés étiquetés tels que $\mathbf{D}_1$ est un revêtement de $\mathbf{D}'_1$ via un homomorphisme $\gamma$ et soit $\mathcal{R}$ une relation de réétiquetage localement engendrée sur les arcs.

Il suffit de prouver le lemme quand $\mathbf{D}'_1 = (D'_1, \eta)\ \mathcal{R}\ (D'_1, \eta') = \mathbf{D}'_2$, soit $r$ la règle de réétiquetage pour ce pas de calcul. Considérons l'arc réécrit $a = (v, v') \in A(D'_1)$ : pour chaque $x \in (V(D'_1) \cup A(D'_1)) \setminus \{a, v, v'\}$, $\eta'(x) = \eta(x)$. Puisque $\mathbf{D}_1$ est un revêtement de $\mathbf{D}'_1$ via $\gamma$ et $\mathbf{D}'_1$ n'a pas de boucle d'après la proposition 8.6 les arcs de $A(D_1)$ qui ont pour image par $\gamma$ le couple $(v, v')$ sont disjoints. Donc $r$ peut être appliquée sur chaque arc $(u, u') \in A(D_1)$ tel que $\gamma(u) = v$ et $\gamma(u') = v'$. Soit $\lambda'$ le nouvel étiquetage de $D_1$ une fois que toutes ces réécritures ont été faites. Pour chaque $u \in \gamma^{-1}(\{v, v'\})$, $\lambda'(u) = \eta'(\gamma(u))$ et pour chaque $u \in V(D_1) \setminus \gamma^{-1}(\{v, v'\})$, $\lambda'(u) = \lambda(u) = \eta(\gamma(u)) = \eta'(\gamma(u))$. Les mêmes relations sont vérifiées pour les arcs. Donc le graphe dirigé étiqueté $\mathbf{D}_2 = (D_1, \lambda')$ est un revêtement de $\mathbf{D}'_2$ via $\gamma$. □

## 8.3 Codage des opérations sur les messages par des calculs locaux sur les arcs

On considère le modèle de communication point-à-point représenté par un graphe connexe. On suppose que l'on dispose d'une numérotation des ports. On considère que la communication se fait par passage de messages en mode asynchrone.

### 8.3.1 Numérotation des ports et graphes dirigés symétriques

Un réseau est représenté par un graphe $(\mathbf{G}, \nu)$ où $\mathbf{G} = (G, \lambda)$ est un graphe simple dont les sommets sont étiquetés et $\nu$ est une numérotation des ports.

Étant donné un graphe $(\mathbf{G}, \nu)$, on lui associe un graphe dirigé étiqueté $(Dir(\mathbf{G}), \nu)$ où chaque sommet $v \in V(Dir(G))$ a la même étiquette que dans $\mathbf{G}$ et où chaque arc $a_{(u,v)} \in A(Dir(G))$ tel que $s(a) = u$ et $t(a) = v$ est étiqueté par $(\nu_u(v), \nu_v(u))$. Soit la fonction $\iota$ définie par : $\iota((p, q)) = (q, p)$, où $p$ et $q$ sont deux entiers. On peut noter que pour chaque arc $a \in A(Dir(G))$ étiqueté par $(p, q)$, l'arc $Sym(a)$ est étiqueté par $(q, p) = \iota((p, q))$, i.e., l'étiquetage des arcs de $Dir(G)$ induit par $\nu$ est symétrique.

Des exemples d'une telle construction sont présentés sur les figures 8.1, 8.2 et 8.3.

### 8.3.2 Codage d'un réseau par un graphe dirigé étiqueté

La construction présentée dans cette section peut apparaître longue et technique mais l'intuition est très naturelle et simple ; cette construction est illustrée sur les figures 8.2 et 8.3.

Étant donné un graphe dirigé étiqueté $\mathbf{D} = (D, \lambda)$ dont les arcs sont non étiquetés, on associe à $\mathbf{D}$ un graphe dirigé étiqueté $\overleftrightarrow{\mathbf{D}}$ défini de la façon suivante.

À chaque arc $a \in A(D)$ dont la source est $u$ et la cible est $v$, on associe l'ensemble $V_a$ de trois sommets noté $\{outbuf_a(u, v), canal_a(u, v), inbuf_a(u, v)\}$ et l'ensemble $A_a$ de quatre arcs : $(u, outbuf_a(u, v)), (outbuf_a(u, v), canal_a(u, v)), (canal_a(u, v), inbuf_a(u, v)), (inbuf_a(u, v), v)$.

Le graphe dirigé $\overleftrightarrow{D}$ est alors défini par :

$$V(\overleftrightarrow{D}) = V(D) \cup (\bigcup_{a \in A(D)} V_a) \text{ et } A(\overleftrightarrow{D}) = \bigcup_{a \in A(D)} A_a.$$

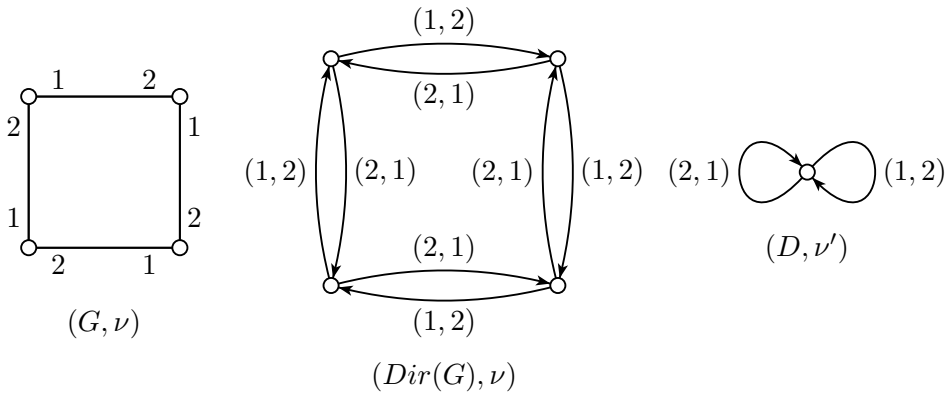



FIGURE 8.1 – Un graphe $G$ avec une numérotation des ports $\nu$ et le graphe dirigé étiqueté associé $(Dir(G), \nu)$ qui est un revêtement de $(D, \nu')$. D'après la proposition 8.9, il n'existe pas d'algorithme d'élection pour $(G, \nu)$. Le même argument s'applique à un anneau quelconque.

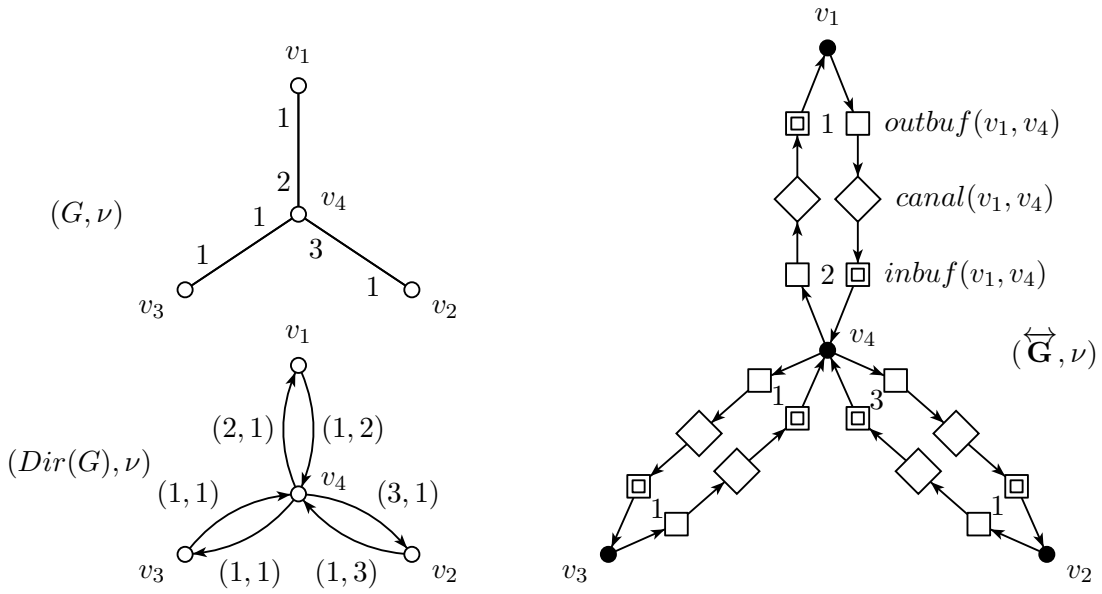



FIGURE 8.2 – On adopte la convention suivante pour les sommets de $(\overleftrightarrow{\mathbf{G}}, \nu)$. Un disque noir correspond à un sommet avec l'étiquette **process**, un carré correspond à un sommet avec l'étiquette **envoyer**, un losange correspond à un sommet avec l'étiquette **transmission**, et un double-carré correspond à un sommet avec l'étiquette **receive**. Il n'y a pas d'arc multiple donc les indices pour *outbuf*, *canal* et *inbuf* sont omis.

Il est nécessaire de mémoriser la signification des différents sommets de $\overleftrightarrow{D}$. Pour ce faire, on considère le graphe dirigé étiqueté $\overleftrightarrow{\mathbf{D}} = (\overleftrightarrow{D}, (\kappa, \lambda))$ où $\lambda$ et $\kappa$ sont deux étiquetages des sommets de $\overleftrightarrow{D}$ définis de la façon suivante. Pour chaque sommet $v \in V(\overleftrightarrow{D})$, $\lambda(v)$ est l'étiquette de $v$ dans $\mathbf{D}$ si $v \in V(D)$ et $\lambda(v) = \epsilon$ sinon. La fonction d'étiquetage $\kappa$ code le rôle de chaque sommet de $\overleftrightarrow{D}$ et est définie de la façon suivante :

— $\forall u \in V(D)$, $\kappa(u) =$ **process**,
— $\forall a \in A(D)$, $\kappa(outbuf_a(s(a), t(a))) =$ **envoyer**,
— $\forall a \in A(D)$, $\kappa(inbuf_a(s(a), t(a))) =$ **receive**,
— $\forall a \in A(D)$, $\kappa(canal_a(s(a), t(a))) =$ **transmission**.

On considère maintenant la fonction d'étiquetage des arcs de $\mathbf{D}$ telle que pour chaque arc $a \in V(D)$, il existe des entiers positifs $p, q$ tels que $\nu(a) = (p, q)$. On étend $\nu$ en une fonction d'étiquetage des sommets de $\overleftrightarrow{D}$ telle que pour chaque arc $a \in A(D)$ dont l'étiquette est $(p, q)$, $\nu(outbuf_a(a)) = p$ et $\nu(inbuf_a(a)) = q$ (pour les autres sommets, $\nu$ est égale à $\epsilon$).

Supposons que $\mathbf{D}$ est un graphe dirigé étiqueté symétrique et que $\nu$ est une fonction d'étiquetage symétrique de $\mathbf{D}$.

On peut remarquer que pour chaque arc $a \in A(D)$, $\nu(inbuf_a(a)) = \nu(outbuf_{Sym(a)}(Sym(a)))$.

Étant donné un réseau $(\mathbf{G}, \nu)$ où $\mathbf{G}$ est un graphe simple étiqueté et $\nu$ est une numérotation des ports de $\mathbf{G}$, on note $(\overleftrightarrow{\mathbf{G}}, \nu)$ le graphe dirigé simple étiqueté obtenu en appliquant la construction décrite ci-dessus sur le graphe dirigé étiqueté $(Dir(\mathbf{G}), \nu)$.

### 8.3.3 Codage des instructions de base avec des calculs locaux sur les arcs

Les actions de base d'un processus sont, en fonction de son état, soit un changement d'état, soit une réception de message à travers un port soit un envoie de message à travers un port.

Étant donnés un réseau $(\mathbf{G}, \nu)$ et sa représentation $(\overleftrightarrow{\mathbf{G}}, \nu)$ comme un simple graphe dirigé étiqueté (décrit plus haut), on explique maintenant comment les instructions de base de chaque processus de $(\mathbf{G}, \nu)$ peuvent être traduites par des calculs sur les arcs de $(\overleftrightarrow{\mathbf{G}}, \nu)$ :

— un événement qui permet à un sommet de modifier son état (i.e., une transition interne à un processus) est codé par une règle qui peut être appliquée sur un sommet $v \in V(\overleftrightarrow{G})$ tel que $\kappa(v) =$ **process**,
— un événement du type "envoyer un message **m** via le port $p$" est codé par une règle qui peut être appliquée sur l'arc $(u, v) \in A(\overleftrightarrow{G})$ où $\kappa(u) =$ **process**, $\kappa(v) =$ **envoyer** et $\nu(v) = p$,
— un événement du type "recevoir a message **m** via le port $q$"est codé par une règle qui peut être appliquée sur l'arc $(v, u) \in A(\overleftrightarrow{G})$ où $\kappa(u) =$ **process**, $\kappa(v) =$ **recevoir** et $\nu(v) = q$,
— un événement concernant le contrôle de la transmission peut être codé par une règle concernant un arc de la forme $(u, v)$ ou $(v, u)$ avec $\kappa(u) \in \{$**envoyer**, **recevoir**$\}$ et $\kappa(v) =$ **transmission**.

**Proposition 8.8** *Soient* $\mathbf{D}$ *et* $\mathbf{D}'$ *deux graphes dirigés étiquetés symétriques. Soit* $\nu$ *(resp.* $\nu'$*) une fonction d'étiquetage symétrique des arcs de* $\mathbf{D}$ *(resp.* $\mathbf{D}'$*) telle que pour chaque arc* $a \in V(D)$*, il existe 2 entiers positifs* $p, q$ *tels que* $\nu(a) = (p, q)$ *et pour chaque arc* $a' \in V(D')$*, il existe deux entiers positifs* $p, q$ *tels que* $\nu'(a') = (p, q)$*. Si* $(\mathbf{D}, \nu)$ *est un revêtement de* $(\mathbf{D}', \nu')$ *alors* $(\overleftrightarrow{\mathbf{D}}, \nu)$ *est revêtement de* $(\overleftrightarrow{\mathbf{D}}', \nu')$*.*

**Preuve :** Soient deux graphes dirigés $((D, \lambda), \nu), ((D', \lambda'), \nu')$ tels que $(\mathbf{D}, \nu)$ est un revêtement de $(\mathbf{D}', \nu')$ via un homomorphisme $\varphi$. On définit un homomorphisme $\gamma$ de $(\overleftrightarrow{\mathbf{D}}, \nu)$ sur $(\overleftrightarrow{\mathbf{D}}', \nu')$ de

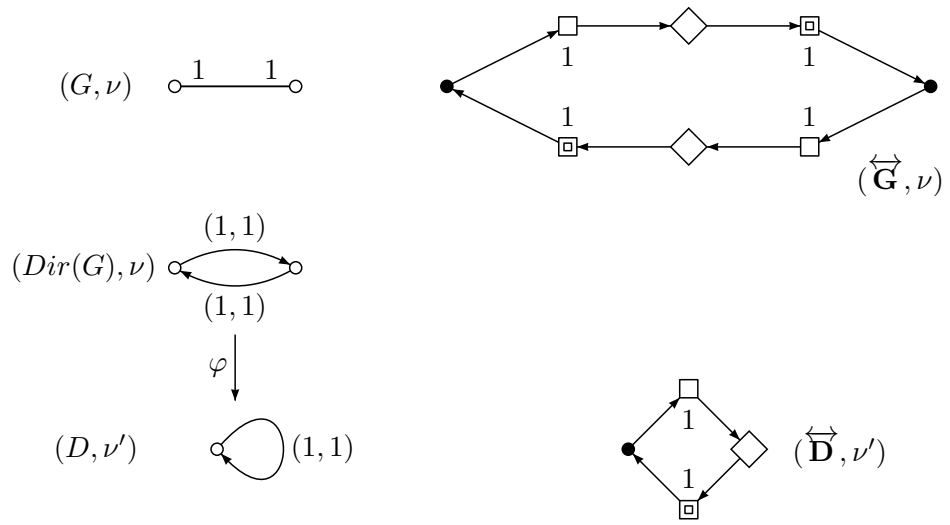



FIGURE 8.3 – Le graphe dirigé $(Dir(G),\nu)$ est un revêtement symétrique propre de $(D,\nu')$ et donc il n'existe pas d'algorithme d'élection pour $(G,\nu)$ d'après la proposition 8.9.

la façon suivante. Pour chaque sommet $v \in V(\overleftrightarrow{D})$, $\gamma(v) = \varphi(v)$ si $v \in V(D)$. Pour chaque arc $a \in A(D)$ tel que $s(a) = u$ et $t(a) = v$,
$\gamma(outbuf_a(u,v)) = outbuf_{\gamma(a)}(\gamma(u),\gamma(v))$,
$\gamma(canal_a(u,v)) = canal_{\gamma(a)}(\gamma(u),\gamma(v))$ et
$\gamma(inbuf_a(u,v)) = inbuf_{\gamma(a)}(\gamma(u),\gamma(v))$.

On vérifie que $(\overleftrightarrow{\mathbf{D}},\nu)$ est un revêtement simple de $(\overleftrightarrow{\mathbf{D}}',\nu')$ via l'homomorphisme $\gamma$. □

## 8.4 Une condition nécessaire pour pouvoir élire

On présente une condition nécessaire que doit vérifier un réseau $(\mathbf{G},\nu)$ pour lequel il existe un algorithme d'élection.

**Proposition 8.9** *Étant donnés un graphe étiqueté* $\mathbf{G}$ *et une numérotation de ports* $\nu$ *pour* $\mathbf{G}$. *Si il existe un algorithme d'élection ou de nommage pour* $\mathbf{G}$ *et* $\nu$ *alors* $(Dir(\mathbf{G}),\nu)$ *est premier pour la relation "être revêtement symétrique".*

**Preuve :** Par contradiction. Soient $\mathbf{G}$ un graphe simple étiqueté, $\nu$ une fonction de numérotation des ports de $\mathbf{G}$ et un graphe dirigé étiqueté $(\mathbf{D},\nu')$ tel que $(Dir(\mathbf{G}),\nu)$ est un revêtement symétrique propre de $(\mathbf{D},\nu')$ via un homomorphisme $\gamma$. D'après la proposition 8.8, on sait que $(\overleftrightarrow{\mathbf{G}},\nu)$ est un revêtement de $(\overleftrightarrow{\mathbf{D}},\nu')$.

Considérons un algorithme à base de messages $\mathcal{A}$ sur $(\mathbf{G},\nu)$ et l'algorithme correspondant $\mathcal{A}'$ sur $(\overleftrightarrow{\mathbf{G}},\nu)$ utilisant des calculs locaux sur les arcs obtenus à l'aide des transformations décrites ci-dessus.

On peut remarquer que s'il existe une exécution infinie de $\mathcal{A}'$ sur $(\overleftrightarrow{\mathbf{D}},\nu')$, alors il existe une exécution infinie de $\mathcal{A}'$ sur $(\overleftrightarrow{\mathbf{G}},\nu)$, et il existe une exécution non finie de $\mathcal{A}$ sur $(\mathbf{G},\nu)$. Finalement, $\mathcal{A}$ n'est pas un algorithme d'élection (ou de nommage) pour $(\mathbf{G},\nu)$.

Considérons une exécution finie $\rho$ de $\mathcal{A}'$ sur $(\overleftrightarrow{\mathbf{D}}, \nu')$. Le graphe dirigé étiqueté $(\overleftrightarrow{\mathbf{D}}, \nu')$ n'a pas de boucle et d'après le lemme 8.7, il existe une exécution $\rho'$ de $\mathcal{A}'$ sur $(\overleftrightarrow{\mathbf{G}}, \nu)$ qui est relevée de cette exécution finie de $\mathcal{A}'$ sur $(\overleftrightarrow{\mathbf{D}}, \nu')$. Par conséquent, comme le revêtement est strict, d'après la proposition 8.4, chaque étiquette qui apparaît dans la configuration finale de $\rho$ dans $(\overleftrightarrow{\mathbf{D}}, \nu')$ apparaît au moins deux fois dans la configuration finale de $\rho'$ dans $(\overleftrightarrow{\mathbf{G}}, \nu)$.

Finalement, il existe une exécution de $\mathcal{A}$ sur $(\mathbf{G}, \nu)$, où dans la configuration finale, aucun sommet n'a une étiquette en propre. Donc $\mathcal{A}$ n'est pas un algorithme d'élection pour $(\mathbf{G}, \nu)$. □

**Remarque 8.10** *Comme conséquences immédiates de ce résultat, on en déduit deux résultats classiques pour le modèle asynchrone à base de messages : il n'existe pas d'algorithme d'élection dans un réseau anonyme formé par 2 processus, et il n'existe pas d'algorithme d'élection dans ce même modèle pour un anneau de taille donnée (le schéma de ces preuves est donné dans les figures 8.1 et 8.3).*

## 8.5 Un algorithme d'énumération

On montre dans cette section que la condition nécessaire donnée ci-dessus (proposition 8.9) est également suffisante. Pour ce faire on présente un algorithme d'énumération noté $\mathcal{M}$.

On donne tout d'abord une description de l'exécution de $\mathcal{M}$ sur $\mathbf{G}$ doté d'une numérotation des ports $\nu$.

Pendant l'exécution de $\mathcal{M}$ chaque sommet $v$ essaie d'obtenir sa propre identité qui sera un numéro compris entre 1 et $|V(G)|$. Dès qu'un sommet $u$ a choisi un numéro $n(u)$ il l'envoie à chaque voisin $v$ avec le numéro de port $\nu_u(v)$. Quand un sommet $v$ reçoit un message d'un voisin $u$, il mémorise le nombre $n(u)$ avec les numéros de ports $\nu_u(v)$ et $\nu_v(u)$. À partir des informations qu'un sommet reçoit de ses voisins chaque sommet construit sa vue locale (qui est l'ensemble des numéros de ses voisins associés aux numéros de port). Puis il diffuse son numéro et sa vue locale. Si un sommet $u$ s'aperçoit qu'il existe un autre sommet $v$ avec le même numéro alors il envisage de changer de numéro. Pour ce faire il compare sa vue locale avec la vue locale de l'autre sommet. Si l'étiquette de $u$ ou sa vue locale est plus faible alors $u$ change de numéro et le diffuse accompagné de sa vue locale. Finalement, si le graphe est premier pour les revêtements symétriques alors chaque sommet aura à la fin un numéro unique compris entre 1 et le nombre de sommets de $G$.

### 8.5.1 Les étiquettes

Soit $(\mathbf{G}, \nu)$ un réseau, où $\mathbf{G} = (G, \lambda)$ est un graphe simple étiqueté et $\nu$ est une fonction de numérotation des ports de $\mathbf{G}$. La fonction $\lambda : V(G) \to L$ est la fonction d'étiquetage initiale et n'est pas modifiée pendant l'exécution de l'algorithme. On suppose qu'il existe un ordre total $<_L$ sur $L$.

L'étiquette de chaque sommet $v$ est un tuple $(\lambda(v), n(v), N(v), M(v))$ représentant l'information suivante :

- $\lambda(v) \in L$ est l'étiquette initiale de $v$ et n'est pas modifiée par l'algorithme,
- $n(v) \in \mathbb{N}$ est le numéro actuel du sommet $v$, ce numéro est calculé par l'algorithme,
- $N(v) \in \mathcal{P}_{\text{fin}}(\mathbb{N}^3)$ [1] est la vue locale de $v$. La vue locale d'un sommet $v$ contient les informations qu'un sommet a sur ses voisins. Si un sommet $v$ a un voisin $u$ tel que $\nu_u(v) = p$

1. Pour un ensemble $S$, $\mathcal{P}_{\text{fin}}(S)$ désigne l'ensemble des sous-ensembles finis de $S$.

et $\nu_v(u) = q$, alors $(m, p, q) \in N(v)$ si le dernier message que $v$ reçoit de $u$ indique que $n(u) = m$.

— $M(v) \in \mathbb{N} \times L \times \mathcal{P}_{\text{fin}}(\mathbb{N}^3)$ est la boîte aux lettres de $v$. Elle contient toutes les informations reçues par $v$ depuis le début de l'exécution de l'algorithme. Si $(m, \ell, \mathcal{N}) \in M(v)$ alors lors de l'exécution d'un pas de l'algorithme, il y avait un sommet $u$ tel que $n(u) = m$, $\lambda(u) = \ell$ et $N(u) = \mathcal{N}$.

Initialement, chaque chaque sommet $v$ a une étiquette de la forme $(\lambda(v), 0, \emptyset, \emptyset)$.

Lors du déroulement de l'algorithme $\mathcal{M}$, les processus échangent des informations de la forme $< (m, n_{old}, M), p >$. Si un sommet $u$ envoie le message $< (m, n_{old}, M), p >$ à son voisin $v$, alors le message contient les informations suivantes :

— $m$ est le numéro actuel $n(u)$ de $u$,

— $n_{old}$ est le numéro précédant de $u$, i.e., le numéro que $u$ a envoyé à $v$ dans son message précédant ; si $u$ n'a pas changé de numéro dans l'intervalle de temps alors $n_{old} = m$,

— $M$ est la boîte aux lettres de $u$,

— $p$ est le numéro du port à travers lequel le message a été envoyé, i.e., $p = \nu_u(v)$.

### 8.5.2 Un ordre sur les vues locales

L'algorithme $\mathcal{M}$ repose sur un ordre total sur les vues, i.e., sur les sous-ensembles finis de $\mathbb{N}^3$. On considère l'ordre lexicographique sur $\mathbb{N}^3$ : $(n, p, q) < (n', p', q')$ si $n < n'$, ou si $n = n'$ et $p < p'$, ou si $n = n'$, $p = p'$ et $q < q'$.

Puis on utilise l'ordre sur les ensembles finis défini par : étant donnés deux ensembles distincts $N_1, N_2 \in \mathcal{P}_{\text{fin}}(\mathbb{N}^3)$, on définit $N_1 \prec N_2$ si le maximum de la différence symétrique $N_1 \triangle N_2 = (N_1 \setminus N_2) \cup (N_2 \setminus N_1)$ appartient à $N_2$.

Si $N(u) \prec N(v)$, alors on dit que la vue locale $N(v)$ de $v$ est plus forte que la vue locale $N(u)$ de $u$ et que $N(u)$ est plus faible que $N(v)$. En utilisant l'ordre total $<_L$ sur $L$, on définit $(\ell, \mathcal{N}) \prec (\ell', \mathcal{N}')$ si soit $\ell <_L \ell'$, soit $\ell = \ell'$ et $\mathcal{N} \prec \mathcal{N}'$. On note $\preceq$ la clôture réflexive de $\prec$.

### 8.5.3 L'algorithme $\mathcal{M}$

### 8.5.4 Preuve de $\mathcal{M}$

On considère un graphe simple connexe et étiqueté $\mathbf{G}$ et une numérotation des ports $\nu$ de $\mathbf{G}$.

Une exécution de l'algorithme sur $(\mathbf{G}, \nu)$ est une suite d'émissions de messages, de réceptions de messages et d'actions internes, où à chaque pas une seule transition est effectuée.

On considère une exécution $\rho$ de $\mathcal{M}$ sur $(\mathbf{G}, \nu)$ et pour chaque sommet $v \in V(G)$, on note $(\lambda(v), n_i(v), N_i(v), M_i(v))$ l'état de $v$ après le $i$ème pas de calcul de $\rho$. On remarque tout d'abord que les pas qui correspondent à des émissions de messages ou des réceptions de messages ne changent pas la valeur de $(\lambda(v), n(v), N(v), M(v))$ pour tout sommet $v \in V(G)$. De plus, pour chaque pas de calcul où un calcul interne est effectué la valeur de $(\lambda(v), n(v), N(v), M(v))$ n'est modifiée que pour au plus un sommet $v \in V(G)$.

On a :

**Lemme 8.11** *Pour chaque sommet $v \in V(G)$ et pour chaque pas $i$ :*

1. $\exists (n, p, q) \in N_i(v) \iff \exists v' \in N_G(v)$ *tel que* $\nu_v(v') = q$ *et* $\nu_{v'}(v) = p$,
2. $n_i(v) \neq 0 \implies (n_i(v), \lambda(v), N_i(v)) \in M_i(v)$,
3. $\forall (n, p, q) \in N_i(v), n \neq 0$ *et* $\exists (n, \ell', \mathcal{N}') \in M_i(v)$,
4. $\forall (n, p, q), (n', p', q') \in N_i(v), q \neq q'$,

**Algorithme 8:** l'algorithme $\mathcal{M}$.

**I** : $\{n(v_0) = 0$ et aucun message n'est parvenu à $v_0\}$
**début**
  $n(v_0) := 1$ ;
  $M(v_0) := \{(n(v_0), \lambda(v_0), \emptyset)\}$;
  **pour** $i := 1$ ***to*** $\deg(v_0)$ **faire**
    **envoyer** $< (n(v_0), 0, M(v_0)), i >$ via port $i$ ;
**fin**

**R** : $\{$Un message $< (n', n'_{old}, M'), p >$ est parvenu à $v_0$ à travers le port $q\}$
**début**
  $M_{old} := M(v_0)$;
  $n_{old} := n(v_0)$;
  $M(v_0) := M(v_0) \cup M'$;
  **si** $n(v_0) = 0$ *ou* $\exists (n(v_0), \ell, \mathcal{N}) \in M(v_0)$ *tel que* $(\lambda(v_0), N(v_0)) \prec (\ell, \mathcal{N})$ **alors**
    $n(v_0) := 1 + \max\{n \mid \exists (n, \ell, \mathcal{N}) \in M(v_0)\}$;
  $N(v_0) := N(v_0) \setminus \{(n'_{old}, p, q)\} \cup \{(n', p, q)\}$;
  $M(v_0) := M(v_0) \cup \{(n(v_0), \lambda(v_0), N(v_0))\}$;
  **si** $M(v_0) \neq M_{old}$ **alors**
    **pour** $i := 1$ ***to*** $\deg(v_0)$ **faire**
      **envoyer** $< (n(v_0), n_{old}, M(v_0)), i >$ à travers le port $i$;
**fin**

5. $\forall (n(v_0), \ell, \mathcal{N}) \in M_i(v)$, $(\ell, \mathcal{N}) \preceq (\lambda(v_0), N(v_0))$,
6. $(n, p, q) \in N_i(v)$ *si et seulement si le dernier message reçu par* $v$ *à travers le port* $q$ *était* $< (n, M), p >$ *pour* $M \subseteq M_i(v)$.

**Lemme 8.12** *Pour chaque sommet* $v$ *et chaque pas* $i$, $n_i(v) \leq n_{i+1}(v)$, $N_i(v) \preceq N_{i+1}(v)$, *et* $M_i(v) \subseteq M_{i+1}(v)$.

**Preuve :** On suppose qu'une action interne est exécutée au pas $i$ par le sommet $v \in V(G)$. La propriété est vraie pour tout sommet $w \in V(G) \setminus \{v\}$ et $M_i(v) \subseteq M_{i+1}(v)$.

Si $n_i(v) \neq n_{i+1}(v)$, alors $n_{i+1}(v) = 1 + \max\{n' \mid (n', \ell', \mathcal{N}') \in M_i(v)\}$ et soit $n_i(v) = 0 < n_{i+1}(v)$ soit $(n_i(v), \lambda(v), N_i(v)) \in M_i(v)$ d'après le lemme 8.11 et donc $n_i(v) < n_{i+1}(v)$.

Si $N_i(v) \neq N_{i+1}(v)$, alors $v$ a reçu le message $< (n', n'_{old}, M'), p >$ à travers le port $q$ et $N_{i+1}(v) = N_i(v) \setminus \{(n'_{old}, p, q)\} \cup \{(n', p, q)\}$. Soit $v'$ le voisin de $v$ tel que $\nu_v(v') = q$ ; on sait que $\nu_{v'}(v) = p$.

Si $(n'_{old}, p, q) \notin N_i(v)$, alors $\max N_{i+1}(v) \triangle N_i(v) = (n', p, q) \in N_{i+1}(v)$ et $N_i(v) \prec N_{i+1}(v)$.

Si $(n'_{old}, p, q) \in N_i(v)$, alors $n'_{old} \neq n'$. Soit $j < i+1$ le pas de calcul où $v'$ a envoyé le message $< (n', n'_{old}, M'), p >$. On sait que $n'_{old} \leq n' = n_j(v')$ et par conséquent $\max N_{i+1}(v) \triangle N_i(v) = (n', p, q) \in N_{i+1}(v)$ et $N_i(v) \prec N_{i+1}(v)$. □

**Lemme 8.13** *Pour chaque sommet* $v \in V(G)$, *chaque pas* $i$ *et chaque* $(m, \ell, \mathcal{N}) \in M_i(v)$, *il existe un sommet* $v' \in V(G)$ *tel que* $n_i(v') = m$.

**Preuve :** On remarque que $(m, \ell, \mathcal{N})$ a été ajouté à $\bigcup_{v \in V(G)} M_i(v)$ à l'étape $i$ s'il existe un sommet $v \in V(G)$ tel que $n_i(v) = m$, $\lambda(v) = \ell$ et $N_i(v) = \mathcal{N}$.

Étant donnés un sommet $v \in V(G)$, une étape $i$ et $(m, \ell, \mathcal{N}) \in M_i(v)$, soit $U = \{(u, j) \in V(G) \times \mathbb{N} \mid j \leq i, n_j(u) = m\}$ et soit $U' = \{(u, j) \in U \mid \forall (u', j') \in U, (\lambda(u'), N_{j'}(u')) \prec (\lambda(u), N_j(u))$ or $(\lambda(u'), N_{j'}(u')) = (\lambda(u), N_j(u))$ and $j' \leq j\}$. Puisque $(m, \ell, \mathcal{N}) \in M_i(v)$, $U$ et $U'$ sont tous les deux non vides il existe $i_0$ tel que pour chaque $(u, j) \in U'$, $j = i_0$.

Si $i_0 < i$, soit $(u, i_0) \in U'$ ; on sait que $n_{i_0+1}(u) \neq n_{i_0}(u)$, mais ceci est impossible du fait de la maximalité de $(\lambda(u), N_{i_0}(u))$, $u$ n'a pas pu modifier son numéro. Donc $i_0 = i$ et il existe $v' \in V(G)$ tel que $n_i(v') = m$. □

**Lemme 8.14** *Pour chaque sommet* $v \in V(G)$ *et chaque étape* $i$, *pour tout* $(m, \ell, \mathcal{N}) \in M_i(v)$, *pour tout* $m' \in [1, m]$, *il existe* $(m', \ell', \mathcal{N}') \in M_i(v)$.

**Preuve :** Par induction sur $i$. Initiallement la propriété est vraie. On suppose que la propriété est vraie à l'étape $i$ et qu'un sommet $v$ exécute une transition interne à l'étape $i+1$. Si $v$ exécute l'action **I** au pas $i+1$, alors $M_i(v) = \{(1, \lambda(v), \emptyset)\}$ et la propriété est vraie.

Si $v$ exécute l'action **R** au aps $i+1$, alors $v$ a reçu un message $< (n', n'_{old}, M), p >$ d'un voisin $v'$. Soit $j < i+1$ le pas de calcul où $v'$ a envoyé ce message ; on sait que $M' = M_j(v')$. Si $v$ ne modifie son numéro au pasdoes not modify its $i+1$, alors $\{m \mid \exists (m, \ell, \mathcal{N}) \in M_{i+1}(v)\} = \{m \mid \exists (m, \ell, \mathcal{N}) \in M_i(v) \cup M_j(v')\}$ et la propriété est vraie au pas $i+1$ par l'hypothèse d'induction. Si $v$ modifie son numéro au aps $i+1$, alors $n_{i+1}(v) = 1 + \max\{m \mid \exists (m, \ell, \mathcal{N}) \in M_i(v) \cup M_j(v')\}$ et $M_{i+1}(v) = M_i(v) \cup M_j(v') \cup \{(n_{i+1}(v), \lambda(v), N_{i+1}(v)\}$. Donc la propriété est également satisfaite au pas $i$ d'après l'hypothèse d'induction. □

**Lemme 8.15** *Toute exécution $\rho$ de $\mathcal{M}$ sur un graphe étiqueté simple $\mathbf{G} = (G, \lambda)$ muni d'une numérotation de port $\nu$ termine et l'étiquetage final $(\lambda, n_\rho, N_\rho, M_\rho)$ des sommets de $G$ vérifie :*

1. *il existe un entier $k \leq |V(G)|$ tel que $\{n_\rho(v) \mid v \in V(G)\} = [1, k]$,*

*et pour tous sommets $v, v' \in V(G)$ :*

2. *$M_\rho(v) = M_\rho(v')$,*
3. *$(n_\rho(v), \lambda(v), N_\rho(v)) \in M_\rho(v')$,*
4. *si $n_\rho(v) = n_\rho(v')$, alors $\lambda(v) = \lambda(v')$ et $N_\rho(v) = N_\rho(v')$,*
5. *$(n, p, q) \in N_\rho(v)$ si et seulement si il existe $w \in N_G(v)$ tel que $\nu_v(w) = q$, $\nu_w(v) = p$ et $n_\rho(w) = n$.*

**Preuve :**

1. D'après les lemmes 8.13 et 8.14 puisque tout sommet a appliqué une des actions $\mathbf{I}, \mathbf{R}$.
2. Puisqu'à chaque fois qu'un sommet modifie sa boîte aux lettres il l'envoie à ses voisins et tous les messages sont parvenus à leur destination.
3. C'est une conséquence de la propriété précédente et du lemme 8.11.
4. D'après le lemme 8.11.
5. D'après le lemme 8.11 et puisque tous les messages sont arrivés.

□

**Proposition 8.16** *Étant donné un graphe étiqueté $\mathbf{G}$ muni d'une numérotation des ports $\nu$, on peut associer à l'étiquetage final de toute exécution $\rho$ de $\mathcal{M}$ sur $(\mathbf{G}, \nu)$, un graphe dirigé $(\mathbf{D}, \nu')$ tel que $(Dir(\mathbf{G}), \nu)$ est un revêtement symétrique de $(\mathbf{D}, \nu')$.*

**Preuve :** On utilise les notations du lemme 8.15.

On considère le graphe dirigé $D$ défini par $V(D) = \{m \in \mathbb{N} \mid \exists v \in V(G), n_\rho(v) = m\}$, $A(D) = \{a_{(n,p,q,m)} \mid \exists \{v, w\} \in E(G)$ tel que $n_\rho(v) = n, n_\rho(w) = m, \delta_v(w) = p, \delta_w(v) = q\}$ et pour chaque arc $a_{(n,p,q,m)}$, $s(a_{(n,p,q,m)}) = n$ et $t(a_{(n,p,q,m)}) = m$. On définit la fonction $Sym : A(D) \to A(D)$ de la façon suivante : pour chaque arc $a_{(n,p,q,m)} \in A(D)$, $Sym(a_{(n,p,q,m)}) = a_{(m,q,p,n)}$. On peut remarquer que pour chaque arc $a_{(n,p,p,n)}$,

$$Sym(a_{(n,p,p,n)}) = a_{(n,p,p,n)}.$$

On définit un étiquetage $\eta$ des sommets de $D$ comme suit. Pour chaque $v \in V(G)$, $\eta(n_\rho(v)) = \lambda(v)$. D'après le lemme 8.15, deux sommets $v, v'$ avec le même numéro final ont la même étiquette $\lambda(v)$ donc cet étiquetage est bien défini. On définit l'étiquetage $\nu'$ des arcs de $D$ par : pour chaque arc $a_{(n,p,q,m)} \in A(D)$, $\nu'(a_{(n,p,q,m)}) = (p, q)$. Donc pour chaque arc $a_{(n,p,q,m)}$, $\nu'(Sym(a_{(n,p,q,m)})) = \nu'(a_{(m,q,p,n)}) = (q, p)$ ainsi l'étiquetage des arcs de $D$ est symétrique.

On définit un homomorphisme $\varphi$ de $(Dir(\mathbf{G}), \nu)$ sur $(\mathbf{D}, \nu)$ par : pour chaque $v \in V(Dir(G))$, $\varphi(v) = n_\rho(v)$ et pour chaque $a \in A(Dir(G))$

$$\varphi(a) = a_{(n_\rho(s(a)), \nu_{s(a)}(t(a)), \nu_{t(a)}(s(a)), n_\rho(t(a)))}.$$

Puisque pour chaque arc $a \in A(Dir(\mathbf{G}))$, $s(\varphi(a)) = n_\rho(s(a)) = \varphi(s(a))$ et $t(\varphi(a)) = n_\rho(t(a)) = \varphi(t(a))$, $\varphi$ est un homomorphisme de $Dir(G)$ sur $D$.

Sachant que pour chaque arc $a \in A(Dir(G))$, $\nu(a) = (\nu_{s(a)}(t(a)), \nu_{t(a)}(s(a))) = \nu'(\varphi(a))$ et que pour chaque sommet $v \in V(Dir(G))$, $\lambda(v) = \eta(\varphi(v))$, $\varphi$ est un homomorphisme de $(Dir(\mathbf{G}), \nu)$ sur $(\mathbf{D}, \nu')$. De plus pour chaque arc $a \in A(Dir(G))$, $\varphi(Sym(a)) = a_{(n_\rho(t(a)),\nu_{t(a)}(s(a)),\nu_{s(a)}(t(a)),n_\rho(s(a)))} = Sym(\varphi(a))$.

Pour chaque arc $a_{(n,p,q,m)} \in A(D)$, il existe une arête $\{v, w\} \in E(G)$ telle que $n_\rho(v) = n$, $\nu_v(w) = q$, $\nu_w(v) = p$ et $n_\rho(w) = m$. D'après le lemme 8.15, on sait que pour chaque $u \in \varphi^{-1}(n)$ (resp. $u \in \varphi^{-1}(m)$), $N_\rho(u) = N_\rho(v)$ (resp. $N_\rho(u) = N_\rho(w)$) et donc, sachant que $\nu$ est une bijection entre $N_G(u)$ et $[1, \deg_G(u)]$, il existe un unique $u' \in N_G(u)$ tel que $n_\rho(u') = m$ (resp. $n_\rho(u') = n$), $\nu_u(u') = p$ (resp. $\nu_u(u') = q$) et $\nu_{u'}(u) = q$ (resp. $\nu_{u'}(u) = p$). Donc pour chaque arc $a_{(n,p,q,m)} \in A(D)$, pour chaque sommet $u \in \varphi^{-1}(s(a))$ (resp. $u \in \varphi^{-1}(t(a))$), il existe un unique arc $a \in Dir(G)$ tel que $\varphi(a) = a_{(n,p,q,m)}$ et $s(a) = u$ (resp. $t(a) = u$). Donc $(Dir(\mathbf{G}), \nu)$ est un revêtement symétrique de $(\mathbf{D}, \nu')$. □

**Théorème 8.17** *Étant donné un graphe étiqueté simple* $\mathbf{G}$ *muni d'une numérotation des ports* $\nu$*, il existe un algorithme d'élection (ou de nommage) pour* $(\mathbf{G}, \nu)$ *si et seulement si* $(Dir(\mathbf{G}), \nu)$ *est premier pour la relation être revêtement symétrique.*

**Remarque 8.18** *On peut remarquer que toute exécution de* $\mathcal{M}$ *termine et que la connaissance de la taille du graphe suffit à la détection de la terminaison.*

On a :

**Théorème 8.19** *Étant donné un graphe simple étiqueté* $\mathbf{G}$*, il existe un algorithme d'élection (ou de nommage) pour* $\mathbf{G}$ *si et seulement si* $Dir(\mathbf{G})$ *est premier pour la relation être revêtement symétrique.*

# Chapitre 9

# Élection dans des familles de graphes

Une caractérisation des graphes pour lesquels il existe un algorithme d'élection a été présentée dans le chapitre précédent. La question suivante concerne l'existence d'un algorithme pour tous les graphes d'une famille. Par exemple Dana Angluin a montré (en considérant son modèle) :

— qu'il existe un algorithme d'élection pour la famille des graphes complets,
— qu'il existe un algorithme d'élection pour la famille des arbres,
— qu'il n'existe pas d'algorithme qui permet d'élire dans la famille formée des arbres et du triangle.

Le but de ce chapitre est de caractériser les familles de graphes admettant un algorithme d'élection.

Les algorithmes qui fonctionnent pour une famille de graphes sont des algorithmes qui fonctionnent sur des graphes dynamiques et finalement sont tolérants à certaines fautes.

De fait, les conséquences de la caractérisation qui est présentée dans ce chapitre peuvent se voir comme un appauvrissement des hypothèses vues dans le chapitre précédent :

— les sommets n'ont pas nécessairement des identités distinctes (graphes partiellement anonymes),
— connaissance d'une borne supérieure de la taille du graphe ou de son diamètre,
— ou toute combinaison de ces hypothèses.

Le modèle considéré est le même que pour le chapitre précédent, c'est à dire, le modèle point-à-point avec passage de messages en mode asynchrone et une numérotation des ports. Ceci étant la caractérisation obtenue "s'étend" à d'autres modèles : celui d'Angluin, le modèle point-à-point avec passage de messages en mode synchrone, plus généralement le modèle des calculs locaux sur les arêtes ou bien le modèle de Mazurkiewicz. (Ce chapitre est extrait de : [CGM12].)

## 9.1 Quelques rappels

Soit $(G, \lambda)$ un graphe étiqueté avec une numérotation des ports $\delta$. On désigne par $(Dir(\mathbf{G}), \delta)$ le graphe orienté symétrique étiqueté $(Dir(G), (\lambda, \delta))$ obtenu de la façon suivante. Les sommets de $Dir(G)$ sont les sommets de $G$ avec les mêmes étiquettes que $\mathbf{G}$. Chaque arête $\{u, v\}$ de $G$ est remplacée dans $(Dir(\mathbf{G}), \delta)$ par 2 arcs $a_{(u,v)}, a_{(v,u)} \in A(Dir(G))$ tels que $s(a_{(u,v)}) = t(a_{(v,u)}) = u$, $t(a_{(u,v)}) = s(a_{(v,u)}) = v$, $\delta(a_{(u,v)}) = (\delta_u(v), \delta_v(u))$ et $\delta(a_{(v,u)}) = (\delta_v(u), \delta_u(v))$. Ce graphe orienté est sans boucle et sans arc multiple. L'objet que l'on utilise dans cette étude est $(Dir(G), (\lambda, \delta))$ et les résultats sont énoncés avec des graphes orientés étiquetés et symétriques.

L'existence d'un algorithme d'élection pour une famille de graphes est liée au problème de la détection de la terminaison d'un algorithme distribué. En effet, le chapitre précédent présente un algorithme "universel" d'élection au sens où cet algorithme permet d'élire dans un graphe dès lors

qu'il est possible d'élire dans ce graphe. Donc si on considère une famille de graphes cet algorithme permet de distinguer un sommet dans chacun des graphes de cette famille : en considérant par exemple le sommet ayant le plus grand numéro (ou le plus petit numéro) ; le problème de l'existence d'un algorithme d'élection pour cette famille se ramène ainsi à la détection de la terminaison de l'algorithme "universel" sur tous les éléments de cette famille et pour un sommet de décider si il a ou non le plus grand numéro (ou le plus petit numéro).

Finalement, l'algorithme d'élection pour une famille qui est présenté dans ce chapitre est la combinaison de l'algorithme universel d'élection du chapitre précédent et d'un algorithme de détection de terminaison. La détection de la terminaison repose sur la notion de quasi-revêtement, qui est présentée précisément plus loin ; les quasi-revêtements généralisent les revêtements et permettent de définir des graphes qui localement se comportent comme des revêtements.

On obtient ainsi le théorème suivant :

**Théorème 9.1** *Soit $\mathcal{I}$ une famille récursive de graphes orientés étiquetés symétriques. Il existe un algorithme d'élection pour $\mathcal{I}$ si et seulement si chaque graphe orienté étiqueté de $\mathcal{I}$ est minimal pour la relation être revêtement symétrique et il existe une fonction calculable $\tau : \mathcal{I} \to \mathbb{N}$ telle que pour chaque graphe orienté etiqueté symétrique $\mathbf{D}$ de $\mathcal{I}$, il n'existe pas de quasi-revêtement de $\mathbf{D}$ de rayon supérieur à $\tau(\mathbf{D})$ dans $\mathcal{I}$, sauf $\mathbf{D}$ lui-même.*

On rappelle qu'une exécution synchrone d'un algorithme distribué est une exécution qui peut être décomposée en round. À chaque round chaque processus reçoit les messages envoyés lors du round précédent, puis, en fonction des messages reçus il peut exécuter des calculs internes et envoyer des messages.

**Remarque 9.2** *Étant donné un graphe étiqueté $\mathbf{G} = (G, \lambda)$ muni d'une numérotation de ports $\delta$, soit $\mathbf{D} = (Dir(\mathbf{G}), \delta)$ le graphe orienté étiqueté correspondant $(Dir(G), (\lambda, \delta))$. Soit $\mathcal{A}$ un algorithme distribué. On parlera indifféremment d'une exécution de $\mathcal{A}$ sur $(\mathbf{G}, \delta)$ ou sur $\mathbf{D}$.*

D'après le chapitre précédent on a :

**Théorème 9.3** *Étant donné un graphe simple étiqueté $\mathbf{G} = (G, \lambda)$ muni d'une numérotation des ports $\delta$, il existe un algorithme d'élection pour $(\mathbf{G}, \delta)$ si et seulement si $(Dir(G), (\lambda, \delta))$ est premier pour la relation être revêtement symétrique.*

On rappelle avec une légère modification l'algorithme d'élection.

### 9.1.1 Un algorithme d'élection pour un graphe étiqueté premier

**Les étiquettes.**

On considère un réseau $(\mathbf{G}, \delta)$ où $\mathbf{G} = (G, \lambda)$ est un graphe simple et $\delta$ est une numérotation des ports de $\mathbf{G}$. La fonction $\lambda : V(G) \to L$ est l'étiquetage initial. On suppose qu'il existe un ordre total $<_L$ sur $L$. On l'étend à $L \cup \{\perp\}$ (en supposant que $\perp \notin L$) par : pour tout $\ell \in L$, $\perp < \ell$.

L'étiquette d'un sommet $v$ est un tuple $(\lambda(v), n(v), N(v), M(v))$ où :

— $\lambda(v) \in L$ est l'étiquette initiale de $v$.

— $n(v) \in \mathbb{N}$ est le numéro courant de $v$ calculé par l'algorithme, initialement $n(v) = 0$.

— $N(v) \in \mathcal{P}_{\text{fin}}(\mathbb{N}^3)$ [1] est la *vue locale* de $v$. À la fin de l'exécution, si $(m, \ell, p, q) \in N(v)$, alors $v$ a un voisin $u$ dont le numéro est $m$, l'étiquette $\ell$ et l'arc allant de $u$ à $v$ est étiqueté $(p, q)$. Initialement $N(v) = \{(0, \perp, 0, q) \mid q \in [1, \deg_G(v)]\}$.

1. Pour tout ensemble $S$, $\mathcal{P}_{\text{fin}}(S)$ est l'ensemble des sous-ensembles finis de $S$.

— $M(v)$ est un ensemble, c'est la boîte aux lettres de $v$ ; initialement $M(v) = \emptyset$. Un élément de $M(v)$ est de la forme : $(m, \ell, N)$ où $m \in \mathbb{N}$, $\ell \in L$ et $N$ est une vue locale. Elles contient toutes les informations reçues par $v$ pendant l'exécution de l'algorithme. Si $(m, \ell, N) \in M(v)$, cela signifie qu'à un pas précédent de l'exécution, il existait un sommet $u$ tel que $n(u) = m$, $\lambda(u) = \ell$ et $N(u) = N$.

**Messages.**

Les processus échangent des messages de la forme $< (n, \ell, M), p >$. Si un sommet $u$ envoie un message $< (n, \ell, M), p >$ à un voisin $v$, alors le message contient les informations suivantes : $n$ est le numéro courant de $u$, $\ell$ est l'étiquette $\lambda(u)$ de $u$, $M$ est la boîte aux lettres de $u$, et $p = \delta_u(v)$.

**Un ordre sur les vues locales.**

Une propriété clé de l'algorithme repose sur un ordre total sur les vues locales défini de la façon suivante. Étant donnés deux ensembles distincts $N_1, N_2 \in \mathcal{P}_{\text{fin}}(\mathbb{N}^3)$, on dit que $N_1 \prec N_2$ si le maximum de la différence symétrique $N_1 \triangle N_2 = (N_1 \setminus N_2) \cup (N_2 \setminus N_1)$ pour l'ordre lexicographique appartient à $N_2$.

On dit également que $(\ell, N) \prec (\ell', N')$ si $\ell <_L \ell'$, ou ($\ell = \ell'$ et $N \prec N'$). On note $\preceq$ la clôture réflexive de $\prec$.

### 9.1.2 L'algorithme "universel" d'élection $\mathcal{M}$

Les conventions et les notations sont identiques à celles du chapitre précédent.

### 9.1.3 Rappel de quelques propriétés de $\mathcal{M}$

On considère une exécution $\rho$ de $\mathcal{M}$ sur $(\mathbf{G}, \delta)$ et pour chaque sommet $v \in V(G)$, on note $(\lambda(v), n_i(v), N_i(v), M_i(v))$ l'état de $v$ après le $i^{ieme}$ pas de calcul de $\rho$ sur $v$. Si le sommet $v$ exécute un pas de calcul entre $i$ et $i+1$, alors il est dit actif au pas $i+1$. On rappelle que :

**Proposition 9.4** *Soit un sommet $v$ et un pas de calcul $i$.*
*Alors, $n_i(v) \leq n_{i+1}(v)$, $N_i(v) \preceq N_{i+1}(v)$, $M_i(v) \subseteq M_{i+1}(v)$.*
*Pour chaque $(m, \ell, N) \in M_i(v)$ et chaque $m' \in [1, m]$, $\exists (m', \ell', N') \in M_i(v), \exists v' \in V(G)$ tels que $n_i(v') = m'$. Si $m = n_i(v)$, $(\ell, N) \leq (\lambda_i(v), N_i(v))$.*

Soit la boîte aux lettres $M = M(v)$ d'un sommet $v$ pendant l'exécution de $\mathcal{M}$ sur le graphe $(\mathbf{G}, \delta)$. On dit que $(n, \ell, N) \in M$ est *maximal* dans $M$ s'il n'existe pas $(n, \ell', N') \in M$ tel que $(\ell, N) \prec (\ell', N')$. On note $S(M)$ l'ensemble des éléments maximaux de $M$. D'après la proposition 9.4, après chaque pas de l'algorithm $\mathcal{M}$, $(n(v), \lambda(v), N(v))$ est maximal dans $M(v)$.

L'ensemble $S(M)$ est dit *cohérent* s'il est non vide et pour tout $(n_1, \ell_1, N_1) \in S(M)$, pour tout $(n_2, \ell_2, p, q) \in N_1$, $p \neq 0$, $n_2 \neq 0$ et $\ell_2 \neq \perp$ et pour tout $(n_2', \ell_2', N_2') \in S(M)$, il existe $(n_2', \ell_2'', p', q') \in N_1$ si et seulement si $\ell_2' = \ell_2''$ et $(n_1, \ell_1, q', p') \in N_2'$.

D'après le chapitre précédent, on sait qu'une fois que $n(v)$, $N(v)$ et $M(v)$ ont atteint leurs valeurs finales pour tout $v$, alors $S(M(v))$ est cohérent pour tout $v$. Donc, si $S(M(v))$ n'est pas cohérent, on sait que $M(v)$ sera modifié.

Si l'ensemble $S(M)$ est cohérent, on peut construire un graphe orienté étiqueté symétrique $\mathbf{D}_M = (D_M, \lambda_M)$ de la façon suivante. L'ensemble des sommets $V(D_M)$ est l'ensemble $\{n \mid \exists (n, \ell, N) \in S(M)\}$. Pour tout $(n, \ell, N) \in S(M)$ et pour tout $(n', \ell', p, q) \in N$, il existe un arc

**Algorithme 9:** Algorithm $\mathcal{M}$.

**I** : $\{n(v_0) = 0$ et aucun message n'est parvenu à $v_0\}$
**begin**
  $n(v_0) := 1$ ;
  $M(v_0) := \{(n(v_0), \lambda(v_0), \emptyset)\}$ ;
  **for** $i := 1$ ***to*** $\deg(v_0)$ **do**
    **send** $< (n(v_0), \lambda(v_0), M(v_0)), i >$ through $i$ ;
**end**

**R** : {Un message $< (n_1, \ell_1, M_1), p_1 >$ est parvenu Ã  $v_0$ via le port $q_1\}$
**begin**
  $M_{old} := M(v_0)$ ;
  $M(v_0) := M(v_0) \cup M_1$ ;
  **if** $n(v_0) = 0$ *ou* $\exists (n(v_0), \ell', N') \in M(v_0)$ *tel que* $(\lambda(v_0), N(v_0)) \prec (\ell', N')$ **then**
    $n(v_0) := 1 + \max\{n' \mid \exists (n', \ell', N') \in M(v_0)\}$ ;
  $N(v_0) := N(v_0) \setminus \{(n', \ell', p', q_1) \mid \exists (n', \ell', p', q_1) \in N(v_0)\} \cup \{(n_1, \ell_1, p_1, q_1)\}$ ;
  $M(v_0) := M(v_0) \cup \{(n(v_0), \lambda(v_0), N(v_0))\}$ ;
  **if** $M(v_0) \neq M_{old}$ **then**
    **for** $i := 1$ ***to*** $\deg(v_0)$ **do**
      **send** $< (n(v_0), \lambda(v_0), M(v_0)), i >$ à travers le port $i$ ;
**end**

$a_{n,n',p,q} \in A(D_M)$ tel que $t(a) = n, s(a) = n'$, $\lambda_M(a) = (p, q)$. Puisque $S(M)$ est cohérent, on peut définir $Sym$ par $Sym(a_{n,n',p,q}) = a_{n',n,q,p}$.

On montre que $\mathcal{M}$ termine et que l'étiquetage final vérifie : $(Dir(G), (\lambda, \delta))$ est un revêtement symétrique de $\mathbf{D}_M$. Donc si $(Dir(G), (\lambda, \delta))$ est premier alors $\mathbf{D}_M$ est isomorphe à $(Dir(G), (\lambda, \delta))$ et donc l'ensemble des numéros est $[1, |V(G)|]$ : chaque sommet a un numéro unique. De plus, la détection de la terminaison de l'algorithme est possible sur $\mathbf{G}$. Quand un sommet a obtenu comme numéro $|V(G)|$ (qui est connu de chaque sommet), il sait que les sommets ont tous des numéros différents qui ne changeront plus et il peut en conclure que l'attribution des numéros est terminée. L'algorithme d'élection est terminé.

## 9.2 Quasi-revêtements et le problème de l'élection pour une famille de graphes orientés étiquetés

Cette section présente le deuxième outil que l'on utilise : les quasi-revêtements ; il nous permettra d'énoncer une condition nécessaire pour l'existence d'un algorithme d'élection pour une famille de graphes étiquetés. On verra par la suite que cette condition est suffisante.

### 9.2.1 Quasi-revêtements

Les quasi-revêtements ont été introduits pour étudier le problème de la détection de la terminaison. Cette définition est illustrée sur la figure 9.1.

**Définition 9.5** *Étant donnés deux graphes orientés étiquetés symétriques* $\mathbf{D}_0, \mathbf{D}_1$*, un entier* $r$*, un sommet* $v_1 \in V(D_1)$ *et un homomorphisme* $\gamma$ *de* $\mathbf{B}_{\mathbf{D}_1}(v_1, r)$ *sur* $\mathbf{D}_0$*, le graphe orienté* $\mathbf{D}_1$ *est un* quasi-revêtement *de* $\mathbf{D}_0$ *de centre* $v_1$ *et de rayon* $r$ *via* $\gamma$ *si il existe un graphe orienté étiqueté symétrique* $\mathbf{D}_2$ *qui est un revêtement symétrique de* $\mathbf{D}_0$ *via un homomorphisme* $\varphi$ *et il existe* $v_2 \in V(D_2)$ *et un isomorphisme* $\psi$ *de* $\mathbf{B}_{\mathbf{D}_1}(v_1, r)$ *sur* $\mathbf{B}_{\mathbf{D}_2}(v_2, r)$ *tel que pour tout* $x \in V(B_{D_1}(v_1, r)) \cup A(B_{D_1}(v_1, r))$, $\gamma(x) = \varphi(\psi(x))$.

*On définit le nombre de feuillets du quasi-revêtement comme étant le cardinal minimum de l'ensemble des préimages des sommets de* $\mathbf{D}_0$ *qui sont dans la boule :* $q = \min_{v \in V(D_0)} |\{w \in \psi^{-1}(v) | B_{\mathbf{D}_1}(w, 1) \subset B_{\mathbf{D}_1}(v_1, r)\}|$.

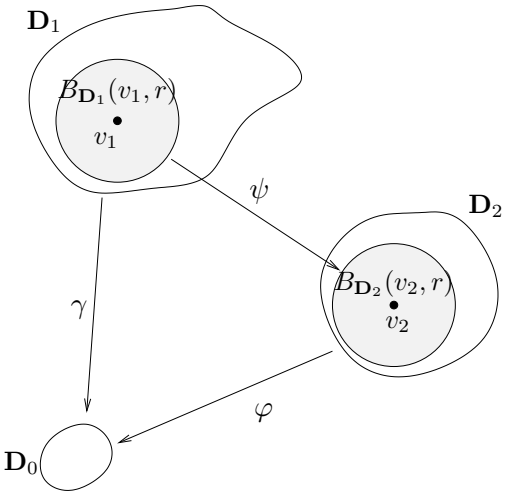



FIGURE 9.1 – Quasi-revêtements : le diagramme de la définition. La boule $B_{\mathbf{D}_1}(v_1, r)$ capture *"l'existence d'une partie relativement importante d'un graphe"* $\mathbf{D}_1$ *"qui ressemble à un autre graphe"* $\mathbf{D}_2$.

En utilisant la définition d'un quasi-revêtement, on dit qu'un quasi-revêtement est *strict* si $B_{\mathbf{D}_1}(v_1, r-1)$ n'est pas égale à $D_1$. On peut remarquer que tout quasi-revêtement qui n'est pas strict est un revêtement. On a :

**Lemme 9.6** *Soit* $\mathbf{D}_1$ *un quasi-revêtement strict de* $\mathbf{D}_0$ *de rayon* $r$ *via* $\gamma$*. Alors, pour tout* $q \in \mathbb{N}$*, si* $r \geq q|V(D_0)|$ *alors* $\gamma$ *a au moins* $q$ *feuillets.*

**Remarque 9.7** *Si un graphe orienté étiqueté* $\mathbf{D}_1$ *est un revêtement symétrique de* $\mathbf{D}_0$*, alors pour tout* $v \in V(D_1)$ *et pour tout* $r \in \mathbb{N}$*,* $\mathbf{D}_1$ *est un quasi-revêtement de* $\mathbf{D}_0$*, de centre* $v$ *et de rayon* $r$*. Il suffit de choisir* $\mathbf{D}_2 = \mathbf{D}_1$*. Dans ce cas, on dit que* $\mathbf{D}_1$ *est un quasi-revêtement de* $\mathbf{D}_0$ *de rayon infini. Inversement, si* $\mathbf{D}_1$ *est un quasi-revêtement de* $\mathbf{D}_0$ *de rayon* $r$ *strictement plus grand que le diamètre de* $\mathbf{D}_1$*, alors* $\mathbf{D}_1$ *est un revêtement de* $\mathbf{D}_0$.

Soit $\mathcal{A}$ un algorithme distribué. Le lemme suivant précise la diminution du rayon du quasi-revêtement après $k$ rounds d'une exécution synchrone de $\mathcal{A}$ :

**Lemme 9.8** *Soit* $\mathbf{D}_1$ *un graphe orienté étiqueté symétrique qui est un quasi-revêtement de* $\mathbf{D}_0$ *de centre* $v_1$ *et de rayon* $r$ *via* $\gamma$*. Soit* $k < r$ *un entier naturel. Pour tout algorithme* $\mathcal{A}$*, soit* $\mathbf{D}'_0$ *le graphe orienté obtenu après* $k$ *rounds d'une exécution synchrone de* $\mathcal{A}$ *sur* $\mathbf{D}_0$*. Alors* $\mathbf{D}'_1$ *obtenu*

*après une exécution synchrone de $\mathcal{A}$ sur $\mathbf{D}_1$ est un quasi-revêtement de $\mathbf{D}'_0$ de centre $v_1$ est de rayon $r-k$.*

**Preuve :** Soit $\mathcal{A}$ un algorithme et $\mathbf{D}_1 = (D_1, \lambda_1)$ un graphe orienté étiqueté symétrique qui est un quasi-revêtement de $\mathbf{D}_0 = (D_0, \lambda_0)$ de centre $v_1$ et de rayon $r$ via $\gamma$. Il existe un graphe orienté étiqueté symétrique $\mathbf{D}_2 = (D_2, \lambda_2)$ qui est un revêtement symétrique de $\mathbf{D}_0$ via un homomorphisme $\varphi$ et un sommet $v_2 \in V(D_2)$ tels que $(B_{D_1}(v,r), \lambda_1)$ est isomorphe à $(B_{D_2}(v,r), \lambda_2)$ via un isomorphisme $\psi$ et pour tout $v \in B(v_1, r)$, $\gamma(v) = \varphi(\psi(v))$.

Soit $\mathbf{D}'_0 = (D_0, \lambda'_0)$ (resp. $\mathbf{D}'_1 = (D_1, \lambda'_1), \mathbf{D}'_2 = (D_2, \lambda'_2)$) des graphes orientés étiquetés où pour chaque $v$, $\lambda'_0(v)$ (resp. $\lambda'_1(v)$, $\lambda'_2(v)$) est l'état de $v$ dans $D_0$ (resp. $D_1, D_2$) après un round $\mathcal{A}$ sur $\mathbf{D}_0$ (resp. sur $\mathbf{D}_1, \mathbf{D}_2$). Pour prouver le lemme, il suffit de montrer que $\mathbf{D}'_1$ est un quasi-revêtement de $\mathbf{D}'_0$ de centre $v_1$ et de rayon $r-1$ via $\gamma$.

D'après le lemme de relèvement du chapitre précédent, on sait que $\mathbf{D}'_2$ est un revêtement de $\mathbf{D}'_0$. De plus, pour chaque $v \in V(B_{D_1}(v_1, r-1))$, $\lambda'_1(v) = \lambda'_2(\varphi(v))$ puisque $(B_{D_1}(v,1), \lambda_1)$ est isomorphe à $(B_{D_2}(\varphi(v),1), \lambda_2)$.

Donc, $(B_{D_1}(v, r-1), \lambda'_1)$ est isomorphe à $(B_{D_2}(v, r-1), \lambda'_2)$ via $\psi$ et ainsi $\mathbf{D}'_1$ est un quasi-revêtement de $\mathbf{D}'_0$ de centre $v_1$ et de rayon $r-1$ via $\gamma$. □

On en déduit :

**Corollaire 9.9** *Soit $\mathbf{D}_1$ un graphe orienté étiqueté symétrique qui est un quasi-revêtement de $\mathbf{D}_0$ de centre $v_1$ et de rayon $r$ via $\gamma$. Pour tout algorithme $\mathcal{A}$, après $r$ rounds d'une exécution synchrone de $\mathcal{A}$ sur $\mathbf{D}_1$, $v_1$ est dans le même état que $\gamma(v_1)$ après $r$ rounds d'une exécution synchrone de $\mathcal{A}$ sur $\mathbf{D}_0$.*

### 9.2.2 Élection dans une famille de graphes étiquetés et quasi-revêtements

**Proposition 9.10 (Condition nécessaire)** *Soit $\mathcal{I}$ une famille récursive de graphes orientés étiquetés et premiers pour la relation être revêtement symétrique telle qu'il existe un algorithme d'élection pour cette famille. Sous ces conditions, il existe une fonction calculable $\tau : \mathcal{I} \to \mathbb{N}$ telle que pour pour tout graphe orienté étiqueté $\mathbf{D}$ de $\mathcal{I}$, il n'y a pas de quasi-revêtement de $\mathbf{D}$, distinct de $\mathbf{D}$, de rayon plus grand que $\tau(\mathbf{D})$ dans $\mathcal{I}$.*

**Preuve :** Soit $\mathcal{A}$ un algorithme d'élection pour $\mathcal{I}$. Pour un graphe orienté étiqueté $\mathbf{D} \in \mathcal{I}$, on pose $\tau(\mathbf{D}) = 2|V(\mathbf{D})| + n$ où $n$ est le nombre de rounds d'une exécution synchrone de $\mathcal{A}$ sur $\mathbf{D}$ donnant un graphe irréductible. La fonction $\tau$ a la propriété souhaitée.

On fait une preuve par contradiction. Soit $\mathbf{D} \in \mathcal{I}$. Soit $\mathbf{D}_i$ le graphe orienté étiqueté obtenu après le $i$ième round dans la définition de $\tau(\mathbf{D})$. Soit $\mathcal{C} = (\mathbf{D} = \mathbf{D}_0, \mathbf{D}_1, ..., \mathbf{D}_n)$ tel qu'aucune action de $\mathcal{A}$ ne puisse être exécutée sur un sommet quelconque de $\mathbf{D}_n$, et $\mathbf{D}_{i+1}$ est obtenu à partir de $\mathbf{D}_i$ par l'exécution d'un round de $\mathcal{A}$. Par hypothèse, l'étiquette *elu* apparaît exactement une fois sur $\mathbf{D}_n$.

Soit $\mathbf{D}' \in \mathcal{I}$ un quasi-revêtement de $\mathbf{D}$ de rayon $\tau(\mathbf{D})$, distinct de $D$. En itérant le lemme 9.8, on obtient $\mathbf{D}$" tel que $\mathbf{D}$" est un quasi-revêtement de $\mathbf{D}_n$ de rayon $\tau(D) - n = 2|V(D)|$. Le graphe orienté étiqueté $\mathbf{D}$ étant premier pour la relation être revêtement symétrique et distinct de $\mathbf{D}'$, le quasi-revêtement $\mathbf{D}$" de $\mathbf{D}_n$ est strict. Donc, d'après le lemme 9.6, l'étiquette *elu* apparaît au moins deux fois dans $\mathbf{D}$". D'où une contradiction. □

## 9.3 Un algorithme d'élection pour une famille de graphes étiquetés

Cette section présente un algorithme d'élection pour une famille de graphes permettant de montrer que la condition nécessaire de la section précédente est suffisante ; ce qui prouve le théorème initial.

Cet algorithme combine l'algorithme d'élection $\mathcal{M}$ pour les graphes orientés symétriques premiers et une généralisation de l'algorithme de Szymanski, Shy et Prywes (SSP dans la suite).

L'algorithme d'énumération $\mathcal{M}$ termine toujours sur tout réseau $(Dir(\mathbf{G}), \delta)$ et pendant son déroulement chaque sommet $v$ peut reconstruire à certains pas de calcul $i$ un graphe orienté étiqueté symétrique $\mathbf{D}_i(v)$ tel que $(Dir(\mathbf{G}), \delta)$ est un quasi-revêtement de $\mathbf{D}_i(v)$. Cependant, cet algorithme ne permet pas à $v$ de calculer le rayon de ce quasi-revêtement. Pour ce faire, on utilise une généralisation de l'algorithme SSP pour que chaque sommet puisse calculer une borne inférieure du rayon de ces quasi-revêtements.

### 9.3.1 Le méchanisme pour la détection de la terminaison

Un sommet détectera si son état est final en utilisant la condition suffisante de la proposition 9.10, donc on ajoute à l'étiquette de chaque sommet deux champs :

— $a(v) \in \mathbb{Z}$ est un compteur, initialement $a(v) = -1$. Ce compteur $a(v)$ représente la distance jusqu'où tous les sommets ont la même boîte aux lettres que $v$ ;

— $A(v) \in \mathcal{P}_{\text{fin}}(\mathbb{N} \times \mathbb{N})$ code l'information que $v$ a sur les valeurs de $a(u)$ pour chaque voisin $u$. Initialement, $A(v) = \{(q, -1) \mid q \in [1, \deg_G(v)]\}$.

Ainsi, maintenant pendant l'exécution du nouvel algorithme l'étiquette de chaque sommet $v$ est un tuple $(\lambda(v), n(v), N(v), M(v), a(v), A(v))$.

Un message envoyé par un sommet $u$ via le port $p$ au sommet $v$ a la forme suivante $< (n, \ell, M, a), p >$ où $n$ est le numéro courant $n(u)$ de $u$, $\ell$ est l'étiquette $\lambda(u)$ de $u$, $M$ est la boîte aux lettres de $u$, $a$ est la valeur du compteur $a(u)$ et $p = \delta_u(v)$.

La description de l'algorithme utilise le prédicat suivant, noté pour un sommet $v$ $\mathbf{QC}(v)$, et défini par :

$$\mathbf{QC}(v) = (S(M(v)) \text{ est cohérent}) \text{ et } (a(v) \neq a_{old}(v)) \text{ et } (\mathbf{D}_{M(v)} \in \mathcal{I}) \text{ et } (a(v) \leq \tau(\mathbf{D}_{M(v)})).$$

### 9.3.2 L'algorithme $\mathcal{M}_{Family}$

L'algorithme $\mathcal{M}_{Family}$ est décrit ci-dessous (algorithm 10). La première règle **I** peut être appliquée par un sommet $v$ à son réveil seulement s'il n'a pas reçu de message : il prend le numéro 1, met à jour sa boîte aux lettres et en informe ses voisins. La deuxième règle **R** décrit les instruction exécutées par un sommet $v$ qui a reçu un message $m$ d'un voisin.

Il met à jour sa boîte aux lettres et sa vue locale en fonction de $m$. Puis, s'il découvre l'existence d'un autre sommet avec le même numéro et une vue locale plus forte alors il prend un nouveau numéro. Si sa boîte aux lettres n'a pas changé, il met à jour $A(v)$ et $a(v)$. Finalement, si $M(v)$ ou $a(v)$ ont été modifiés, il en informe ses voisins.

En utilisant les informations de sa boîte aux lettres chaque sommet pourra construire, sous certaines conditions, un graphe orienté étiqueté $\mathbf{D}$ tel que $(Dir(\mathbf{G}), \delta)$ est identique à $\mathbf{D}$ jusqu'à distance $a(v)$.

**Algorithme 10:** Algorithme $\mathcal{M}_{Family}$.

**I** : $\{n(v_0) = 0$ et aucun message n'est parvenu à $v_0\}$
**begin**
  $n(v_0) := 1$ ;
  $M(v_0) := \{(n(v_0), \lambda(v_0), \emptyset)\}$ ;
  $a(v_0) := -1$ ;
  **for** $i := 1$ ***to*** $\deg(v_0)$ **do**
    **send** $< (n(v_0), \lambda(v_0), M(v_0), a(v_0)), i >$ through port $i$ ;
**end**

**R** : {Un message $< (n_1, \ell_1, M_1, a_1), p_1 >$ est parvenu à $v_0$ à travers le port $q_1\}$
**begin**
  $M_{old} := M(v_0)$ ;
  $a_{old} := a(v_0)$ ;
  $M(v_0) := M(v_0) \cup M_1$ ;
  **if** $n(v_0) = 0$ *ou* $\exists (n(v_0), \ell', N') \in M(v_0)$ *tel que* $(\lambda(v_0), N(v_0)) \prec (\ell', N')$ **then**
    $n(v_0) := 1 + \max\{n' \mid \exists (n', \ell', N') \in M(v_0)\}$ ;
  $N(v_0) := N(v_0) \setminus \{(n', \ell', p', q_1) \mid \exists (n', \ell', p', q_1) \in N(v_0)\} \cup \{(n_1, \ell_1, p_1, q_1)\}$ ;
  $M(v_0) := M(v_0) \cup \{(n(v_0), \lambda(v_0), N(v_0))\}$ ;
  **if** $M(v_0) \neq M_{old}$ **then**
    $a(v_0) := -1$ ;
    $A(v_0) := \{(q', -1) \mid 1 \leq q' \leq deg(v_0)\}$;
  **if** $M(v_0) = M_1$ **then**
    $A(v_0) := A(v_0) \setminus \{(q_1, a') \mid \exists (q_1, a') \in A(v_0)\} \cup \{(q_1, a_1)\}$ ;
  **if** $(\forall (q', a') \in A(v_0), a(v_0) \leq a'$ *et* $\mathbf{QC}(v_0))$ **then** $a(v_0) := a(v_0) + 1$ ;
  **if** $M(v_0) \neq M_{old}$ *ou* $a(v_0) \neq a_{old}$ **then**
    **for** $i := 1$ ***to*** $\deg(v_0)$ **do**
      **send** $< (n(v_0), \lambda(v_0), M(v_0), a(v_0)), i >$ via $i$ ;
**end**

**Propriétés de l'algorithme $\mathcal{M}_{Family}$.**

On considère un graphe $\mathbf{G}$ avec une numérotation des ports $\delta$ et une exécution $\rho$ de l'algorithme $\mathcal{M}_{Family}$ sur $(\mathbf{G}, \delta)$.

Pour chaque sommet $v \in V(G)$, on note $(\lambda_i(v), n_i(v), N_i(v), M_i(v),\ a_i(v), A_i(v))$ l'état de $v$ après le $i$ième pas de calcul de $\rho$.

Un corollaire intéressant de la proposition 9.4 est : il existe un pas $i_0$ tel que après ce pas de calcul pour tout sommet $v$, les valeurs de $\lambda(v), n(v), N(v)$ et de $M(v)$ ne sont plus modifiées.

**Proposition 9.11** *Soit un sommet $v$ et un pas $i$.*

*Si $M_i(v) = M_{i+1}(v)$ et si $v$ est actif au pas $i+1$, alors $a_i(v) \leq a_{i+1}(v) \leq a_i(v) + 1$ et $a_{i+1}(v) \geq \min\{a \mid \exists (q,a) \in A_{i+1}(v)\}$ si $\exists (q,a) \in A_{i+1}(v)$.*

*Si $a_{i+1}(v) \geq 1$, pour chaque $w \in N_G(v)$, il existe un pas $j \leq i$ tel que $a_j(w) \geq a_{i+1}(v) - 1$ et $M_j(w) = M_{i+1}(v)$.*

**Preuve :** On fait une preuve par induction sur $i$. Soit un sommet $v$ qui modifie son état au pas $i+1$.

Si $v$ a appliqué la règle **I** alors la propriété est clairement vérifiée.

Supposons maintenant que $v$ a appliqué la règle **R** après la réception du message $m_1 =< (n_1, l_1, M_1, a_1), p_1 >$ via le port $q_1$. Nécessairement, on a : $M_i(v) \subseteq M_{i+1}(v)$.

Si $M_i(v) = M_{i+1}(v)$ et $a_i(v) \neq a_{i+1}(v)$ alors $a_{i+1}(v) = a_i(v) + 1$.

Soit $min_A = min\{a | \exists (q,a) \in A_{i+1}(v)\}$.

Si $min_A \neq a_1$, alors d'après l'hypothèse d'induction $a_{i+1} \geq min_A$. Si $M_{i+1}(v) \neq M_i(v)$ alors $a_{i+1}(v) \geq -1 \geq min_A$. Supposons que $M_{i+1}(v) = M_i(v)$. Si $(q_1, a_1) \in A_i(v)$ alors $A_{i+1}(v) = A_i(v)$ et par induction $a_{i+1}(v) \geq min_A$. Si $M_1 \neq M_{i+1}(v)$ alors $A_{i+1}(v) = A_i(v)$ et $a_{i+1}(v) \geq min_A$.

Supposons maintenant que $M_1 = M_{i+1}(v)$. Si $a_1 = 0$ alors $v$ peut augmenter $a(v)$ s'il est égal à $-1$. Donc $a_{i+1}(v) \geq 0$ et par conséquent $a_{i+1}(v) \geq min_A$.

Sinon, on peut supposer que $min_A = a_1 > 0$, $M_{i+1}(v) = M_i(v) = M_1$ et $(q_1, a_1) \notin A_i(v)$. Soit $m_2 =< (n_2, \ell_2, M_2, a_2), p_1 >$ le message précédent reçu via le port $q_1$. Comme les canaux de communications sont FIFO, $m_2$ a été envoyé avant $m_1$. Comme $a_1 > 0$, $M_2 = M_1$ et donc par induction, $a_2 = a_1 - 1$. Soit $j \leq i$ le pas où $v$ reçoit $m_2$. Comme $M_2 \subseteq M_j(v) \subseteq M_i(v) = M_1 = M_2$, $M_j(v) = M_2$. Par conséquent, $(q, a_2) \in A_i(v)$. Sachant que $a_1 = min_A$, $a_2 = a_1 - 1 = min\{a | \exists (q,a) \in A_i(v)\}$. Si $a_i(v) \geq a_1$ alors $a_{i+1}(v) \geq a_i(v) \geq min_A$. Sinon, par induction, $a_i(v) = a_2 = a_1 - 1$, et, comme $a_i(v) < a_1$ et $a_i(v) \leq min_A$, $v$ incrémente $a_i(v)$. Donc $a_{i+1}(v) = 1 + a_i(v) = a_1 \geq min_A$.

Supposons quet $a_{i+1}(v) \geq 1$ et considérons un sommet $w \in N_G(v)$. Il existe $(\delta_v(w), a) \in A_{i+1}(v)$ avec $a \geq a_{i+1}(v) - 1 \geq 0$ et $v$ reçoit un message $m =< (n, \ell, M, a), \delta_w(v) >$ de $w$ à un pas $i' \leq i+1$. Soit $j < i+1$ le pas où $w$ envoie son message. Comme $a \geq 0$, $M_{i'}(v) = M_{i+1}(v) = M_j(v)$ et $a_j(w) = a \geq a_{i+1}(v) - 1$.

□

La proposition suivante met en évidence des similarités entre $\mathbf{D}_{M(v)}$ et $(Dir(\mathbf{G}), \delta)$.

**Proposition 9.12** *Si $S(M(v))$ est cohérent, alors $(Dir(\mathbf{G}), \delta)$ est un quasi-revêtement de $\mathbf{D}_{M(v)}$ de rayon $a(v)$ et de centre $v$.*

On montre tout d'abord une autre proposition qui permet de présenter une définition équivalente des quasi-revêtements qu'on utilisera dans la preuve de la proposition proposition 9.12.

On va utiliser la définition d'une vue que l'on donne maintenant.

**Définition 9.13** *Soit un graphe orienté symétrique $\mathbf{D} = (D, \lambda) \in \mathcal{D}_L$ et un sommet $v \in V(D)$. La* vue *de $v$ dans $\mathbf{D}$ est l'arbre enraciné infini noté $\mathbf{T_D}(v) = (T_D(v), \lambda')$ et défini par :*

- *$V(T_D(v))$ est l'ensemble de chemins qui ne rebroussent pas sur eux-mêmes $\pi = a_1, \ldots, a_p$ dans $\mathbf{D}$ avec $s(a_1) = v$. Pour chaque chemin $\pi = a_1, \ldots, a_p$, $\lambda'(\pi) = \lambda(t(a_p))$.*
- *pour chaque $\pi, \pi' \in V(T_D(v))$, il y a 2 arcs $a_{\pi,\pi'}, a_{\pi',\pi} \in A(T_D(v))$ tels que $Sym(a_{\pi,\pi'}) = a_{\pi',\pi}$ si et seulement si $\pi' = \pi, a$. dans ce cas, $\lambda'(a_{\pi,\pi'}) = \lambda(a)$ et $\lambda'(a_{\pi',\pi}) = \lambda(Sym(a))$.*
- *la racine de $T_D(v)$ est le sommet correspondant au chemin vide et son étiquette est $\lambda(v)$.*

**Remarque 9.14** *Pour tous sommets $u$ et $v$ : $T_D(u)$ est isomorphe à $T_D(v)$. On note $T_D$ ce graphe défini à un isomorphisme près. C'est le revêtement universel de $D$. Il est très utile pour construire des quasi-revêtements.*

Soit $\mathbf{T_D}(v)$ la vue d'un sommet $v$ dans un graphe orienté $\mathbf{D} \in \mathcal{D}_L$ et un arc $a$ tel que $s(a) = v$. On définit $\mathbf{T_{D-}}_a(v)$ comme étant l'arbre infini obtenu à partir de $\mathbf{T_D}(v)$ en retirant le sous-arbre enraciné au sommet correspondant au chemin $a$.

**Proposition 9.15** *Étant donnés deux graphes orientés étiquetés symétriques $\mathbf{D}_0, \mathbf{D}_1$, un entier $r$, un sommet $v_1 \in V(D_1)$ et un homomorphisme $\gamma$ de $\mathbf{B}_{\mathbf{D}_1}(v_1, r)$ sur $\mathbf{D}_0$, $\mathbf{D}_1$ est un* quasi-revêtement *de $\mathbf{D}_0$ de centre $v_1$ et de rayon $r$ via $\gamma$ si et seulement si on a :*

*(i) pour chaque arc $a \in A(B_{D_1}(v_1, r))$, $\gamma(Sym(a)) = Sym(\gamma(a))$,*

*(ii) pour tout $v \in \mathbf{B}_{\mathbf{D}_1}(v_1, r)$, $\gamma$ induit une injection entre les arcs entrants (resp.sortants) de $v$ et les arcs entrants (resp. sortants) de $\gamma(v)$,*

*(iii) pour tout $v \in \mathbf{B}_{\mathbf{D}_1}(v_1, r-1)$, $\gamma$ induit une surjection entre les arcs entrants (resp.sortants) de $v$ et les arcs entrants (resp. sortants) de $\gamma(v)$.*

**Preuve :** Si $\mathbf{D}_1$ est un quasi-revêtement de $\mathbf{D}_0$ de centre $v_1$ et de rayon $r$ via $\gamma$, alors on vérifie que $\gamma$ satisfait ces propriétés.

Inversement, on construit un revêtement infini $\mathbf{D}_2 = (D_2, \lambda_2)$ de $\mathbf{D}_0 = (D_0, \lambda_0)$ de la façon suivante. Tout d'abord on prend une copie $\mathbf{B}_1$ de $\mathbf{B}_{\mathbf{D}_1}(v_1, r)$ et on note $\delta$ l'isomorphisme de $\mathbf{B}_{\mathbf{D}_1}(v_1, r)$ sur $\mathbf{B}_1$. Ensuite, on considère un sommet $v$ tel que $\mathrm{dist}_{B_1}(v_1, v) = r$ et un arc $a_0 \in A(D_0)$ tel que $v \in \gamma^{-1}(t(a_0))$. S'il n'y a pas d'arc $a \in A(B_1)$ tel que $t(a) = v$ et $\gamma(a) = a_0$, alors on ajoute une copie de $\mathbf{T_{D}}_{0-a_0}(s(a_0))$ à $\mathbf{D}_2$ et on note $v_2(a_0)$ la racine de cet arbre. On ajoute deux arcs $a_2, a_2'$ tels que $t(a_2) = s(a_2') = v$, $s(a_2) = t(a_2') = v_2(a_0)$, $\lambda_2(a_1) = \lambda_0(a_0)$, $\lambda_2(a_2) = \lambda_0(Sym(a_0))$ et $Sym(a_2) = a_2'$.

Le graphe orienté étiqueté $\mathbf{D}_2$ est le graphe orienté une fois que ces constructions ont été effectuées pour chaque sommet $v \in V(B_1)$ tel que $\mathrm{dist}_{B_1}(v, v_1) = r$. On vérifie aisément que $\mathbf{D}_2$ est un revêtement symétrique de $\mathbf{D}_0$ via un homomorphisme $\varphi$ et que pour tout $v \in V(B_{\mathbf{D}_1}(v, r))$, $\gamma(v) = \varphi(\delta(v))$. □

On utilise la proposition 9.15 pour prouver la proposition 9.12.

**Preuve :** [de la proposition 9.12] Soit un pas de calcul $i$ et un processus $v$ tel que $S(M_i(v))$ est cohérent. Si $a_i(v) = 0$ alors on obtient le résultat. Supposons que $a_i(v) \geq 1$. D'après la proposition 9.11, pour chaque $w \in V(Dir(G))$ tel que $\mathrm{dist}_{Dir(G)}(v, w) \leq a_i(v)$, il existe un pas $j_w \leq i$ tel que $a_{j_w}(w) \geq a_i(v) - \mathrm{dist}_{Dir(G)}(v, w)$ et $M_{j_w}(w) = M_i(v)$.

Donc pour chaque $w \in V(B_{Dir(\mathbf{G})}(v, a_i(v)))$, $j_w$ est défini et $a_{j_w}(w) \geq 0$ et pour chaque $w \in V(B_{Dir(\mathbf{G})}(v, a_i(v)-1))$, $a_{j_w}(w) \geq 1$. Pour chaque $w \in V(B_{Dir(\mathbf{G})}(v, a_i(v)))$, $(n_{j_w}(w), \lambda(v), N_{j_w}(w)) \in S(M_{j_w}(w)) = S(M_i(v))$ et donc $\deg_{G'}(w) = \deg_{D_{M_i(v)}}(n_{j_w}(w))$. Donc on peut définir $\gamma(w) = n_{j_w}(w) \in V(D_{M_i(v)})$ et on a $\lambda_{M_i(v)}(\gamma(w)) = \lambda(w)$.

Pour chaque arc $a \in A(B_{Dir(\mathbf{G})}(v, a_i(v)))$, soit $w = t(a), w' = s(a)$ (resp. $w = s(a), w' = t(a)$) et supposons, sans perdre de généralité, que $\text{dist}_G(v, w) \leq a_i(v) - 1$. Soit $m =< (n, \ell, M, a), p >$ le dernier message reçu via le port $\delta_w(w')$ avant le pas $j_w$. Comme $a_{j_w}(w) \geq 1$, $M = M_{j_w}(w) = M_{j_{w'}}(w')$, $n = n_{j_{w'}}(w')$, $p = \delta_{w'}(w)$ et $a \geq 0$. Donc on peut définir $\gamma(a) = a_{n,n',p,q}$ (resp. $\gamma(a) = a_{n',n,q,p}$) où $n = n_{j_w}(w)$, $n' = n_{j_{w'}}(w')$, $p = \delta_{w'}(w)$ et $q = \delta_w(w')$. On vérifie que $\lambda_{M_i(v)}(\gamma(a)) = \lambda(a) = (p, q)$ (resp. $\lambda_{M_i(v)}(\gamma(a)) = \lambda(a) = (q, p)$) et que $Sym(\gamma(a)) = \gamma(Sym(a))$.

Par conséquent $\gamma$ est un homomorphisme de $\mathbf{B}_{Dir(\mathbf{G})}(v, a_i(v))$ sur $\mathbf{D}_{M_i(v)}$.

Pour chaque $w \in V(B_{Dir(\mathbf{G})}(v, a_i(v)))$, pour tous arcs $a, a'$ tels que $s(a) = s(a') = w$ (resp. $t(a) = t(a') = w$), $\lambda(a) \neq \lambda(a')$ et donc $\gamma(a) \neq \gamma(a')$. Pour chaque $w \in V(B_{Dir(\mathbf{G})}(v, a_i(v) - 1))$, comme $\deg_{G'}(w) = |N_{j_w}(w)| = \deg_{D_{M_i(v)}}(n_{j_w}(w))$, $\gamma$ induit une bijection entre les arcs entrants (resp. sortants) de $w$ dans $Dir(\mathbf{G})$ et les arcs entrants (resp. sortants) de $n_{j_w}(w)$ dans $\mathbf{D}_{M_i(v)}$.

D'après la proposition 9.15, $Dir(\mathbf{G})$ est un quasi-revêtement de $\mathbf{D}_{M_i(v)}$ de centre $v$ et de rayon $a_i(v)$ via $\gamma$.

□

L'algorithme $\mathcal{M}_{Family}$ est la combinaison de l'algorithm $\mathcal{M}$ et une généralisation de l'algorithme de Szymanski, Shy et Prywes. l'algorithme d'énumération termine toujours sur n'importe quel réseau $(Dir(\mathbf{G}), \delta)$. Pendant son exécution, chaque processus $v$ peut reconstruire à certains pas de calcul $i$ un graphe orienté étiqueté symétrique $\mathbf{D}_i(v)$ tel que $(Dir(\mathbf{G}), \delta)$ est un quasi-revêtement de $\mathbf{D}_i(v)$.

On rappelle que le calcul de $a(v)$ et sa pogression sont conditionnés par l'appartenance de $\mathbf{D}_{M(v)}$ à $\mathcal{I}$. Lorsque l'algorithme d'énumération est terminé chaque sommet $v$, après le pas $t_v$, il existe un pas $t'_v$ tel que $a(v) \geq \tau(\mathbf{D}_{M(v)})$. Cette connaissance et la connaissance du graphe reconstruit $\mathbf{D}_{M(v)}$ implique que chaque sommet sait que l'algorithme d'énumération est terminé et peut décider si son numéro est ou non le plus grand parmi les numéros attribués. Finallement chaque sommet peut décider s'il est élu ou non. Ce qui achève la preuve du théorème énoncé dans l'introduction.

On peut noter que ce résultat peut être appliqué à d'autres modèles : celui de Mazurkiewicz, d'Angluin, plus généralement aux réécritures d'arêtes étiquetées.

### 9.3.3 Quelques applications

Comme premier exemple d'application on en déduit que la connaissance d'une borne de la taille ou du diamètre d'un graphe minimal suffit pour élire.

Dans le modèle de Mazurkiewicz ou d'Angluin, il existe un algorithme d'élection pour la famille des arbres ou la famille des graphes complets. Mais il n'existe pas d'algorithme d'élection pour une famille contenant les arbres et un graphe qui n'est pas un arbre.

S'il existe un algorithme d'élection pour un anneau de taille première il n'existe pas d'algorithme pour la famille des anneaux premiers.

# Chapitre 10

# Élection et Calculs Locaux sur des Étoiles Fermées

## 10.1 Introduction

Dans ce chapitre, on étudie les *calculs locaux sur les étoiles fermées* qui correspondent aux relations de réétiquetage localement engendrées sur les étoiles fermées. Dans ce modèle, les graphes considérés sont des graphes simples. En un pas de calcul, un sommet peut observer l'état de ses voisins, l'état des arêtes qui lui sont incidentes et modifier son état ainsi que l'état de ses voisins et des arêtes qui lui sont incidentes.

On présente une caractérisation des graphes admettant un algorithme d'élection et de nommage obtenue par Mazurkiewicz (dans ce modèle, élection et nommage peuvent être résolus sur les mêmes graphes). Cette caractérisation est basée sur la notion de *revêtements simples* (qui correspondent aux homomorphismes de graphes simples localement bijectifs). (Ce chapitre est extrait de [God02, Cha06].)

## 10.2 Revêtements Simples

Dans ce chapitre, les homomorphismes localement bijectifs de graphe simples, i.e., les revêtements simples, sont les homomorphismes qui permettent de donner des conditions nécessaires que doivent vérifier les graphes admettant un algorithme de nommage ou d'élection utilisant des calculs locaux sur les étoiles fermées.

**Définition 10.1** *Un graphe simple $G$ est un revêtement d'un graphe simple $H$ à travers un homomorphisme $\gamma\colon G \to H$ si pour tout sommet $v \in V(G)$, $\gamma$ induit une bijection entre $N_G(u)$ et $N_H(\varphi(u))$, i.e., si les conditions suivantes sont vérifiées :*

— $|N_G(u)| = |N_H(\gamma(u))|$,

— $\gamma(N_G(u)) = N_H(\gamma(u))$.

*On dit alors que l'homomorphisme $\gamma$ est* localement bijectif

*Un graphe $G$ est un* revêtement simple propre *de $H$ si $\gamma$ n'est pas un isomorphisme et $G$ est* minimal pour les revêtements simples *si $G$ n'est pas revêtement propre d'un graphe.*

*Naturellement, un graphe simple étiqueté $(G, \lambda)$ est un revêtement simple d'un graphe simple étiqueté $(H, \eta)$ à travers $\gamma$ si $G$ est un revêtement simple de $H$ à travers $\gamma$ et si $\gamma$ conserve l'étiquetage.*

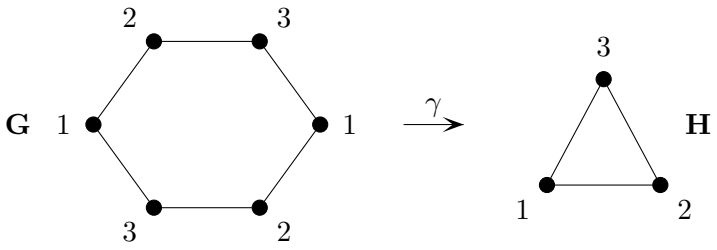



FIGURE 10.1 – Le graphe **G** est un revêtement simple de **H** à travers l'homomorphisme $\gamma$ qui envoie chaque sommet de **G** étiqueté $i$ sur l'unique sommet de **H** dont l'étiquette est $i$.

**Exemple 10.2** *Le graphe* **G** *de la figure 10.1 est un revêtement simple de* **H** *à travers l'homomorphisme* $\gamma$.

*Le graphe* **G** *est un revêtement propre de* **H** *et n'est donc pas minimal pour les revêtements simples ; le graphe* **H** *est minimal pour les revêtements simples.*

Dans le lemme suivant, on montre que si **G** est un revêtement simple de **H**, pour tout sommet $v$ de $V(H)$, l'image inverse de l'étoile de **H** centrée en $v$ est une union disjointe d'étoiles de **G**.

**Lemme 10.3** *On considère des graphes simples* $\mathbf{G}, \mathbf{H}$ *tels que* **G** *est un revêtement simple de* **H** *à travers un homomorphisme* $\gamma$. *Soit* $v$ *un sommet de* $V(H)$ *et soient* $u_1, u_2$ *deux sommets distincts de* $\gamma^{-1}(v)$. *Pour tous sommets* $u'_1 \in N_G(u_1) \cup \{u_1\}$ *et* $u'_2 \in N_G(u_2) \cup \{u_2\}$, $u'_1 \neq u'_2$.

**Preuve :** On considère des graphes simples $\mathbf{G}, \mathbf{H}$ tels que **G** est un revêtement simple de **H** à travers un homomorphisme $\gamma$. Soit $v$ un sommet de $V(H)$ et soient $u_1, u_2$ deux sommets de $\gamma^{-1}(v)$. Puisque **H** est un graphe simple, on sait que $u_2 \notin N_G(u_1)$. S'il existe $u \in N_G(u_1) \cap N_G(u_2)$, alors $\gamma$ n'est pas localement bijectif en $u$ puisque $u$ a deux voisins qui ont la même image par $\gamma$. □

La relation ≪être revêtement simple≫ est une relation transitive comme le montre la proposition suivante.

**Proposition 10.4** *Pour tous graphes simples* $\mathbf{G}, \mathbf{H}, \mathbf{K}$, *si* **G** *est un revêtement simple de* **H** *a travers un homomorphisme* $\gamma$ *et si* **H** *est un revêtement simple de* **K** *à travers un homomorphisme* $\varphi$, *alors* **G** *est un revêtement simple de* **K** *à travers l'homomorphisme* $\varphi \circ \gamma$.

**Preuve :** Il est clair que $\varphi \circ \gamma$ est un homomorphisme de **G** dans **K**. De plus, pour tout sommet $u \in V(G)$, pour tout voisin $w' \in N_K(\varphi(\gamma(u)))$, il existe un unique sommet $v' \in N_H(\gamma(u))$ telle que $\varphi(v') = w'$ et par conséquent, il existe un unique sommet $u' \in N_G(u)$ telle que $\varphi(\gamma(u')) = w'$. Ainsi l'homomorphisme $\varphi \circ \gamma$ est localement bijectif et **G** est un revêtement simple de **K** à travers $\varphi \circ \gamma$. □

Puisqu'on ne considère que des graphes simples connexes, tout homomorphisme localement bijectif est surjectif.

**Proposition 10.5** *Si un graphe simple* **G** *est un revêtement simple d'un graphe simple connexe* **H** *à travers* $\gamma$, *alors* $\gamma$ *est surjectif.*

**Preuve :** Soit **G** un revêtement simple d'un graphe simple connexe **H** à travers un homomorphisme $\gamma$.

On considère un sommet $v \in \gamma(V(G))$. Il existe donc $u \in V(G)$ tel que $\gamma(u) = v$. Puisque $\gamma$ induit une bijection entre $N_G(u)$ et $N_H(v)$, pour tout sommet $v' \in N_H(v)$, il existe un sommet $u' \in N_G(u)$ telle que $\gamma(u') = v'$ et donc $v' \in \gamma(V(G))$. Ainsi, puisque **H** est connexe, on sait que $\gamma$ est un homomorphisme surjectif de **G** dans **H**. □

Si un graphe simple **G** est un revêtement simple d'un graphe simple **H** connexe, alors tous les sommets de **H** ont le même nombre d'antécédents, qui est appelé le *nombre de feuillets* du revêtement.

**Proposition 10.6** *Si un graphe simple* **G** *est un revêtement simple d'un graphe simple connexe* **H** *à travers* $\gamma$*, alors il existe une constante* $q$ *telle que pour tout* $v \in V(H)$, $|\gamma^{-1}(v)| = q$.

*Cette constante* $q$ *est appelée le* nombre de feuillets *du revêtement.*

**Preuve :** On considère un graphe simple **G** qui est un revêtement simple d'un graphe simple connexe **H** à travers un homomorphisme $\gamma$. On considère un sommet $v \in V(H)$ et un sommet $v' \in N_H(v)$.

Puisque $\gamma$ est un homomorphisme localement bijectif, pour tout sommet $u \in \gamma^{-1}(v)$, il existe un unique sommet $u' \in N_G(u)$ tel que $\gamma(u') = v'$. De plus, puisque $\gamma$ est localement bijectif en $u'$, pour tout sommet $u'' \in N_G(u')$ différent de $u$, $\gamma(u'') \neq \gamma(u)$. Par conséquent, $|\gamma^{-1}(v)| \leq |\gamma^{-1}(v')|$ et par symétrie, $|\gamma^{-1}(v)| = |\gamma^{-1}(v)|$. Ainsi, puisque le graphe **H** est connexe, pour tout $v, v' \in V(H)$, $|\gamma^{-1}(v)| = |\gamma^{-1}(v')|$. □

**Exemple 10.7** *On présente ici quelques exemples de graphes connexes simples qui sont minimaux pour les revêtements simples.*

- *les graphes dont le nombre d'arêtes et le nombre de sommets sont premiers entre eux,*
- *les arbres,*
- *les anneaux de taille première.*

Reidemeister a décrit dans une méthode pour construire tous les revêtements d'un graphe simple $H$ donné. Étant donné un arbre couvrant $T$ de $H$, Reidemeister a montré que tout revêtement de $H$ à $q$ feuillets peut être obtenu en prenant $q$ copies de $T$ et en reliant les copies de $T$ entre elles en fonction de permutations de $[1, q]$ associées à chaque arête de $H$ qui n'est pas dans $T$.

**Théorème 10.8** *Soit* $\mathbf{H} = (H, \eta)$ *un graphe simple et* **T** *un arbre couvrant de* **H**. *Un graphe simple* **G** *est un revêtement simple de* **H** *si et seulement s'il existe un entier* $q$ *et un ensemble* $\Sigma = \{\sigma_{(u,v)} \mid \{u, v\} \in E(H) \setminus E(T)\}$ *de permutations de* $[1, q]$ *(avec la propriété* $\sigma_{(u,v)} = \sigma^{-1}_{(v,u)}$*) tels que* **G** *est isomorphe au graphe* $\mathbf{H}_{T,\Sigma} = (H_{T,\Sigma}, \lambda)$ *défini de la manière suivante.*

$$\begin{aligned} V(H_{T,\Sigma}) &= \{(u,i) \mid v \in V(H) \text{ et } i \in [1,q]\}, \\ E(H_{T,\Sigma}) &= \{\{(u,i),(v,i)\} \mid \{u,v\} \in E(T)\} \cup \\ &\quad \{\{(u,i),(v,\sigma_{(u,v)}(i))\} \mid \{u,v\} \in E(H) \setminus E(T) \text{ et } i \in [1,q]\}. \end{aligned}$$

*De plus, pour tout* $(u,i) \in V(H_{T,\Sigma})$, $\lambda((u,i)) = \eta(u)$ *et pour tout* $\{(u,i),(v,j))\} \in E(H_{T,\Sigma})$, $\lambda(\{(u,i),(v,j)\}) = \eta(\{u,v\})$.

Un exemple de construction de revêtement en utilisant le théorème de Reidemeister apparaît sur la figure 10.2.

**Remarque 10.9** *La construction de Reidemeister permet d'obtenir tous les revêtements à* $q$ *feuillets d'un graphe* **H**. *Cependant, certains des graphes obtenus ne sont pas connexes et on peut obtenir plusieurs fois certains graphes, puisque certaines permutations produisent des graphes isomorphes.*

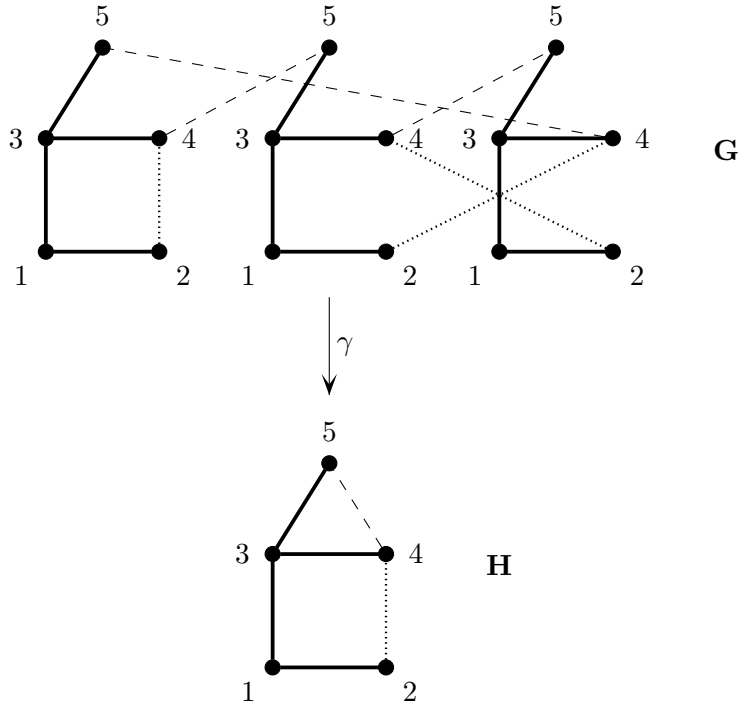



FIGURE 10.2 – Le graphe **G** est obtenu par la construction de Reidemeister d'un revêtement à 3 feuillets de **H**. Les arêtes de l'arbre couvrant de **H** sont représentés en gras et les permutations associées aux arêtes $\{2,4\}$ et $\{4,5\}$ de **H** sont respectivement $\sigma_{(2,4)} = (1)(23)$ et $\sigma_{(4,5)} = (123)$.

## 10.3 Calculs Locaux sur les Étoiles Fermées

Dans cette partie, on montre que les calculs locaux sur les étoiles fermées correspondent aux relations de réétiquetage localement engendrées sur les étoiles, puis on étudie leurs relations avec les revêtements simples.

### 10.3.1 Définitions

On rappelle qu'informellement, les calculs locaux sur les étoiles fermées permettent en un pas de calcul à un sommet de modifier son étiquette, les étiquettes de ses voisins et et les étiquettes des arêtes qui lui sont incidentes. L'application d'un tel pas de calcul ne dépend que de son étiquette, des étiquettes de ses voisins et des étiquettes qui lui sont incidentes.

On rappelle la définition des relations de réétiquetage localement engendrée sur les étoiles.

**Définition 10.10** *Une relation de réétiquetage $\mathcal{R}$ est localement engendrée sur les étoiles si la condition suivante est satisfaite. Pour tous graphes $(G,\lambda)$, $(G,\lambda')$, $(H,\eta)$, $(H,\eta')$, pour tous sommets $v \in V(G)$ et $w \in V(H)$ tels qu'il existe un isomorphisme $\varphi : B_G(v) \to B_G(w)$, si les conditions suivantes sont vérifiées :*

1. *pour tout $x \in V(B_G(v)) \cup E(B_G(v))$, $\lambda(x) = \eta(\varphi(x))$ et $\lambda'(x) = \eta'(\varphi(x))$,*
2. *pour tout $x \notin V(B_G(v)) \cup E(B_G(v))$, $\lambda'(x) = \lambda(x)$,*
3. *pour tout $x \notin V(B_H(v)) \cup E(B_H(v))$, $\eta'(x) = \eta(x)$,*

*alors $(G,\lambda)\mathcal{R}(G,\lambda')$ si et seulement si $(H,\eta)\mathcal{R}(H,\eta')$.*

Par définition, les *calculs locaux sur les étoiles fermées* correspondent aux relations de réétiquetage localement engendrées sur les étoiles.

### 10.3.2 Revêtements Simples et Calculs Locaux sur les Étoiles Fermées

On présente maintenant le lemme qui met en évidence le lien entre les calculs locaux sur les étoiles fermées et les revêtements simples.

**Lemme 10.11** *On considère un graphe simple* $\mathbf{G}$ *qui est un revêtement simple d'un graphe simple* $\mathbf{H}$ *à travers un homomorphisme* $\gamma$ *et une relation* $\mathcal{R}$ *de réétiquetage localement engendrée sur les étoiles. Si* $\mathbf{H}\mathcal{R}^*\mathbf{H}'$*, alors il existe* $\mathbf{G}'$ *tel que* $\mathbf{G}\mathcal{R}^*\mathbf{G}'$ *et* $\mathbf{G}'$ *est un revêtement simple de* $\mathbf{H}'$ *à travers* $\gamma$*.*

**Preuve :** Il suffit de prouver ce lemme pour un pas de calcul. On considère deux graphes simples $(G, \lambda)$ et $(H, \eta)$ tels que $(G, \lambda)$ est un revêtement simple de $(H, \eta)$ à travers $\gamma$. On considère un pas de réétiquetage qui modifie les étiquettes d'une étoile $B_H(v)$ pour un sommet $v \in V(H)$. On note $\eta'$ l'étiquetage de $H$ obtenu après l'application de ce pas de réétiquetage.

Puisque $\gamma$ est un homomorphisme localement bijectif, on sait que pour tout sommet $u \in \gamma^{-1}(v)$, $(B_G(u), \lambda)$ est isomorphe à $(B_H(v), \eta)$. De plus, d'après le lemme 10.3, on sait que pour tous $u, u' \in \gamma^{-1}(v)$, les étoiles $B_G(u)$ et $B_G(u')$ sont disjointes. On peut donc appliquer le pas de réétiquetage sur chacune des étoiles $B_G(u)$ pour tout $u \in \gamma^{-1}(v)$ de telle sorte que tout sommet $u' \in V(B_G(u))$ soit réétiqueté avec la même étiquette que $\gamma(u')$. On note $\lambda'$ l'étiquetage de $G$ obtenu après l'application de tous ces réétiquetages. Le graphe $(G, \lambda')$ ainsi obtenu est un revêtement de $(H, \eta')$ à travers $\gamma$. □

Le diagramme suivant représente la propriété du Lemme 10.11.

$$\begin{array}{ccc} \mathbf{G} & \xrightarrow{\mathcal{R}^*} & \mathbf{G}' \\ \text{revêtement simple}\downarrow & & \downarrow\text{revêtement simple} \\ \mathbf{H} & \xrightarrow[\mathcal{R}^*]{} & \mathbf{H}' \end{array}$$

## 10.4 Énumération, Nommage et Élection

On s'intéresse maintenant aux problèmes d'élection et de nommage dans le cadre des calculs locaux sur les étoiles fermées. On va montrer que les graphes où on peut résoudre l'élection, le nommage et l'énumération dans ce modèle sont les graphes minimaux pour les revêtements simples. Le fait que les graphes admettant un algorithme d'élection admettent un algorithme de nommage est lié à la Proposition 10.6 : si on arrive à élire un sommet, i.e., à donner à un sommet une étiquette qui n'apparaît qu'une fois, alors on peut donner des étiquettes uniques à tous les sommets.

On donne d'abord le résultat d'impossibilité d'Angluin *qui a montré qu'il ne peut pas exister d'algorithme d'élection ou de nommage pour un graphe simple $\mathbf{G}$ qui n'est pas minimal pour les revêtements simples. On présente ensuite l'algorithme d'énumération de Mazurkiewicz qui permet de résoudre l'énumération sur les graphes simples minimaux pour les revêtements simples.

### 10.4.1 Résultats d'Impossibilité

**Proposition 10.12** *Soit* **G** *un graphe simple étiqueté qui n'est pas minimal pour les revêtements simples. Il n'existe pas d'algorithme de nommage, d'énumération ou d'élection pour le graphe* **G** *utilisant des calculs locaux sur les étoiles fermées.*

**Preuve :** On considère un graphe simple étiqueté **H** qui n'est pas isomorphe à **G** et tel que **G** soit un revêtement simple de **H** à travers un un homomorphisme $\gamma$. Étant donné un algorithme $\mathcal{R}$ utilisant des calculs locaux sur les étoiles fermées, on considère une exécution de $\mathcal{R}$ sur **H**. Si cette exécution est infinie sur **H**, alors d'après le lemme 10.11, il existe une exécution infinie de $\mathcal{R}$ sur **G** ; auquel cas, $\mathcal{R}$ n'est ni un algorithme d'énumération, ni de nommage, ni d'élection.

On suppose maintenant qu'il existe une exécution finie de $\mathcal{R}$ sur **H** et on considère la configuration finale $\mathbf{H}'$. D'après le lemme 10.11, il existe une exécution de $\mathcal{R}$ sur **G** qui permet d'atteindre une configuration $\mathbf{G}'$ telle que $\mathbf{G}'$ est un pseudo-revêtement de $\mathbf{H}'$ à travers $\gamma$. Si $\mathbf{G}'$ n'est pas une configuration finale de $\mathcal{R}$, alors il existe un sommet $u \in V(G)$ tel qu'on puisse appliquer une règle de réétiquetage sur l'étoile $B_{G'}(u)$. Dans ce cas là, on peut aussi appliquer cette même règle de réétiquetage sur l'étoile $B_{H'}(\gamma(u))$ et la configuration $\mathbf{H}'$ n'est pas une configuration finale de $\mathcal{R}$. Par conséquent, $\mathbf{G}'$ est une configuration finale de $\mathcal{R}$. Mais puisque $\mathbf{G}'$ n'est pas isomorphe à $\mathbf{H}'$, on sait d'après la Proposition 10.6 que chaque étiquette de $\mathbf{H}'$ apparaît au moins deux fois dans $\mathbf{G}'$ et par conséquent, $\mathcal{R}$ n'est ni un algorithme de nommage, ni un algorithme d'énumération, ni un algorithme d'élection. □

### 10.4.2 L'Algorithme de Mazurkiewicz

On va maintenant décrire l'algorithme d'énumération $\mathcal{M}$ de Mazurkiewicz qui permet de résoudre le problème de l'énumération sur les graphes simples minimaux pour les revêtements simples.

Durant l'exécution de l'algorithme, chaque sommet $v$ essaie d'obtenir une identité qui est un numéro entre 1 et $|V(G)|$. À chaque fois qu'un sommet modifie son numéro, il en informe immédiatement ses voisins. Chaque sommet $v$ peut donc tenir à jour la liste des numéros de ses voisins, qui sera appelée la *vue locale* de $v$. Lorsqu'un sommet $v$ modifie son numéro ou sa vue locale, il diffuse dans le réseau son numéro accompagné de son étiquette initiale et de sa vue locale.

Si un sommet $u$ découvre qu'un autre sommet $v$ a le même numéro que lui, alors le sommet $u$ doit décider s'il modifie son identité. Pour cela, il compare son étiquette $\lambda(u)$ et sa vue locale avec l'étiquette $\lambda(v)$ et la vue locale de $v$ : si l'étiquette de $u$ est plus faible que l'étiquette de $v$ ou si les deux sommets ont la même étiquette et que la vue locale de $u$ est plus ≪faible≫ (pour un ordre qu'on expliquera par la suite), alors le sommet $u$ choisit un nouveau numéro (sa nouvelle identité temporaire) et informe ses voisins de ce changement de numéro. Ensuite, les numéros et les vues locales qui ont été modifiées sont diffusés à nouveau dans le graphe. Lorsque l'exécution est terminée, si le graphe **G** est minimal pour les revêtements simples, alors chaque sommet a un numéro unique : l'algorithme permet de résoudre le problème du nommage.

#### Étiquettes

On considère un graphe $\mathbf{G} = (G, \lambda)$ où $\lambda\colon V(G) \cup E(G) \to L$ est un étiquetage initial, qui ne sera pas modifié par l'algorithme. Lors de l'exécution, les étiquettes des arêtes ne vont pas

être modifiées et chaque sommet va obtenir une étiquette de la forme $(\lambda(v), n(v), N(v), M(v))$ qui représente les informations suivantes :

- la première composante $\lambda(v)$ est l'étiquette initiale et ne sera pas modifiée lors de l'exécution.
- $n(v) \in \mathbb{N}$ est le *numéro* courant du sommet $v$ qui est modifié lors de l'exécution de l'algorithme,
- $N(v) \in \mathcal{P}_{\text{fin}}(L \times \mathbb{N})$ [1] est la *vue locale* du sommet $v$ qui contient des informations sur les voisins de $v$ ; c'est un ensemble fini de couples $(\ell, n) \in L \times \mathbb{N}$. À tout moment de l'exécution, pour chaque voisin $v'$ de $v$ tel que $n(v') \neq 0$, la vue locale de $v$ contient le couple $(\lambda(\{v, v'\}), n(v'))$.
- $M(v) \subseteq \mathbb{N} \times L \times \mathcal{P}_{\text{fin}}(L \times \mathbb{N})$ est la *boîte-aux-lettres* de $v$. Elle va contenir toute l'information reçue par $v$ lors de l'exécution de l'algorithme, i.e., les couples de numéros et de vues locales qui auront été diffusées par tous les sommets du graphe.

Initialement, chaque sommet a une étiquette de la forme $(\lambda(v), 0, \emptyset, \emptyset)$ qui signifie qu'au début de l'algorithme, $v$ n'a pas choisi de numéro et qu'il n'a aucune information à propos de ses voisins, ni à propos des autres sommets du graphe.

### Un Ordre sur les Vues Locales

Les bonnes propriétés de l'algorithme de Mazurkiewicz sont basées sur un ordre total sur les vues locales. Cet ordre permet à un sommet $u$ de modifier son numéro s'il découvre qu'il existe un autre sommet avec le même numéro, la même étiquette et une vue locale «plus forte». Afin d'éviter des exécutions infinies, il faut que lorsque la vue locale d'un sommet est modifiée, elle ne puisse pas devenir plus faible, et ce pour éviter qu'un sommet ne modifie son numéro à cause d'un message qu'il ait lui même envoyé lors d'une étape précédente.

On définit donc un ordre total sur les ensembles finis de couples de $L \times N$. Pour cela, on considère que $L \times N$ est muni de l'ordre lexicographique usuel : $(\ell, n) \leq (\ell', n')$ si $\ell <_L \ell'$, ou si $\ell = \ell'$ et $n \leq n'$.

Ensuite, étant données deux ensembles $N_1, N_2 \in \mathcal{P}_{\text{fin}}(L \times \mathbb{N})$ distincts, on dit que $N_1 \prec N_2$ si le maximum pour l'ordre lexicographique sur $L \times N$ de la différence symétrique $N_1 \triangle N_2 = (N_1 \setminus N_2) \cup (N_2 \setminus N_1)$ appartient à $N_2$.

On peut aussi voir cet ordre comme l'ordre lexicographique usuel sur les ensembles ordonnés $N_1$ et $N_2$. On note $(\ell_1, n_1), (\ell_2, n_2), \ldots, (\ell_k, n_k)$ et $(\ell'_1, n'_1), (\ell'_2, n'_2), \ldots, (\ell'_l, n'_l)$ les éléments respectifs de $N_1$ et de $N_2$ dans l'ordre décroissant (pour l'ordre lexicographique sur $L \times \mathbb{N}$) : $(\ell_1, n_1) \geq (\ell_2, n_2) \geq \cdots \geq (\ell_k, n_k)$ et $(\ell'_1, n'_1) \geq (\ell'_2, n'_2) \geq \cdots \geq (\ell'_l, n'_l)$. Alors $N_1 \prec N_2$ si l'une des conditions suivantes est vérifiée :

- $k < l$ et pour tout $i \in [1, k]$, $(\ell_i, n_i) = (\ell'_i, n'_i)$,
- $(\ell_i, n_i) < (\ell'_i, n_i)$ où $i$ est le plus petit indice pour lequel $(\ell_i, n_i) \neq (\ell'_i, n_i)$.

Si $N(u) \prec N(v)$, alors on dit que la vue locale $N(v)$ de $v$ est *plus forte* que celle de $u$ et que $N(u)$ est *plus faible* que $N(v)$. En utilisant l'ordre total $<_L$ de $L$, on étend l'ordre $\prec$ pour obtenir un ordre total sur $L \times \mathcal{P}_{\text{fin}}(L \times \mathbb{N})$ : $(\ell, N) \prec (\ell', N')$ si $\ell <_L \ell'$ ou bien si $\ell = \ell'$ et $N \prec N'$. Par la suite, on notera $\preceq$ la clôture réflexive de $\prec$.

### Les Règles de Réétiquetage

On décrit maintenant l'algorithme d'énumération grâce à des règles de réétiquetage. La première règle $\mathcal{M}_0$ est une règle spéciale qui permet à chaque sommet $v$ de modifier son étiquette $\lambda(v)$ pour

1. Pour tout ensemble $S$, on note $\mathcal{P}_{\text{fin}}(S)$ l'ensemble des parties finies de $S$.

obtenir l'étiquette $(\lambda(v), 0, \emptyset, \emptyset)$.

Les règles sont décrites pour une étoile $B(v_0)$ de centre $v_0$. L'étiquette d'un sommet $v \in N_G(v_0) \cup \{v_0\}$ avant l'application de la règle est notée $(\lambda(v), n(v), N(v), M(v))$ et on note $(\lambda(v), n'(v), N'(v), M'(v))$ l'étiquette de $v$ après l'application de la règle de réétiquetage. Par ailleurs, afin de rendre plus lisible les règles, on ne mentionne pas les différents champs des étiquettes qui ne sont pas modifiés.

La première règle permet aux sommets d'une même étoile d'échanger les informations dont ils disposent (i.e., contenues dans leur boîte-aux-lettres) à propos des étiquettes présentes dans le graphe. Cette règle ne peut être appliquée que si un tel échange d'information est nécessaire (i.e., tous les sommets de $B(v_0)$ n'ont pas la même boîte-aux-lettres).

$\mathcal{M}_1$: **Règle de Diffusion**

*Précondition :*
— $\exists v \in N_G(v_0)$ tel que $M(v) \neq M(v_0)$.
*Réétiquetage :*
— $\forall v \in V(B(v_0))$, $M'(v) := \bigcup_{w \in V(B(v_0))} M(w)$.

La deuxième règle permet à un sommet $v_0$ de changer de numéro s'il n'a pas encore de numéro (i.e., $n(v_0) = 0$) ou s'il sait qu'il existe un sommet dans le graphe qui a le même numéro que lui et qui a une étiquette ou une vue locale plus forte que la sienne. Dans ce cas là, $v_0$ choisit un nouveau numéro et modifie la vue locale de ses voisins. Toutes les boîtes-aux-lettres des sommets de l'étoile $B(v_0)$ sont modifiées : on ajoute à chaque boîte-aux-lettres tous les couples $(n'(v), \lambda(v), N'(v))$ pour tous les sommets $v'$ de l'étoile $B(v_0)$.

$\mathcal{M}_2$: **Règle de Renommage**

*Précondition :*
— $\forall v \in N_G(v_0)$, $M(v) = M(v_0)$,
— $n(v_0) = 0$ ou
$\exists (n(v_0), \ell, N) \in M(v_0)$ tel que $(\lambda(v_0), N(v_0)) \prec (\ell, N)$
*Réétiquetage :*
— $n'(v_0) := 1 + \max\{n' \mid \exists (n', \ell', N') \in M(v_0)\}$ ;
— $\forall v \in N_G(v_0)$, $N'(v) := N(v) \setminus \{(\lambda(\{v_0, v\}), n(v_0))\} \cup \{(\lambda(\{v_0, v\}), n'(v_0))\}$ ;
— $\forall v \in V(B(v_0))$, $M'(v) := M(v) \cup \bigcup_{w \in V(B(v_0))} \{(n'(w), \lambda(w), N'(w))\}$.

### 10.4.3 Correction de l'Algorithme d'Énumération

On considère un graphe simple étiqueté $\mathbf{G}$. Pour tout sommet $v \in V(G)$, on note $(\lambda(v), n_i(v), N_i(v), M_i(v))$ l'étiquette du sommet $v$ après la $i$ème étape de réétiquetage de l'algorithme $\mathcal{M}$ décrit ci-dessus. On présente d'abord quelques propriétés qui sont satisfaites par n'importe quelle exécution de l'algorithme.

#### Propriétés Satisfaites lors de l'Exécution

Le lemme suivant, qui peut être facilement prouvé par une récurrence sur le nombre d'étapes, rappelle quelque propriétés simples qui sont toujours satisfaites par l'étiquetage.

**Lemme 10.13** *Pour tout sommet $v \in V(G')$, et pour toute étape $i$,*

1. $n_i(v) \neq 0 \implies (n_i(v), \lambda(v), N_i(v)) \in M_i(v)$,

2. $\exists(\ell_e, n) \in N_i(v) \iff \exists v' \in N_G(v)$ *tel que* $n_i(v') = n > 0$ *et* $\lambda(\{v, v'\}) = \ell_e$,
3. $\forall(\ell_e, n) \in N_i(v), n \neq n_i(v)$ *et* $\exists(n, \ell, N) \in M_i(v)$,
4. $\forall v', v'' \in N_G(v), n_i(v), n_i(v') > 0 \implies n_i(v') \neq n_i(v'')$.

L'algorithme $\mathcal{M}$ a des propriétés de monotonicité intéressantes qui sont données dans le lemme suivant.

**Lemme 10.14** *Pour chaque sommet $v$ et chaque étape $i$,*
— $n_i(v) \leq n_{i+1}(v)$,
— $N_i(v) \preceq N_{i+1}(v)$,
— $M_i(v) \subseteq M_{i+1}(v)$.
*De plus, à chaque étape $i$, il existe un sommet $v$ telle qu'au moins une de ces inégalités (ou inclusions) est stricte pour $v$.*

**Preuve :** La propriété est trivialement vraie pour les sommets qui ne sont pas réétiquetés lors de la $(i+1)$ème étape. De plus, il est facile de voir que quelque soit la règle appliquée à l'étape $i+1$, on a toujours $M_i(v) \subseteq M_{i+1}(v)$ pour tout sommet $v \in V(G)$.

Pour chaque sommet $v$ tel que $n_i(v) \neq n_{i+1}(v)$, alors la règle $\mathcal{M}_2$ a été appliquée sur l'étoile $B(v)$ et on sait que $n_{i+1}(v) = 1 + \max\{n' \mid \exists(n', \ell', N') \in M_i(v)\}$. De plus, ou bien $n_i(v) = 0 < n_{i+1}(v)$, ou alors d'après le lemme 10.13, $(n_i(v), \lambda(v), N_i(v)) \in M_i(v)$ et donc $n_i(v) < n_{i+1}(v)$.

Pour chaque sommet $v$ tel que $N_i(v) \neq N_{i+1}(v)$, alors la règle $\mathcal{M}_2$ a été appliquée sur une étoile $B(v_0)$ telle que $v_0 \in N_G(v)$. On sait que $N_{i+1}(v) = N_i(v) \setminus \{(\lambda(\{v_0, v\}), n_i(v_0))\} \cup \{(\lambda(\{v_0, v\}), n_{i+1}(v_0))\}$ et que pour tout $(n, \ell, N) \in M_i(v)$, $n_{i+1}(v_0) > n$. Ainsi, on a $\max N_{i+1}(v) \triangle N_i(v) = (\lambda(\{v_0, v\}), n_{i+1}(v_0)) \in N_{i+1}(v)$. Par conséquent, $N_i(v) \prec N_{i+1}(v)$.

Puisque chaque application d'une règle modifie l'étiquette d'au moins un sommet $v$, on sait que l'une de ces inégalités est stricte pour $v$. □

Les informations dont dispose chaque sommet $v$ dans sa boîte-aux-lettres permettent d'obtenir des informations vérifiées par la configuration globale du graphe. Les deux lemmes suivants permettent de prouver que si un sommet $v$ connaît un numéro $m$ à une étape $i$ (i.e., il existe $\ell, N$ tels que $(m, \ell, N) \in M_i(v)$), alors pour chaque $m' \leq m$, il existe un sommet $w$ tel que $n_i(w) = m$. On montre d'abord que si $v$ connaît un numéro $m$, alors il existe un sommet $w$ tel que $n_i(w) = m$.

**Lemme 10.15** *Pour chaque sommet $v \in V(G)$ et chaque étape $i$, pour tout $(m, \ell, N) \in M_i(v)$, il existe un sommet $w \in V(G)$ tel que $n_i(w) = m$.*

**Preuve :** On remarque d'abord qu'un triplet $(m, \ell, N)$ est ajouté à une étape $i$ dans $\bigcup_{v \in V(G)} M_i(v)$ seulement s'il existe un sommet $v$ tel que $n_i(v) = m$, $\lambda(v) = \ell$ et $N_i(v) = N$.

Étant donné un sommet $v$, une étape $i$ et un triplet $(m, \ell, N) \in M_i(v)$, on note $U = \{(u, j) \in V(G) \times \mathbb{N} \mid j \leq i, n_j(u) = m\}$. On considère ensuite l'ensemble $U' = \{(u, j) \in U \mid \forall(u', j') \in U, (\lambda(u'), N_{j'}(u')) \prec (\lambda(u), N_j(u))$ ou $(\lambda(u'), N_{j'}(u')) = (\lambda(u), N_j(u))$ et $j' \leq j\}$. Puisque $(m, \ell, N) \in M_i(v)$, $U$ et $U'$ sont deux ensembles non-vides. On remarque aisément qu'il existe $i_0$ tel que pour tout $(u, j) \in U'$, $j = i_0$.

Si $i_0 < i$, il existe exactement un élément $(u, i_0) \in U'$ puisqu'à chaque étape, le numéro d'au plus un sommet peut être modifié. Le numéro $n_{i_0}(u) = m$ a donc été modifié à l'étape $i_0 + 1$, mais par maximalité de $(\lambda(u), N_{i_0}(u))$, la règle $\mathcal{M}_2$ n'a pas pu être appliquée à $u$ à l'étape $i_0$. Par conséquent, $i_0 = i$ et il existe donc un sommet $w$ tel que $n_i(w) = m$. □

Dans le lemme suivant, on montre que si un sommet $v$ connaît un numéro $m$, alors il connaît tous les numéros inférieurs à $m$.

**Lemme 10.16** *Pour chaque sommet $v$ et chaque étape $i$, pour tout $(m, \ell, N) \in M_i(v)$, pour tout $m' \in [1, m]$, il existe $(m', \ell', N') \in M_i(v)$.*

**Preuve :** On montre ce lemme par récurrence sur $i$. Initialement, la propriété est trivialement vraie. On suppose que la propriété est vérifiée pour $i \geq 0$. La propriété est trivialement vraie à l'étape $i+1$ pour tout sommet $w \in V(G)$ dont l'étiquette n'est pas modifiée à l'étape $i+1$. Soit $v$ un sommet dont l'étiquette est modifiée à l'étape $i+1$.

Si la règle $\mathcal{M}_1$ a été appliquée à l'étape $i+1$, alors pour tout $(m, \ell, N) \in M_{i+1}(v)$, il existe $v'$ tel que $(m, \ell, N) \in M_i(v')$ et $M_i(v') \subseteq M_{i+1}(v)$. Par conséquent, par hypothèse de récurrence, pour tout $m' < m$, il existe $(m', \ell', N') \in M_i(v') \subseteq M_{i+1}(v)$. On suppose maintenant que la règle $\mathcal{M}_2$ a été appliquée à l'étape $i+1$ sur une étoile $B(v_0)$ qui contient $v$. Soit $(m, \ell, N) \in M_{i+1}(v) \setminus M_i(v)$. Si $m = n_{i+1}(v_0) = 1 + \max\{m' \mid \exists (m', \ell', N') \in M_i(v)\}$, pour tout $m' < m$, il existe $(m', \ell', N') \in M_i(v) \subseteq M_{i+1}(v)$. Si $m \neq n_{i+1}(v_0)$, alors il existe $v' \in N_G(v_0)$ tel que $m = n_i(v') = n_{i+1}(v')$ et d'après le lemme 10.13, on sait qu'il existe $(m, \ell', N') \in M_i(v') = M_i(v)$. Ainsi la propriété est vérifiée à l'étape $i+1$. □

On veut maintenant montrer que toute exécution de l'algorithme $\mathcal{M}$ termine sur $\mathbf{G}$. D'après les lemmes 10.15 et 10.16, on voit qu'à chaque étape de l'exécution, les numéros des sommets forment un ensemble $[1, k]$ ou un ensemble $[0, k]$ avec $k \leq |V(G)|$. Par conséquent, d'après le lemme 10.14, on sait qu'il existe une étape $i_0$ telle que pour tout sommet $v$ et toute étape $i \geq i_0$, $n_{i+1}(v) = n_i(v)$.

De plus, pour chaque sommet $v$ et chaque étape $i$, si $N_i(v)$ contient un couple $(\ell_e, n')$, alors il existe $v' \in N_G(v)$ tel que $n_i(v') = n'$ et $\lambda(\{v, v'\}) = \ell_e$. Par conséquent $N(v)$ ne peut prendre qu'un nombre fini de valeurs et il en est de même pour $M(v)$. Ainsi le nombre de valeurs différentes que peut prendre l'étiquette de chaque sommet est fini (mais dépend de la taille du graphe). Par ailleurs, les étiquettes consécutives de chaque sommet $v$ forment une suite croissante et puisqu'à chaque étape $i$, l'étiquette d'au moins un sommet est modifiée, toute exécution de l'algorithme termine : la relation $\mathcal{M}$ est noethérienne.

### Propriétés Satisfaites par l'Étiquetage Final

Puisqu'on sait que l'algorithme termine toujours, on s'intéresse maintenant aux propriétés satisfaites par l'étiquetage final.

**Lemme 10.17** *Toute exécution $\rho$ de l'algorithme $\mathcal{M}$ sur un graphe simple étiqueté $\mathbf{G} = (G, \lambda)$ termine et l'étiquetage final $(\lambda, n_\rho, N_\rho, M_\rho)$ vérifie les propriétés suivantes :*

1. *il existe un entier $k \leq |V(G)|$ tel que $\{n_\rho(v) \mid v \in V(G)\} = [1, k]$,*

*et pour tous sommets $v, v'$ :*

2. *$M_\rho(v) = M_\rho(v')$,*
3. *$(n_\rho(v), \lambda(v), N_\rho(v)) \in M_\rho(v')$,*
4. *si $n_\rho(v) = n_\rho(v')$, alors $\lambda(v) = \lambda(v')$ et $N_\rho(v) = N_\rho(v')$,*
5. *$\forall w, w' \in N_G(v), n_\rho(w) \neq n_\rho(w')$,*
6. *$(\ell_e, n) \in N_\rho(v)$ si et seulement s'il existe $w \in N_G(v)$ tel que $n_\rho(w) = n$ et $\lambda(\{v, w\}) = \ell_e$ ; auquel cas, $(\ell_e, n_\rho(v)) \in N_\rho(w)$.*

**Preuve :**

1. D'après les lemmes 10.15 et 10.16 et puisque la règle $\mathcal{M}_2$ ne peut pas être appliquée.
2. Dans le cas contraire, la règle $\mathcal{M}_1$ peut être appliquée.
3. C'est une conséquence directe de la propriété précédente d'après le lemme 10.13.
4. Dans le cas contraire, la règle $\mathcal{M}_2$ peut être appliquée à $v$ ou à $v'$.
5. D'après le lemme 10.13 et puisque la règle $\mathcal{M}_2$ ne peut pas être appliquée.
6. D'après le lemme 10.13.

□

Grâce au Lemme 10.17, on peut prouver que l'étiquetage final permet de construire un graphe **H** tel que **G** est un revêtement de **H**.

**Proposition 10.18** *Étant donné un graphe simple* **G***, on peut construire, à partir de l'étiquetage final obtenu après une exécution* $\rho$ *de* $\mathcal{M}$*, un graphe simple* **H** *tel qu'il existe un homomorphisme localement bijectif de* **G** *dans* **H**.

**Preuve :** On utilise les notations du Lemme 10.17.

On considère le graphe $H$ défini par $V(H) = \{m \in \mathbb{N} \mid \exists v \in V(G), n_\rho(v) = m\}$ et $E(H) = \{\{m, m'\} \mid \exists v, v' \in V(G); n_\rho(v) = m, n_\rho(v') = m' \text{ et } \{v, v'\} \in E(G)\}$. D'après le lemme 10.13, on sait qu'il n'existe pas d'arête $\{v, v'\} \in E(G)$ telle que $n_\rho(v) = n_\rho(v')$ : le graphe $H$ ne contient donc pas de boucle. De plus, de par la définition de $E(H)$, on sait que $H$ ne contient pas d'arêtes multiples. Ainsi le graphe $H$ est bien un graphe simple. On définit un étiquetage $\eta$ de $H$ en posant $\eta(n_\rho(v)) = \lambda(v)$ et pour toute arête $\{v, v'\} \in E(H)$, $\eta(\{n_\rho(v), n_\rho(v')\}) = \lambda(\{v, v'\})$. D'après le lemme 10.17, si deux sommets $v, v' \in V(G)$ ont le même numéro, alors $\lambda(v) = \lambda(v')$. De plus, pour toutes arêtes $\{v, v'\}, \{w, w'\} \in E(G)$ telles que $n_\rho(v) = n_\rho(w) = n$ et $n_\rho(v') = n_\rho(w') = n'$, on sait d'après le lemme 10.17 que $(\lambda(\{v, v'\}), n') \in N_\rho(v)$ et que $(\lambda(\{w, w'\}), n') \in N_\rho(w)$. Puisque $n_\rho(v) = n_\rho(w)$, on a $N_\rho(w) = N_\rho(v)$. Ainsi, puisqu'il ne peut exister deux couples distincts $(\ell_1, n'), (\ell_2, n')$ dans $N_\rho(v) = N_\rho(w)$, $\lambda(\{w, w'\}) = \lambda(\{v, v'\})$. Par conséquent, l'étiquetage $\eta$ de $H$ est bien défini.

On considère maintenant la fonction $n_\rho : V(G) \to V(H)$. Par définition de $H$, on sait que si $\{v, v'\} \in E(G)$, alors $\{n_\rho(v), n_\rho(v')\} \in E(G)$ et $\eta(\{n_\rho(v), n_\rho(v')\}) = \lambda(\{v, v'\})$ . De plus, pour tout $v \in V(G)$, $\eta(n_\rho(v)) = \lambda(v)$ et par conséquent, $n_\rho$ est un homomorphisme de **G** dans **H**. D'après le lemme 10.17, pour tout sommet $v$, si $m \in N_H(n_\rho(v))$, alors il existe $w \in N_G(v)$ tel que $n_\rho(w) = m$ et par conséquent, $n_\rho(N_G(v)) = N_H(n_\rho(v))$. De plus, pour tous $w, w' \in N_G(v)$, $n_\rho(w) \neq n_\rho(w')$ et on a donc $|N_G(v)| = |N_H(n_\rho(v))|$. Ainsi, l'homomorphisme $n_\rho$ est localement bijectif et **G** est un revêtement de **H**. □

On considère maintenant un graphe simple **G** qui est minimal pour les revêtements simples. Pour chaque exécution $\rho$ de $\mathcal{M}$ sur **G**, le graphe obtenu à partir de l'étiquetage final est isomorphe à **G**. Par conséquent, l'ensemble des numéros des sommets est exactement $[1, |V(G)|]$ : chaque sommet a un identifiant unique. L'algorithme $\mathcal{M}$ permet de résoudre le nommage et l'énumération sur la famille des graphes minimaux pour les revêtements simples, mais si aucune information à propos de **G** n'est disponible, les sommets ne peuvent pas détecter la terminaison.

Cependant, il est possible de détecter la terminaison pour un graphe simple **G** donné (l'algorithme dépend alors de **G**). En effet, une fois qu'un sommet a obtenu le numéro $|V(G)|$, d'après les lemmes 10.15 et 10.16, il sait que tous les sommets de **G** ont un numéro unique qui ne va plus

être modifié. Dans ce cas là, on peut aussi résoudre le problème de l'élection, puisque ce sommet peut prendre l'étiquette ÉLU et diffuser ensuite l'information qu'un sommet a été élu.

Par ailleurs, d'après la Proposition 10.12, on sait que pour tout graphe simple $\mathbf{G}$ qui n'est pas minimal pour les revêtements simples, il n'existe aucun algorithme utilisant des calculs locaux sur les étoiles fermées qui permettent de résoudre les problèmes du nommage ou de l'élection sur $\mathbf{G}$. On a donc prouvé le théorème suivant.

**Théorème 10.19** *Pour tout graphe simple étiqueté* $\mathbf{G}$*, les assertions suivantes sont équivalentes :*

1. *il existe un algorithme de nommage (ou d'énumération) pour* $\mathbf{G}$ *utilisant des calculs locaux sur les étoiles fermées,*
2. *il existe un algorithme d'élection et un algorithme de nommage (ou d'énumération) avec détection de la terminaison pour* $\mathbf{G}$ *utilisant des calculs locaux sur les étoiles fermées,*
3. $\mathbf{G}$ *est minimal pour les revêtements simples.*

**Remarque 10.20** *Étant donné un graphe simple* $\mathbf{G}$ *minimal pour les revêtements simples, pour détecter que l'algorithme* $\mathcal{M}$ *a attribué un identifiant unique à chaque sommet, il suffit de connaître le nombre de sommets de* $\mathbf{G}$*. Ainsi, l'algorithme* $\mathcal{M}$ *permet de résoudre l'élection ainsi que le nommage avec détection de la terminaison sur les graphes minimaux pour les revêtements simples de taille donnée.*

## 10.5 Importance de la Connaissance Initiale

Dans la partie précédente, on a montré que l'algorithme de Mazurkiewicz permettait de résoudre le problème de l'élection et du nommage avec détection de la terminaison sur la famille des graphes minimaux pour les revêtements simples de taille donnée. Cependant, étant donné un graphe $\mathbf{G}$ minimal pour les revêtements simples, l'algorithme de Mazurkiewicz à lui seul ne permet pas de résoudre l'élection sur $\mathbf{G}$ si on ne connaît qu'une borne sur la taille de $\mathbf{G}$. Par ailleurs, étant donné un graphe $\mathbf{G}$ qui n'est pas minimal pour les revêtements, l'algorithme de Mazurkiewicz ne permet pas de détecter qu'on ne peut pas élire dans $\mathbf{G}$ si on ne connait que la taille de $\mathbf{G}$.

Cependant, il n'est pas forcément nécessaire de connaître la taille pour pouvoir élire ; d'autres types de connaissance initiales permettent de résoudre le problème de l'élection, ou du nommage avec détection de la terminaison. Par exemple, si on sait que le graphe $\mathbf{G}$ est un arbre, on peut toujours élire un sommet, sans connaître d'autre information sur $\mathbf{G}$. En effet, l'algorithme d'élection pour la famille des arbres présenté dans le Chapitre **??** peut être implémenté avec des calculs locaux sur les étoiles fermées et on a donc le théorème suivant.

**Théorème 10.21** *Il existe un algorithme d'élection pour la famille des arbres utilisant des calculs locaux sur les étoiles fermées.*

On va maintenant présenter une extension de l'algorithme de détection de la terminaison de Szymansky, Shi et Prywes qui permet d'obtenir un algorithme d'élection pour des classes de graphes minimaux pour les revêtements simples qui n'ont pas tous la même taille.

### 10.5.1 L'Algorithme de Szymansky, Shi et Prywes

Cet algorithme a été initialement décrit pour détecter la terminaison d'un algorithme dans un modèle où les processus où les sommets communiquent en échangeant des messages. Étant donné

un algorithme où chaque processus sait qu'il a calculé sa valeur finale, l'algorithme de Szymansky, Shi et Prywes (noté SSP) permet aux processus de détecter, sous certaines conditions, un instant où tous les sommets ont calculé leurs valeurs finales : le calcul est globalement terminé.

Étant donné un graphe $G$, on associe à chaque sommet $v \in V(G)$ un prédicat $P(v)$ et un entier $a(v)$, son *niveau de confiance*. Initialement $P(v)$ est FAUX et $a(v)$ est égal à $-1$. Lorsqu'un processus a calculé sa valeur finale, il modifie la valeur de $P(v)$ qui devient VRAI (et qui ne changera plus). À chaque fois qu'un processus modifie la valeur de $a(v)$, il en informe ses voisins.

La modification de la valeur de $a(v_0)$ d'un sommet $v_0$ dépend seulement de la valeur de $P(v_0)$ et des informations dont $v_0$ dispose à propos des valeurs $\{a(v_1), \ldots, a(v_d)\}$ de ses voisins :

— Si $P(v_0) =$FAUX, alors $a(v_0) = -1$ ;

— si $P(v_0) =$VRAI, alors $a(v_0) = 1 + \min\{a(v_k) \mid 0 \leq k \leq d\}$.

Le principe de l'algorithme de SSP généralisé (noté GSSP) introduite et utilisée dans est le suivant. Au lieu de considérer que la fonction $P$ est un prédicat qui ne peut changer qu'une fois de valeur pour passer de FAUX à VRAI, la fonction $P$ peut prendre une valeur quelconque et peut être modifiée un nombre arbitraire de fois. Cependant, on suppose que pour tout sommet $v$, si $P(v)$ a une valeur $\alpha$ qui est modifiée par la suite, alors $P(v)$ ne pourra plus jamais être égal à $\alpha$. Autrement dit, entre deux moments où un sommet $v$ est tel que $P(v) = \alpha$, la fonction $P(v)$ est constante. On dit alors que la fonction $P$ est *connexe par valeurs*.

### 10.5.2 Une Adaptation de l'Algorithme de Mazurkiewicz

On va utiliser l'algorithme GSSP pour vérifier que tous les sommets du graphe ont la même boîte-aux-lettres et qu'aucun d'eux ne peut appliquer la règle $\mathcal{M}_2$ de l'algorithme de Mazurkiewicz, i.e., changer de numéro. Si on peut s'assurer que ces deux propriétés sont vraies, alors on sait que l'exécution de l'algorithme est terminée et on sait que les propriétés du Lemme 10.17 et de la Proposition 10.18 sont vérifiées. D'après le lemme 10.14, on sait que les boîtes aux lettres des sommets ne peuvent qu'augmenter pour l'inclusion et $M : v \mapsto M(v)$ est donc une fonction connexe par valeurs.

L'étiquette des sommets de $\mathbf{G} = (G, \lambda)$ sont alors de la forme $(\lambda(v), n(v), N(v), M(v), a(v))$ où la seule différence avec l'algorithme de Mazurkiewicz est le champ $a(v)$ qui est le *rayon de confiance* du sommet $v$. Initialement, tous les sommets ont l'étiquette $(\lambda(v), 0, \emptyset, \emptyset, -1)$.

Les deux premières règles de l'algorithme de Mazurkiewicz sont adaptées de telle sorte qu'à chaque fois que la boîte-aux-lettres d'un sommet est modifiée, alors son *rayon de confiance* est réinitialisé à $-1$ puisque le sommet a modifié sa boîte-aux-lettres.

$\mathcal{M}_1'$: **Règle de Diffusion**

*Précondition :*

— $\exists v \in N_G(v_0)$ tel que $M(v) \neq M(v_0)$.

*Réétiquetage :*

— $\forall v \in V(B(v_0))$, $M'(v) := \bigcup_{w \in V(B(v_0))} M(w)$ ;

— $\forall v \in V(B(v_0))$, $a'(v) := -1$.

$\mathcal{M}_2'$: **Règle de Renommage**

*Précondition :*

— $\forall v \in N_G(v_0)$, $M(v) = M(v_0)$,

— $n(v_0) = 0$ ou
$\exists (n(v_0), \ell, N) \in M(v_0)$ tel que $(\lambda(v_0), N(v_0)) \prec (\ell, N)$

*Réétiquetage :*
— $n'(v_0) := 1 + \max\{n' \mid \exists(n', \ell', N') \in M(v_0)\}$ ;
— $\forall v \in N_G(v_0)$, $N'(v) := N(v) \setminus \{(\lambda(\{v_0, v\}), n(v_0))\} \cup \{(\lambda(\{v_0, v\}), n'(v_0))\}$ ;
— $\forall v \in V(B(v_0))$, $M'(v) := M(v) \cup \bigcup_{w \in V(B(v_0))} (n'(w), \lambda(w), N'(w))$ ;
— $\forall v \in V(B(v_0))$, $a'(v) := -1$.

Une troisième règle est ajouté qui permet à un sommet $v$ qui ne peut pas modifier son numéro d'augmenter son rayon de confiance si tous ses voisins ont la même boîte-aux-lettres que $v$ et si leurs rayons de confiances sont tous supérieurs ou égaux à celui de $v$.

$\mathcal{M}'_3$: **Règle GSSP**
*Précondition :*
— $\forall v \in N_G(v_0)$, $M(v) = M(v_0)$,
— $n(v_0) > 0$ et $\forall (n(v_0), \ell, N) \in M(v_0), (\ell, N) \preceq (\lambda(v_0), N(v_0))$,
— $\forall v \in N_G(v_0)$, $a(v) \geq a(v_0)$
*Réétiquetage :*
— $a'(v_0) := 1 + \min\{a(v) \mid v \in N_G(v_0)\}$.

### 10.5.3 Propriétés Satisfaites par l'Algorithme

Comme précedemment, on considère une exécution de l'algorithme $\mathcal{M}'$ sur un graphe simple étiqueté $\mathbf{G}$. Pour tout sommet $v \in V(G)$, on note $(\lambda(v), n_i(v), N_i(v), M_i(v), a_i(v))$ l'étiquette du sommet $v$ après la $i$ème étape de réétiquetage.

Le lemme suivant qui peut être facilement prouvé par induction sur le nombre d'étapes montre que le rayon de confiance d'un sommet ne peut qu'augmenter tant que sa boîte-aux-lettres n'est pas modifiée.

**Lemme 10.22** *Pour tout sommet $v$ et toute étape $i$, si $M_i(v) = M_{i+1}(v)$, alors $a_{i+1}(v) \geq a_i(v)$. De plus, si $a_i(v) \geq 0$, alors la règle $\mathcal{M}'_2$ ne peut être appliquée sur l'étoile de centre $v$.*

Dans le lemme suivant, on montre que le rayon de confiance d'un sommet permet d'obtenir des informations sur le contenu des boîtes-aux-lettres d'autres sommets du graphe lors d'étapes précédentes de l'exécution.

**Lemme 10.23** *Pour tout sommet $v \in V(G)$ et toute étape $i$, pour tout sommet $w \in V(G)$ tel que $\text{dist}_G(v, w) \leq a_i(v)$, il existe une étape $j \geq i$ telle que $a_j(w) \geq a_i(v) - \text{dist}_G(v, w)$ et $M_j(v) = M_i(v)$.*

**Preuve :** On fait une démonstration par récurrence sur la distance $k$ entre $v$ et $w$. Si $k = 0$, la propriété est trivialement vraie. On suppose maintenant que la propriété est vraie pour tous sommets $v, w$ tels que $\text{dist}_G(v, w) \leq k$.

On considère deux sommets $v, w$ et une étape $i$ tels que $a_i(v) \geq k + 1$ et $\text{dist}_G(v, w) = k + 1$. Il existe un sommet $u \in N_G(v)$ tels que $\text{dist}_G(u, w) = k$. On considère la dernière étape $j'$ où la règle $\mathcal{M}'_3$ a été appliquée au sommet $v$. On sait que $a_{j'}(u) \geq a_{j'}(v) - 1 = a_i(v) - 1$ et $M_{j'}(u) = M_{j'}(v) = M_i(v)$. Par hypothèse de récurrence, on sait qu'il existe une étape $j < j'$ telle que $a_j(w) \geq a_{j'}(u) - k \geq a_i(v) - (k + 1)$ et $M_j(w) = M_{j'}(u) = M_i(v)$. Ainsi, la propriété est vérifiée pour tous sommets $v, w$ à distance $k + 1$. □

Dans le lemme suivant, on montre que si à un moment donné, un sommet a un rayon de confiance supérieur au diamètre du graphe, alors tous les sommets du graphe ont la même boîte-aux-lettres et ont tous un rayon de confiance supérieur à 0.

**Lemme 10.24** *S'il existe un sommet $v$ et une étape $i$ telle que $a_i(v) \geq D(G)$, alors pour tout $w \in V(G)$, $M_i(w) = M_i(v)$ et $a_i(v) \geq 0$.*

**Preuve :** Puisque $a_i(v) > D(G)$, on sait d'après le lemme 10.23 que pour tout sommet $w \in V(G)$, il existe une étape $i_w < i$ telle que $a_{i_w} \geq 0$ et $M_{i_w}(w) = M_i(v)$.

Supposons qu'il existe un sommet $w$ tel que $M_i(w) \neq M_{i_w}(w)$. Soit $j$ l'étape de l'exécution lors de laquelle pour la première fois, la boîte-aux-lettres d'un sommet $w$ qui valait $M_i(v)$ a été modifiée. Autrement dit, pour tout sommet $w \in V(G)$, il existe une étape $j' \geq j-1$ telle que $M_{j'}(w) = M_i(v)$ et il existe un sommet $w$ tel que $M_{j-1}(w) = M_i(v) \subsetneq M_j(w)$. Cela signifie que la règle $\mathcal{M}'_1$ ou $\mathcal{M}'_2$ a été appliquée lors de l'étape $j$ ; on note $v_0$ le centre de l'étoile sur laquelle cette règle a été appliquée.

On sait que pour tout sommet $w \in V(B_G(V_0))$, $M_{j-1}(w) = M_i(v) \subsetneq M_j(w)$ et qu'il existe une étape $i_w \leq j-1$ telle que $M_{i_w}(w) = M_i(v)$ et $a_{i_w}(w) \geq 0$. D'après le lemme 10.22, on sait que pour tout sommet $w \in V(B(V_0))$, $M_{j-1}(w) = M_i(v)$ et $a_{j-1}(w) \geq 0$. Par conséquent, puisque tous les sommets de $B_G(v_0)$ ont la même boîte-aux-lettres à l'étape $j-1$, la règle $\mathcal{M}'_1$ ne peut pas être appliquée sur l'étoile $B_G(v_0)$ lors de l'étape $j$. De plus, $a_j(v_0) \geq 0$ et d'après le lemme 10.22, on sait que la règle $\mathcal{M}'_2$ ne peut pas être appliquée sur l'étoile de centre $v_0$. Par conséquent, pour tout sommet $w \in V(G)$, $M_i(w) = M_i(v)$. □

Ainsi, si on connaît une borne $B$ sur le diamètre de $\mathbf{G}$, l'algorithme $\mathcal{M}'$ permet de détecter que l'exécution de l'algorithme de Mazurkiewicz sous-jacent est terminé. En effet, une fois qu'un sommet a un rayon de confiance supérieur ou égal à $B$, il sait que tous les sommets de $\mathbf{G}$ ont la même boîte-aux-lettres et qu'ils ont leurs numéros et vues locales finaux.

Si on sait que le graphe $\mathbf{G}$ est minimal pour les revêtements simples, on sait alors d'après la Proposition 10.18 que tous les sommets ont un identifiant unique et dans ce cas là, le sommet dont le numéro est 1 peut prendre l'étiquette ÉLU et diffuser l'information.

**Théorème 10.25** *Pour tout entier $B$, il existe un algorithme d'élection et un algorithme de nommage avec détection de la terminaison utilisant des calculs locaux sur les étoiles fermées pour la famille des graphes minimaux pour les revêtements simples dont le diamètre est bornée par $B$.*

Ainsi, il n'est pas nécessaire de connaître la taille pour pouvoir élire dans un graphe $\mathbf{G}$ que l'on sait minimal pour les revêtements simples : une borne sur la taille ou le diamètre est suffisante.

De plus, Angluin a montré qu'on ne pouvait pas résoudre l'élection si les sommets ne disposaient d'aucune information à propos de $\mathbf{G}$, i.e., il n'existe pas d'algorithme d'élection universel pour la famille des graphes minimaux pour les revêtements simples.

Cependant, on ne sait pas nécessairement si le graphe $\mathbf{G}$ est minimal pour les revêtements simples. On peut donc souhaiter avoir un algorithme qui permet ou bien de résoudre l'élection sur $\mathbf{G}$, ou bien de détecter que le graphe $\mathbf{G}$ n'est pas minimal pour les revêtements simples. On va montrer que l'algoritme $\mathcal{M}'$ permet de résoudre ce problème si on a une borne serrée sur la taille de $\mathbf{G}$.

**Théorème 10.26** *Pour tout entier $B$, il existe un algorithme effectif d'élection et de nommage avec détection de la terminaison utilisant des calculs locaux sur les étoiles fermées pour la classe des graphes $\mathbf{G}$ tels que $V(G) \leq B < 2|V(G)|$.*

**Preuve :** On sait d'après la Proposition 10.18 et le lemme 10.24 qu'avec la connaissance d'une borne serrée $B$ sur la taille d'un graphe $\mathbf{G}$, toute exécution de $\mathcal{M}'$ sur $\mathbf{G}$ permet de construire un graphe $\mathbf{H}$ tel que $\mathbf{G}$ est un revêtement simple de $\mathbf{H}$. Si $\mathbf{G}$ est un revêtement simple propre de $\mathbf{H}$, on sait d'après la Proposition 10.6 que $2|V(H)| \leq |V(G)|$. Puisque $|V(G)| \leq B < 2|V(G)|$, on sait que si $\mathbf{G}$ est un revêtement simple propre de $\mathbf{H}$, $2|V(H)| \leq B$. Au contraire, si $\mathbf{G}$ est isomorphe à $\mathbf{H}$, alors $B < 2|V(G)| = 2|V(H)|$. Par conséquent, il suffit de tester si la taille du graphe $\mathbf{H}$ construit à partir de l'étiquetage final obtenu après l'exécution de $\mathcal{M}'$ sur $\mathbf{G}$ vérifie l'inégalité $2|V(H)| \leq B$. Si l'inégalité est vérifiée, alors le graphe $\mathbf{G}$ n'est pas minimal pour les revêtements simples, et dans le cas contraire, le graphe $\mathbf{H}$ est isomorphe à $\mathbf{G}$ et on peut donc résoudre l'élection et le nommage avec détection de la terminaison sur $\mathbf{G}$. □

On remarque que si on ne connait qu'une borne sur la taille de $\mathbf{G}$ et que cette borne n'est pas serrée, on ne peut pas trouver d'algorithme utilisant des calculs locaux sur les étoiles fermées qui permet de résoudre le problème de l'élection sur $\mathbf{G}$ ou de détecter que $\mathbf{G}$ n'est pas minimal pour les revêtements simples. En effet, il suffit de considérer deux graphes $\mathbf{G}$ et $\mathbf{H}$ de telle sorte que $\mathbf{G}$ soit un revêtement simple à deux feuillets de $\mathbf{H}$ (ce qui est toujours possible d'après le théorème de Reidemeister), comme par exemple les graphes de la figure 10.1. Si la borne fournie à l'algorithme est $|V(G)|$ et qu'elle n'est pas serrée, l'algorithme doit résoudre le problème de l'élection sur $\mathbf{H}$ et par conséquent, d'après le lemme 10.11, il existe une exécution de l'algorithme sur $\mathbf{G}$ telle que dans la configuration finale, l'étiquette ÉLU apparaît deux fois dans $\mathbf{G}$.

# Chapitre 11

# Détection de propriétés stables

Définir des points de reprise d'un système distribué, savoir si le système est au repos ou bien s'il y a un interblocage sont des questions clés dans la gestion des systèmes distribués. Ce chapitre donne quelques exemples de réponses à travers la détection de la terminaison d'une tâche ou bien le calcul d'un état global.

Les algorithmes distribués que l'on ausculte (observe), pour détecter certaines propriétés stables, vérifient :

1. un processus qui reçoit un message est actif : s'il était actif, il reste actif et s'il était passif il devient actif ;
2. un processus actif qui ne reçoit pas de message devient au bout d'un certain temps passif ;
3. un processus actif devient passif à travers un événement interne.

Par ailleurs, les algorithmes qui auscultent (observent) un système distribué doivent vérifier les propriétés suivantes :

1. ils ne doivent pas interférer avec le système,
2. dans le cas de la détection de la terminaison : si l'exécution de l'algorithme ausculté est terminée cela doit être détecté au bout d'un temps fini après la fin de l'exécution de l'algorithme et si la terminaison est détectée alors effectivement l'exécution de l'algorithme doit être finie.

## 11.1 Propriétés stables dans un système distribué

Soit **G** un graphe. Soit $\mathcal{D}$ un algorithme distribué à base de messages et soit $\mathcal{E}$ une exécution de $\mathcal{D}$ sur **G**.

Une propriété $P$ des configurations de $\mathcal{E}$ est stable si :

si $P$ est vrai pour une configuration $(\texttt{state}, M)$ alors $P$ est vrai pour toute configuration issue de $(\texttt{state}, M)$.

Parmi les propriétés stables des systèmes distribués détectées à l'aide d'un état global on considère : la terminaison, l'interblocage, la perte de jetons et le ramasse-miettes.

### Terminaison

Une exécution $\mathcal{E}$ est terminée si et seulement si tous les processus sont passifs et tous les canaux de communication sont vides.

**Interblocage**

Un interblocage apparaît dans un système distribué s'il existe un cycle de processus chacun étant en attente d'un message de son prédécesseur et aucun message n'est en transit entre eux.

Il peut être détecté en construisant le graphe de dépendance ; les sommets sont les processus et il existe un arc allant du processus $p$ vers le processus $q$ si :

— $p$ est bloqué,

— il n'y a pas de message dans le canal allant de $p$ à $q$,

— $p$ attend un message de $q$.

Il y a un interblocage si et seulement si ce graphe contient un cycle.

**Perte de jetons**

Certains systèmes distribués utilisent des jetons qui se déplacent de processeurs en processeurs. Des jetons peuvent disparaître ou être consommés. Ainsi les propriétés "il y a exactement $k$ jetons" ou bien "il y a au plus $k$ jetons" peuvent être des propriétés stables.

**Ramasse-miettes**

Le but est de décider si un objet est utile (Schiper and Sandoz). Un système est composé d'un ensemble d'objets $O$, et d'un sous-ensemble fixe $Root \subseteq O$, dont les éléments sont des objets sources. Les sources sont invoquées par certains processus. L'invocation de l'objet $o_i$ entraîne l'exécution d'actions par $o_i$. L'ensemble des objets peut être vu comme un ensemble de processus échangeant des messages sur requête et les messages transportent des références.

On définit la relation de descendance $\mathcal{D}$ sur l'ensemble $O$ des objets par : $o_i \mathcal{D} o_j$ si l'objet $o_i$ fait référence à l'objet $o_j$ ou une référence à $o_j$ est en cours dans $o_i$.

Un objet $o_i$ est atteignable si $o_i \in Root$ ou si $o_i$ est descendant d'un objet atteignable. Par définition un objet est utile s'il est atteignable. Seul un objet atteignable peut envoyer des références à un autre objet. Un objet non atteignable est dit inutile et doit être supprimé.

Finalement les objets atteignables sont les sommets $o$ pour lesquels il existe un chemin d'une racine à $o$.

## 11.2 Un algorithme pour détecter une propriété locale stable

On considère un algorithme distribué qui termine dès que chaque processus a atteint sa propre condition locale de terminaison ; on suppose que chaque processus peut localement savoir s'il a terminé sa tâche, par exemple il doit résoudre localement une équation et peut déterminer un instant où il a réalisé cette tâche.

L'algorithme que l'on présente ici, noté SSP, permet à chaque processus de détecter un instant où tous les processus ont réalisé leur propre tâche si chaque processus a la connaissance du diamètre du réseau (ou bien d'un majorant du diamètre comme par exemple la taille du réseau). Cet algorithme a été introduit dans [SSP85].

Soit $G$ un graphe, à chaque sommet $v$ de $G$ est associé un prédicat $P(v)$ et un entier $a(v)$. Initialement $P(v)$ est faux et $a(v)$ est égal à $-1$.

Les transformations de $a(v)$ sont définies par les règles suivantes.

Chaque calcul local du sommet $v_0$ agit sur $a(v_0)$ ; la nouvelle valeur de $a(v_0)$ dépend des valeurs associées aux sommets de $B(v_0)$. Plus précisément, soit $v_0$ un sommet et soit $\{v_1, ..., v_d\}$ l'ensemble des voisins de $v_0$.

— Si $P(v_0) = faux$ alors $a(v_0) = -1$;
— Si $P(v_0) = vrai$ alors $a(v_0) = 1 + Min\{a(v_k) \mid 0 \leq k \leq d\}$.

On fait l'hypothèse suivante : pour chaque sommet $v$ la valeur de $P(v)$ devient vrai au bout d'un temps fini et reste vrai jusqu'à la fin de l'exécution de l'algorithme.

Nous donnons des propriétés de l'algorithme de SSP sous cette hypothèse.

Nous utilisons les notations suivantes. Soit $(\mathbf{G}_i)_{0\leq i}$ une chaîne de calcul associée à l'exécution de SSP. On note $a_i(v)$ (resp. $P_i(v)$) l'entier (resp. le booléen) associé au sommet $v$ de $\mathbf{G}_i$.

Ce premier lemme énonce des propriétés de base.

**Lemme 11.1** *Soit $(\mathbf{G}_i)_{0\leq i}$ une chaîne de calcul associée à une exécution de l'algorithme de SSP. On a :*

1. $a_i(v) \leq i$,
2. $a_{i-1}(v) \leq a_i(v)$,
3. $w \in B(v) \Rightarrow |a_i(v) - a_i(w)| \leq 1$,
4. *si* $h = a_i(v) \geq 0$ *alors* $\forall w \in V(G)\ d(v,w) \leq h \Rightarrow a_{i-h}(w) \geq 0$.

Des propriétés *2* et *4* de ce lemme, on déduit immédiatement que le prédicat $P$ est vrai pour tout sommet de la boule de centre $v$ et de rayon $a_i(v)$, i.e., :

**Proposition 11.2** *Soit $(\mathbf{G}_i)_{0\leq i\leq n}$ une chaîne de calcul associée à une exécution de SSP ; soit $v$ un sommet de $G$; si $h = a_i(v) \geq o$ alors :*

$$\forall w \in V(G) \qquad d(v,w) \leq h \Rightarrow a_i(w) \geq 0.$$

Finalement on en déduit que si chaque sommet connaît le diamètre $D$ du réseau alors chaque sommet $v$ peut détecter un instant où tous les sommets ont exécuté leur tâche : il suffit que $a(v) \geq D$.

**Remarque 11.3** *la fonction a vérifie : $a(v)$ peut croître indéfiniment dès que $a(v)$ a atteint la valeur de $D$.*

## 11.3 L'algorithme de détection de la terminaison globale de Dijkstra-Scholten

On veut détecter la terminaison globale d'un algorithme distribué $\mathcal{A}$ qui démarre à partir d'un unique sommet $p_0$. L'algorithme qui observe le déroulement de $\mathcal{A}$, pour en détecter sa terminaison, maintient un arbre dont la racine est $p_0$ et qui contient tous les sommets actifs ou qui ont un descendant actif. Cet arbre est dynamique : il peut croître puis décroître et de nouveau se développer d'une manière différente. Un sommet $v$ devient feuille de cet arbre s'il est en dehors de l'arbre et s'il reçoit un message d'un sommet $u$, qui, de fait, est dans l'arbre. Le sommet $u$ devient le père de $v$. Dans ce cas, $v$ n'accuse pas réception du message qui l'a fait devenir feuille (il le fera plus tard lorsqu'il quittera l'arbre).

Un sommet quitte cet arbre lorqu'il devient passif et qu'il est une feuille. C'est à ce moment qu'il accuse réception du message qui l'a activé. Un sommet qui est dans l'arbre et qui reçoit un message accuse réception de ce message.

La terminaison de $\mathcal{A}$ sera détectée par $p_0$ quand l'arbre sera réduit au sommet $p_0$ et que $p_0$ sera lui même passif.

**Algorithme 11:** L'algorithme de Dijkstra-Scholten.

**$S_p$** : $\{state_p = actif\}$
**début**
  **envoyer**$< \mathbf{mes}, q >$ ;
  (*Ce message est envoyé par l'algorithme dont on veut détecter la terminaison.*)
  $sc_p := sc_p + 1$
**fin**

**$R_p$** : {Un message $< \mathbf{mes}, q >$ arrive sur $p$}
**début**
  **recevoir**$< \mathbf{mes}, q >$ ;
  (*Ce message fait partie de l'algorithme dont on veut détecter la terminaison.*)
  $state_p := actif$;
  **si** $father_p = udef$ **alors**
    $father_p := q$
  **sinon**
    **envoyer** $< \mathbf{sig}, q >$ à $q$
**fin**

**$I_p$** : $\{state_p = actif\}$
**début**
  $state_p := passif$;
  **si** $sc_p = 0$ **alors**
    **si** $father_p = p$ **alors**
      **c'est fini**
    **sinon**
      **envoyer** $< \mathbf{sig}, father_p >$ à $father_p$ ;
      $father_p := udef$
**fin**

**$A_p$** : {Un message $< \mathbf{sig}, p >$ arrive sur le sommet $p$ }
**début**
  **recevoir**$< \mathbf{sig}, q >$ ;
  $sc_p := sc_p - 1$ ;
  **si** $sc_p = 0$ *et* $state_p = passif$ **alors**
    **si** $father_p = p$ **alors**
      **c'est fini**
    **sinon**
      **envoyer** $< \mathbf{sig}, father_p >$ à $father_p$ ;
      $father_p := udef$
**fin**

## 11.4 L'algorithme de détection de la terminaison de Dijkstra-Feijen-Van Gasteren

On suppose que le réseau est un anneau orienté dont les processeurs sont numérotés de 0 à $n-1$.

Chaque processus $p$ dispose de 2 variables : $state_p$ et $color_p$. La variable $state_p$ est $actif$ ou $passif$. La variable $color_p$ est $blanc$ ou $noir$. On suppose que les communications entre les processus se font par échange de messages en mode synchrone.

Le sommet $p_0$ est distingué, il active la détection de la terminaison en générant un jeton blanc qu'il envoie au sommet $p_{n-1}$. Un sommet qui détient le jeton l'envoie à son prédécesseur dès qu'il devient passif.

Les sommets et le jeton sont blanc ou noir. Un sommet qui envoie un message est noir. Lorsque le jeton quitte un sommet celui-ci est blanc. Le jeton qui traverse un sommet blanc garde sa couleur. S'il traverse un sommet noir sa couleur est noir. Le sommet $p_0$ détecte la fin de l'exécution de l'algorithme dès qu'il reçoit le jeton blanc en étant blanc et passif.

**Algorithme 12:** L'algorithme de Dijkstra-Feijen-Van Gasteren.

```
C_pq : { state_p = actif}
début
    (* p envoie un message à q (ce message appartient à l'algorithme dont on veut détecter la terminaison) *)
    color_p := noir ;
    state_q := actif
fin
I_p : { state_p = actif}
début
    state_p := passif
fin
Init_p : {Initialisation de l'algorithme pour la détection de la terminaison par p_0}
début
    Envoyer< tok, blanc > à p_{n-1}
fin
T_p : {Le processus p a le message < tok, c >; dès qu'il devient passif il : }
début
    si p = p_0 alors
        si (c = blanc et color_p = blanc) alors
            C'est fini
        sinon
            Envoyer< tok, blanc > à p_{n-1}
    sinon
        si color_p = blanc alors
            Envoyer< tok, c > à Next_p
        sinon
            Envoyer< tok, noir > à Next_p
    color_p = blanc
fin
```

La validité de cet algorithme repose sur la remarque suivante :

**Remarque 11.4** *Supposons qu'au moins un sommet observé passif par le jeton devienne actif. Considérons la première fois où une telle situation se produit, et soit $v$ le sommet qui devient actif. Nécessairement $v$ devient actif en recevant un message d'un sommet actif, noté $w$, qui n'a pas encore était dépassé par le jeton. Par conséquent $w$ devient noir et le jeton en quittant ou en traversant le sommet $w$ deviendra noir.*

## 11.5 L'algorithme de calcul d'un état global de Chandy-Lamport avec un sommet distingué

Étant donné un instant $t_0$, l'état global d'un système distribué à l'instant $t_0$ est constitué de l'ensemble des états de chaque composante du système. Dans le cas où le système est constitué de processeurs et de canaux de communication, l'état du système est défini par l'état de chaque processeur et l'état de chaque canal. Cela correspond à une photographie instantanée à l'instant $t_0$. Par définition, l'état local d'un sommet est constitué de l'état du sommet et de l'état de chaque canal d'entrée. L'état global du système est constitué de la collection des états locaux.

La connaissance d'un état global permet d'avoir des points de reprise en cas de défaillance du système ; elle peut permettre également de détecter la terminaison d'un algorithme, de détecter un inter-blocage, la perte d'un jeton circulant ou bien de récupérer de la mémoire inutile. Enfin, cette connaisance peut aider à la mise au point d'un algorithme distribué.

Dans un système distribué sans contrôle central une telle photographie n'est pas possible.

On suppose qu'un seul processus, noté $r$ lance le calcul d'un état global. L'algorithme de Chandy-Lamport (Algorithme 13) calcule un état "virtuel" du système dans lequel le système peut être ou bien atteignable à partir de l'état où est le système lors du déclenchement du calcul de l'état global. Le booléen *marque* est initalisé à $faux$. On suppose que l'état du processus $p$ est mémorisé à tout moment dans la variable $state_p$.

**Algorithme 13:** L'algorithme de Chandy-Lamport.

```
Init_p : {Lancement du calcul de l'état global par le processus r}
début
    Enregistrer(state_r) ;
    marque := vrai;
    Pour chaque port q, r enregistre les messages qui arrivent par q;
    Envoyer< marqueur > par chaque port de sortie
fin
R_p : {Un marqueur arrive sur p par le port q_0}
début
    recevoir< marqueur >;
    si non marque alors
        marque := vrai;
        Enregistrer(state_p) ;
        Pour chaque port q, différent de q_0, p enregistre les messages qui arrivent par q;
        (*Le canal associé au port d'entrée q_0 sera vide pour le calcul de cet état global.*)
        Envoyer< marqueur > par chaque port de sortie
    sinon
        Ce marqueur indique la fin de l'enregistrement sur p des messages reçus par le port q_0; p connaît maintenant
        l'état du canal correspondant.
fin
```

**Remarque 11.5** *L'instant $t_0$ est l'instant où le sommet distingué lance le calcul de l'état global. Pour un sommet quelconque, l'instant $t_0$ est l'instant où il reçoit pour la première fois un marqueur. Étant donné un canal, les messages qui se trouvent dans le canal à l'instant $t_0$ sont les messages qui ont été envoyés dans ce canal avant $t_0$ et reçus après $t_0$, c'est à dire ils sont reçus par le sommet alors que ce sommet a déjà reçu un marqueur et sont suivis dans ce canal par un marqueur.*

Si l'algorithme est initialisé par un seul sommet $r$ alors le calcul de l'état global s'effectue par la remontée des états de chaque sommet avec les états de ses canaux incidents au sommet $r$ qui peut

alors reconstituer l'état global du réseau. Cette remontée peut se faire le long de l'arbre associé à ce calcul.

Si l'on veut autoriser plusieurs sommets à déclencher simultanément le calcul d'un état global se pose alors le problème de la reconstruction du réseau par au moins un sommet. Une solution possible passe par l'hypothèse que chaque sommet a un identificateur (une couleur) unique.

**Remarque 11.6** *Le calcul local d'un sommet consiste en la mémorisation de son état et le calcul de l'état de chaque canal incident, ainsi chaque sommet peut savoir quand il dispose effectivement de ces informations. Si chaque sommet a la connaissance d'une borne du diamètre du réseau alors en utilisant l'algorithme SSP chaque sommet peut savoir quand les informations locales sont effectivement calculées sur l'ensemble du réseau, en associant un numéro à chaque calcul de l'état global.*

## 11.6 L'algorithme de Chandy-Lamport : cas général

On présente dans cette section l'algorithme de Chandy-Lamport (Algorithme 14) et son utilisation en supposant que : le réseau est anonyme, plusieurs processus peuvent prendre simultanément l'initiative du lancement de l'algorithme et chaque processus connaît une borne supérieure du diamètre du réseau. L'état local ou snapshot local d'un processus est constitué de l'état du processus et des états des canaux d'entrée du processus.

Chaque processus $p$ dispose des variables suivantes :

— un booléen $taken_p$ initialisé à faux, il indique si le processus a enregistré son état ;

— un booléen $local\text{-}snapshot_p$ initialisé à $faux$, il indique si le processus $p$ a mémorisé son état et les états des canaux d'entrée ;

— un multi-ensemble de messages $M_{p,i}$, initialement $M_{p,i} = \emptyset$, pour chaque canal d'entrée de $p$.

On suppose que l'algorithme 14 est activé par au moins un processus qui : sauvegarde son état, envoie un marqueur à travers chaque port de sortie et qui, pour chaque port d'entrée, mémorise les messages qui arrivent par ce port jusqu'à l'arrivée d'un marqueur par ce port.

Quand un processeur reçoit pour la première fois un marqueur, il exécute les mêmes instructions qu'un initiateur ; le canal par lequel arrive le marqueur est considéré comme vide.

**Lemme 11.7** *Une fois que l'algorithme de Chandy-Lamport a été activé par au moins un processus, au bout d'un temps fini chaque processus p a calculé son snapshot local (local-snapshot$_p$ = true).*

Une fois le calcul du snapshot local effectué par tous les processus du réseau, la connaissance du snapshot est distribué sur l'ensemble du réseau ; se pose alors la question de l'exploitation de cette connaissance.

Une première réponse est le calcul de l'état global (une carte du réseau étiquetée par les processus).

## 11.7 Détection de la terminaison de l'algorithme de Chandy-Lamport

L'algorithme de Szymanski, Shy, and Prywes permet de détecter la terminaison dés lors que chaque processus connaît une borne supérieure du diamètre.

**Algorithme 14:** L'algorithme de Chandy-Lamport.

**I**$nit\text{-}CL_p$ : {Pour activer l'algorithme par au moins un processus $p$ tel que $taken_p = false$}
**début**
  **record**(`state`($p$)) ;
  $taken_p := true$;
  **Envoyer**$< mkr >$ à chaque voisin de $p$ ;
  Pour chaque port $i$ le processus $p$ enregistre les messages qui arrivent via $i$
**fin**

**R**-$CL_p$ : {Un marqueur arrive sur $p$ via le port $j$}
**début**
  **Recevoir**$< mkr >$;
  **Marquer** le port $j$;
  **si** *not* $taken_p$ **alors**
    $taken_p := true$;
    **Enregistrer**(`state`($p$)) ;
    **Envoyer**$< mkr >$ via chaque port de sortie ;
    Pour chaque $i \neq j$ le processus $p$ enregistre les messages qui arrivent via $i$ dans $M_{p,i}$
  **sinon**
    Le processus $p$ arrête d'enregistrer les messages pour le canal $j$ ;
    **Enregistrer**($M_{p,j}$)
  **si** *$p$ a reçu un message à travers chaque canal d'entrée* **alors**
    $local\text{-}snapshot_p := true$
**fin**

### 11.7.1 L'algorithme SSP

Cette section présente l'algorithme de SSP avec des messages : Algorithme 15.

Soit $G$ un graphe, à chaque processus $p$ est associé un prédicat $P(p)$ et un entier $a(p)$. Initialement $P(p)$ est faux et $a(p)$ est égal à $-1$.

Les transformations de $a(p)$ sont définies par les règles suivantes.

Soit $p_0$ un processus et soit $\{p_1, ..., p_d\}$ les processus voisins de $p_0$.

— Si $P(p_0) = false$ alors $a(p_0) = -1$;

— Si $P(p_0) = true$ alors $a(p_0) = 1 + Min\{a(p_k) \mid 1 \leq k \leq d\}$.

On suppose que pour chaque processus $p$, $P(p)$ devient vrai au bout d'un certain temps.

Chaque processus $p$ dispose de 2 variables :

— $a(p) \in \mathbb{Z}$ est un compteur, initialement $a(p) = -1$, $a(p)$ représente la distance minimale jusqu'où les processus ont la valeur du prédicat à vrai.

— $A(p)$ mémorise les informations que $p$ a sur la valeur de $a$ pour ses voisins. Initialement, $A(p) = \{(i, -1) \mid i \in [1, \deg_G(p)]\}$.

---

**Algorithme 15:** L'algorithme SSP.

---

**I***nit-SSP*$_p$ : {Pour activer l'algorithme de détection de la terminaison sur un processus $p$ tel que $P(p) = true$ }

**début**

  $a(p) := 0$;

  $m := Min\{x \mid (i, x) \in A(p)\}$;

  **si** $m \geq a(p)$ **alors**

    $a(p) := m + 1$ ;

  **Envoyer**$< a(p) >$ à chaque voisin de $p$

**fin**

**R**-*SSP*$_p$ : {Un entier $< \alpha >$ parvient à $p$ via le port $j$}

**début**

  **recevoir**$< \alpha >$;

  $A(p) := (A(p) \setminus \{(j, x)\}) \cup \{(j, \alpha)\}$;

  $m := Min\{x \mid (i, x) \in A(p)\}$;

  **si** $(m \geq a(p)$ *et* $P(p) = true)$ **alors**

    $a(p) := m + 1$;

  **si** $a(p) \geq \beta$ **alors**

    $p$ détecte la terminaison de l'algorithme de Chandy-Lamport pour tous les processus

  **sinon**

    **Envoyer**$< a(p) >$ via chaque port

**fin**

---

On considère une exécution $\mathcal{E}$ de SSP sur le graphe **G** et soit $(a_i(p))_{i \geq 0}$ la suite des valeurs de $a(p)$ pour $\mathcal{E}$. (Soit $p$ un processus, $B(p)$ désigne les voisins de $p$.)

**Lemme 11.8** *On a :*

1. $a_i(p) \leq i$,
2. $a_{i-1}(p) \leq a_i(p)$,
3. $q \in B(p) \Rightarrow |a_i(p) - a_i(q)| \leq 1$,
4. *Si* $h = a_i(p) \geq 0$ *alors* $\forall q \in V(G)\ d(p,q) \leq h \Rightarrow a_{i-h}(q) \geq 0$.

**Proposition 11.9** *Soit $p$ un processus de $G$, on suppose que $h = a_i(p) \geq 0$. Alors :*

$$\forall q \in V(G) \quad d(p,q) \leq h \Rightarrow a_i(q) \geq 0.$$

Donc un processus $p$ tel que $a(p)$ est supérieur ou égal au diamètre du graphe sait que tous les prédicats sont à vrai.

### 11.7.2 Un algorithme pour détecter la terminaison des calculs des snapshots locaux

Dans cette section, on compose l'algorithme de Chandy-Lamport et l'algorithme SSP : ainsi chaque processus détectera la fin de l'exécution de Chandy-Lamport par l'ensemble des processus.

L'application de SSP nécessite 3 items supplémentaires :

— $a(p) \in \mathbb{Z}$ est un compteur, initialement $a(p) = -1$ ; $a(p)$ représente la distance à laquelle les processus ont achevé le calcul de leur snapshot local ;

— $A(p) \in \mathcal{P}_{\text{fin}}(\mathbb{N} \times \mathbb{Z})$ mémorise la valeur de $a(q)$ pour chaque voisin $q$. Initialement, $A(p) = \{(i,-1) \mid i \in [1, \deg_G(p)]\}$ ;

— $\textit{snapshot-number}_p$ indique le numéro du snapshot (initialement, $\textit{snapshot-number}_p = 0$) ; cette variable n'est pas nécessiare dans cette section, on l'utilisera plus tard.

Si un processus a terminé le calcul de son snapshot local alors il met la valeur de $a(p)$ à 0 et il informe ses voisins. Quand un processus reçoit une valeur $a(q)$ pour un voisin $q$ via le port $i$ alors il remplace $(i,x)$ par $(i,a(q))$ dans $A(p)$. Puis, $p$ calcule $a(p) = 1 + Min\{x \mid (i,x) \in A(p)\}$.

Un processus $p$ sait que tous les processus ont terminé le calcul des snapshots locaux dès que $a(p) \geq \beta$ (où $\beta$ est une borne supérieure du diamètre du réseau). La variable *snapshot*, initialisée à faux, indique si le processus sait si tous les processus ont terminé le calcul des snapshots locaux.

On a :

**Proposition 11.10** *Soit $(G,\lambda)$ un réseau. Soit $\beta$ une borne supérieure du diamètre de $G$ connue par chaque processus. Une fois l'algorithme de Chandy-Lamport activé, l'algorithme 16 permet à chaque processus de détecter la fin du calcul des snapshots locaux.*

## 11.8 Deux applications : "Point de contrôle et rétablissement d'une configuration" et "Détection de la terminaison"

Cette section présente 2 applications de l'algorithme de Chandy-Lamport et de l'algorithme 16 dans un contexte anonyme autorisant plusieurs initiateurs en supposant que les processus connaissent un majorant du diamètre du réseau :

1. pour calculer des configurations à partir desquelles on peut relancer le réseau en cas de défaillance,
2. pour détecter la terminaison de l'exécution d'un algorithme distribué sur un réseau.

**Algorithme 16:** Détection de la terminaison de l'algorithme de Chandy-Lamport.

**I**$nit\text{-}SSP_p$ : {Pour activer la détection de la terminaison sur le processus $p$ tel que $local\text{-}snapshot_p = true$, $a(p) = -1$ et $snapshot = false$}
**début**
  $a(p) := 0$;
  $m := Min\{x \mid (i,x) \in A(p)\}$;
  **si** $m \geq a(p)$ **alors**
    $a(p) := m + 1$ ;
  **Envoyer**$< a(p) >$ à chaque voisin de $p$
**fin**

**R**-$SSP_p$ : {Un entier $< \alpha >$ est parvenu à $p$ via le port $j$}
**début**
  **Recevoir**$< \alpha >$;
  $A(p) := (A(p) \setminus \{(j,x)\}) \cup \{(j,\alpha)\}$;
  $m := Min\{x \mid (i,x) \in A(p)\}$;
  **si** $(m \geq a(p)$ *et* $local\text{-}snapshot_p = true)$ **alors**
    $a(p) := m + 1$;
  **si** $a(p) \geq \beta$ **alors**
    $snapshot\text{-}number_p := snapshot\text{-}number_p + 1$;
    $snapshot_p := true$
  **sinon**
    **Envoyer**$< a(p) >$ via chaque port
**fin**

### 11.8.1 Point de contrôle et restitution

Une solution est obtenue en répétant les actions suivantes :

1. au moins un processus active l'algorithme de Chandy-Lamport (Algorithme 14) ;
2. chaque processus $p$ détecte un instant où le calcul de son snapshot local est terminé : $local\text{-}snapshot_p = true$ ;
3. chaque processus détecte un instant où tous les snapshots locaux sont calculés : $snapshot_p = true$ (Algorithme 16) ;
4. un nouveau numéro (obtenu en ajoutant 1 au compteur $snapshot\text{-}number_p$, initialement $snapshot\text{-}number_p = 0$) est associé à ce snapshot et chaque processus donne ce numéro à son snapshot local. Chaque processus $p$ sauvegarde son snapshot local associé au numéro $snapshot\text{-}number_p$. Cela permet un redémarrage en cas de panne.
5. Finalement, les variables de l'algorithme 14 et de l'algorithme 16 sont réinitialisées à la fin.

**Remarque 11.11** *On suppose que le snapshot initial, dont le numéro est* 0*, correspond à l'initialisation du système, i.e., l'état de chaque processus est son état initial et chaque canal est vide.*

**Remarque 11.12** *La solution présentée est totalement distribuée, plusieurs processus peuvent prendre simultanément l'initiative du calcul du snapshot.*

**Remarque 11.13** *Il n'y a pas de chevauchement entre des appels consécutifs des algorithmes 14 et 16 grâce aux variables* $taken_p$ *qui ne sont réinitialisées qu'à la fin de l'algorithme 17 et grâce au fait que les canaux sont FIFO.*

### 11.8.2 Détection de la terminaison d'un algorithme distribué.

Soit $\mathcal{A}$ un algorithme distribué. Soit $\mathcal{E}$ une exécution de $\mathcal{A}$. L'objectif est de permettre à chaque processus de détecter la fin de $\mathcal{E}$.

Une exécution de $\mathcal{E}$ est terminée si et seulement si tous les processus sont passifs et tous les canaux sont vides. Ainsi pour détecter la fin de l'exécution de $\mathcal{E}$, il suffit de lancer de temps en temps un calcul de snapshot et de vérifier la propriété ci-dessus avec une activation de SSP. Si les variables d'un processus indiquent que l'exécution n'est pas finie alors $q$ émet un signal à travers le réseau pour en informer chaque élément.

Les principales idées sont :

1. au moins un processus active le calcul d'un snapshot (Algorithme 14) ;
2. si le snapshot local d'un processus $p$ est tel que l'état de $p$ est passif et tous les canaux d'entrée sont vides et tous les snapshots locaux sont calculés (Algorithme 16) alors $p$ active une occurrence de SSP (Algorithme 16) pour tester si l'exécution de $\mathcal{A}$ est terminée ;
3. si le snapshot local d'un processus $p$ est tel que l'état de $p$ est actif ou bien au moins un canal d'entrée n'est pas vide alors $p$ envoie un signal pour informer tous les processus que l'exécution de l'algorithme n'est pas finie et qu'il faut au moins un autre calcul d'état global. Les variables de l'algorithme 14 et de l'algorithme 16 sont réinitialisées.

On a besoin d'une autre variable $Term\text{-}Detection_p$ sur chaque processus $p$, initialement $Term\text{-}Detection_p = false$, cette variable indique si une tentative de détection de terminaison est en cours.

Les variables $aTD_p$ et $ATD_p$ sont utilisées par SSP.

**Algorithme 17:** Point de contrôle.

**I***nit-Checkpoint* : {Pour activer un point de contrôle par un processus $p$ tel que $snapshot = true$}
**début**
  **Memoriser** les informations numérotées avec *snapshot-number*;
  $next_p := snapshot\text{-}number + 1$;
  $snapshot_p := false$;
  $local\text{-}snapshot_p := false$;
  $a(p) := -1$;
  $A(p) = \{(i, -1) \mid i \in [1, \deg_G(p)]\}$;
  **Demarquer** chaque port de $p$;
  **Envoyer** $<$ *new-snapshot*,$next_p >$ via chaque port de $p$;
  $taken_p := false$
**fin**

**R**-$NewS_p$ : {le message *new-snapshot* est parvenu à $p$ via le port $j$}
**début**
  **Recevoir**$<$ *new-snapshot*, $k >$;
  **si** $k > snapshot\text{-}number_p$ **alors**
    **Memoriser** les informations concernant le snapshot local ayant le numéro *snapshot-number*;
    $next_p := snapshot\text{-}number + 1$;
    $snapshot_p := false$;
    $local\text{-}snapshot_p := false$;
    $a(p) := -1$;
    $A(p) = \{(i, -1) \mid i \in [1, \deg_G(p)]\}$;
    **Demarquer** chaque port de $p$;
    **Envoyer** $<$ *new-snapshot*,$next_p >$ via chaque port de $p$;
    $taken_p := false$
**fin**

**Algorithme 18:** Un appel pour l'algorithme de détection de la terminaison.

**I**$nit\text{-}Term_p$ : {Pour activer la détection de la terminaison par $p$ tel que : $snapshot_p = true$, $p$ est passif, les canaux d'entrée de $p$ sont vides et $Term\text{-}Detection_p = false$.}
**début**
  $Term\text{-}Detection_p := true$; $aTD_p := 0$; $ATD_p := \{(i,-1) \mid i \in [1, \deg_G(p)]\}$ ;
  **Envoyer**$< TD, aTD_p >$ à chaque voisin de $p$
**fin**
**R**$TD_p$ : {Un message $< TD, \alpha >$ parvient à $p$ via le port $j$}
**début**
  **Recevoir**$< TD, \alpha >$;
  **si** $Term\text{-}Detection_p = true$ **alors**
    $ATD_p := (ATD_p \setminus \{(j,x)\}) \cup \{(j,\alpha)\}$; $m := Min\{x \mid (i,x) \in ATD_p\}$ ;
    **si** $m \geq aTD_p$ **alors**
      $aTD_p := m + 1$;
    **si** $aTD_p \geq \beta$ **alors**
      TERM :=true
    **sinon**
      **Envoyer**$< TD, aTD_p >$ via chaque port de $p$ ;
  **sinon**
    **si** $Term\text{-}Detection_p = false$ *et $p$ est passif, et les canaux d'entrée de $p$ sont vides et* $local\text{-}snapshot_p = true$ **alors**
      $Term\text{-}Detection_p := true$; $aTD_p := 0$;
      $ATD_p := \{(i,-1) \mid i \in [1, \deg_G(p)]\}$ ; $ATD_p := (ATD_p \setminus \{(j,x)\}) \cup \{(j,\alpha)\}$;
      **Envoyer**$< TD, aTD_p >$ via chaque port de $p$ ;
**fin**
**I**$nit\text{-}Term_p$ : {Pour envoyer un signal de non terminaison à partir d'un processus $p$ tel que : $snapshot_p = true$, $p$ n'est pas passif ou au moins un canal d'entrée de $p$ n'est pas vide.}
**début**
  $next_p := snapshot\text{-}number_p + 1$;
  **Envoyer**$< not\text{-}terminated, next_p >$ à chaque voisin de $p$;
  $snapshot_p := false$; $local\text{-}snapshot_p := false$; $a(p) := -1$;
  $A(p) = \{(i,-1) \mid i \in [1, \deg_G(p)]\}$; **Demarquer** chaque port de $p$;
  **Envoyer** $< new\text{-}snapshot, next_p >$ via chaque port de $p$; $taken_p := false$
**fin**
**R**$T_p$ : {Un message $< not\text{-}terminated, \alpha >$ est parvenu à $p$ via le port $j$}
**début**
  **Recevoir**$< not\text{-}terminated, \alpha >$;
  **si** $snapshot\text{-}number_p < \alpha$ **alors**
    $next_p := snapshot\text{-}number_p + 1$;
    **Envoyer**$< not\text{-}terminated, next_p >$ à chaque voisin de $p$;
    $snapshot_p := false$; $local\text{-}snapshot_p := false$; $a(p) := -1$;
    $A(p) = \{(i,-1) \mid i \in [1, \deg_G(p)]\}$; **demarquer** chaque port de $p$;
    **Envoyer** $< new\text{-}snapshot, next_p >$ via chaque port de $p$; $taken_p := false$
**fin**

**Remarque 11.14** *Comme pour les algorithmes précédents, il n'y pas de chevauchements pour les algorithmes 14 et 16 grâce au booléen* $taken_p$ *qui n'est réinitialisé qu'après l'algorithme 18 et grâce à la propriété FIFO des canaux.*

# Chapitre 12

# Stabilisation

Différents phénomènes peuvent apparaître lors de l'exécution d'un algorithme ; en particulier peuvent se produire des changements inopinés dans le réseau (ajout ou disparition d'un sommet ou d'un lien) ou bien encore des altérations de messages ou de mémoires. Un algorithme est dit auto-stabilisant s'il se termine correctement en dépit de l'apparition de ces phénomènes. Un corollaire immédiat et très important de cette propriété est qu'un algorithme auto-stabilisant ne nécessite pas d'initialisation particulière.

## 12.1 3-Coloration d'un anneau à l'aide de réécritures d'étoiles

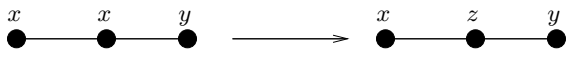
Considérons un anneau ayant au moins 3 sommets. Le problème de la 3-coloration consiste à attribuer aux sommets de l'anneau des couleurs choisies parmi 3 couleurs données de telle sorte que 2 sommets voisins ont des couleurs différentes. En algorithmique distribuée, la coloration des sommets est utilisée en particulier pour l'allocation de ressources.

Nous donnons un système de réécriture d'étoiles qui 3-colorie un anneau à partir d'une configuration initiale quelconque. Soit $\mathcal{R}_7 = (\mathcal{L}_7, \mathcal{I}_7, P_7)$ le système de réécriture défini par : $\mathcal{L}_7 = \{x, y, z\}$, $\mathcal{I}_7 = \{x, y, z\}$ et $P_7 = \{R_1, R_2\}$ où $R_1$, $R_2$ sont les règles suivantes :

$R_1$ :


; $y \neq x$

$R_2$ :




; $x \neq y$, $z \neq x$, $z \neq y$

On vérifie aisément que ce système a la propriété de terminaison, qu'il 3-colorie un anneau quelconque et qu'il est auto-stabilisant.

## 12.2 Présentation de l'algorithme du jeton circulant de Dijkstra à l'aide des réécritures d'arcs

Par définition au plus un processus peut être dans une section critique à un instant donné. Pour résoudre ce problème d'exclusion mutuelle, on peut utiliser un jeton qui donne le privilège d'entrer dans la section critique avec les règles du jeu suivantes :

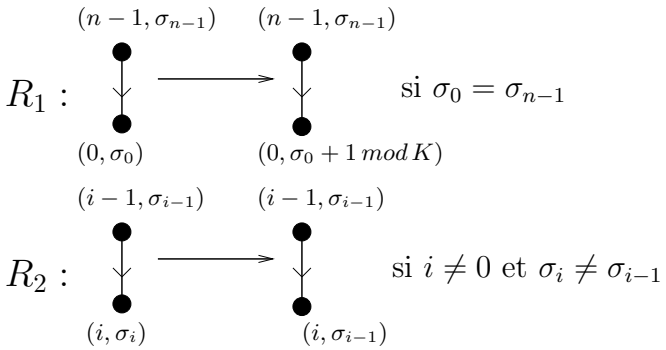
1. à tout moment au plus un processus détient le jeton,

2. tout processus détient infiniment souvent le jeton.

L'algorithme que l'on présente donne une solution logicielle auto-stabilisante à ce problème. Cette solution est logicielle dans la mesure où la pocession du jeton n'est pas la pocession d'un "message" mais cette pocession est concrétisée par une condition que doit vérifier l'état du sommet "détenteur du jeton".

On suppose que le réseau est un anneau dirigé dont les processus sont numérotés de 0 à $n-1$. À tout processus est associé un état, noté $\sigma_i$, qui est un entier appartenant à l'intervalle $0, \ldots K-1$, où $K$ est un entier strictement plus grand que $n$. Le processus $i \neq 0$ a le privilège si $\sigma_i \neq \sigma_{i-1}$. Le processus 0 a le privilège si $\sigma_0 = \sigma_{n-1}$. Un processus qui a le privilège peut changer d'état suivant les règles ci-après.

L'étiquette associée à chaque sommet de l'anneau est le couple formé du numéro $i$ du sommet et de l'état $\sigma_i$ de ce sommet.




La validité de ce système repose sur les remarques suivantes.

1. Quelques soient les valeurs des états des sommets de l'anneau une transformation peut avoir lieu.
2. Si le sommet 0 ne change pas de valeur au bout d'un temps fini le système n'évolue plus.
3. Sachant que $K > n$, au cours de l'exécution des transformations l'état du sommet 0 prendra une première fois une valeur, notée $\pi$, égale à aucun autre état de l'anneau. Pour que le sommet 0 change de nouveau de valeur il faut et il suffit que tous les sommets de l'anneau soient dans l'état $\pi$.

## 12.3 Calcul d'un arbre recouvrant

On suppose qu'il existe un sommet distingué, $v_0$, qui est la racine de l'arbre recouvrant. Chaque sommet $v$ est doté d'une variable $m(v)$ qui indique la plus courte distance à $v_0$. Le sommet $v_0$ est l'unique sommet dont la variable $m(v_0)$ est égale à 0.

L'idée de l'algorithme est la suivante : chaque sommet $v$ consulte les valeurs de ses voisins et choisit comme père un voisin $u$ qui correspond à la plus petite valeur $min = m(u)$ si $m(v) > min + 1$.

# Chapitre 13

# Synchronisation et synchroniseurs

Les algorithmes distribués sont étudiés sous différents modèles. Un critère fondamental est le caractère synchrone ou asynchrone du modèle.

Dans le modèle synchrone on suppose, a priori, qu'il y a une horloge globale et que les opérations des composantes se font simultanément à chaque top horloge. Ce modèle est idéal et cette hypothèse ne peut pas être garantie dans la plupart des systèmes.

En fait la synchronisation dans les réseaux peut être vue comme une structure qui permet de contrôler le déroulement des actions sur les différentes composantes ; cette idée peut être illustrée par les exemples suivantes :

1. Les processus exécutent les actions en rondes et à chaque ronde un processus $p$ exécute la suite de pas :
   (a) envoyer un message à chaque voisin,
   (b) recevoir un message de chaque voisin,
   (c) effectuer un calcul interne ;
2. une autre hypothèse est l'exécution des calculs du tour $r$ par un sommet après les calculs du tour $r-1$ et tous les messages envoyés au tour $r$ sont reçus avant les calculs du tour $r+1$ ;
3. si chaque processus dispose d'un compteur des actions locales, on peut demander que la différence entre deux compteurs soit au plus 1 ; plus généralement pour un entier positif $k$ donné on peut demander que la différence entre deux compteurs soit bornée par $k$ ;
4. certains algorithmes utilisent des barrières de synchronisation associées à des groupes de processus : tous les membres du groupe sont bloqués à cette barrière jusqu'à ce que tous les membres aient atteint cette barrière.

Dans le modèle asynchrone il n'y a pas d'horloge globale et chaque composante progresse à son propre rythme. Aucune hypothèse n'est faite sur les délais d'acheminement des messages.

Il existe également des modèles intermédiaires où on suppose que les vitesses d'exécution des différents processus sont liées.

## 13.1 Un premier exemple de synchroniseur

Chaque sommet $v$ dispose d'un compteur $c(v)$ dont la valeur initiale est 0. À chaque pas de calcul la valeur du compteur $c(v_0)$ dépend des valeurs des compteurs des voisins de $v_0$. Plus précisément si $c(v_0) = i$ et si pour chaque voisin $v$ de $v_0$ : $c(v) = i$ ou $c(v) = i+1$ alors la nouvelle valeur de $c(v_0)$ est $i+1$.

R: **la règle de synchronisation**
*Précondition :*
— $c(v_0) = i$,
— $\forall v \in B(v_0, 1) : c(v) = i$ ou $c(v) = i + 1$.
*Réétiquetage :*
— $c(v_0) := i + 1$.

Une simple induction sur la distance entre deux sommets montre que :

**Proposition 13.1** *Pour tous sommets : $v_1$ et $v_2$*

$$|c(v_1) - c(v_2)| \leq d(v_1, v_2).$$

**Remarque 13.2** *Cette synchronisation ne nécessite aucune connaissance sur le graphe et en particulier sa taille.*

Cette synchronisation est équivalente à celle décrite dans l'introduction où les processus fonctionnent en rondes : à chaque ronde un processus envoie un message à chaque voisin, reçoit un message de chaque voisin et effectue un calcul interne.

En fait, pour implémenter cette synchronisation il suffit de disposer d'un compteur modulo 3 : chaque processus compare la valeur de son compteur avec celle de ses voisins. Chaque processus $v_0$ détermine pour chaque voisin $v$ si $c(v_0) = c(v) - 1$, ou $c(v_0) = c(v)$, ou $c(v_0) = c(v) + 1$. Finalement la synchronisation peut être codée par :

R: **la règle de synchronisation**
*Précondition :*
— $c(v_0) = i$,
— $\forall v \in B(v_0, 1)$ $c(v) = i \bmod 3$ ou $c(v) = (i + 1) \bmod 3$.
*Réétiquetage :*
— $c(v_0) := (i + 1) \bmod 3$.

**Remarque 13.3** *Ce synchroniseur peut être implémenté par des messages : un sommet informe chacun de ses voisins à chaque réalisation d'un pas de calcul.*

## 13.2 Un synchroniseur pour un graphe dont on connaît la taille ou le diamètre

Soit $u$ un sommet, l'entier $c(u)$ désigne le numéro de la ronde associée au sommet $u$. Le diamètre du graphe $G$ est noté $D$. Un sommet $v$, dont le numéro de ronde est $c(v)$, peut démarrer la ronde suivante quand l'ensemble des sommets du réseau a atteint la ronde $c(v)$. Lorsqu'il démarre une nouvelle ronde il met à 0 la variable $a(v)$ qui va lui indiquer à quelle distance les sommets sont dans la même ronde. On obtient ainsi l'algorithme suivant :

R: **La règle d'observation**
*Précondition :*
— $\lambda(v_0) = (c(v_0), a(v_0))$,
— $a(v_0) < D$,
— $\forall v \in B(v_0, 1)$ $c(v) \geq c(v_0)$,

— $a(v_0) = Min\{a(v) \mid v \in B(v_0, 1) \text{ et } c(v) = c(v_0)\}$.
*Réétiquetage :*
— $\lambda'(v_0) := (c(v_0), a(v_0) + 1)$.

R: **Le changement de ronde**
*Précondition :*
— $\lambda(v_0) = (c(v_0), D)$.
*Réétiquetage :*
— $\lambda'(v_0) := (c(v_0) + 1, 0)$.

Soit $(\mathbf{G}_i)_{0 \leq i}$ la suite des étiquetages de $G$. Soit $c_i(v)$ (resp. $a_i(v)$) les valeurs des variables $c(v)$ et $a(v)$ de $\mathbf{G}_i$.

**Proposition 13.4** *Ce synchroniseur garantit que la différence entre les compteurs de deux sommets quelconques est au plus* 1.

Pour montrer ce résultat, on a :

**Fait. 1**

$$c_{i+1}(v) \geq c_i(v).$$

**Fait. 2** *Si* $c_{i+1}(v) = c_i(v) + 1$ *alors* $a_i(v) = D$.

**Lemme 13.5** *Si* $c_i(v) = \pi$ *et* $a_i(v) = h$ *alors :*

$$\forall w \in V(G) \quad d(v, w) \leq h \Rightarrow c_{i-h}(w) \geq \pi.$$

**Preuve :** Par induction sur $i$. Si $i = 0$ la propriété est vraie.

On suppose que $c_{i+1}(v) = c_i(v) = \pi$ et $a_{i+1}(v) = a_i(v) = h$. D'après l'hypothèse d'induction : $d(v, w) \leq h \Rightarrow c_{i-h}(w) \geq \pi$. D'après le fait 1 $c_{i-h+1}(w) \geq c_{i-h}(w) \geq \pi$. Donc $c_{(i+1)-h}(w) \geq \pi$.

On suppose que : $c_{i+1}(v) = c_i(v) = \pi$ et $a_{i+1}(v) = a_i(v) + 1 = h$. Si $d(v, w) \leq h = a_i(v) + 1$ alors soit $u$ tel que $d(v, u) = 1$ et $d(u, w) = a_i(v) = h - 1$.

On a $a_{i+1}(v) = a_i(v) + 1 \Rightarrow c_i(u) \geq c_i(v) \geq \pi \quad \text{and} \quad a_i(u) \geq h - 1$ (précondition de la règle d'observation). En appliquant l'hypothèse d'induction au sommet $u$, $c_{i-(h-1)}(w) \geq \pi$ et finalement $c_{(i+1)-h}(w) \geq \pi$.

Le dernier cas est $c_{i+1}(v) = c_i(v) + 1$, nécessairement $a_{i+1}(v) = 0$ ce qui termine la preuve. □

De ce lemme et du Fait 1, on en déduit :

**Corollaire 13.6** *Si* $c_i(v) = \pi \quad et \quad a_i(v) = h$ *alors* $d(v, w) \geq h \Rightarrow c_i(w) \geq \pi$.

**Lemme 13.7** *Si* $c_i(v) = \pi$ *et* $c_i(w) = \pi + 1$ *alors* $\forall u \in V(G)$ $(c_i(u) = \pi$ *ou* $c_i(u) = \pi + 1)$.

**Preuve :** Soit $j$ tel que $c_j(w) = \pi$ et $c_{j+1}(w) = \pi + 1$, par la précondition de la règle sur la ronde et le corollaire précédent :

$$\forall u \in V(G) \quad c_j(u) \geq \pi.$$

Pour les mêmes raisons, comme $c_i(v) = \pi$, il n'existe pas $u$ tel que $c_i(u) > \pi + 1$. □

**Remarque 13.8** *Le même résultat peut être obtenu avec un borne du diamètre, avec la taille du graphe ou une borne de la taille.*

## 13.3 Trois synchroniseurs

Cette section présente brièvement trois synchroniseurs classiques.

Le but d'un synchroniseur est de simuler une exécution synchrone qui, par définition, est régit par une horloge et qui se traduit par une exécution en rondes (ou rounds). Le méchanisme de base d'un synchroniseur est la génération d'impulsions sur chaque sommet : la génération d'une impulsion sur un sommet autorisant une opération (ou plus généralement une suite d'opérations) sur ce sommet. Cette impulsion sur le sommet $v$ se traduit par l'incrémentation d'un compteur $P_v$. La propriété suivante doit être vérifiée :

Un message émis par le processeur $v$ en direction de son voisin $u$ à l'impulsion $p$ de $v$ doit être reçu par $u$ avant l'impulsion $p+1$ de $u$.

Un processeur $v$ est prêt pour l'impulsion $p$, ce qui est noté $Ready(v,p)$, s'il a reçu tous les messages que ses voisins lui ont envoyés à l'impulsion $p-1$.

Un processeur $v$ est à jour par rapport à $p$, ce qui est noté $Ajour(v,p)$, si les messages envoyés par $v$ pendant l'impulsion $p$ sont arrivés.

On a :

**Lemme 13.9** *Si chaque voisin $w$ de $v$ satisfait $Ajour(w,p)$ alors $v$ satisfait $Ready(v,p+1)$.*

Ces propriétés peuvent être facilement garanties pour des algorithmes tels que chaque processeur communique avec tous ses voisins à chaque round.

Si des processeurs ne communiquent pas avec certains voisins pendant un ou plusieurs rounds, ces propriétés sont vérifiées dès lors que l'on respecte les phases suivantes.

- Phase A : Chaque processeur envoie son message et chaque processeur recevant d'un voisin un message lui retourne un accusé de réception.
- Phase B : Faire en sorte que chaque processeur puisse savoir quand ses voisins sont à jour.
- Phases C : Faire en sorte que chaque processeur sache quand tous ses voisins sont prêt pour l'impulsion $p$.

1. Le synchroniseur $\alpha$ correspond à une synchronisation par ronde.
2. Le synchroniseur $\beta$ correspond à une synchronisation par ronde qui est contrôlée par la racine d'un arbre qui envoie le long des branches de l'arbre le top pour la ronde suivante une fois qu'elle a reçue de tous ses descendants un accusé de réception indiquant que la ronde précédente est finie.
3. Le synchroniseur $\gamma$ correspond à une synchronisation par ronde qui est contrôlée par les racines des arbres qui constituent une partition du réseau, les racines étant synchronisées entre elles par des chemins les liant.

# Chapitre 14

# Agents mobiles

Ce chapitre présente la notion d'agents mobiles et un lien avec les algorithmes distribués. Un agent mobile dans un réseau peut être vu comme un programme avec ses données et sa mémoire ayant la capacité de se déplacer à travers ce réseau, d'interagir avec les noeuds et avec d'autres agents mobiles. (Ce chapitre a été présenté dans [CGMO06].)

## 14.1 Description du modèle

### 14.1.1 Système d'agents mobiles.

Un système d'agents mobiles est composé :
— d'un ensemble $\mathbb{P}$ de places (d'exécution),
— d'un système de navigation $\mathbb{S}$,
— d'un ensemble $\mathbb{A}$ d'agents mobiles,
— d'une application injective $\pi_0 : \mathbb{A} \longrightarrow \mathbb{P}$ décrivant le placement initial des agents,
— d'un étiquetage initial $\lambda$ des places et des agents.

**Remarque 14.1** *L'étiquetage $\lambda$ des places et des agents peut coder des places anonymes (certianes places ont la même étiquette), des agents anonymes (certains agents ont la même étiquette) ou une connaissance initiale (des agents ou bien des places). Comme exemples de connaissance initiale on peut donner : (une borne sur) le nombre de places, (une borne sur) le nombre d'agents, (une borne sur) le diamètre du système de navigation, la topologie du système de navigation, le placement initial des agents, l'anonymat des agents ou des places.*

Le système de navigation est décrit par un graphe connexe $G = (V, E)$, où $V$ correspond aux places et $E$ aux canaux de communication entre les places. Dans la suite, on identifie d'une part les places et les sommets et d'autre part les canaux et les arêtes.

On suppose que $G$ est muni d'une numérotation des ports donc le système de navigation est défini par $\mathbb{S} = (G, \delta)$ où $\delta$ est une fonction de numérotation des ports.

Le système est supposé asynchrone : pas d'horloge globale, la migration d'un agent est asynchrone : le temps de déplacement d'un agent le long d'un canal n'est pas prévisible.

### 14.1.2 Algorithme associé à un agent mobile

À chaque agent mobile est associé un système de transition qui interagit avec les places et le système de navigation.

Soit $Q_{\mathbb{P}}$ un ensemble (récursif) d'états associés aux places, et soit $Q_{\mathbb{A}}$ un ensemble (récursif) d'états associés aux agents. L'état initial de chaque agent $\mathbf{a}$ est $\lambda(\mathbf{a})$ et l'état initial de chaque place $\mathbf{p}$ est $\lambda(\mathbf{p})$.

Soit $\mathbf{p}$ une place. On note $\texttt{state}(\mathbf{p})$ l'état associé à $\mathbf{p}$. Soit $\mathbf{a}$ un agent. On note $\texttt{state}(\mathbf{a})$ l'état associé à $\mathbf{a}$.

La transition associée à l'agent mobile $\mathbf{a}$ dans l'état $s$ sur la place $\mathbf{p}$ dans l'état $q$, transforme $s$ en $s'$, $q$ en $q'$ et soit $\mathbf{a}$ ne bouge pas soit il migre sur une place voisine à travers le port *out*. Cette transition est notée :

$$(s, q, in) \vdash_{\mathbf{p}}^{\mathbf{a}} (s', q', out),$$

cela signifie que l'agent mobile $\mathbf{a}$ a migré sur la place $\mathbf{p}$ à travers le port *in* ou après la transition il quitte la place $\mathbf{p}$ à travers le port *out*, avec la convention que si l'agent était déjà sur la place et il ne bouge pas après la transition alors $in = 0$ et $out = 0$; de plus *in* et *out* ne peuvent pas être simultanément différent de 0.

**Remarque 14.2** *En général on ne dispose pas du nom des places. Néanmoins, pour facilité la description, un agent* $\mathbf{a}$ *en transit est noté par* $(p, \mathbf{a}, p')$ *où* $p$ *est la place de départ et* $p'$ *est la place d'arrivée.*

Une configuration du système est décrite par l'état de chaque place, l'état de chaque agent, l'ensemble $\mathbb{M}$ des agents en transit (initialement $\mathbb{M}$ est vide) et une application $\pi$ décrivant l'emplacement des agents qui ne sont pas dans les canaux.

Un événement dans le système est défini par une transition associée à un agent $\mathbf{a}$ sur la place $\mathbf{p}$, il a la forme :

$$(s, q, in) \vdash_{\mathbf{p}}^{\mathbf{a}} (s', q', out),$$

l'état des agents différents de $\mathbf{a}$ n'est pas affecté, l'état des places différentes de $\mathbf{p}$ n'est pas affecté, le nouvel état de $\mathbf{a}$ est $s'$ (il était $s$ avant l'événement), le nouvel état de $\mathbf{p}$ est $q'$ (il était $q$ avant l'événement), et :

- si $in = 0$ et $out = 0$ alors $\pi$ et $\mathbb{M}$ ne sont pas affectés,
- si $in = 0$ et $out \neq 0$ alors l'ensemble des agents en transit après l'événement est $\mathbb{M} \cup \{(\mathbf{p}, \mathbf{a}, \mathbf{p}')\}$ (où $\mathbf{p}'$ est la place adjacente à $\mathbf{p}$ correspondant au port *out*) et $\pi$ n'est plus définie pour $\mathbf{a}$ et inchangée pour les autres agents,
- si $in \neq 0$ et $out = 0$ alors l'ensemble des agents en transit après l'événement est $\mathbb{M} \setminus \{(\mathbf{p}', \mathbf{a}, \mathbf{p})\}$ (où $\mathbf{p}'$ est la place adjacente à $\mathbf{p}$ correspondant au port *in*), $\pi(\mathbf{a}) = \mathbf{p}$ et $\pi$ est inchangée pour les autres agents.

### 14.1.3 Exécution.

Une exécution de l'algorithme associé aux agents mobiles est définie par une suite $(\texttt{state}_0, \mathbb{M}_0), (\texttt{state}_1, \mathbb{M}_1), ..., (\texttt{state}_i, \mathbb{M}_i), ...$ telle que :

- $\mathbb{M}_0 = \emptyset$,
- pour chaque agent $\mathbf{a}$, $\texttt{state}_0(\mathbf{a}) = \lambda(\mathbf{a})$ est un état initial,
- pour chaque place $\mathbf{p}$, $\texttt{state}_0(\mathbf{p}) = \lambda(\mathbf{p})$ est un état initial,
- pour chaque $i$ il existe une unique place $\mathbf{p}$ et un unique agent $\mathbf{a}$ tels que :
  - si $\mathbf{p}' \neq \mathbf{p}$ alors $\texttt{state}_{i+1}(\mathbf{p}') = \texttt{state}_i(\mathbf{p}')$,
  - si $\mathbf{a}' \neq \mathbf{a}$ alors $\texttt{state}_{i+1}(\mathbf{a}') = \texttt{state}_i(\mathbf{a}')$,
  - $(\texttt{state}_{i+1}(\mathbf{a}), \texttt{state}_{i+1}(\mathbf{p}), \mathbb{M}_{i+1})$ est obtenu à partir de $(\texttt{state}_i(\mathbf{a}), \texttt{state}_i(\mathbf{p}), \mathbb{M}_i)$ par un événement de la forme :

$$(s, q, in) \vdash_{\mathbf{p}}^{\mathbf{a}} (s', q', out).$$

Une configuration est définie par $(\texttt{state}_i, \mathbb{M}_i)$. Une configuration terminale est une configuration à partir de laquelle aucun événement ne peut se produire ; on peut noter que dans ce cas aucun agent n'est en transit. Par définition la longueur de la suite est la longueur de l'exécution.

Une place qui est l'emplacement initial d'un agent est appelée base. Le placement initial des agents peut être codé dans l'état des places donc on suppose qu'une place peut savoir si elle est base ou non. Néanmoins, en général, un agent ne peut pas savoir si une base-initiale est la sienne ou non.

Finalement, le système d'agents mobiles est défini par :

$$(\mathbb{A}, \mathbb{P}, \mathbb{S}, \pi_0, \lambda),$$

l'algorithme à base d'agents mobiles est défini par :

$$\mathcal{A} = (\vdash^{\mathbf{a}}_{\mathbf{p}})_{\mathbf{a}\in\mathbb{A},\mathbf{p}\in\mathbb{P},}$$

et une exécution $\mathcal{E}$ est définie par :

$$\mathcal{E} = (\texttt{state}_i, \mathbb{M}_i)_{i\geq 0}.$$

### 14.1.4 Détection de la terminaison

Un agent $\mathbf{a}$ dans l'état $s$ sur la place $\mathbf{p}$ dans l'état $q$ est dit passif si aucune transition n'est associée à cette configuration. Dans une configuration terminale tous les agents sont passifs. La terminaison est dite implicite si aucun agent ne peut la détecter. Si au moins un agent peut détecter la terminaison alors elle est dite explicite.

## 14.2 Exécutions équivalentes

On considère des algorithmes à base d'agents mobiles et des algorithmes à base de messages tels que les graphes sous-jacents sont identiques.

Différents types d'équivalences peuvent être définis entre les algorithmes à base d'agents et les algorithmes à base de messages. Ici on ne considère que des algorithmes qui terminent. Dans ce cas ces algorithmes sont équivalents pour les configurations terminales si l'ensemble des graphes correspondant au système de navigation étiquetés par les états finaux des places et l'ensemble des graphes correspondant au système de communications étiquetés par les états finaux des processeurs sont égaux et si la terminaison des deux algorithmes est implicite ou la terminaison des deux algorithmes est explicite.

## 14.3 Simuler un algorithme à base d'agents par un algorithme à base de messages

Le but de cette section est de vérifier que l'on peut implémenter un système à base d'agents mobiles à l'aide d'un système à base de messages. Pour ce faire, on montre que tous les pas élémentaires des agents peuvent être simuler dans le système à base de messages.

Soit $(\mathbb{A}, \mathbb{P}, \mathbb{S}, \pi_0, \lambda)$ un système à base d'agents mobiles et soit $\mathcal{A} = (\vdash^{\mathbf{a}}_{\mathbf{p}})_{\mathbf{a}\in\mathbb{A},\mathbf{p}\in\mathbb{P}}$ un algorithme à base d'agents mobiles implémenté sur ce système. Soit $\mathbb{S} = (V, E, \delta)$ le système de navigation correspondant. On suppose que le système est composé de $k$ agents. On définit un étiquetage additionel $\chi_{\pi_0}$ des sommets de $G$ par $\chi_{\pi_0}(v) = 1$ si il existe un agent $\mathbf{a}$ tel que $\pi_0(\mathbf{a}) = v$, et

$\chi_{\pi_0}(v) = 0$ sinon. On construit un algorithme à base de messages $(P, C, \lambda') = (V, E, \delta, \lambda')$. à partir du système à base d'agents mobiles Sur chaque sommet $v$ qui correspond à une place $\mathbf{p}$ on installe un processus $p$. Soit $\sharp$ une nouvelle étiquette. La fonction d'étiquetage $\lambda'$ code sur chaque processus $p$ l'étiquette de la place correspondante $\mathbf{p}$, et, dans le cas d'une base, l'étiquette de l'agent $\mathbf{a}$, i.e., si $v$ correspond à la base de $\mathbf{a}$ $\lambda'(v) = (\lambda(v), 1, \lambda(\mathbf{a}))$ et si non $\lambda'(v) = (\lambda(v), 0, \sharp))$.

Maintenant on consruit un algorithme à base de messages $\mathcal{D}$ tel que chaque exécution $\mathcal{E}$ de $\mathcal{A}$ peut être simulée par une exécution $\mathcal{E}'$ de $\mathcal{D}$. L'état d'un processus est défini par : - l'état de la place correspondante, - la présence d'un agent mobile est codé par un jeton et l'état correspondant de l'agent mobile est codé par la valeur du jeton. Finalement, l'ensemble des états possibles des processus est l'ensemble des états définis par les places, la présence du jeton et si le jeton est présent par l'état de l'agent.

La présence d'un agent $\mathbf{a}$ sur un sommet $u$ est représentée par le jeton $t(\mathbf{a})$ situé sur $u$. Chaque jeton a une *base* qui correspond à la position initiale de l'agent mobile correspondant. À chaque jeton est associé un état : l'état courant du jeton $t(\mathbf{a})$ est égal à l'état de l'agent $\mathbf{a}$.

La traduction de l'événement :

$$(s, q, in) \vdash_{\mathbf{p}}^{\mathbf{a}} (s', q', out)$$

de l'algorithme à base d'agents mobiles dans un événement de l'algorithme à base de messages se fait suivant les règles suivantes.

L'état de chaque jeton différent du jeton associé à $\mathbf{a}$ n'est pas affecté, de même, l'état de chaque processus différent de $p$ n'est pas affecté, le nouvel état du jeton associé à $\mathbf{a}$, i.e., $t(\mathbf{a})$, est $s'$ (il était $s$ avant l'événement), le nouvel état de $p$ est $q'$ (il était $q$ avant l'événement), et :

— si $in = 0$ et $out = 0$ alors le jeton $t(\mathbf{a})$ ne bouge pas,

— si $in = 0$ et $out \neq 0$ alors le jeton $t(\mathbf{a})$ est envoyé via le port $out$,

— si $in \neq 0$ et $out = 0$ alors le jeton $t(\mathbf{a})$ est reçu par le processus $p$ via le port $in$.

Soit $\mathcal{D}_p$ l'algorithme induit par cette construction sur le processus $p$. Soit $\mathcal{D} = (\mathcal{D}_p)_{p \in P}$. Par induction sur la longueur des exécutions, on montre que si l'algorithme à base d'agents mobiles termine alors l'algorithme à base de messages défini ci-dessus termine. L'algorithme à base de messages $\mathcal{D}$ termine explicitement si et seulement si $\mathcal{A}$ termine explicitement. Un graphe correspondant au système de navigation étiqueté par les états finaux des places est obtenu avec $\mathcal{A}$ si et seulement si il peut être obtenu comme un graphe de communication étiqueté par les états finaux des processus avec $\mathcal{D}$. Finalement :

**Proposition 14.3** *Soit $(\mathbb{A}, \mathbb{P}, \mathbb{S}, \pi_0, \lambda)$ un système d'agents mobiles. Soit $\mathcal{A} = (\vdash_{\mathbf{p}}^{\mathbf{a}})_{\mathbf{a} \in \mathbb{A}, \mathbf{p} \in \mathbb{P}}$ un algorithme à base d'agents mobiles implémenté sur ce système. Soit $(P, C, \lambda')$ le système à base de messages construit ci-dessus. Soit $\mathcal{D} = (\mathcal{D}_p)_{p \in P}$ l'algorithme à base de messages défini plus haut. Les exécutions de $\mathcal{D}$ sont équivalentes aux exécutions de $\mathcal{A}$.*

## 14.4 Simulation d'un algorithme à base de messages par un algorithme à base d'agents mobiles

Cette section présente une procédure qui, étant donnés un algorithme à base de message $\mathcal{D}$ sur le système $(V, E, \delta, \lambda)$ et un nombre $k \geq 1$, engendre un algorithme équivalent à base d'agents $\mathcal{A}$ avec $k$ agents sur un système à base d'agents.

La procédure fonctionne de la façon suivante. le système de navigation correspond à $(V, E, \delta)$. L'état de la place $\mathbf{p}$ qui correspond au processus $p$ et (qui est identifié au sommet $v$) est défini

par l'état du processus $p$ (avec la même initialisation) et par les valeurs des variables définies ci-dessous. Les états et les algorithmes associés aux agents mobiles sont définis dans la suite.

**Procédure 1**

Pas 1 : En se réveillant, un agent $\mathbf{a}$ construit un arbre $T_{\mathbf{a}}$ à travers un parcours d'arbre. Ceci conduit à la construction d'une forêt couvrante de $k$ arbres où chaque arbre est construit par un agent.

Pas 2 : L'agent $\mathbf{a}$ exécute l'algorithme $\mathcal{D}$ sur les sommets de $T_{\mathbf{a}}$. Cette exécution est faite par des rondes, où à chaque ronde, $T_{\mathbf{a}}$ est parcouru et, si c'est possible, $d$ ($d \geq 1$) pas de calcul de $\mathcal{D}$ sont exécutés sur chaque sommet de $T_{\mathbf{a}}$. Donc, $\mathbf{a}$ est en charge des pas de calcul exécutés sur les sommets de l'arbre qu'il a construit.

Du fait que les sommets et les agents peuvent être anonymes et en l'absence d'une orientation globale du réseau, il peut être compliqué pour un agent de trouver son chemin dans le réseau. Pour résoudre ce problème, les agents doivent utiliser des informations locales pour mémoriser leur chemin. En conséquence, on suppose que chaque arête e $e = \{u, v\}$ dispose de 2 étiquettes $\delta_u(v)$ et $\delta_v(u)$ qui correspondent aux étiquettes de $e$ sur $u$ et $v$.

Quand un agent travers le graphe il mémorise la suite ordonnée des étiquettes des arêtes traversées. Dans la suite, *ePath* (chemin d'exploration) désignera cette suite. Quand une arête $e$ est traversée par un agent $\mathbf{a}$ de $u$ à $v$, alors l'étiquette $\delta_v(u)$ est concaténée au chemin associé à l'agent $\mathbf{a}$. Ceci permet à $\mathbf{a}$ de revenir au sommet précédent (i.e., $u$) dès qu'il le souhaite. Quand il le fait, l'étiquette $\delta_v(u)$ est retirée de *ePath*. Ainsi, à tout moment pendant l'éxécution de l'algorithme, *ePath* contient la suite des étiquettes des arêtes que $\mathbf{a}$ a parcouru (dans l'ordre inverse) pour retourner à sa base à partir du sommet où il se trouve.

### 14.4.1 Calcul de l'arbre par un agent

Chaque agent $\mathbf{a}$ calcule son arbre $T_{\mathbf{a}}$ de la façon suivante. À partir de sa base, $\mathbf{a}$ exécute un parcours partiel du graphe. Pendant ce parcours, il marque tous les sommets visités pour la première fois. À chaque sommet $w$ marqué par $\mathbf{a}$, $\mathbf{a}$ choisit au hasard une arête non explorée incidente à $w$ et la traverse. Chaque fois que $\mathbf{a}$ traverse une arête $e$ pour atteindre un sommet non marqué $v$, il marque $v$ comme *visité* et marque $e$ comme une arête de $T$. Par ailleurs, quand il atteint un sommet $u$ déjà visité (le sommet $u$ est déjà marqué), l'arête menant à $u$ est marqué comme ne faisant pas partie de l'arbre et l'agent retourne au sommet précédent (marqué par lui) et essaie une autre arête non explorée incidente à ce sommet. Quand il n'y plus d'arête non explorée l'agent retourne en arrière au sommet précédent (grâce au lien dans *ePath*) et essaie d'explorer. Finalement, quand l'agent est de retour sur sa base et qu'il n'y a plus de lien à explorer il arrête la construction de l'arbre.

**Remarque 14.4** *Une marque sur une arête peut être faite en marquant les ports correspondants sur les extrémités de cette arête.*

La procédure de calcul de l'arbre par chaque agent est décrite par l'algorithme 19.

**Fait. 3** *Chaque sommet du graphe est marqué par exactement un agent.*

**Fait. 4** *Si 2 sommets $u_1$ et $u_2$ sont marqués par le même agent alors il existe exactement un seul chemin de $T$ les reliant. si 2 sommets $u_1$ et $u_2$ sont marqués par 2 agents distincts, alors tout chemin les liant contient au moins une arête marquée $NT$.*

**Fait. 5** *Il n'existe pas de cycle ne contenant que des arêtes marquées $T$.*

**Algorithme 19:** Construction de l'arbre par un agent mobile

**début**
  Marquer la base ;
  Initialiser *ePath* à vide ;
  **pour** *une arête non explorée* $e = (u, v)$ *incidente au sommet* $u$ *où se trouve l'agent* **faire**
    traverser $e$ pour aller sur le sommet $v$ ;
    **si** $v$ *est déjà marqué ou* $v$ *contient un agent* $a_1$, **alors**
      retourner sur $u$ et marquer le lien $e$ avec l'étiquette $NT$
    **sinon**
      marquer le lien $e$ comme étant $T$ et marquer $v$ exploré ;
  **si** *il n'y plus de lien à explorer sur le sommet courant* **alors**
    **si** *ePath est non vide,* **alors**
      retirer le dernier lien de *ePath*, traverser ce lien et aller à la ligne 3
    Fin du calcul de l'arbre ;
**fin**

### 14.4.2 Codage des actions à base de messages

Parmi les pas de calcul composant un système à base de messages les opérations *envoyer un message via le port j* et *recevoir un message via le port j* sont fondamentales. Pour coder ces opérations, on dispose sur chaque place d'une varibale appelée *in-buf*, où sont mémorisés les messages reçus. Dans ce travail, la première partie d'un message contient toujours le port d'où provient ce message.

**Envoyer le message m via le port j.** Soit $e = \{u, v\}$ et soit **a** un agent situé sur $u$ avec $\delta_u(v) = j$. L'exécution de l'opération *Envoyer le message m via le port j* par l'agent **a** consiste à traverser l'arête $e$, écrire le message $\langle \delta_v(u), m \rangle$ dans *in-buf* de $v$ et revenir à travers l'arête $e$.

**Recevoir le message m via le port j.** Soit $e = \{u, v\}$ et soit **a** un agent sur le sommet $u$ avec $\delta_u(v) = j$. L'exécution de l'opération *recevoir le message m via le port j* par l'agent **a** consiste à regarder le premier message arrivé via le port $j$ dans *in-buf* de $u$ (si l'algorithme origine nécessite des canaux FIFO, i.e., les messages doivent être reçus dans l'ordre où ils ont été émis ; si ce n'est pas le cas alors on prend les messages dans n'importe quel ordre dans *in-buf*). Une fois que l'on a pris ce message (il est retiré de *in-buf*) et mémorisé dans une variable temporaire à des fins de calcul.

**Opération interne.** Il suffit d'appliquer la transformation de l'état du processus à l'état de la place.

**Remarque 14.5** *Les exécutions de envoyer et recevoir autorisent un agent* **a** *à écrire des informations dans in-buf des sommets quin'appartiennent pas nécessairement à* $T_{\mathbf{a}}$. *De plus, la simulation exécutée par la procédure 1* **??** *peut être vue comme un calcul distribué dans un réseau dont les proccessus sont décomposés en clusters, et où les processus, d'un même cluster, exécutent à tour de rôle leur pas de calcul.*

**Remarque 14.6** *Si à chaque ronde du Pas 2 de la procédure 1, on demande qu'à chaque fois qu'un pas de calcul peut être exécuté sur un sommet appartenant à l'arbre d'un agent, il le simule en un nombre fini de rondes alors tout état global du système à base de messages obtenu par l'algorithme peut être obtenu par les agents.*

### 14.4.3 Transformer un algorithme à base de messages qui termine en un algorithme à base d'agents qui termine

Dans cette partie nous montrons que si $\mathcal{D}$ est un algorithme qui se termine alors l'algorithme à base d'agents $\mathcal{A}$ se termine également. Pour cette raison, nous adaptons le comportement de chaque agent.En fait, on ajoute sur chaque place 2 variables *fatherLink* et *fState*. La variable *fatherLink* d'un sommet $u$, contient le numéro du port à travers lequel l'agent $\mathbf{a}$, situé surt $u$, peut atteindre le *father* de $u$ dans l'arbre qui contient $u$. Soit $u$ la base de $\mathbf{a}$, la variable *fState* de $u$ indique à $\mathbf{a}$ qu'il doit effectuer une ronde de plus dans l'arbre $T_{\mathbf{a}}$. La variable *fState* est soit le jeton *Finished* ou le jeton *NotFinished*. Initialement la variable *fatherLink* de chaque base contient 0 et la variable *fState* de chaque base est initialisée à *NotFinished*. La variable *fatherLink* des autres sommets est initialisée à 0 et la variable *fState* est initialisée à *Finished*. Tous ces changements conduisent à la procédure 2.

**Procédure 2**

Pas 1 : En se réveillant, chaque agent $\mathbf{a}$ construit un arbre $T_{\mathbf{a}}$ à travers un parcours partiel du graphe. Ceci construit une forêt couvrante de $k$ arbre où chaque arbre est construit par exactement un agent. Pendant cette construction, la variable *fatherLinks* de tous les sommets, autres que la base, qui ont été marqués par $\mathbf{a}$ sont mis à jour.

Pas 2 : L'agent $\mathbf{a}$ exécute les événements de $\mathcal{D}$ sur les sommets de $T_{\mathbf{a}}$. Cette exécution est faite à travers des rondes. L'agent $\mathbf{a}$ est autorisé à exécuter la ronde $r$ si et seulement si au début de la ronde $r$ la variable *fState* de sa base a la valeur *NotFinished* et il a simulé tous les pas possibles dans la ronde précédente sinon il s'endort. À chaque ronde, $\mathbf{a}$ met la variable *fState* de sa base à *Finished*, ensuite $T_{\mathbf{a}}$ est parcouru et, si c'est possible, $d$ ($d \geq 1$) pas de calcul de $\mathcal{D}$ sont exécutés à chaque sommet de $T_{\mathbf{a}}$. Si un agent simule un envoi de message au sommet $u$, l'agent utilise la variable *fatherLink* pour retrouver la base $w$ de l'agent qui a construit l'arbre contenant $u$. Quand il arrive sur $w$, il met la variable *fState* de $w$ à *NotFinished* (s'il trouve un agent endormi, il le réveille) puis il retourne sur $u$.

On vérifie que :

**Lemme 14.7** *L'algorithme $\mathcal{A}$, à base d'agents mobiles, construit suivant la procédure 2 est équivalent à l'algorithme $\mathcal{D}$.*

Finalement :

**Théorème 14.8** *Il existe un algorithme à base d'agents mobiles $\mathcal{A}$ qui résoud un problème $\mathcal{P}$ sur un système à base d'agents mobiles $(G, \delta, \lambda)$ avec un placement initial $\pi_0$ si et seulement si il existe un algorithme à base de messages $\mathcal{D}$ qui résoud $\mathcal{P}$ sur $(G, \delta, \lambda')$ ($\lambda'$ étant définie dans la section 4).*

Comme exemple d'applications en traduisant la caractérisation des réseaux pour lesquels il existe un algorithme d'élection on en déduit une caractérisation des systèmes à base d'agents mobiles dans lesquels on peut élire.

# Chapitre 15

# Algorithmes distribués probabilistes

Ce chapitre présente et illustre quelques notions sur les algorithmes distribués probabilistes qui seront utilisées dans les différents exemples développés.

## 15.1 Le réseau

On considère des réseaux avec une communication par messages en mode point-à-point décrits par un graphe connexe $G = (V(G), E(G))$ $(= (V, E))$.

L'état de chaque processus (resp. canal de communication) est représenté par une étiquette $\lambda(v)$ (resp. $\lambda(e)$) associée au sommet correspondant $v \in V(G)$ (resp. arête $e \in E(G)$) ; un tel graphe est noté $\mathbf{G} = (G, \lambda)$.

Chaque processeur est une entité capable de calculer, d'envoyer et/ou de recevoir des messages. On suppose que les processeurs et les canaux de communication sont fiables. Les communications sont FIFO.

### 15.1.1 Le modèle synchrone

On travaillera souvent dans le modèle synchrone, i.e., tous les processeurs démarrent en même temps et le temps se décompose en cycles (rounds) synchronisés. Néanmoins, en utilisant un synchroniseur local fonctionnant par cycles, cette présentation peut être étendue aux réseaux asynchrones.

### 15.1.2 Connaissance du réseau et des processeurs

Sauf précisions contraires, le réseau $G = (V(G), E(G))$ est anonyme : on ne dispose pas des identités ou bien on ne peut pas garantir l'unicité d'un identificateur d'un processeur. On ne dispose d'aucune connaissance globale sur le réseau comme sa taille ou une borne de sa taille. Chaque processeur peut distinguer l'origine des messages qu'il reçoit : il sait par quel canal il reçoit un message ou émet un message. Finalement le réseau est représenté par un graphe muni d'une numérotation des ports.

## 15.2 Algorithme distribué probabiliste

Un *algorithme probabiliste* est un algorithme qui fait des choix aléatoires au cours de son exécution comme, par exemple, faire un tirage du type pile ou face ou bien tirer un entier au

hasard dans un intervalle donné. Les algorithmes probabilistes permettent de fournir des solutions parfois plus "efficaces" que les solutions déterministes, ou encore des solutions pour des problèmes qui n'admettent pas de solution déterministe.

Un algorithme distribué probabiliste est un ensemble d'algorithmes probabilistes locaux ; de plus si le réseau est anonyme alors deux processeurs qui ont le même nombre de voisins exécutent des algorithmes identiques ayant la même distribution de probabilité.

Plus formellement (voir [Tel00] chapitre 9) un *processus probabiliste*, est défini par un quadruple $p = (\mathcal{E}, \mathcal{J}, \rightarrow^0, \rightarrow^1)$ où :

— $\mathcal{E}$ est l'ensemble des états du processus,
— $\mathcal{J} \subset \mathcal{E}$ est un sous ensemble d'états initiaux,
— $\rightarrow^0$ et $\rightarrow^1$ sont des relations binaires sur $\mathcal{E}$ telles que le $i^{\text{ième}}$ pas du processus est exécuté selon la transition $\rightarrow^0$ si le résultat du tirage est 0 et selon la transition $\rightarrow^1$ sinon.

Un *algorithme probabiliste distribué* $\mathcal{A}_p$ est par conséquent une collection d'algorithmes locaux $\mathcal{A}_p = (\mathcal{A}_i)_{1 \leq i \leq n}$ telle que chaque algorithme $\mathcal{A}_i$ est un algorithme probabiliste.

## 15.3 Analyse des algorithmes distribués probabilistes

L'analyse des algorithmes distribués probabilistes se fait à travers deux mesures : le temps d'exécution de l'algorithme et le nombre ou la taille des messages échangés. On s'intéresse principalement à l'espérance mathématique de la valeur de la mesure considérée.

### 15.3.1 Complexité en temps

Un cycle (un round ou une ronde) d'un processeur est composé de trois pas : 1. Envoyer un message à certains voisins, 2. Recevoir un message de certains voisins, 3. Calculer localement. La complexité en temps est le nombre de cycles nécessaires pour que tous les processeurs aient terminé leur calcul.

L'analyse de la complexité en temps des algorithmes proposés utilise l'hypothèse de synchronisme. C'est une hypothèse forte, néanmoins elle permet d'obtenir une "bonne" estimation du temps d'exécution de l'algorithme si le système est asynchrone.

### 15.3.2 Complexité en bits

Le coût d'un algorithme distribué est à la fois le temps et le nombre de bits : suivant qu'un message contient un bit ou l'encyclopédie universalis n'a pas la même incidence. Par définition la complexité en bits (pour un canal) d'un algorithme distribué est le nombre de bits qui transitent (par un canal) pendant son exécution. Dans chaque cycle de bit chaque noeud envoie/reçoit au plus un bit. Finalement on définit la complexité en bits comme étant le nombre de cycles de bits jusqu'à la terminaison de l'algorithme.

### 15.3.3 Complexité moyenne

On considère un système distribué synchrone $\mathcal{S}$ à $n \geq 1$ entités. Soit $\mathcal{T}$ une tâche à réaliser sur $\mathcal{S}$ et $\mathcal{A}_p = (\mathcal{A}_i)_{1 \leq i \leq n}$ un algorithme probabiliste permettant de réaliser la tâche $\mathcal{T}$. Sans perte de généralité, on considère que l'algorithme $\mathcal{A}_p$ démarre à l'instant $t = 0$. Nous définissons la v.a. $T$ comme le nombre d'unités de temps qui s'écoulent entre l'instant $t = 0$ et la plus petite valeur de $t$ telle que $\forall i \in \{1, 2, \cdots, n\}$, l'algorithme $\mathcal{A}_i$ est terminé à l'instant $t$. La complexité moyenne de l'algorithme $\mathcal{A}_p$ est donc l'espérance mathématique de la v.a. $T$.

Dans l'analyse des algorithmes probabilistes, on s'intéresse principalement à l'espérance de $T$. Cependant, une analyse plus fine, et parfois plus complexe, permet de calculer, du moins asymptotiquement, la distribution de $T$.

Dans beaucoup de cas, nous pouvons montrer que la probabilité que $T$ dépasse $C(n)$ (une fonction de la taille $n$ du graphe) est en $o(n^{-1})$. On dit alors que $T \leq C(n)$ avec forte probabilité.

## 15.4 Las Vegas - Monte Carlo

**Algorithmes du type Las Vegas**

Un algorithme probabiliste $\mathcal{A}$ est du type *Las Vegas* si $\mathcal{A}$ fournit *toujours* un résultat *correct* mais la complexité en temps est une v.a. non nulle $T$. En général il se termine avec probabilité 1.

**Algorithmes du type Monte Carlo**

Un algorithme probabiliste $\mathcal{A}$ est du type *Monte Carlo* s'il se termine et le résultat est correct avec une probabilité non nulle.

## 15.5 Un premier exemple : un algorithme distribué probabiliste pour élire dans un graphe complet

Dans cette section, nous allons illustrer quelques unes des notions et notations introduites ci-dessus. Rappelons que le problème de l'élection consiste à distinguer un processus dans un ensemble. Dans cet exemple, nous présentons et analysons un algorithme distribué probabiliste pour élire un sommet dans un graphe anonyme et complet à $n$ sommets : $K_n$.

**Remarque 15.1** *Le graphe étant anonyme il n'existe pas d'algorithme d'élection déterministe.*

L'algorithme étudié est simple. Il est paramétré par un entier $N \geq 1$. Chaque sommet $v$ tire aléatoirement et uniformément un nombre $r(v)$ dans l'ensemble $\{1, 2, \cdots, N\}$, envoie $r(v)$ à tous ses voisins et reçoit $r(u)$ de tous ses voisins. Le sommet $v$ compare ensuite $r(v)$ avec les nombres tirés par ses voisins. Si $v$ est le seul à avoir tiré la plus grande valeur alors $v$ est élu sinon il est battu. L'analyse de l'algorithme consiste à étudier la probabilité pour un sommet $v$ donné d'être élu puis la probabilité pour qu'un sommet du graphe soit élu.

**Algorithme 20:** $ElectionProba_1()$, élection dans un graphe complet.

$v$ tire aléatoirement et uniformément un élément $r(v)$ dans l'ensemble $\{1, 2, \cdots, N\}$;
$v$ envoie $r(v)$ à tous ses voisins;
$v$ reçoit $r(u)$ de chaque voisin $u$;
**si** $r(v) > r(u)$ *pour tout sommet* $u$ *de* $V$ **alors**
  $etat_v := Elu$;
**sinon**
  $etat_v := NonElu$;
$v$ envoie $etat_v$ à tous ses voisins;
$v$ reçoit $etat_u$ de chaque voisin $u$;

**Probabilité pour un sommet d'être élu.**

Nous avons :

**Proposition 15.2** *Soit $v$ un sommet quelconque de $K_n$. Soit $\mathcal{E}(v)$ l'événement "$v$ est élu". Alors :*

$$\mathbb{P}\left(\mathcal{E}(v)\right) = \frac{1}{N}\sum_{i=2}^{N}\left(\frac{i-1}{N}\right)^{n-1}. \tag{15.1}$$

**Preuve.** Soit $v$ un sommet de $K_n$. Soit $\mathcal{E}(v)$ l'événement "$v$ est élu." L'événement $\mathcal{E}(v)$ a lieu si et seulement si la valeur tirée par $v$ est plus grande que les valeurs tirées par tous les autres sommets de $K_n$. Si $r(v) = 1$, il est clair que $v$ ne peut pas être élu. Si $r(v) = 2$, alors $v$ est élu si et seulement si tous les autres sommets ont tiré la valeur 1. De manière générale, si $r(v) = i \in \{2, 3, \cdots, N\}$, alors $v$ est élu si et seulement si tous les autres sommets ont tiré une valeur $j < i$. Pour tout $i \in \{2, 3, \cdots, N\}$, notons $A_i(v)$ l'événement "$v$ a tiré la valeur $i$" et $A_{<i}(v)$ l'événement "$v$ a tiré une valeur $j < i$". L'événement $\mathcal{E}(v)$ s'écrit :

$$\mathcal{E}(v) = \bigcup_{i=2}^{N}\left[A_i(v) \cap \bigcap_{u\neq v} A_{<i}(u)\right]. \tag{15.2}$$

On en déduit que :

$$\mathbb{P}\left(\mathcal{E}(v)\right) = \sum_{i=2}^{N}\left(\mathbb{P}\left(A_i(v)\right) \times \prod_{u\neq v}\mathbb{P}\left(A_{<i}(u)\right)\right). \tag{15.3}$$

Par ailleurs, tous les nombres de l'ensemble $\{1, 2, \cdots, N\}$ ont la même probabilité d'être tirés par $v$. Il en résulte que pour tout $i \in \{1, 2, \cdots, N\}$ :

$$\mathbb{P}\left(A_i(v)\right) = \frac{1}{N} \text{ et } \mathbb{P}\left(A_{<i}(u)\right) = \frac{i-1}{N}. \tag{15.4}$$

La relation (15.3) devient alors :

$$\begin{aligned}\mathbb{P}\left(\mathcal{E}(v)\right) &= \sum_{i=2}^{N}\left(\frac{1}{N} \times \prod_{u\neq v}\frac{i-1}{N}\right)\\ &= \frac{1}{N}\sum_{i=2}^{N}\left(\frac{i-1}{N}\right)^{n-1}.\end{aligned} \tag{15.5}$$

Ce qui termine la preuve.

Le paramètre $N$ est fondamental dans l'analyse de l'algorithme. En effet :

— Si $N = 1$, l'algorithme échoue et aucun sommet ne peut être élu.

— Si $N = 2$, cela revient à faire des tirages dans l'ensemble $\{1, 2\}$. Un sommet $v$ est donc élu si et seulement si il est le seul à avoir tiré 2. L'expression (15.1) devient alors :

$$\mathbb{P}\left(\mathcal{E}(v)\right) = \frac{1}{2^n}.$$

— Si $N \to \infty$ et $n$ reste fixé, nous avons :

$$\begin{aligned}\sum_{i=2}^{N}(i-1)^{n-1} &= \sum_{i=1}^{N-1} i^{n-1} \\ &\sim \int_1^{N-1} x^{n-1}dx \\ &= \frac{(N-1)^n-1}{n}.\end{aligned}$$

D'où

$$\begin{aligned}\mathbb{P}(\mathcal{E}(v)) &\sim \frac{1}{n}\left((1-\frac{1}{N})^n - \frac{1}{N^n}\right) \\ &\to \frac{1}{n}, \text{ quand } N \to \infty.\end{aligned}$$

— Si $N = n$, un raisonnement similaire donne :

$$\begin{aligned}\mathbb{P}(\mathcal{E}(v)) &\sim \frac{1}{n}\left((1-\frac{1}{n})^n - \frac{1}{n^n}\right), \text{ quand } n \to \infty \\ &\sim \frac{1}{ne}, \text{ quand } n \to \infty.\end{aligned}$$

**Probabilité d'obtenir une élection.**

Soit $\mathcal{E}$ l'événement *"Il y a un sommet élu dans le graphe"*. Il est alors clair que :

$$\mathcal{E} = \{\exists v \in V \text{ tel que } \mathcal{E}(v)\}.$$

Par ailleurs, pour tous $u, v \in V$, $\mathcal{E}(u) \cap \mathcal{E}(v) = \emptyset$. Il en résulte que :

$$\begin{aligned}\mathbb{P}(\mathcal{E}) &= \mathbb{P}(\{\exists v \in V \text{ tel que } \mathcal{E}(v)\}) \\ &= \sum_{v \in V}\mathbb{P}(\mathcal{E}(v)) \\ &= \frac{n}{N}\sum_{i=2}^{N}\left(\frac{i-1}{N}\right)^n.\end{aligned}$$

Si $N = n$, il résulte de ces expressions que :

$$\begin{aligned}\mathbb{P}(\mathcal{E}) &\sim \frac{1}{e} \\ &\sim \frac{1}{e} \geq 1/3 (n \to \infty).\end{aligned}$$

La complexité en temps de l'algorithme $ElectionProba_1()$ est 1. La probabilité que $ElectionProba_1()$ échoue est strictement supérieure à 0. Nous avons également le résultat suivant :

Si $N = n^2$, alors $ElectionProba_1()$ réalise une élection avec forte probabilité.

En effet, si on remplace $N$ par $n^2$ dans (15.6), nous obtenons :

$$\begin{aligned}\mathbb{P}(\mathcal{E}) &\sim \left(1-\frac{1}{n^2}\right)^n \\ &\sim e^{-1/n} (n \to \infty) \\ &> 1-\frac{1}{n}.\end{aligned}$$

On en déduit que la probabilité que l'algorithme $ElectionProba_1()$ réalise une élection est supérieure à $1-\frac{1}{n}$. Autrement dit, l'algorithme réussit avec forte probabilité.

On considère maintenant l'algorithme suivant :

**Algorithme 21:** L'algorithme $ElectionProba_2()$.

**répéter**
  $ElectionProba_1()$ ;
**jusqu'à** $etat_v = Elu$ *ou* $\exists u$ *tel que* $etat_u = Elu$;

L'algorithme $ElectionProba_2()$ ne s'arrête que si un sommet du graphe est élu. Donc si $ElectionProba_2()$ termine, alors le résultat est correct. Nous allons étudier sa complexité en temps (pour différentes valeurs de $N$). Cette complexité est une v.a. $T$.

Pour cela, nous définissons le nouvel événement :

$$\mathcal{E}_i = \{\text{il y a une élection à la } i^{\text{ième}} \text{ itération }\} \text{ pour tout } i \geq 1.$$

Il est alors clair que :

$$\text{Pour tout } i \geq 1,\ T = i \text{ si et seulement si } \mathcal{E}_i.$$

L'événement $\mathcal{E}_i$ correspond à la situation suivante : pendant les $(i-1)$ premières itérations de l'algorithme $ElectionProba_1()$, aucun sommet n'a été élu, et à la $i^{\text{ième}}$ exécution de $ElectionProba_1()$, un sommet a été élu. Si on interprète le fait d'obtenir un sommet élu comme un *succès* et l'événement inverse comme un *échec*, la v.a. $T$ correspond au nombre d'itérations avant d'obtenir un succès. La v.a. $T$ est par conséquent une v.a. suivant une loi géométrique de paramètre $p = \mathbb{P}(\mathcal{E})$. Autrement dit :

$$\mathbb{P}(T = i) = (1-p)^{i-1} p.$$

Nous en déduisons :

— Si $N = 2$,

$$\mathbb{P}(T = i) = \left(1-\frac{1}{2^n}\right)^{i-1} \frac{1}{2^n} \text{ et } \mathbb{E}(T) = \frac{1}{p} = 2^n.$$

— Si $N = n$,

$$\mathbb{P}(T = i) \sim \left(1-\frac{1}{e}\right)^{i-1} \frac{1}{e},\ \mathbb{E}(T) \sim \frac{1}{p} = e \text{ et } \mathbb{V}ar(T) \sim e-1.$$

L'espérance de la v.a. est asymptotiquement une constante et sa variance également. On cherche à mesurer l'écart de la v.a. $T$ de son espérance $e$. Une première tentative consiste à utiliser l'inégalité de Markov (voir (16.12)) :

$$\mathbb{P}(T > k) \leq \frac{\mathbb{E}(T)}{k} = \frac{e}{k}.$$

Si $k = e$, l'inégalité devient :

$$\mathbb{P}(T > e) \leq \frac{\mathbb{E}(T)}{e} = 1.$$

Ce qui est une majoration triviale.
De même, si on utilise l'inégalité de Bienaymé-Tchebicheff (voir (16.13)), nous avons :

$$\mathbb{P}(\mid T - \mathbb{E}(T) \mid \geq k) \leq \frac{\mathbb{V}ar(T)}{t^2} = \frac{e-1}{k^2}.$$

Si $k = e$, l'inégalité devient :

$$\mathbb{P}(\mid T - e \mid \geq e) \leq \frac{e-1}{e^2} \leq 0,2325....$$

Ce qui donne une majoration non triviale mais insuffisante. Cependant, il est clair que pour tout $k \geq 1$,

$$\begin{aligned}
\mathbb{P}(T \geq k) &= \frac{1}{e}\sum_{i\geq k}\left(1-\frac{1}{e}\right)^{i-1} \\
&= \frac{1}{e}\sum_{j\geq O}\left(1-\frac{1}{e}\right)^{j+k-1} \\
&= \frac{1}{e}\left(1-\frac{1}{e}\right)^{k-1}\sum_{j\geq O}\left(1-\frac{1}{e}\right)^{j} \\
&= \frac{1}{e}\left(1-\frac{1}{e}\right)^{k-1}\frac{1}{1-\left(1-\frac{1}{e}\right)} \\
&= \left(1-\frac{1}{e}\right)^{k-1}
\end{aligned}$$

Il en résulte que pour tout $k \geq e\log n + 1$,

$$\mathbb{P}(T \geq k) \leq \frac{1}{n}.$$

La complexité de l'algorithme $ElectionProba_2()$ est ainsi en $O(\log n)$ avec forte probabilité.

## 15.6 Élection dans un anneau anonyme

Cette section présente un algorithme d'élection dans un anneau anonyme où chaque processus connaît le nombre de sommets de l'anneau. Cet algorithme est probabiliste du type Las Vegas. Si les processus ne disposent pas de la connaissance du nombre de sommets alors il n'existe pas d'algorithme probabiliste du type Las Vegas seulement du type Monte Carlo.

On suppose que le système est asynchrone et fonctionne par échange de messages en mode asynchrone.

**Remarque 15.3** *Avec ces hypothèses il n'existe pas d'algorithme déterministe d'élection.*

L'algorithme présenté repose sur le même principe que l'algorithme de Chang et Roberts où le sommet ayant la plus grande identité est le sommet élu. Chaque processus tire au hasard et uniformément une identité dans un ensemble fini puis fait circuler cette valeur à travers l'anneau. Si plusieurs processus détectent qu'ils ont tiré la même plus grande valeur alors ils participent à un nouveau tournoi. Un processus ayant tiré, lors d'un tournoi, d'une manière unique la plus grande valeur est élu.

Un message est composé de 4 champs $(id, tournoi, saut, unique)$ :

— $id$ est un entier tiré au hasard et uniformément dans l'intervalle $[1, k]$ par le sommet initiateur de ce message ;

— $tournoi$ est le numéro du tournoi auquel correspond le message, initialement la valeur de $tournoi$ est 1;

— $saut$ est le nombre de sommets visités par ce message, initialement la valeur de saut est 1;

— $unique$ est un booléen qui détecte si un sommet différent du sommet émetteur du message a tiré le même numéro, initialement la valeur de $unique$ est $vrai$.

On suppose que l'anneau est orienté et chaque processus est initialement actif. Soit $n$ le nombre de sommets de l'anneau.

**Description de l'algorithme d'Itai-Rodeh (du à Fokkink et Pang)**

— Initialement tous les processus sont actifs, et chaque processus $p_i$ tire aléatoirement et uniformément son identité $id_i$ parmi les entiers de l'intervalle $[1, k]$, son numéro de tournoi est égal à 1 ($tournoi_i := 1$) puis envoie le message $(id_i, 1, 0, true)$ à son successeur ;

— Si un message $(id, tournoi, saut, unique)$ est reçu par un processus passif alors celui-ci transmet le message $(id, tournoi, saut + 1, unique)$ à son successeur ; s'il est reçu par un processus $p_i$ actif ($etat_i = actif$) alors, suivant le cas, $p_i$ effectue une des actions suivantes :

  — si $saut = n$ et $unique = true$ alors $p_i$ est élu et $etat_i := elu$;

  — si $saut = n$ et $unique = false$ alors $p_i$ tire aléatoirement et uniformément une nouvelle identité, $id_i$, parmi les entiers de $[1, k]$, puis ($tournoi_i := tournoi_i + 1$) et finalement il envoie à son successeur le message $(id_i, tournoi_i, 1, true)$;

  — si $(id, tournoi) = (id_i, tournoi_i)$ et $saut < n$ alors $p_i$ envoie le message $(id, tournoi, saut + 1, false)$ ;

  — si $(tournoi, id) >_{lex} (tournoi_i, id_i)$ alors $p_i$ devient passif ($etat_i := passif$) et il transmet le message $(id, tournoi, saut + 1, unique)$;

  — si $(tournoi, id) < (tournoi_i, id_i)$ alors $p_i$ supprime le message.

La connaissance du nombre de sommets du réseau permet d'assurer l'élection d'au plus un processus. Par ailleurs les calculs de la section précédente montrent que la probabilité qu'un sommet soit élu dans un tournoi est strictement positive ; le premier tournoi étant celui pour lequel cette probabilité est la plus faible. On en déduit que cet algorithme probabiliste est du type Las Vegas.

En utilisant les revêtements on montre qu'en l'absence d'information sur la taille de l'anneau il n'existe pas d'algorithme probabiliste du type Las Vegas permettant une élection.

# Chapitre 16

# Quelques rappels sur les probabilités

Cette section présente quelques éléments de la théorie des probabilités. Il s'agit d'éléments simples et utilisés pour l'analyse des algorithmes présentés dans ce travail.

### 16.0.1 Espace probabilisé discret

**Définition 16.1** *On appelle* espace probabilisé discret *tout couple* $(\Omega, \mathbb{P})$ *où* $\Omega$ *est un ensemble au plus dénombrable*[1] *non vide et où* $\mathbb{P}$ *est une application de* $\mathcal{P}(\Omega)$[2] *dans* $[0,1]$ *qui vérifie les deux propriétés(ou axiomes) suivantes :*

1. $\mathbb{P}(\Omega) = 1$,
2. $\mathbb{P}\left(\cup_{i\geq 1} A_i\right) = \sum_{i\geq 1} \mathbb{P}(A_i)$, *pour toute suite, finie ou infinie,* $A_i \in \mathcal{P}(\Omega)$, $i \geq 1$, *de parties deux à deux disjointes.*

**Terminologie :**

- $\Omega$ s'appelle l'espace des événements.
- Une partie $A \in \mathcal{P}(\Omega)$ s'appelle un événement.
- La fonction $\mathbb{P}$ s'appelle loi de probabilité ou, en abrégé, probabilité.
- Le nombre $\mathbb{P}(A)$ s'appelle probabilité de $A$.

**Remarque 16.2** *Etant donné* $\Omega$, $\mathbb{P}$ *est entièrement determinée par la donnée de* $\mathbb{P}(\omega) = \mathbb{P}(\{w\})$, $\omega \in \Omega$ *: si* $\mathbb{P} : \Omega \to [0,1]$ *est telle que* $\sum_{\omega\in\Omega} \mathbb{P}(\omega) = 1$, *alors elle peut être prolongée de façon unique sur* $\mathcal{P}(\Omega)$ *telle que* $(\Omega, \mathbb{P})$ *soit un espace probabilisé.*

**Propriétés**

- $\mathbb{P}(\emptyset) = 0$,
- $\mathbb{P}\left(\overline{A}\right) = 1 - \mathbb{P}(A)$,
- Soient $A$ et $B$ deux éléments de $\mathcal{P}(\Omega)$,
  - $\mathbb{P}(A \cup B) = \mathbb{P}(A) + \mathbb{P}(B) - \mathbb{P}(A \cap B)$,
  - si $A$ et $B$ sont disjoints, alors : $\mathbb{P}(A \cup B) = \mathbb{P}(A) + \mathbb{P}(B)$.

1. Un ensemble est au plus dénombrable s'il est en bijection avec un sous-ensemble de $\mathbb{N}$.
2. $\mathcal{P}(\Omega)$ est l'ensemble des parties de $\Omega$.

— Généralisation : Principe d'inclusion-exclusion

$$\begin{aligned}\mathbb{P}\left(\cup_{i=1}^{n} A_i\right) &= \mathbb{P}(A_1) + \mathbb{P}(A_2) + \cdots + \mathbb{P}(A_n) \\ &- \mathbb{P}(A_1 \cap A_2) - \mathbb{P}(A_1 \cap A_3) - \cdots \\ &+ \mathbb{P}(A_1 \cap A_2 \cap A_3) + \mathbb{P}(A_1 \cap A_2 \cap A_4) + \cdots \\ &- \cdots \\ &+ (-1)^{n+1}\mathbb{P}(A_1 \cap A_2 \cap \cdots \cap A_n).\end{aligned}$$

### 16.0.2 Probabilités conditionnelles

**Définition 16.3** *Soit $(\Omega, P)$ un espace probabilisé dénombrable et soit $A$ un événement de probabilité non nulle. On définit sur $\mathcal{P}(\Omega)$, l'application $P(. \mid A)$ à valeurs dans $[0,1]$ par :*

$$\mathbb{P}(B \mid A) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(A)},\ \forall B \in \mathcal{P}(\Omega).$$

*On appelle $\mathbb{P}(B \mid A)$ la probabilité conditionnelle de $B$ sachant $A$.*

**Proposition 16.4** *Soit $A_1, A_2, \cdots, A_k$ une partition de l'ensemble $\Omega$ telle que $\mathbb{P}(A_i) \neq 0$, $\forall i \in \{1, 2, \cdots, k\}$. Pour tout événement $A$, nous avons :*

$$\mathbb{P}(A) = \sum_{i=1}^{k} \mathbb{P}(A \mid A_i)\,\mathbb{P}(A_i). \tag{16.1}$$

Par ailleurs :

$$\mathbb{P}(A_1 \cap A_2) = \mathbb{P}(A_1 \mid A_2)\,\mathbb{P}(A_2) = \mathbb{P}(A_2 \cap A_1)\,\mathbb{P}(A_1),$$

il en résulte la formule de Bayes :

**Proposition 16.5** *Soit $A_1, A_2, \cdots, A_k$ une partition de l'ensemble $\Omega$ telle que $\mathbb{P}(A_i) \neq 0$, $\forall i \in \{1, 2, \cdots, k\}$. Pour tout événement $A$, nous avons :*

$$\begin{aligned}\mathbb{P}(A_i \mid A) &= \frac{\mathbb{P}(A_i \cap A)}{\mathbb{P}(A)}, \\ &= \frac{\mathbb{P}(A \mid A_i)\,\mathbb{P}(A_i)}{\sum_{j=1}^{k} \mathbb{P}(A \mid A_j)\,\mathbb{P}(A_j)}.\end{aligned} \tag{16.2}$$

### 16.0.3 Variables aléatoires

Dans l'analyse des algorithmes probabilistes, il est courant de s'intéresser au comportement d'une métrique (nombre de rounds, nombre de messages, etc). Cette métrique est en général une variable aléatoire :

**Définition 16.6** *Soit $(\Omega, \mathbb{P})$ un espace probabilisé dénombrable et soit $\Omega'$ un ensemble non vide au plus dénombrable. Une variable aléatoire (v.a.) $X$ à valeurs dans $\Omega'$ est une application de $\Omega$ dans $\Omega'$. Nous prenons souvent pour $\Omega'$ un sous-ensemble de $\mathbb{N}$ ou de $\mathbb{R}$.*

*On pourra munir $\Omega'$ d'une loi de probabilité $\mathbb{P}_X$ en posant, pour tout $\omega \in \Omega'$ : $\mathbb{P}_X(\omega) = \mathbb{P}\left(X^{-1}(\omega)\right)$.*

**Définition 16.7** — *La* fonction de répartition $F_X : \mathbb{R} \to [0,1]$ *d'une v.a. $X$ est définie par :*

$$F_X(x) = \mathbb{P}(X \leq x). \tag{16.3}$$

— *La* fonction de densité $p_X : \mathbb{R} \to [0,1]$ *d'une v.a.* $X$ *est définie par :*

$$p_X(x) = \mathbb{P}(X = x). \tag{16.4}$$

**Définition 16.8** — *La* fonction de répartition jointe $F_{X,Y} : \mathbb{R} \times \mathbb{R} \to [0,1]$ *de deux v.a.* $X$ *et* $Y$ *est définie par :*

$$F_{X,Y}(x,y) = \mathbb{P}(\{X \leq x\} \cap \{Y \leq y\}). \tag{16.5}$$

— *La* fonction de densité jointe $p_{X,Y} : \mathbb{R} \times \mathbb{R} \to [0,1]$ *de deux v.a.* $X$ *et* $Y$ *est définie par :*

$$p_{X,Y}(x,y) = \mathbb{P}(\{X = x\} \cap \{Y = y\}). \tag{16.6}$$

Ainsi

$$\mathbb{P}(Y = y) = \sum_x p(x,y).$$

Nous pouvons alors définir l'indépendance de deux v.a. $X$ et $Y$ :

**Définition 16.9** *Deux v.a.* $X$ *et* $Y$ *sont dites* indépendantes *si pour tout* $x, y \in \mathbb{R}$ *:*

$$p(x,y) = \mathbb{P}(X = x)\,\mathbb{P}(Y = y),$$

*ou encore,*

$$\mathbb{P}(X = x \mid Y = y) = \mathbb{P}(X = x).$$

### 16.0.4 Espérance mathématique

Une v.a. admet un certain nombre de paramètres. Nous considérons dans la suite les v.a. à valeurs dans $\mathbb{R}$. En arithmétique, la moyenne de $n$ nombres est définie par leur somme divisée par $n$. En calcul des probabilités, l'espérance d'une v.a. est définie comme la somme des valeurs prises pondérées par les probabilités respectives, c'est-à-dire :

$$\mathbb{E}(X) = \sum_{x \in X(\Omega)} x\mathbb{P}(X = x), \tag{16.7}$$

lorsque cette somme converge absolument. ($X(\Omega)$ est l'ensemble des valeurs prises par la v.a. $X$). Sinon, on dit que $X$ n'admet pas d'espérance.

**Exemple.** Soit $K_2$ le graphe complet de taille 2. On considère que le graphe est anonyme. On considère que les deux sommets exécutent l'algorithme suivant afin de briser la symétrie qui résulte de l'anonymat : chaqu'un des sommets tire uniformément une valeur $b$ dans l'ensemble $\{0,1\}$, il envoie $b$ à l'autre voisin. L'algorithme s'arête quand les deux sommets ont tiré deux valeurs différentes.

Soit $X$ la v.a. qui compte le nombre d'échanges avant que la symétrie ne soit brisée entre $u$ et $v$. Pour tout $k \geq 1$, nous avons :

$$\mathbb{P}(X = k) = \left(\frac{1}{2}\right)^k,$$

on en déduit alors que :

$$\begin{aligned} \mathbb{E}(X) &= \sum_{k \geq 1} k \left(\frac{1}{2}\right)^k \\ &= 2. \end{aligned}$$

Le calcul de l'espérance en utilisant la formule (16.7) suppose la connaissance de la distribution de la v.a. $X$. Dans la pratique, et en particulier pour l'analyse d'algorithmes, cette distribution est difficile à caractériser. La proposition suivante fournit un outil qui permet de contourner dans plusieurs cas ce problème.

**Proposition 16.10** *(Linéarité de l'espérance) Soient $X$ et $Y$ deux v.a. définies sur le même espace probabilisé dénombrable $(\Omega, \mathbb{P})$ et admettant toutes les deux une espérance. Soit $a \in \mathbb{R}$. Alors :*

$$\mathbb{E}(aX) = a\mathbb{E}(X),$$

$$\mathbb{E}(X+Y) = \mathbb{E}(X) + \mathbb{E}(Y).$$

**Proposition 16.11** *Si $X$ et $Y$ sont deux v.a. indépendantes admettant une espérance, alors la v.a. produit $XY$ admet une espérance et :*

$$\mathbb{E}(XY) = \mathbb{E}(X)\,\mathbb{E}(Y).$$

On dit que la v.a $X$ *domine* la v.a. $Y$ si, pour tout $z \in \mathbb{R}$, $\mathbb{P}(X > z) \geq \mathbb{P}(Y > z)$. La proposition suivante donne un ensemble de propriétés intéressantes :

**Proposition 16.12** *Soit $X$ et $Y$ deux v.a. admettant chacune une espérance mathématique.*

1. *Si $X$ domine $Y$, alors $\mathbb{E}(X) \geq \mathbb{E}(Y)$. Sous la même hypothèse, l'égalité est réalisée si et seulement si $X$ et $Y$ ont la même distribution.*
2. *$|\,\mathbb{E}(X)\,| \leq \mathbb{E}(|X|)$.*
3. *Pour toute v.a. $X$ à valeurs entière positives,*

$$\mathbb{E}(X) = \sum_{x=1}^{\infty} \mathbb{P}(X \geq x).$$

### 16.0.5 Moments

**Définition 16.13** *Soit $m$ un entier positif et soit $X$ une v.a. telle que $X^m$ possède une espérance. Le* moment d'ordre *$m$ de $X$ est l'espérance de la v.a. $X^m$, i.e., le nombre $\mathbb{E}(X^m)$. Le nombre $\mathbb{E}(X - \mathbb{E}(X))^m$, s'il existe, s'appelle le moment centré d'ordre $m$ de $X$ autour de son espérance.*

#### Variance

Soit $X$ une v.a. ayant un moment d'ordre 2 fini. La *variance* de $X$ est définie par :

$$\mathbb{V}ar(X) = \mathbb{E}(X - \mathbb{E}(X))^2. \tag{16.8}$$

De même que pour l'espérance, la propriété suivante est utile dans de nombreux calculs pratiques :

$$\mathbb{V}ar(X) = \mathbb{E}\left(X^2\right) - (\mathbb{E}(X))^2.$$

La racine carrée de la variance est appelée *écart-type* et est désignée par $\sigma$ :

$$\sigma(X) = \sqrt{\mathbb{V}ar(X)}.$$

**Remarque 16.14** *L'écart-type permet de caractériser la dispersion d'une v.a. autour de sa moyenne.*

Nous avons également la proposition suivante :

**Proposition 16.15** *Si $X$ et $Y$ sont deux v.a. indépendantes admettant chacune une variance, alors :*

$$\mathbb{V}ar\left(X+Y\right)=\mathbb{V}ar\left(X\right)+\mathbb{V}ar\left(Y\right),$$

*et pour tout $a\in\mathbb{R}$ :*

$$\mathbb{V}ar\left(aX\right)=a^{2}\mathbb{V}ar\left(X\right),\ \textit{et}\ \mathbb{V}ar\left(X+a\right)=\mathbb{V}ar\left(X\right).$$

### 16.0.6 Fonctions génératrices

**Définition 16.16** *Soit $X$ une v.a. à valeurs entières positives admettant $p_X$ comme fonction de densité. La* fonction génératrice de probabilités *de $X$ est définie par :*

$$G_X(z)=\mathbb{E}\left(z^X\right)=\sum_{k=0}^{\infty}p_X(k)z^k. \tag{16.9}$$

La somme dans la définition de $G_X(z)$ converge pour $|z|\leq 1$, nous supposons donc que la variable symbolique $z$ appartient toujours à l'intervalle $[-1,1]$.

**Proposition 16.17** *Soit $X$ une v.a. à valeurs entières positives admettant $G_X(z)$ comme fonction génératrice de probabilités. On a :*

1. $G(1)=1$.
2. $\mathbb{E}\left(X\right)=G'_X(1)$.
3. $\mathbb{E}\left(X^2\right)=G''_X(1)+G'_X(1)$.
4. $\mathbb{V}ar\left(X\right)=G''_X(1)+G'_X(1)-G'_X(1)^2$.

**Proposition 16.18** *Soient $X_1,X_2,\cdots,X_k$ des v.a. indépendantes admettant comme fonctions génératrices de probabilités respectivement $G_1(z),G_2(z),\cdots,G_k(z)$. La fonction génératrice de la v.a. $Y=\sum_{i=1}X_i$ est donnée par :*

$$G(z)=\prod_{i=1}^{k}G_i(z). \tag{16.10}$$

**Proposition 16.19** *Soit $X_1,X_2,\cdots$ une suite de v.a. indépendantes de même distribution admettant la même fonction génératrice de probabilités $G_X(z)$. Si $Y$ est une v.a. ayant pour fonction génératrice de probabilités $G_Y(z)$ et si $Y$ est indépendante de chacune des v.a. $X_i$, alors $S=X_1+X_2+\cdots+X_Y$ admet comme fonction génératrice de probabilités :*

$$G_S(z)=G_Y(G_X(z)). \tag{16.11}$$

**Définition 16.20** *Soit $X$ une v.a. ayant une densité $p$. La fonction génératrice des moments de $X$ est donnée par*

$$M_X(z)=\mathbb{E}\left(e^{zX}\right).$$

**Proposition 16.21**

$$\mathbb{E}\left(X^k\right)=M^{(k)}(z)\mid_{z=0}.$$

**Proposition 16.22** *Soient $X_1, X_2, \cdots, X_k$ des v.a. ayant pour fonctions génératrices de moments $M_1(z), M_2(z), \cdots, M_k(z)$. La fonction génératrice des moments de la v.a. $Y = \sum_{i=1}^{k} X_i$ est donnée par :*

$$M(z) = \prod_{i=1}^{k} M_i(z).$$

### 16.0.7 Quelques distributions usuelles

#### Loi de Bernoulli

La v.a. $X$ prend deux valeurs : 1 avec probabilité $p$ et 0 avec la probabilité $q$ ; on suppose que $p, q \in [0,1]$ et $p + q = 1$.

Propriétés :

$$\mathbb{E}(X) = p, \ \ \mathbb{V}ar(X) = pq, \ \ \sigma(X) = \sqrt{pq}.$$

Cette loi intervient souvent de façon implicite lorsqu'on veut traiter une probabilité comme une espérance. En effet c'est la loi de la v.a. qui est la fonction indicatrice d'un événement $A$ de probabilité $p$ :

$$\mathbb{P}(A) = \mathbb{E}(1_A).$$

Propriétés :

$$\mathbb{E}(X) = p, \ \ \mathbb{V}ar(X) = pq, \ \ G_X(z) = q + pz.$$

#### Loi binomiale

On effectue $n$ épreuves identiques et indépendantes ; la probabilité de succès dans chacune étant supposée égale à $p$ et celle d'échec à $q = 1 - p$. Posons $X$ la v.a. qui compte le nombre total de succès obtenus dans les épreuves. La v.a. $X$ peut prendre la valeur entière $k$ dans l'intervalle $[0, n]$ avec la probabilité :

$$p_k = \mathbb{P}(X = k) = \binom{n}{k} p^k q^{n-k}.$$

On dit alors que $X$ suit une loi binomiale de paramètres $n$ et $p$ et on note $BIN(n,p)$ cette loi.

Propriétés :

$$\mathbb{E}(X) = np, \ \ \mathbb{V}ar(X) = npq, \ \ G_X(z) = (q + pz)^n.$$

**Remarque 16.23** *Une loi binomiale de paramètres $p$ et $n$ est la somme de $n$ v.a. indépendantes et identiques de Bernoulli, chacune de paramètre $p$.*

#### Loi géométrique

Soit $p > 0$ la probabilité de succès dans une épreuve, et $q = 1 - p$. Nous répétons la même épreuve indépendamment jusqu'à l'obtention du premier succès. Soit $X$ la v.a. désignant le nombre d'épreuves effectuées. C'est une v.a. qui peut prendre la valeur $k$ (entier naturel non nul) avec la probabilité :

$$p_k = \mathbb{P}(X = k) = q^{k-1} p.$$

On dit que $X$ suit une loi géométrique de paramètre $p$.

Propriétés :

$$\mathbb{E}(X) = \frac{1}{p}, \ \ \mathbb{V}ar(X) = \frac{q}{p^2}, \ \ G_X(z) = \frac{pz}{1 - qz}.$$

**Remarque 16.24** *La loi géométrique est une loi* sans mémoire *: soient $k$ et $m$ deux entiers positifs, nous avons :*

$$\mathbb{P}(X = k + l \mid X > l) = \mathbb{P}(X = k).$$

**Loi de Poisson**

Cette distribution intervient dans l'étude du nombre d'événements intervenant dans un intervalle de temps (file d'attente). Une v.a. $X$ suit une loi de poisson de paramètre $\lambda$, si $X$ peut prendre la valeur $k \in \mathbb{N}$ avec la probabilité :

$$p_k = \mathbb{P}(X = k) = \frac{\lambda^k}{k!} e^{-\lambda}.$$

Propriétés :

$$\mathbb{E}(X) = \lambda, \ \ \mathbb{V}ar(X) = \lambda, \ \ G_X(z) = e^{\lambda(z-1)}.$$

**Remarque 16.25** *La loi de Poisson est une bonne approximation de la distribution binomiale $BIN(n, \lambda/n)$ pour $n$ assez grand.*

### 16.0.8 Inégalités usuelles

**Proposition 16.26** *(Inégalité de Markov) Soit $f$ une fonction réelle positive croissante et non nulle (sauf éventuellement en 0). Soit $X$ une v.a. sur $(\Omega, \mathbb{P})$ telle que la v.a. $f(\mid X \mid)$ admette une espérance. Alors, pour tout $t > 0$, on a :*

$$\mathbb{P}(\mid X \mid \geq t) \leq \frac{\mathbb{E}(f(\mid X \mid))}{f(t)}. \tag{16.12}$$

En posant $f(x) = x^2$, et en définissant la nouvelle v.a. $Y = X - \mathbb{E}(X)$, nous obtenons :

**Proposition 16.27** *(Inégalité de Bienaymé-Tchebicheff)*

$$\mathbb{P}(\mid X - \mathbb{E}(X) \mid \geq t) \leq \frac{\mathbb{V}ar(X)}{t^2}. \tag{16.13}$$

**Proposition 16.28** *(Inégalité de Chernoff) Soit $BIN(n, p)$ une v.a. binomiale de paramètres $n$ et $p$. Pour tout $0 < a < np$, nous avons*

$$\mathbb{P}(|BIN(n, p)|) < 2e^{-a^2/3np}. \tag{16.14}$$

### 16.0.9 Convergence d'une suite de v.a.

**Convergence en probabilité**

Une suite de v.a. $X_1, X_2, \cdots, X_n$ converge en probabilité vers la v.a. $X$ si et seulement si :

$$\lim_{n \to \infty} \mathbb{P}(|X_n - X| \geq \varepsilon) = 0.$$

**Convergence en loi**

Une suite de v.a. $X_1, X_2, \cdots, X_n$ converge en loi vers la v.a. $X$ si et seulement si :

$$\lim_{n \to \infty} \mathbb{P}(X_n \leq a) = \mathbb{P}(X \leq a).$$

## 16.1 Quelques formules et notations utiles

Cette section présente quelques rappels mathématiques, et inégalités utiles dans la suite.

### 16.1.1 Notations asymptotiques

Soient $f$ et $g$ deux fonctions de $\mathbb{R}$ dans $\mathbb{R}^+$. On dit que :
- $f(n) = O(g(n))$ si et seulement si il existe une constante $c > 0$ et un entier $n_0$ tels que $\forall n > n_0$, $f(n) \leq cg(n)$.
- $f(n) = \Omega(g(n))$ si et seulement si il existe une constante $c > 0$ et un entier $n_0$ tels que $\forall n > n_0$, $f(n) \geq cg(n)$.
- $f(n) = \Theta(g(n))$ si et seulement si il existe deux constantes $c_1 < c_2$ et un entier $n_0$ tels que $\forall n > n_0$, $c_1 g(n) \leq f(n) \leq c_2 g(n)$.
- $f(n) = o(g(n))$ si $f(n)$ est négligeable devant $g(n)$, c'est-à-dire, $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$. Dans ce cas, on dit également que $g(n) = \omega(f(n))$.
- $f(n) \sim g(n)$ si $f(n)$ est asymptotiquement équivalent à $g(n)$, c'est-à-dire, $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$.

**Remarque 16.29** *Les notation $O$ et $\Omega$ sont parfois étendues au cas où les inégalités correspondantes sont vraies pour une infinité de $n$.*

### 16.1.2 Quelques propriétés

Rappelons que pour tous $n, k$ tels que $n \geq k \geq 0$, le coefficient binomial $\binom{n}{k}$ donne le nombre de sous-ensembles différents ayant $k$ éléments que l'on peut former à partir d'un ensemble contenant $n$ éléments. Il est défini par :

$$\binom{n}{k} = \binom{n}{n-k} = \frac{n!}{k!(n-k)!}.$$

Si $k > n \geq 0$, on définit $\binom{n}{k} = 0$.

**Proposition 16.30** *(Formule de Stirling)*

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \frac{1}{12n} + O\left(\frac{1}{n^2}\right)\right). \tag{16.15}$$

Nous avons également les relations suivantes :

**Proposition 16.31** *Soit $n \geq k \geq 0$.*

1. $\binom{n}{k} \leq \frac{n^k}{k!}$.
2. *Si $n$ est assez grand alors* $\binom{n}{k} \sim \frac{n^k}{k!}$.
3. $\binom{n}{k} \leq \left(\frac{ne}{k}\right)^k$.
4. $\binom{n}{k} \geq \left(\frac{n}{k}\right)^k$.

### 16.1.3 Quelques égalités ou inégalités utiles

Nous avons les développements suivants :

$$e^z = 1 + z + \frac{z^2}{2!} + \frac{z^3}{3!} + \cdots \tag{16.16}$$

$$\ln(1+z) = z - \frac{z^2}{2} + \frac{z^3}{3} - \frac{z^4}{4} + \cdots \tag{16.17}$$

$$\frac{1}{1-z} = 1 + z + z^2 + \cdots \tag{16.18}$$

**Proposition 16.32** *1. Pour tout $t \in \mathbb{R}$ :*

$$e^t \geq 1 + t. \tag{16.19}$$

*L'égalité est réalisée si $t = 0$.*

*2. Pour tous $t, n \in \mathbb{R}$, tels que $n \geq 1$ et $|t| \leq n$ :*

$$e^t \left(1 - \frac{t^2}{n}\right) \leq \left(1 - \frac{t}{n}\right)^n \leq e^t. \tag{16.20}$$

*3. Pour tous $t, n \in \mathbb{R}^+$ :*

$$\left(1 + \frac{t}{n}\right)^n \leq e^t \leq \left(1 + \frac{t}{n}\right)^{n + t/2}. \tag{16.21}$$

Étant donnés $n$ réels strictement positifs $x_1, x_2, \cdots, x_n$. Nous avons l'inégalité suivante dite Inégalité arithmético-géométrique :

$$\frac{1}{n}\left(\sum_{i=1}^{n} x_i\right) \geq \left(\prod_{i=1}^{n} x_i\right)^{\frac{1}{n}}. \tag{16.22}$$

Nous avons également les égalités suivantes :

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \tag{16.23}$$

$$\sum_{i=1}^{n} i^2 \sim \frac{n^3-1}{3}. \tag{16.24}$$

$$\tag{16.25}$$

# Chapitre 17

# Coloration distribuée des sommets d'un graphe

## 17.1 Introduction

Soit $G = (V, E)$ un graphe simple non orienté. Une *coloration* des sommets de $G$ (coloration de $G$ dans la suite) est une fonction qui affecte une couleur $c(v)$ à chaque sommet $v$ de $G$ telle que pour toute arête $\{u, v\} \in E$, $c(u) \neq c(v)$.

Ce chapitre présente un algorithme distribué probabiliste simple et efficace permettant de réaliser une coloration pour tout graphe $G$ [MRSDZ10].

## 17.2 L'algorithme *Coloration*

L'algorithme opère en rounds. À la fin de chaque round, les sommets qui obtiennent leurs couleurs définitives arrêtent d'exécuter l'algorithme, ils sont alors supprimés du graphe avec leurs arêtes adjacentes. Les autres sommets continuent d'exécuter l'algorithme dans le graphe résiduel.

Formellement, chaque sommet $u$ maintient un ensemble $actifs_u$ de sommets voisins actifs, c'est-à-dire l'ensemble des sommets voisins non encore coloriés et avec lesquels la symétrie n'est pas encore brisée. Initialement, $actifs_u$ est égale à $N(u)$. La couleur $couleur_u$ d'un sommet $u$ est un mot de bits initialisé au mot vide. À chaque round, $u$ génère un bit $b_u$, le concatène à $couleur_u$ (initialement mot vide) et envoie $b_u$ à tous les sommets dans l'ensemble $actifs_u$. Il reçoit ensuite les bits envoyés par ses voisins encore actifs et met à jour l'ensemble $actifs_u$. Le sommet $u$ répète cet ensemble d'actions jusqu'à ce que la symétrie soit brisée avec *tous ses voisins*, sa couleur est alors le mot $couleur_u$. La figure 17.2 illustre une exécution de l'algorithme.

**Remarque 17.1** *La couleur d'un sommet $u$ est la concaténation de tous les bits générés par $u$ depuis le début de l'exécution de l'algorithme. Cette couleur peut donc être interprétée comme un entier.*

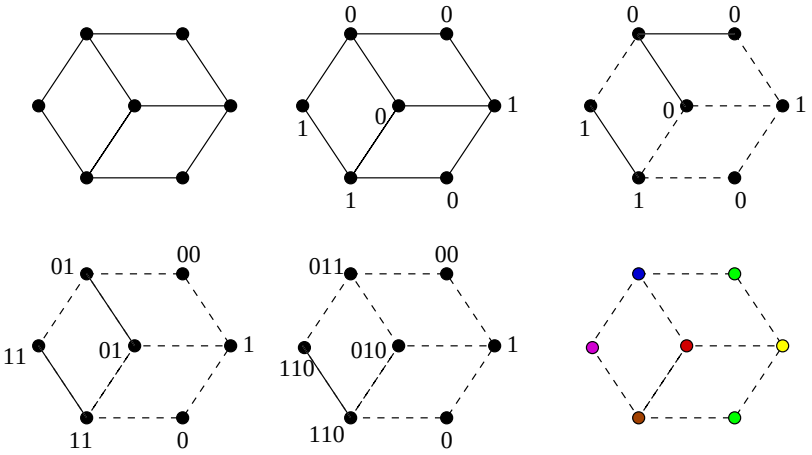



FIGURE 17.1 – Une exécution de l'algorithme *Coloration* : initialement, tous les sommets sont dans le même état. À chaque tirage, les arêtes (en pointillés) pour lesquelles les deux extrémités ont tiré des bits différents (la symétrie est brisée) sont retirées du graphe. À la fin de l'algorithme (quand toutes les arêtes sont supprimées) les mots sur chaque sommet sont interprétés comme des couleurs.

## 17.3 Analyse de l'algorithme

**Espérance du temps d'exécution**

Chaque sommet actif $u$ génère un bit $b_u$, envoie $b_u$ à ses voisins actifs et reçoit $b_v$ de chaque voisin actif $v$. Si $b_u \neq b_v$, alors $u$ met à jour l'ensemble des sommets actifs en supprimant $v$ de l'ensemble $actifs_u$. En terme de graphe, cela revient à supprimer l'arête $\{u, v\}$ du graphe $G$. La probabilité que l'événement $b_v \neq b_u$ se produise est égale à $1/2$. Nous en déduisons :

**Lemme 17.2** *Soit $(G_i)_{i\geq 0}$ la suite des graphes résiduels obtenus au cours de l'exécution de l'algorithme Coloration. L'espérance mathématique du nombre d'arêtes supprimées du graphe résiduel $G_i$ après le $(i+1)^{ieme}$ round est égale à la moitié du nombre de ses arêtes.*

**Preuve :** La preuve utilise la linéarité de l'espérance (voir 16.10). En effet, soit $X$ le nombre total d'arêtes supprimées du graphe à la fin d'une phase. Cette v.a. peut être écrite comme somme de v.a. $X_{\{u,v\}}$ comme suit :

$$X = \sum_{\{u,v\}\in E} X_{\{u,v\}},$$

où $X_{\{u,v\}} = 1$ si et seulement si $b_u \neq b_v$, c'est-à-dire, si la symétrie est brisée entre $u$ et $v$. Or il est facile de voir que $X_{\{u,v\}}$ est une v.a. de Bernoulli de paramètre $1/2$ et donc d'espérance $1/2$ (voir 16.0.7). On en déduit que :

$$\begin{aligned} \mathbb{E}(X) &= \sum_{\{u,v\}\in E} \mathbb{E}\left(X_{\{u,v\}}\right) \\ &= \sum_{\{u,v\}\in E} \frac{1}{2} \\ &= |E|/2. \end{aligned}$$

□

**Algorithme 22:** L'algorithme *Coloration*

```
1:  var :
2:      couleur_v : mot Init mot-vide ;
3:      actifs_v : ⊆ N(v) Init N(v) ;
4:      b_v : ∈ {0,1} ;
5:  Tant que actifs_v ≠ ∅ faire
6:      b_v ← flip(0,1) ;
7:      couleur_v ← b_v ⊕ couleur_v ;
8:       Pour tout u ∈ actifs_v Faire
9:          envoyer b_v à u ;
10:         recevoir b_u de u ;
11:         Si b_v ≠ b_u alors
12:             active_v ← active_v \ {u} ;
13:         Fin Si
14:      Fin Pour
15: Fin Tant que
```

On obtient le théorème suivant :

**Théorème 17.3** *Il existe deux constantes $k_1$ et $K_1$ telles que pour tout graphe $G$ à $n \geq 1$ sommets, le nombre de rounds nécessaires pour supprimer toutes les arêtes de $G$ est*

1. *inférieur à $k_1 \log n$ en moyenne,*
2. *inférieur à $K_1 \log n$ avec grande probabilité.*

**Preuve :**

— Le point 1 est une conséquence directe du lemme 17.2 et du fait que $\mid E \mid < n^2$.

— Soit $e = \{u, v\}$ une arête de $G$, et soit $r > 0$. L'arête $e$ n'est pas supprimée du graphe après $r$ itérations si et seulement si la symétrie entre $u$ et $v$ n'a pas été brisée pendant $r$ itérations. Autrement dit, pendant $r$ tirages, $u$ et $v$ ont tiré le même bit. Cet événement se produit avec probabilité $1/2^r$. Si $X_r$ désigne la v.a. qui compte le nombre d'arêtes encore dans le graphe après $r$ tirages, il est facile de voir que $X_r$ est une v.a. binomiale de paramètres $\mid E \mid$ et $1/2^r$ (voir 16.0.7). Son espérance est donc donnée par :

$$\mathbb{E}(X_r) = \frac{\mid E \mid}{2^r}.$$

Par ailleurs, en utilisant l'inégalité de Markov (voir 16.26), nous avons :

$$\mathbb{P}(X_r > 1) \leq \mathbb{E}(X_r). \tag{17.1}$$

En prenant $r \geq 4 \log_2 n - 1$, et en remarquant que $\mid E \mid \leq n^2/2$, l'inégalité (17.1) devient :

$$\begin{aligned} \mathbb{P}(X_r > 1) &\leq \frac{n^2}{2^{4 \log_2 n - 1 + 1}} \\ &= \frac{1}{n^2}. \end{aligned} \tag{17.2}$$

Or, l'algorithme termine dès que $X_r = 0$, ce qui se produit pour $r \geq 4 \log_2 n$ avec probabilité

$$\begin{aligned} \mathbb{P}(X_r = 0) &= 1 - \mathbb{P}(X_r = 0) \\ &\geq 1 - \frac{1}{n^2} \\ &= 1 - o\left(\frac{1}{n}\right). \end{aligned} \tag{17.3}$$

Il suffit donc de prendre $K_1 = 4$ pour obtenir le second point du théorème.

□

Le théorème suivant résume les résultats de cette section.

**Théorème 17.4** *L'algorithme Coloration calcule une coloration de tout graphe de taille $n$ en $O(\log n)$ rounds avec forte probabilité en n'utilisant que des messages de 1 bit.*

## 17.4 Réduction du nombre de couleurs

L'algorithme *Coloration* permet d'obtenir une coloration de tout graphe $G$ de taille $n$ en $O(\log n)$ rounds en moyenne et avec forte probabilité. Cependant, le nombre de couleurs utilisées peut être élevé. La figure 17.2 montre comment ce nombre peut être assez grand : le nombre de couleurs utilisées est 6 alors que 2 suffisent.

Dans cette section, nous allons voir comment réduire ce nombre pour n'utiliser qu'au plus $\Delta + 1$ couleurs pour tout graphe $G$ de degré maximum $\Delta$.

L'algorithme que nous étudions ici opère en deux phases : la première consiste à exécuter l'algorithme *Coloration*, la deuxième se base sur les mots générés lors de la première phase afin de calculer un coloriage n'utilisant que $\Delta + 1$ couleurs.

### 17.4.1 L'algorithme $\Delta$-*Coloration*

L'algorithme $\Delta$-*Coloration* se décompose en 2 phases.

La première phase est l'algorithme *Coloration*. Les couleurs sont des mots de bits (la variable $couleur_v$ dans l'algorithme). Ces mots peuvent être intérprétés comme des codages binaires de nombres. Dans cette section, nous verrons que si on intérprete $couleur_v$ comme le codage binaire d'un nombre réel de l'intervalle $[0, 1]$, alors nous pouvons construire un coloriage utilisant un nombre réduit de couleurs. En l'occurrence, $\Delta + 1$ couleurs au maximum, où $\Delta$ est le degré maximum du graphe.

La deuxième phase, (l'algorithme $\Delta$-*Coloration*) opère comme suit : quand un sommet $v$ termine l'exécution de la phase 1 (l'algorithme *Coloration*), il obtient une couleur propre. Le but de l'algorithme $\Delta$-*Coloration* est d'affecter au sommet $v$ une couleur dans l'ensemble $\{0, 1, \cdots, d(v)\}$, où $d(v)$ est le degré de $v$. Pour tout sommet $v$, nous définissons $x(v)$ comme le nombre dont le codage binaire est donné par $0.b_1b_2...$ où $b_1$, $b_2$, ... sont les bits du mot $couleur_v$ suivis d'une suite infinie de bits aléatoires uniformes. Notons que les nombres $x(u)$ sont des v.a. indépendantes suivant une même loi.

Pour chaque voisin $u$ de $v$, si $x(v) < x(u)$, alors l'arête $\{u, v\}$ est orientée de $u$ vers $v$. Ensuite, un sommet $v$ qui n'a que des arcs sortants, et qui par conséquent a un nombre $x(v)$ plus petit que tout ses voisins, prend la plus petite couleur encore disponible dans l'ensemble $\{0, 1, \cdots, d(v)\}$, il informe alors ses voisins que cette couleur lui est affectée et donc elle leur est interdite (voir l'algorithme 23). La figure 17.4.1 donne un exemple d'exécution de l'algorithme.

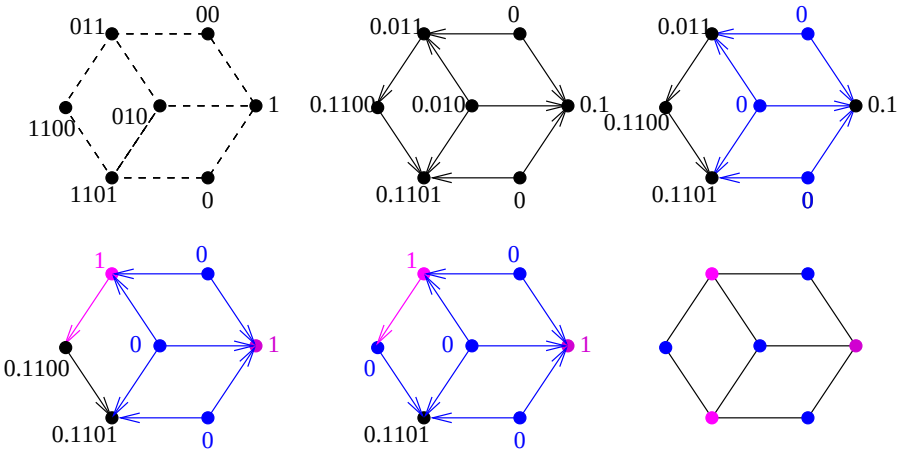



FIGURE 17.2 – Une exécution de l'algorithme $\Delta$-*Coloration*.

### 17.4.2 Analyse de l'algorithme

Nous avons le théorème suivant :

**Théorème 17.5** *Pour tout graphe $G = (V, E)$ de degré maximum $\Delta$, l'algorithme $\Delta$-Coloration réalise une $(\Delta + 1)-$coloration de $G$ en au plus $e\Delta + 3 \log n$ rounds avec forte probabilité.*

**Preuve :** Un sommet $v$ dont le nombre $x(v)$ vérifie la propriété $\{x(v) < x(u), \forall u \in N(v)\}$ prend la plus petite couleur encore disponible. Si le degré de $v$ est $d(v)$, cette couleur appartient à l'ensemble $\{0, 1, \cdots, d(v)\}$. Il en résulte que si $G$ est de degré maximum $\Delta$, alors le nombre de couleurs utilisées dans tout le graphe ne peut excéder $\Delta + 1$.

Soit $v$ un sommet quelconque et notons par $T_v$ la v.a. qui compte le nombre de phases avant que $v$ ne soit colorié. On peut remarquer que si $v$ n'est pas colorié après $t > 0$ phases, alors cela veut dire qu'il existe au moins une chaîne $P = \{v_0, v_1, \cdots, v_t\}$ telle que $v_0 = v$ et $x(v_0) > x(v_1) > \cdots > x(v_t)$. On en déduit que :

$$\begin{aligned} \mathbb{P}(T_v > t) &= \mathbb{P}(\exists P = \{v_0 = v, v_1, \cdots, v_t\} \mid x(v_0) > x(v_1) > \cdots > x(v_t)) \\ &\leq \sum_{P=\{v_0=v, v_1, \cdots, v_t\}} \mathbb{P}(x(v_0) > x(v_1) > \cdots > x(v_t)). \end{aligned} \tag{17.4}$$

Soit $P = \{v_0, v_1, \cdots, v_t\}$ un chemin dans $G$ tel que $v_0 = v$. Nous avons :

$$\mathbb{P}(x(v_0) > x(v_1) > \cdots > x(v_t)) = \frac{1}{(t+1)!}. \tag{17.5}$$

Par ailleurs, si $G$ est de degré maximum $\Delta$, alors le nombre maximum de chemins $P$ de longueur $t$ et ayant une extrémité égale à $v$ est majoré par $\Delta(\Delta - 1)^t$. L'expression (17.4) devient :

$$\begin{aligned} \mathbb{P}(T_v \geq t) &\leq \frac{\Delta(\Delta - 1)^{t-1}}{t!} \\ &\leq \frac{\Delta^t}{t!}. \end{aligned} \tag{17.6}$$

**Algorithme 23:** L'algorithme $\Delta$-*Coloration*.

**var :**
  $FC_u$ : entier **Init** $couleur_v$ ; (* couleur finale de $v$ *)
  $IN_v : \subset N(v)$ **Init** $\emptyset$ ;
  $OUT_v : \subset N(u)$ **Init** $\emptyset$ ;
  $Couleurs_v$ : ensemble de couleurs **Init** $\{0, 1, \cdots, d(v)\}$ ;
**Pour chaque** $v \in N(v)$
  **Si** $(x(u) < x(v))$ **alors**
    $OUT_v = OUT_v \cup \{v\}$ ;
  **Sinon**
    $IN_v = IN_v \cup \{v\}$ ;
  **Fin Si** ;
**Fin Pour**
**Tant que** $IN_v \neq \emptyset$ **faire**
  recevoir $< c, w >$ d'un voisin $w$ ;
  $IN_v = IN_v \setminus \{w\}$ ;
  $Colours_v = Colours_v \setminus \{c\}$ ;
**Fin Tant que**
$FC_v = \min\{Colours_v\}$ ;
**Pour chaque** $u \in OUT_v$
  envoyer $< FC_v, u >$ à $v$ ;
**Fin Pour** ;

La formule de Stirling donne $t! \sim \sqrt{2\pi t}\left(\frac{t}{e}\right)^t$, ce qui permet d'établir l'inégalité suivante :

$$\begin{aligned} \mathbb{P}(T_v \geq t) &\leq \left(\frac{e\Delta}{t}\right)^t \\ &= \left(1 - \frac{t - e\Delta}{t}\right)^t. \end{aligned} \tag{17.7}$$

Utilisons maintenant l'inégalité $(1-x)^k \leq e^{-kx},\ \forall x \in [0,1]$ pour obtenir :

$$\mathbb{P}(T_v \geq t) \leq e^{-(t-e\Delta)}. \tag{17.8}$$

Avec $t = e\Delta + 3\log n$, l'inégalité (17.8) devient :

$$\mathbb{P}(T_v \geq e\Delta + 3\log n) \leq \frac{1}{n^3}. \tag{17.9}$$

Si $T$ est la v.a. qui compte le nombre de phases avant que l'algorithme $\Delta$-*Coloration* soit terminé dans tout le graphe, nous avons :

$$\begin{aligned} \mathbb{P}(T \geq e\Delta + 3\log n) &= \mathbb{P}(\exists v \in V \mid T_v \geq e\Delta + 3\log n) \\ &\leq \sum_{v \in V} \mathbb{P}(T_v \geq e\Delta + 3\log n) \\ &\leq \frac{n}{n^3}. \end{aligned} \tag{17.10}$$

Il en résulte que $\mathbb{P}(T \geq e\Delta + 3\log n) = o\left(\frac{1}{n}\right)$. Ce qui termine la preuve du théorème. □

# Chapitre 18

# Synchronisation entre 2 sommets voisins dans un réseau

Dans de nombreux modèles pour les algorithmes distribués, un pas élémentaire de calcul se réalise après une synchronisation entre 2 sommets voisins : le modèle d'Angluin, le modèle de communication synchrone (modèle de Hoare) et d'une façon plus générale le modèle de réécriture d'arêtes étiquetées.

Le but de ce chapitre est de présenter et d'étudier un algorithme distribué probabiliste permettant de réaliser "régulièrement" une synchronisation entre 2 sommets voisins quelconques [MSZ03] d'une manière totalement décentralisée.

On verra également que cet algorithme permet de construire un couplage maximal.

## 18.1 L'algorithme *Rendez-vous*

Chaque noeud $v$ choisit au hasard et uniformément parmi ses voisins un sommet $w$, il lui envoie le bit 1 et il envoie le bit 0 aux autres voisins. Il y a rendez-vous entre deux sommets voisins $u$ et $v$ si et seulement si $u$ et $v$ se sont choisis mutuellement ($u$ envoie 1 à $v$ et $v$ envoie 1 à $u$, voir la figure 18.1). Plus formellement, la suite d'instructions est décrite plus loin par l'algorithme *Rendez-vous*.

Pour étudier les qualités de cet algorithme on peut essayer de calculer plusieurs paramètres :
— probabilité qu'il y ait au moins un rendez-vous à un tirage,
— étant donnés 2 sommets voisins $u$ et $v$, probabilité d'un rendez-vous entre $u$ et $v$,

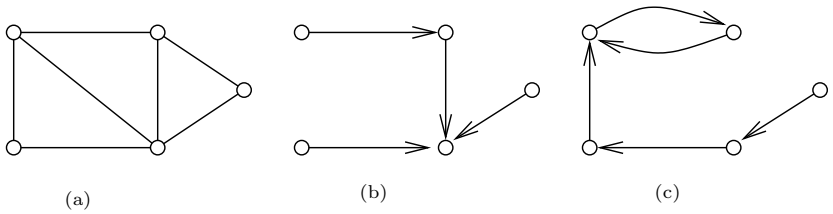



FIGURE 18.1 – (a) un graphe $G$, (b) une exécution de l'algorithme *Rendez-vous* sur $G$ ne donnant pas lieu à un rendez-vous, (c) une exécution de l'algorithme *Rendez-vous* sur $G$ donnant lieu à un rendez-vous (l'arc issu d'un sommet indique le choix de ce sommet).

**Algorithme 24:** L'algorithme *Rendez-vous*

1: Chaque sommet $v$ répète à l'infini :
2: le sommet $v$ choisit aléatoirement et uniformément un de ses voisins $c(v)$ ;
3: le sommet $v$ envoie 1 à $c(v)$ ;
4: le sommet $v$ envoie 0 à ses voisins différents de $c(v)$ ;
5: le sommet $v$ reçoit un message de chaque voisin ;
(* Il y a un rendez-vous entre $v$ et $c(v)$ si $v$ reçoit 1 de $c(v)$. *)

— temps moyen d'attente entre 2 rendez-vous pour 2 sommets donnés,
— étant donné un tirage, nombre moyen de rendez-vous.
probabilité

## 18.2 Probabilité de succès

**Définition 18.1** Soit $G = (V, E)$ un graphe. Un *appel* sur $G$ est une fonction $c$ de $V$ dans $V$ qui associe à chaque sommet un de ses voisins.

Soit $c$ un appel, par définition, un rendez-vous dans le graphe $G$ est un couple de sommets $(v, w)$ tels que $c(v) = w$ et $c(w) = v$.

Un appel $c$ sur $G$ est un *succès*, s'il contient au moins un rendez-vous, sinon, ce sera un *échec*.

On peut représenter un appel sur un graphe $G$ par le graphe orienté $G_c = (V, A)$, où $A$ est tel que pour tout $\{v, w\} \in E$, l'arc $(v, w)$ appartient à $A$ si et seulement si $w = c(v)$, voir la figure 18.1, partie (c).

Clairement, $G_c$ est un graphe orienté, sans cycle de longueur 1 dont tous les sommets sont de degré sortant 1. Il possède donc $n = |V|$ arcs. De plus, il est facile de voir que :

**Lemme 18.2** *Soit $c$ un appel sur le graphe $G$, $c$ est un échec si et seulement si $G_c$ est sans cycle de longueur 2.*

**Lemme 18.3** *Si $G$ est un arbre, alors tout appel sur $G$ est un succès.*

On suppose que tous les sommets adjacents à $v$ ont la même probabilité $\frac{1}{d(v)}$ d'être choisis, où $d(v)$ est le degré du sommet $v$. Ainsi, chaque arête $e = \{v, w\} \in E$ a une probabilité $\frac{1}{d(v)d(w)}$ d'être un lien sur lequel a lieu un rendez-vous entre $v$ et $w$. Dans la suite de ce chapitre, on suppose que les choix des sommets sont indépendants et sans mémoire.

Chaque sommet $v$ a $d(v)$ choix possibles, considérons maintenant la mesure de probabilité, qui associe à chaque appel sur $G$ la probabilité $\alpha(G)$ donnée par :

$$\alpha(G) = \prod_{v \in V} \frac{1}{d(v)}. \tag{18.1}$$

Soit $s(G)$ la probabilité de succès et $f(G)$ celle d'échec. Du lemme 18.2, on déduit que :

**Lemme 18.4** *Nous avons :*

$$f(G) = \alpha(G)N(G) \tag{18.2}$$

*et*

$$s(G) = 1 - \alpha(G)N(G), \tag{18.3}$$

*où $N(G)$ est le nombre d'appels $c$ sur $G$ pour lesquels $G_c$ n'a pas de cycle de longueur 2.*

Pour obtenir une expression exacte de la distribution de probabilité du nombre de rendez-vous pour un appel aléatoire, on considère les couplages.

Un couplage sur $G = (V, E)$ est un sous-ensemble $M$ de $E$ tel que pour toute paire d'arêtes $e$ et $e'$ de $M$, $e \cap e' = \emptyset$. On associe à un couplage $M$ le rendez-vous correspondant aux rendez-vous entre les extrémités des arêtes du couplage.

Soit $e = \{v, w\}$ une arête, soit $e^{(1)}$ l'événement correspondant à un rendez-vous sur $e$, et $e^{(0)}$ son complémentaire. La probabilité d'un rendez-vous sur $e$ est :

$$\mathbb{P}r(e^{(1)}) = \mathbb{P}r(\{v, w\}^{(1)}) = \frac{1}{d(v)d(w)}. \tag{18.4}$$

Soit $M = \{e_1, ..., e_k\}$ un couplage, de la même façon, la probabilité $\mathbb{P}r(M)$ d'avoir des rendez-vous sur $M$ est :

$$\mathbb{P}r(M) = \mathbb{P}r(e_1^{(1)} \wedge e_2^{(1)} \wedge ... \wedge e_k^{(1)}) = \prod_{\{v,w\} \in M} \frac{1}{d(v)d(w)} = \prod_{e \in M} \mathbb{P}r(e^{(1)}). \tag{18.5}$$

Pour un entier $k$, un *$k$-couplage* sur $G$ est un couplage de taille $k$. Soit $\mathcal{M}_k$ l'ensemble des $k$-couplages.

Soit :

$$q_k = \sum_{M \in \mathcal{M}_k} \mathbb{P}r(M),\ k = 0, 1, ..., \lfloor n/2 \rfloor. \tag{18.6}$$

Avec cette définition, on peut noter que $q_0 = 1$. Par une application directe du principe d'inclusion-exclusion, nous obtenons :

**Proposition 18.5** *Soit la suite $q_k, k = 0, 1, ..., \lfloor n/2 \rfloor$ comme définie ci-dessus pour un graphe connexe $G$ de taille $n$. Alors, pour tout entier $l$ entre 1 et $\lfloor n/2 \rfloor$, la probabilité d'obtenir au moins $l$ rendez-vous sur $G$ est :*

$$P_l = \sum_{l \leq i \leq \lfloor n/2 \rfloor} (-1)^{i+l} \binom{i}{l} q_i. \tag{18.7}$$

*En particulier, la probabilité de succès est :*

$$s(G) = P_1 = \sum_{0 \leq i \leq \lfloor n/2 \rfloor - 1} (-1)^i q_{i+1}. \tag{18.8}$$

Il est aussi possible de dériver des expressions simples pour la probabilité de succès $s(G)$ et l'espérance mathématique du nombre d'appels nécessaires pour obtenir un rendez-vous dans des classes spéciales de graphes. En effet :

**Exemple 18.6** Soit $G$ un cycle de taille $n \geq 2$. Le nombre $N(G)$, introduit dans le lemme 18.4, est égal à 2. Ainsi :

$$f(G) = \frac{1}{2^{n-1}}, \tag{18.9}$$

et

$$s(G) = 1 - \frac{1}{2^{n-1}}. \tag{18.10}$$

Le nombre moyen d'appels nécessaires pour obtenir un succès est donc :

$$(1 - \frac{1}{2^{n-1}}) + 2\frac{1}{2^{n-1}}(1 - \frac{1}{2^{n-1}}) + 3(\frac{1}{2^{n-1}})^2(1 - \frac{1}{2^{n-1}}) + ... \tag{18.11}$$

c'est-à-dire :

$$\frac{2^{n-1}}{2^{n-1} - 1}. \tag{18.12}$$

Il est intéressant de voir l'impact de l'ajout d'une nouvelle arête au graphe sur le comportement de la probabilité de succès. En effet, ce comportement n'est pas monotone :

- Si on ajoute une arête à un arbre, la probabilité d'avoir au moins un rendez-vous diminue.
- Le graphe $G$ de la figure 18.2, dû à H. Austinat et V. Diekert, montre que l'ajout d'une arête peut augmenter cette probabilité. En effet, pour ce graphe, nous avons $s(G) = 1156/1600 = 0.7225$. Soit $G'$ le graphe obtenu à partir de $G$ en ajoutant l'arête $\{1, 2\}$. Nous avons $s(G') = 4742/6400 = 0.7409....$

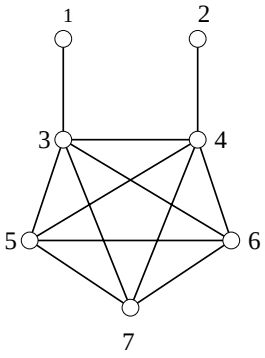



FIGURE 18.2 – Un exemple où l'ajout d'une arête augmente la probabilité d'obtenir au moins un rendez-vous.

Des résultats plus précis ont été obtenus pour des classes particulières de graphes. En effet, nous avons :

**Proposition 18.7** *Soit $G = (V, E)$ un graphe à degré d-borné, et $s(G)$ la probabilité de succès. Alors :*

$$s(G) \geq 1 - \left(1 - \frac{1}{d^2}\right)^{|E|}. \tag{18.13}$$

Le majorant ci-dessus devient très intéressant si le rapport $|E|$ sur $d$ est important. En effet, la formule ci-dessus montre que :

$$f(G) \leq e^{\frac{-|E|}{d^2}}. \tag{18.14}$$

En particulier, dans le cas des graphes $d$-réguliers, nous avons $|E| = \frac{nd}{2}$ et finalement :

**Corollaire 18.8** *Soit $G$ un graphe $d-$régulier, la probabilité d'échec $f(G)$ vérifie :*

$$f(G) \leq e^{\frac{-n}{2d}}. \tag{18.15}$$

Dans le cas où, $G$ est un graphe complet, nous avons :

**Proposition 18.9** *Soit $K_n$ le graphe complet de taille $n$, alors :*
- $s(K_n) = \sum_{k\geq 1}(-1)^{k+1}\frac{n!}{k!2^k(n-2k)!}\frac{1}{(n-1)^{2k}}$,
- $s(K_n)$ *est asymptotiquement* $1 - e^{-1/2}$,
- *et le nombre moyen d'appels nécessaires pour obtenir un succès vaut asymptotiquement* $\frac{\sqrt{e}}{\sqrt{e}-1}$.

La proposition 18.7 donne une borne inférieure pour la probabilité de succès si le graphe est à degré borné par $d$. Le corollaire 18.8 montre combien cette borne est importante si $d$ est suffisamment petit par rapport à $n = |V|$. Cependant, cette borne devient trop large si $d$ est trop grand et si $|E|$ n'est pas suffisamment grand. Il est donc intéressant de trouver une borne uniforme ne dépendant ni de $d$, ni de $|E|$. Cette section a pour objectif de donner un tel minorant. En effet, nous avons le théorème suivant :

**Théorème 18.10** *La probabilité $s(G)$ de succès dans un appel sur tout graphe $G = (V, E)$ est minorée par $1 - e^{-\overline{M}(G)}$, où $\overline{M}(G)$ est le nombre moyen de rendez-vous dans $G$.*

En utilisant le théorème 18.10 et la proposition 18.17, il vient :

**Corollaire 18.11** *La probabilité $s(G)$ d'un succès sur tout graphe $G = (V, E)$ est minorée par $1 - e^{-1/2}$.*

Et par conséquent :

**Corollaire 18.12** *Le nombre moyen d'appels nécessaires pour obtenir un succès est majoré par $\frac{\sqrt{e}}{\sqrt{e}-1}$.*

## 18.3 Nombre moyen de rendez-vous

Soit $X$ le nombre de rendez-vous pour un appel sur $G$, le nombre moyen de rendez-vous sur $G$, noté $\overline{M}(G)$, est $E(X)$ : c'est l'espérance mathématique de $X$. Ce paramètre peut être considéré comme une mesure du degré de parallélisme réalisé par l'algorithme de rendez-vous.

Pour tout arête $e \in E$, on définit $\chi_e$ par :

$$\chi_e = \begin{cases} 1 & \text{s'il y a rendez-vous sur } e, \\ 0 & \text{sinon.} \end{cases} \tag{18.16}$$

On a

$$X = \sum_{e \in E} \chi_e. \tag{18.17}$$

Donc :

$$E(X) = \sum_{e \in E} E(\chi_e). \tag{18.18}$$

Or

$$E(\chi_{\{v,w\}}) = \frac{1}{d(v)d(w)}, \tag{18.19}$$

donc :

$$E(X) = \sum_{\{v,w\} \in E} \frac{1}{d(v)d(w)}, \tag{18.20}$$

et finalement :

**Proposition 18.13** *Le nombre moyen de rendez-vous sur $G$ est :*

$$\overline{M}(G) = \sum_{\{v,w\} \in E} \frac{1}{d(v)d(w)}. \tag{18.21}$$

Considérons les cas particuliers suivants :

**Exemple 18.14** Si $G$ est le graphe complet de taille $n \geq 2$, alors

$$\overline{M}(G) = \binom{n}{2} \frac{1}{(n-1)^2} = \frac{n}{2(n-1)}. \tag{18.22}$$

**Exemple 18.15** Si $G$ est le cycle de taille $n \geq 3$, alors

$$\overline{M}(G) = \frac{n}{4}. \tag{18.23}$$

**Exemple 18.16** Si $G = (V, E)$ est de degré borné par $d$, alors

$$\overline{M}(G) \geq \frac{|E|}{d^2}. \tag{18.24}$$

Si on considère le cas d'un arbre $T$ de taille $n$ et de degré borné par $d$, nous obtenons :

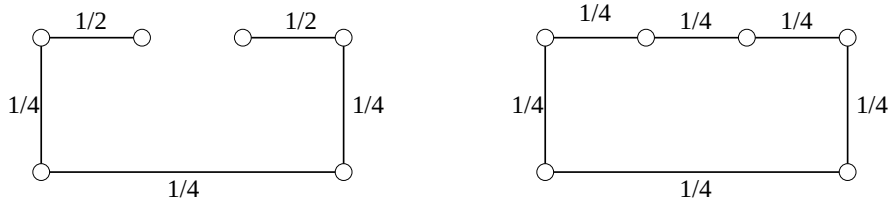
$$\overline{M}(T) \geq \frac{n-1}{d^2}. \tag{18.25}$$

Dans le cas des graphes réguliers de degré $d$ :

$$\overline{M}(G) = \frac{n}{2d}, \tag{18.26}$$

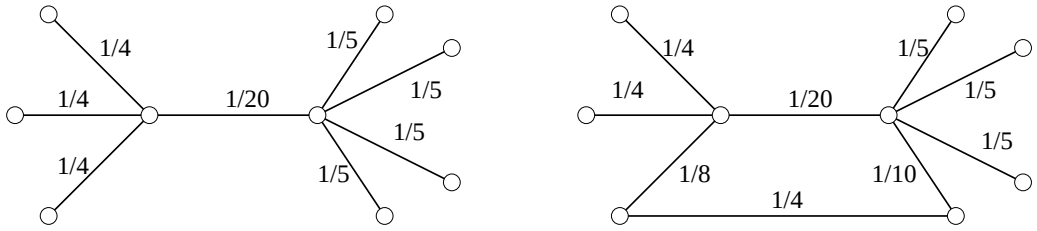
où $n$ est la taille du graphe.

Nous nous intéressons maintenant à l'impact de l'ajout d'une nouvelle arête sur le comportement de $\overline{M}(G)$. Les exemples suivants illustrent le fait que le nombre d'arêtes ne favorise pas nécessairement les rendez-vous. En effet, les figures 18.3 et 18.4 montrent que le nombre moyen de rendez-vous n'est pas monotone en fonction du nombre d'arêtes.




$\overline{M}(G) = 7/4$ $\overline{M}(G) = 3/2$

FIGURE 18.3 – Un exemple où le nombre moyen de rendez-vous diminue si on ajoute une arête.




$\overline{M}(G) = 8/5$ $\overline{M}(G) = 13/8$

FIGURE 18.4 – Un exemple où le nombre moyen de rendez-vous augmente si on ajoute une arête.

La proposition 18.17 donne un minorant du nombre moyen de rendez-vous.

**Proposition 18.17** *Pour un entier positif fixé $n$, le graphe complet $K_n$ minimise le nombre moyen de rendez-vous sur les graphes de taille $n$. La valeur moyenne minimale réalisée par $K_n$ est $\frac{n}{2(n-1)}$.*

**Preuve.** Soit $G = (V, E)$ un graphe, et $\overline{M}(G)$ le nombre moyen de rendez-vous sur $G$, nous avons :

$$\overline{M}(G) = \frac{1}{2} \sum_{v \in V} p(v), \tag{18.27}$$

où $p(v)$ désigne la probabilité que $v$ obtienne un rendez-vous.

Sachant que :

$$p(v) = \frac{1}{d(v)} \sum_{\{w,v\} \in E} \frac{1}{d(w)}, \tag{18.28}$$

et que $d(w) \leq n-1$, nous avons $p(v) \geq \frac{1}{n-1}$. En sommant sur tous les sommets, nous obtenons :

$$\overline{M}(G) \geq \frac{n}{2(n-1)}. \tag{18.29}$$

Or, dans le cas du graphe complet de taille $n$ :

$$\overline{M}(G) = \frac{n}{2(n-1)}. \tag{18.30}$$

D'où la proposition.

## 18.4 Construction d'un couplage maximal

Nous avons vu dans les sections précédentes que si les sommets d'un graphe $G = (V, E)$ exécutent la procédure *Rendez-vous*, l'ensemble des arêtes sur lesquelles ont lieu des rendez-vous forme un couplage dans $G$. Par ailleurs, un couplage $M$ est dit *maximal* si pour toute arête $e \in E \setminus M$, l'ensemble $M \cup \{e\}$ n'est pas un couplage. L'objectif de cette section est de montrer comment en itérant la procédure *Rendez-vous* on peut obtenir un couplage maximal.

**Remarque 18.18** *L'algorithme présenté ici n'est pas "le meilleur algorithme" en terme de complexité en temps. En effet, nous verrons dans la section suivante que sa complexité en moyenne, en général, est $O(m)$ phases. Or il existe un algorithme qui construit un couplage maximal en $O(\log n)$ phases, en moyenne. L'algorithme étudié ici est donc présenté dans un but pédagogique.*

On considère l'algorithme 25 désigné par l'algorithme $\mathcal{C}$.

L'algorithme opère par des phases (ou rounds) successives. Dans une phase, un sommet, encore dans le graphe, exécute la procédure *Rendez-vous* (avec ses voisins encore dans le graphe résiduel). Un sommet ayant "réussi" à établir un rendez-vous avec un voisin rentre dans le couplage maximal et en informe ses voisins. A la fin de chaque phase, les sommets ayant obtenu un rendez-vous sont retirés du graphe (avec toutes les arêtes adjacentes). L'algorithme s'arête lorsque toutes les arêtes sont retirées (tous les sommets sont soit dans le couplage maximal, soit des sommets isolés).

### 18.4.1 Analyse de l'algorithme $C$

#### Cas général

Soit $G = (V, E)$ un graphe à $n > 2$ sommets et $m > 0$ arêtes. Nous avons vu dans le corollaire 18.12 que si on note par $s(G)$ la probabilité d'avoir au moins un rendez-vous dans le graphe $G$, alors $s(G) > 1 - e^{-1/2}$. Cela permet d'obtenir la proposition suivante :

**Proposition 18.19** *Soit $T$ la v.a. qui dénote le nombre de phases nécessaires pour construire un couplage maximal. Nous avons :*

$$\mathbb{E}(T) = O(n).$$

**Algorithme 25:** L'algorithme $C$.

**var :**
$actifs_v$ : ensembles de sommets actifs **Init :** $N(v)$;
$etat_v$ : etat du sommet $v$ **Init :** $\perp$;
**répéter**
le sommet $v$ choisit, aléatoirement et uniformément, un de ses voisins actifs $c(v)$;
le sommet $v$ envoie 1 à $c(v)$;
le sommet $v$ envoie 0 à ses voisins actifs différents de $c(v)$;
le sommet $v$ reçoit les messages de tous ses voisins actifs;
**si** *($\exists w \in actifs_v$ tel que $c(v) = w$ et $c(w) = v$)* **alors**
$etat_v \rightarrow InC$;
envoyer $etat_v$ à tous les sommets dans $actifs_v$;
recevoir $etat_w$ de chaque sommets $w$ dans $actifs_v$;
$actifs_v \rightarrow actifs_v \setminus \{w \mid etat_w = InC\}$;
**si** $actifs_v = \emptyset$ **alors**
$etat_v \rightarrow NotInC$;
**jusqu'à** $etat_v = InC$ *ou* $etat_v = NotInC$;

**Preuve :** Pour tout $i \geq 1$, soit $G_i = (V_i, E_i)$ le graphe avant l'exécution de la phase $i$. Soit $X_i$ la v.a. qui compte le nombre d'arête dans $G_i$ et $Y_i$ le nombre d'arêtes supprimées du graphe à l'issue de la phase $i$. Nous avons :

$$X_{i+1} = X_i - Y_i, \tag{18.31}$$

il en résulte, par la linéarité de l'espérance, que :

$$\begin{aligned} \mathbb{E}(X_{i+1} \mid G_i) &= \mathbb{E}(X_i \mid G_i) - \mathbb{E}(Y_i \mid G_i) \\ &= X_i - \mathbb{E}(Y_i \mid G_i). \end{aligned} \tag{18.32}$$

Par ailleurs,

$$\begin{aligned} \mathbb{E}(Y_i \mid G_i) &\geq \mathbb{P}(Y_i \mid G_i \geq 1) \\ &= s(G_i) \\ &> 1 - e^{-1/2}. \end{aligned} \tag{18.33}$$

Donc

$$\mathbb{E}(X_{i+1} \mid G_i) < X_i - \left(1 - e^{-1/2}\right). \tag{18.34}$$

Définissons à présent la v.a. $Z_i$ par $Z_i = X_i + \left(1 - e^{-1/2}\right) * i$. Il est facile de vérifier l'inégalité suivante :

$$\mathbb{E}(Z_{i+1} \mid G_i) < Z_i. \tag{18.35}$$

On en déduit que

$$\begin{aligned} \mathbb{E}(Z_{i+1}) &= \mathbb{E}(\mathbb{E}(Z_{i+1} \mid G_i)) \\ &< \mathbb{E}(Z_i) \\ &< \mathbb{E}(Z_0) = m. \end{aligned} \tag{18.36}$$

Or

$$\mathbb{E}(Z_i) = \mathbb{E}(X_i) + \left(1 - e^{-1/2}\right) * i, \tag{18.37}$$

ce qui permet d'établir que :

$$\mathbb{E}(X_i) < m - \left(1 - e^{-1/2}\right) * i. \tag{18.38}$$

Cependant, il est facile de voir que $T < i$ si, et seulement si, $X_i < 1$. Il suffit donc de prendre $i > \frac{m}{1-e^{-1/2}}$ pour que $\mathbb{E}(X_i) < 1$. Ce qui termine la preuve. □

**Remarque 18.20** *La borne $O(m)$ est atteinte pour les graphes complets de taille $n$.*

### Graphes à degré maximum borné

Nous avons vu que l'algorithme $\mathcal{C}$ permet, pour tout graphe $G$ à $m$ arêtes, de construire un couplage maximal de $G$ en $O(m)$ phases. Nous allons voir dans cette section, que si on se limite aux graphes à degré borné par $\Delta$ (une constante), alors l'algorithme calcule un couplage maximal en $O(\log n)$ phases. En effet, nous avons le résultat suivant :

**Proposition 18.21** *Pour tout graphe $G = (V, E)$ à $n > 1$ sommets et $m > 0$ arêtes, et à degré maximum $\Delta$, l'algorithme $\mathcal{C}$ conctruit un couplage maximal dans $G$ en $O(\log n)$ en moyenne.*

**Preuve :** La preuve est simple. Il suffit de voir que si le degré maximum est borné, alors chaque phase supprime au moins une fraction (constante) du nombre d'arêtes dans le graphe résiduel. Ce qui est vrai : le nombre moyen de rendez-vous dans le graphe à l'issue d'une phase est égal à $\sum_{\{u,v\}\in E} \frac{1}{d(u)d(v)} \geq \frac{m}{\Delta^2}$. On en déduit alors que le nombre moyen de phases nécessaires pour supprimer toutes les arêtes du graphe est $O(\log m) = O(\log n)$. □

# Chapitre 19

# Élection distribuée probabiliste dans des boules de rayon 1 ou de rayon 2

Étant donné un entier positif $k$, une élection $k$-locale permet à partir d'une configuration initiale d'un réseau $(G, \lambda)$ d'atteindre une configuration $(G, \lambda')$ telle qu'il existe un ensemble non vide $\mathcal{E}$ de sommets vérifiant :

— chaque sommet de $\mathcal{E}$ est dans l'état *Elu*,

— pour chaque sommet $v$ de $\mathcal{E}$ et pour chaque sommet $w$ $(w \neq v)$ si $d(v, w) \leq k$ alors $w$ est dans l'état *non-Elu*.

On suppose que le réseau est anonyme, le système est asynchrone, les processeurs communiquent par messages en mode asynchrone et que le système est muni d'une numérotation des ports.

**Remarque 19.1** *Sous ces hypothèses, il n'existe pas d'algorithme déterministe autorisant une telle élection.*

Le problème de l'élection locale dans des boules de rayon 1 ou de rayon 2 est motivé par différents paradigmes ou applications dans les systèmes distribués :

— calculer un M.I.S. (maximal independent set) ou un 2-M.I.S.,

— implémentation totalement distribuée des calculs locaux sur des étoiles ou sur des étoiles ouvertes,

— plus généralement, déterminer une structuration initiale d'un réseau autorisant des processeurs à agir en parallèle.

Ce chapitre présente des algorithmes distribués probabilistes (du type Las Vegas) pour des élections 1-locale et 2-locale [MSZ03]. On montre ensuite qu'il n'existe pas d'élection $k$-locale probabiliste du type Las Vegas pour $k > 2$.

## 19.1 Algorithmes probabilistes d'élection 1-locale et 2-locale

Dans ce chapitre $K$ désigne un ensemble non vide muni d'une relation d'ordre total.

**$LR_1$ : Élection probabiliste 1-locale**

Chaque sommet $v$ répète à l'infini les actions suivantes :

— *Le sommet $v$ choisit un élément $rand(v)$ au hasard et uniformément dans l'ensemble $K$*;

— *le sommet $v$ envoie à ses voisins $rand(v)$*;

— *le sommet $v$ reçoit $rand(w)$ de chaque voisin $w$.*
*(* Le sommet $v$ est élu dans $B(v,1)$ si pour chaque voisin $w$ de $v$ : $rand(v) > rand(w)$. *)*

**$LR_2$ : Élection probabiliste 2-locale**

Chaque sommet $v$ répète à l'infini les actions suivantes :
— *Le sommet $v$ choisit un élement $rand(v)$ au hasard et uniformément dans l'ensemble $K$;*
— *le sommet $v$ envoie à ses voisins $rand(v)$;*
— *le sommet $v$ reçoit $rand(w)$ de chaque voisin $w$;*
— *quand il a reçu de chaque voisin un élément de $K$ il envoit à chaque voisin $w$ la valeur maximale qu'il a reçue de ses voisins différents de $w$;*
— *il reçoit de chaque voisin un élément de $K$.*
*(* Le sommet $v$ est élu dans $B(v,2)$ si $rand(v)$ est strictement plus grand que tous les éléments qu'il a reçus. *)*

**Remarque 19.2** *Si les procédures $LR_1$ et $LR_2$ sont utilisées pour implémenter les calculs locaux alors la terminaison globale va dépendre de la terminaison de l'algorithme codé par les calculs locaux et la détection de cette terminaison.*

## 19.2 Analyse probabiliste de $LR_1$ et $LR_2$

Si les procédures $LR_1$ and $LR_2$ sont utilisées pour implémenter les calculs locaux on doit savoir si un pas de calcul peut être exécuté quelque part, est-ce qu'un pas de calcul sera exécuté ? Quel est le temps moyen d'attente entre deux pas de calcul centrés sur un sommet donné ? Quel est le nombre moyen de pas de calcul pouvant être effectués en parallèle ?

On suppose que : $K = [0,1]$.

### 19.2.1 Probabilité pour un sommet d'être élu

Un sommet est élu localement par $LR_1$ (resp. par $LR_2$) si pour tout $w \in B(v,1) \setminus \{v\}$ (resp. $w \in B(v,2) \setminus \{v\}$), on a : $rand(w) < rand(v)$.

Si on suppose que les sommets choisissent uniformément et indépendamment un réel dans l'intervalle $[0,1]$, on a :

**Fait. 6** *La probabilité pour un sommet $v$ d'être élu localement dans un round de $LR_1$ est :*

$$p_1(v) = \frac{1}{d(v)+1}.$$

*(Où $d(v)$ est le degré du sommet $v$.)*

On rappelle que $d(v)+1$ est le cardinal de $B(v,1)$.

Pour un sommet $v$, on note $N_2(v)$ le nombre de sommets à distance au plus 2 de $v$ ($v$ compris), i.e., $N_2(v)$ est le cardinal de la boule $B(v,2)$.

**Fait. 7** *La probabilité pour un sommet $v$ d'être élu localement dans un round de $LR_2$ est :*

$$p_2(v) = \frac{1}{N_2(v)}.$$

On en déduit :

**Fait. 8** *Soit $v$ un sommet de $G$. Le nombre moyen de rounds entre deux élections successives d'un sommet $v$ par $LR_1$ (resp. $LR_2$) est : $d(v)+1$, (resp. $N_2(v)$).*

On obtient pour certaines classes de graphes.

**Exemple 19.3** *Si $G$ est un cycle de taille $n$ alors pour un sommet $v$ on a :*

$$p_1(v) = \frac{1}{3} \text{ and } p_2(v) = \frac{1}{5}.$$

*Le temps moyen d'attente pour un sommet $v$ d'être élu est 3 avec $LR_1$ et 5 avec $LR_2$.*

**Exemple 19.4** *Si $G$ est le graphe complet de taille $n$ alors pour tout sommet $v$ on a :*

$$p_1(v) = p_2(v) = \frac{1}{n}.$$

*Le temps moyen d'attente pour un sommet $v$ d'être élu par $LR_1$ ou par $LR_2$ est $n$.*

**Remarque 19.5** *On suppose que chaque sommet $v \in V$ choisit au hasard uniformément et indépendamment un entier $rand(v)$ dans $\{1, ..., N\}$. Soit $X \subseteq V$ un ensemble de sommets contenant un sommet $v$ donné. Soit $|X| = h$. Alors*

$$Pr\left(rand(v) > rand(w), \forall w \in X \setminus \{v\}\right) = \frac{1}{N} \sum_{i=2}^{N} \left(\frac{i-1}{N}\right)^{h-1}. \tag{19.1}$$

*On obtient les expressions pour $LR_1$ ou $LR_2$ en considérant pour $X$ les boules de rayon 1 ou 2.*

*Pour simplifier l'expression ci-dessus on suppose que $N$ est suffisamment grand pour que la probabilité de coincidence de rand dans $B(v,1)$ ou dans $B(v,2)$ soit négligeable. Cette hypothèse est équivalente à un tirage aléatoire d'un réel dans $[0,1]$.*

### 19.2.2 Nombre moyen de sommets élus à chaque round

Le nombre moyen de sommets élus à chaque round mesure le degré de parallélisme pour l'exécution d'un algorithme.

Soit $\overline{M_1}(G)$ (resp. $\overline{M_2}(G)$) le nombre moyen de sommets élus par $LR_1$ (resp. $LR_2$) dans un graphe $G$. En sommant $p_i(v), i = 1, 2$, sur $V$ dans les calculs précédents, on obtient :

**Fait. 9**

$$\overline{M_1}(G) = \sum_{v \in V} \frac{1}{d(v)+1},$$

*et*

$$\overline{M_2}(G) = \sum_{v \in V} \frac{1}{N_2(v)}.$$

Considérons de nouveau les cycles et les graphes complets.

**Exemple 19.6** *Si $G$ est un cycle de taille $n \geq 3$ alors :*

$$\overline{M_1}(G) = \frac{n}{3} \text{ et } \overline{M_2}(G) = \frac{n}{5}.$$

**Exemple 19.7** *Si $G$ est un graphe complet alors*

$$\overline{M_1}(G) = \overline{M_2}(G) = 1.$$

Étant donné un entier naturel $k$, on définit la *$k$-densite* du graphe $G = (V, E)$ par

$$\mathcal{D}_k(G) = \frac{\sum_{v \in V} d(v)^k}{|V|}.$$

**Théorème 19.8** *Soit $G = (V, E)$ un graphe connexe de taille $n \geq 2$ avec $m = |E|$. On a :*

$$\overline{M_1}(G) \geq \frac{n}{\mathcal{D}_1(G) + 1} = \frac{n^2}{2m + n},$$

*et*

$$\overline{M_2}(G) \geq \frac{n}{\mathcal{D}_2(G) + 1}.$$

Dans le cas des graphes de degré borné, on obtient :

**Corollaire 19.9** *Soit $G$ un graphe dont les sommets sont de degré inférieur ou égal à $\Theta$. alors :*

$$\overline{M_1}(G) \geq \frac{n}{\Theta + 1}$$

*et*

$$\overline{M_2}(G) \geq \frac{n}{\Theta^2 + 2\Theta + 1}.$$

Dans le cas des arbres, la première assertion du théorème devient :

**Corollaire 19.10** *Soit $T$ un arbre de taille $n \geq 2$. On a :*

$$\overline{M_1}(T) \geq \frac{n^2}{3n - 2} > \frac{n}{3}.$$

## 19.3 Quelques résultats d'impossibilité

Cette section présente des résultats d'impossibilité concernant l'existence d'algorithmes distribués probabilistes du type Las Vegas pour briser des symmétries à distance $k$ pour $k \geq 3$ dans des graphes anonymes.

Tout d'abord on a :

**Proposition 19.11** *Soit $k$ un entier naturel tel que $k \geq 3$. Il n'existe pas d'algorithme distribué probabiliste du type Las Vegas pour résoudre le problème de la coloration à distance $k$ ou le problème du $k$-M.I.S.*

**Preuve :** Par contradiction. Soit $k$ un entier naturel supérieur ou égal à 3. On suppose qu'il existe un algorithme distribué probabiliste du type Las Vegas $\mathcal{A}$ qui résoud le problème de la coloration à distance $k$. On considère un hexagone $H$ et un triangle $T$. Le graphe $H$ est un revêtement de $T$ via le morphisme $\varphi$ défini sur la figure 19.1.

Une exécution $(\mathbf{T}_i)_{0 \leq i \leq \ell}$ de $\mathcal{A}$ sur $\mathbf{T}$ (avec $\mathbf{T}_0 = \mathbf{T}$) induit une exécution $(\mathbf{H}_i)_{0 \leq i \leq \ell}$ de $\mathcal{A}$ sur $\mathbf{H}$ (avec $\mathbf{H}_0 = \mathbf{H}$) via $\varphi^{-1}$.

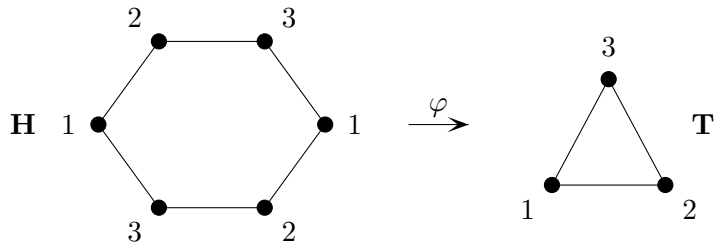



FIGURE 19.1 – Le graphe **H** est un revêtement du graphe **T**, à 2 feuillets, via l'homomorphisme $\varphi$ qui envoie un sommet de $H$ étiqueté $i$ sur un sommet de $T$ étiqueté $i$. (Les étiquettes de **H** et de **T** ne servent qu'à la définition de $\varphi$.)

Si un sommet $v$ de $T$ a une couleur finale $c$ alors il existe 2 sommets de $H$ avec cette couleur $c$, ce qui induit une contradiction.

En utilisant la même construction, on montre que s'il existe un algorithme distribué du type Las Vegas $\mathcal{A}$ qui résoud le problème du $k$-M.I.S. alors on aboutit à une contradiction : il existe dans $T$ un sommet dont l'état indique qu'il est dans le $k$-M.I.S. et donc 2 sommets dans $H$ sont dans le $k$-M.I.S. □

Le même type de construction permet de montrer :

**Lemme 19.12** *Il n'existe pas d'algorithme distribué du type Las Vegas permettant de détecter dans un graphe étiqueté si* 2 *sommets à distance au plus* 3 *ont la même étiquette.*

# Chapitre 20

# Quelques Algorithmes Probabilistes du type Monte Carlo

Cette section présente tout d'abord quelques résultats de non existence d'algorithmes probabilistes pour des réseaux anonymes sur lesquels on n'a aucune information. Elle se poursuit par la description et l'analyse de procédures de fractionnement et de nommage. Enfin on applique ces procédures aux problèmes du calcul d'un arbre recouvrant, du comptage du nombre de sommets d'un anneau et de l'élection [MRZ15].

On considère des réseaux anonymes et, plus généralement, partiellement anonymes (réseaux dans lesquels l'unicité des identités n'est pas garantie). On n'a aucune connaissance globale comme par exemple le nombre de sommets ou bien une borne supérieure de ce nombre. Les sommets n'ont aucune connaissance sur leur position respective ou distance.

Dans ce cadre, les seules solutions pour des problèmes distribués classiques comme la construction d'arbres recouvrant, le comptage du nombre de sommets ou l'élection sont des algorithmes probabilistes du type Monte Carlo. Ce chapitre présente des procédures distribuées probabilistes de fractionnement (division ou séparation) et de nommage qui sont des briques de base pour résoudre les problèmes ci-dessus.

## 20.1 Quelques Rappels

### 20.1.1 le modèle

Le modèle considéré est le modèle point-à-point avec passage de messages en mode asynchrone et une numérotation des ports. Le réseau est représenté par un graphe simple connexe $G = (V(G), E(G)) = (V, E)$ où les sommets correspondent aux processus et les arêtes aux liens de communication.

Un *algorithme probabiliste* est un algorithme qui fait des choix aléatoires, suivant une distribution, au cours de son exécution comme, par exemple, faire un tirage du type pile ou face ou bien tirer un entier au hasard dans un intervalle donné. Les algorithmes probabilistes permettent de fournir des solutions parfois plus "efficaces" que les solutions déterministes, ou encore des solutions pour des problèmes qui n'admettent pas de solution déterministe.

Un algorithme distribué probabiliste est un ensemble d'algorithmes probabilistes locaux ; de plus si le réseau est anonyme alors deux processeurs qui ont le même nombre de voisins exécutent des algorithmes identiques ayant la même distribution de probabilité.

Un algorithme (du type) Las Vegas est un algorithme probabiliste qui termine avec une probabilité positive (en général 1) et lorsqu'il se termine il produit un résultat correct.

Un algorithme (du type) Monte Carlo est un algorithme probabiliste qui se termine toujours mais le résultat peut être faux avec une certaine probabilité.

Les algorithmes que l'on présente ont une terminaison implicite : les processus sont passifs et les canaux de communication sont vides, mais a priori aucun sommet ne le sait.

On parle de terminaison explicite si au moins un sommet sait que l'exécution de l'algorithme est terminée.

Des résultats sur des graphes ayant $n$ sommets sont exprimés avec forte probabilité ce qui signifie avec probabilité $1 - o(n^{-1})$ (a.f.p. en abrégé) ou avec très forte probabilité ce qui signifie avec probabilité $1 - o(n^{-c})$ pour n'importe quel $c \geq 1$ (a.t.f.p. en abrégé).

On rappelle que $\log^* n = min\{i | \log^{(i)} n \leq 2\}$, où $\log^{(1)} n = \log n$ et $\log^{(i+1)} n = \log(\log^{(i)} n)$.

## 20.2 Résultats d'impossibilité

L'anonymat des réseaux et l'absence de connaissance globale font qu'il n'existe pas d'algorithme déterministe ou du type Las Vegas pour élire, pour compter le nombre de sommets ou bien pour calculer un arbre recouvrant. On rappelle qu'un algorithme distribué résoud le problème de l'élection si il se termine. De plus, on suppose qu'initialement chaque noeud est dans l'état *undetermined* et que dans la configuration finale exactement un processus est dans l'état *elected* et tous les autres sont étiquetés *non-elected*. Enfin on suppose que *elected* et *non-elected* sont des étiquettes terminales : une fois qu'elles sont apparues sur un sommet elles restent jusqu'à la fin de l'exécution de l'algorithme. Pour les graphes anonymes sans connaissance initiale on a :

**Théorème 20.1** *Il n'existe pas d'algorithme d'élection pour les anneaux correct avec une probabilité $\alpha > 0$.*

Ce théorème est une conséquence directe du théorème ci-dessous :

**Theorem 4.2** *Il n'existe pas d'algorithme avec une terminaison explicite pour calculer la taille d'un anneau qui est correcte avec une probabilité $\alpha > 0$.*

Ainsi dans la suite on utilise le terme élection en supposant que l'état *elected* peut ne pas être terminal.

## 20.3 Analyse d'une procédure de fractionnement et de nommage

Cette section présente et analyse une procédure probabiliste distribuée, appelée Splitting-Naming-whp (voir l'algorithme 26), qui associe à chaque sommet $v$ d'un graphe $G$ une étiquette $(t_v, id_v)$ définie de la façon suivante.

Chaque sommet $v$ tire uniformément au hasard (u.a.h. en abrégé) un bit $b_v$ jusqu'à ce que $b_v = 1$. On note $t_v$ le nombre de bits générés par $v$ : c'est la durée de vie de $v$.

Le nombre $id_v$ est obtenu en tirant un nombre choisi u.a.h. dans l'ensemble : $\{0, ..., 2^{t_v + 3\log_2(t_v)} - 1\}$.

On définit l'ordre, noté $<$, sur les couples par : $(t_v, id_v) < (t'_v, id'_v)$ si :

— $t_v < t'_v$

— ou $t_v = t'_v$ et $id_v < id'_v$.

**Algorithme 26:** Procedure Splitting-Naming-whp($v$).

1: $t_v := 0$
2: **repeat**
3: draw uniformly at random a bit $b(v)$ ;
4: $t_v := t_v + 1$
5: **until** $b(v) = 1$
6: choose uniformly at random a number $id_v$ in the set $\{0, \cdots, 2^{t_v + 3\log_2(t_v)} - 1\}$ ;

La suite de cette section est consacrée à l'analyse de la procédure Splitting-Naming-whp($v$), et plus précisément à l'analyse :

1. de la durée de vie de chaque sommet,
2. du nombre de sommets qui ont la durée de vie maximale,
3. du nombre de sommets qui ont la durée de vie maximale et ont tiré le plus grand numéro,
4. de la taille des étiquettes.

### 20.3.1 Analyse du nombre maximal de bits tirés par chaque sommet

On analyse tout d'abord la valeur de $t_v$ pour chaque sommet et la valeur maximale de $t_v$ parmi les sommets du graphe.

Chaque sommet a la probabilité 1/2 de tirer le bit 1. Ainsi $t_v$, pour chaque sommet $v$, est une variable aléatoire géométrique de paramètre 1/2. La durée de vie maximale est simplement $\max_{v \in V} t_v$, le maximum de $n$ variables aléatoires géométriques indépendentes et par conséquent :

**Proposition 20.2** *Soit $G = (V, E)$ un graphe ayant $n > 0$ sommets. On considère une exécution de la procédure Splitting-Naming-w.h.p. par les sommets de $G$. Soit $T$ la plus grande valeur de l'ensemble $\{t_v | v \in V\}$ ; $T$ satisfait les inégalités suivantes :*

1. $T < 2\log_2 n + \log^* n$ *a.f.p.,*
2. $T > \log_2 n - \log_2(2\log n)$ *a.f.p.*

**Preuve :** Initialement le nombre de sommets est égal à $n$. Donc après $t$ tirages aléatoires de bits l'espérance du nombre de sommets encore vivant est $\frac{n}{2^t}$. En particulier après $2\log_2 n + \log^* n$ tirages sa valeur est $\frac{1}{n 2^{\log^* n}}$ et donc la probabilité qu'un sommet survive est inférieure à $\frac{1}{n 2^{\log^* n}}$. Ce qui prouve la première inégalité.

Par ailleurs, pour tout $t > 0$, on a :

$$\begin{aligned} \mathbb{P}r(T > t) &= \mathbb{P}r(\exists v \in V \text{ tel que } t_v > t) \\ &= 1 - \left(1 - \frac{1}{2^t}\right)^n . \end{aligned} \tag{20.1}$$

Sachant que $(1-a)^n \sim e^{-an}$ quand $n \to \infty$, on a :

$$\mathbb{P}r(T > t) \sim 1 - e^{-\frac{n}{2^t}}, \quad \text{quand } n \to \infty. \tag{20.2}$$

En prenant $t = \log_2 n - \log_2(2\log n)$ dans (20.2), on obtient :

$$\mathbb{P}r(T > t) \geq 1 - o\left(\frac{1}{n}\right), \tag{20.3}$$

ce qui prouve la deuxième inégalité. □

**Remarque 20.3** *Dans la proposition 20.2, le terme* $\log^* n$ *peut être remplacé par n'importe quelle fonction (lentement) croissante non bornée.*

La Proposition 20.2 implique :

**Corollaire 20.4** *L'espérance de la valeur de* $T$ *est égale à* $\Theta(\log n)$.

**Preuve :** À la fin de chaque ronde, la moitié des sommets actifs deviennent inactifs. Donc $\mathbb{E}(T)$, la valeur de l'espérance de $T$, est $O(\log n)$.

De plus, en utilisant l'inégalité de Markov, on obtient :

$$\mathbb{E}(T) \geq (\log_2 n - \log_2(2\log n)) \times \mathbb{P}r(T > \log_2 n - \log_2(2\log n)). \tag{20.4}$$

Finalement, en utilisant la deuxième inégalité de la proposition 20.2, on a $\mathbb{E}(T) = \Omega(\log n)$. Ce qui termine la preuve. □

### 20.3.2 Analyse du nombre de sommets ayant la durée de vie maximale

Pour tout $t \geq 0$, $X_t$ désigne le nombre de sommets vivant à l'instant $t$. Pour tout $i > 0$, on a :

$$\mathbb{P}(X_{t+1} = 0 \mid X_t = i) = \frac{1}{2^i}. \tag{20.5}$$

Donc :

**Fait. 10**

$$\mathbb{P}(X_{t+1} = 0 \mid X_t \geq 2\log_2 n) \leq \frac{1}{n^2}. \tag{20.6}$$

On peut maintenant prouver :

**Proposition 20.5** *Le nombre de sommets qui ont une durée de vie maximale est, a.f.p., au plus* $2\log_2 n$.

**Preuve :** On considère la probabilité $C_t$ qu'à l'instant $t$, il y a plus que $2\log n$ sommets vivants et qu'ils disparaissent tous l'instant d'après. La somme de tous les $C_t$ est égale à la probabilité que plus de $2\log_2 n$ sommets ont la même durée de vie maximale. On a :

$$\begin{aligned} C_t &= \mathbb{P}r(\text{que le nombre de sommets vivants à l'instant } t > 2\log_2 n) \\ &\quad \times \mathbb{P}r(\text{que tous les sommets vivants disparaissent} \mid \text{ils sont } > 2\log_2 n) \\ &\leq \mathbb{P}r(\text{que tous les sommets vivants disparaissent } \mid \text{ils sont} > 2\log_2 n). \end{aligned}$$

Puis, en utilisant (20.6), on en déduit :

$$C_t \leq 2^{-2\log_2 n} = n^{-2}. \tag{20.7}$$

Par ailleurs :

$$\mathbb{P}(X_T \geq 2\log_2 n) \leq \mathbb{P}(T > 2\log_2 n) + \sum_{t \leq 2\log_2 n} C_t = o\left(\frac{1}{n}\right). \tag{20.8}$$

Ce qui termine la preuve.

□

### 20.3.3 Analyse du nombre de sommets qui ont la même durée de vie maximale et tirent le même nombre maximal

À la fin de la phase d'initialisation, chaque sommet $v$ dispose d'un entier $t_v$. Il choisit u.a.h. un nombre $id_v$ :

$$id_v \in \left\{0, \cdots, 2^{t_v+3\log_2(t_v)} - 1\right\}.$$

La propriété suivante est vérifiée :

**Proposition 20.6** *Avec forte probabilité, il existe un unique sommet $v$ avec l'étiquette $(t_v, id_v)$ tel que pour tout $w \in V \setminus \{v\}$ : $(t_v, id_v) > (t_w, id_w)$.*

**Preuve :** Soit $S = \{v_1, \cdots, v_k\}$ l'ensemble des sommets qui partagent la même durée de vie maximale. La deuxième inégalité de la proposition 20.2, implique qu'avec forte probabilité, pour tout sommet $v$ dans $S$ : $t_v > \log_2 n - \log_2(2\log n)$. Donc, chaque sommet $v$ dans $S$ choisira u.a.h. un nombre $id_v$ dans un ensemble qui contient a.f.p. l'ensemble :

$$\left\{0, \cdots, \frac{n}{2\log n} \times \left(\log_2\left(\frac{n}{2\log n}\right)\right)^3 - 1\right\}.$$

Soit $f(n) = \frac{n}{2\log n} \times \left(\log_2\left(\frac{n}{2\log n}\right)\right)^3$, un argument d'énumération implique que la probabilité pour $v$ d'être l'unique sommet avec le plus grand numéro est donnée par :

$$\begin{aligned}
\mathbb{P}r\left(id_v > id_w, \forall w \in S \setminus \{v\}\right) &\geq \sum_{i=1}^{f(n)-1} \frac{1}{f(n)} \times \left(\frac{i-1}{f(n)}\right)^{k-1} \\
&\sim \frac{1}{k}\left(1 - \frac{1}{f(n)}\right)^k \text{ quand } n \to \infty. \qquad (20.9)
\end{aligned}$$

Donc la probabilité qu'un unique sommet dans $S$ a le plus grand numéro est :

$$\mathbb{P}r\left(\exists v \text{ tel que } id_v > id_w, \forall w \in S \setminus \{v\}\right) \sim \left(1 - \frac{1}{f(n)}\right)^k \text{ quand } n \to \infty. \qquad (20.10)$$

Maintenant, de la proposition 20.5, on en déduit que $k < 2\log_2 n$, a.f.p., donc :

$$\begin{aligned}
\mathbb{P}r\left(\exists v \text{ tel que } id_v > id_w, \forall w \in S \setminus \{v\}\right) &\geq \left(1 - \frac{1}{f(n)}\right)^{2\log_2 n} \\
&\sim e^{-2\frac{\log_2 n}{f(n)}} \\
&= 1 - o\left(\frac{1}{n}\right).
\end{aligned}$$

Ce qui termine la preuve. □

### 20.3.4 Analyse de la taille des nombres tirés au hasard

Un sommet $v$ choisit u.a.h. un nombre $id_v$ dans l'ensemble :

$$\left\{0, \cdots, 2^{t_v+3\log_2(t_v)} - 1\right\}.$$

Donc ce nombre aléatoire a une taille d'au plus $2\log_2 n + O(\log_2 \log_2 n)$ bits a.f.p. De plus l'espérance de la valeur de $T$ est $\Theta(\log n)$ donc l'espérance de la valeur de la taille de $id_v$ est $O(\log n)$.

## 20.4 Analyse d'une variante de la procédure probabiliste de fractionnement et de nommage

Cette section présente et analyse une variante de la procédure Splitting-Naming-whp, elle est notée Splitting-Naming-wvhp (voir l'algorithme 27).

L'étiquette du sommet $v$ est le couple $(t_v, id_v)$ où $t_v$ est toujours la durée de vie de $v$. La différence réside dans le tirage de $id_v$ : le nombre $id_v$ est obtenu en le tirant u.a.h. dans l'ensemble : $\{0, ..., 2^{t_v \log^* t_v} - 1\}$.

**Algorithme 27:** Procedure Splitting-Naming-wvhp($v$).

1: $t_v := 0$
2: **repeat**
3: draw uniformly at random a bit $b(v)$ ;
4: $t_v := t_v + 1$
5: **until** $b(v) = 1$
6: choose uniformly at random a number $id_v$ in the set $\{0, ..., 2^{t_v \log^* t_v} - 1\}$ ;

### 20.4.1 Analyse du nombre maximal de bits tirés par chaque sommet

**Proposition 20.7** *Soit $G = (V, E)$ un graphe ayant $n > 0$ sommets. On considère une exécution de la procédure Splitting-Naming-wvhp. Soit $T$ la plus grande valeur de l'ensemble $\{t_v | v \in V\}$, $T$ vérifie :*

1. $T < (\log_2 n) \log^* n$. *a.t.f.p.*
2. $T > \frac{1}{2} \log_2 n$ *a.t.f.p.*

**Preuve :** En prenant $t = \frac{1}{2} \log_2 n$ dans (20.2), on obtient :

$$\begin{aligned} \mathbb{P}r\left(T > \frac{1}{2}\log_2 n\right) &\sim \quad 1 - e^{-\sqrt{n}}, \quad \text{quand } n \to \infty \\ &\sim \quad 1 - o\left(\tfrac{1}{n^c}\right) \text{ pour tout } c \geq 1. \end{aligned} \tag{20.11}$$

D'autre part, après $(\log^* n) \log_2 n$ rondes, l'espérance du nombre de sommets encore vivant est $\frac{n}{n^{\log^* n}}$ donc la probabilité qu'un sommet survive est bornée par $\frac{1}{n^{\log^* n - 1}}$ qui est $o(\frac{1}{n^c})$ pour tout $c \geq 1$. □

**Remarque 20.8** *Dans la proposition 20.7, comme dans la proposition 20.2, le terme* $\log^* n$ *peut être remplacé par n'importe quelle fonction (lentement) croissante et non bornée.*

### 20.4.2 Analyse du nombre de sommets qui ont le même temps de vie maximal et tirent le même nombre maximal

**Proposition 20.9** *Avec très forte probabilité, il existe un unique sommet $v$ avec l'étiquette $(t_v, id_v)$ tel que pour tout $w \in V \setminus \{v\}$ : $(t_v, id_v) > (t_w, id_w)$.*

**Preuve :** Le nombre $id_v$ est obtenu par un choix u.a.h. dans l'ensemble : $\{0, ..., 2^{t_v \log^* t_v} - 1\}$.

On rappelle que $S = \{v_1, \cdots, v_k\}$ est l'ensemble des sommets ayant eu une durée de vie maximale.

Donc, chaque sommet $v$ dans $S$ choisit u.a.h. un nombre dans un ensemble qui contient avec très forte probabilité l'ensemble :

$$\left\{0, \cdots, 2^{(\frac{\log_2 n}{2})(\log^*(\frac{\log_2 n}{2}))} - 1\right\}.$$

La probabilité qu'au moins 2 sommets avec la même valeur maximale $t_v = T$, tirent le même nombre est :

$$\begin{aligned} \mathbb{P}r\left(\exists u, v \in S \text{ tel que } u \neq v \text{ et } id_u = id_v\right) &< \frac{n^2}{2^{(\frac{\log_2 n}{2})(\log^*(\frac{\log_2 n}{2}))}} \\ &= o\left(\frac{1}{n^c}\right) \text{ pour tout } c \geq 1. \end{aligned}$$

Ce qui termine la preuve. □

### 20.4.3 Analyse de la taille des nombres aléatoires

**Proposition 20.10** *La taille des nombres $id_v$ est a.t.f.p. $O\left((\log n)(\log^* n)^2\right)$. La valeur de son espérance est $O\left((\log n)(\log^* n)\right)$.*

**Preuve :** La première partie est une conséquence directe de la proposition 20.7 et du choix de $id_v$ dans l'ensemble :

$$\left\{0, \cdots, 2^{T \log^* T} - 1\right\}.$$

Pour la deuxième partie, soit $S(id_v)$ la taille de $id_v$, et soit $S_{max} = \max_{v \in V} S(id_v) = T \log^* T$. On a :

$$\begin{aligned} \mathbb{E}\left(S_{max}\right) &= \sum_{x \geq 1} \mathbb{P}r\left(S_{max} \geq x\right) \\ &= \sum_{x=1}^{(\log n)(\log^* n)} \mathbb{P}r\left(S_{max} \geq x\right) + \sum_{x > (\log n)(\log^* n)} \mathbb{P}r\left(S_{max} \geq x\right), \end{aligned}$$

donc :

$$\mathbb{E}\left(S_{max}\right) \leq (\log n)(\log^* n) + \sum_{x \geq (\log n)(\log^*(\log n))} \mathbb{P}r\left(S_{max} \geq x\right). \tag{20.12}$$

Finalement, la seconde partie de la somme dans l'expression (20.12) satisfait :

$$\sum_{x \geq (\log n)(\log^*(\log n))} \mathbb{P}r\left(S_{max} \geq x\right) = \sum_{t \geq \log n} \sum_{y = t \log^* t}^{(t+1)\log^*(t+1)-1} \mathbb{P}r\left(S_{max} \geq y\right). \tag{20.13}$$

Par ailleurs, pour tout $y$ dans l'intervalle $[t\log^* t, (t+1)\log^*(t+1)-1]$ :

$$\begin{aligned}\mathbb{P}r\left(S_{max}\geq y\right) &= \mathbb{P}r\left(S_{max}\geq t\log^* t\right)\\ &= \mathbb{P}r\left(T\geq t\right)\leq\frac{n}{2^t},\end{aligned}\tag{20.14}$$

ce qui donne :

$$\sum_{x\geq(\log n)(\log^*(\log n))}\mathbb{P}r\left(S_{max}\geq x\right) \leq \sum_{t\geq\log n}\frac{n((t+1)\log^*(t+1)-t\log^* t)}{2^t}.\tag{20.15}$$

Sachant que $\log^*(t+1)\leq\log^* t+1$, on obtient :

$$\begin{aligned}\sum_{x\geq(\log n)(\log^*(\log n))}\mathbb{P}r\left(S_{max}\geq x\right) &\leq \sum_{t\geq\log n}\frac{n\left(t+\log^* t+1\right)}{2^t}\\ &= O\left(\log n\right).\end{aligned}\tag{20.16}$$

Donc :

$$\begin{aligned}\mathbb{E}\left(S_{max}\right) &\leq (\log n)(\log^* n)+O\left(\log n\right)\\ &= O\left((\log n)(\log^* n)\right),\end{aligned}\tag{20.17}$$

ce qui termine la preuve. □

## 20.5 Analyse du nombre de messages nécessaires pour diffuser la valeur maximale dans un réseau

Cette section analyse la complexité en messages de la diffusion de la plus grande valeur dans un réseau. Cette analyse est très utile pour l'analyse d'autres algorithmes.

Soit $G$ un graphe et $L$ un ensemble d'étiquettes totalement ordonné par la relation $>$. Chaque sommet $v$ de $G$ choisit une étiquette dans $L$ (on suppose que les sommets utilisent la même distribution pour choisir les étiquettes), la mémorise dans $max_v$ et l'envoie à ses voisins. Si un sommet $v$ reçoit une étiquette $l$ plus grande que $max_v$ alors il la mémorise dans $max_v$ et l'envoie à ses voisins. Ceci est décrit précisément dans l'algorithme 28.

Le nombre de messages engendrés par cette procédure vérifie :

**Proposition 20.11** *Soit $G$ un graphe et $L$ un ensemble totalement ordonné d'étiquettes. Chaque sommet $v$ de $G$ choisit une étiquette dans $L$ ; on suppose que tous les sommets utilisent la même distribution pour le choix des étiquettes. Chaque exécution de l'algorithme Broadcasting-Max engendre un nombre de messages égal à $O\left(\log n\right)$ a.f.p..*

**Preuve :** Soit $G$ un graphe ayant $n$ sommets et soit $v$ un sommet de $G$. Soit $max_v^{(i)}$ la valeur de $max_v$ à la $i$ième ronde de l'algorithme Broadcasting-Max. Par convention, $max_v^{(0)}=label_v$.

Soit $L_i=\{l\in L|\exists w\in V, label_w=l \text{ et } l>max_v^{(i)}\}$ et soit $X_i$ le cardinal de $L_i$. On a : $L_0\subseteq L$ et $X_0=\mid L_0\mid\leq n$.

**Algorithme 28:** Broadcasting-Max.

```
I : {If label_v is not defined}
begin
    choose at random label_v from L ;
    max_v := label_v;
    send < (label_v) > to all neighbours
end
B : {A Message (label) has arrived at v through port p}
begin
    if label_v is not defined then
        choose at random label_v from L ;
        max_v := label_v ;
        send < label_v > to all neighbours
    if label > max_v then
        max_v := label;
        send < (label) > to all neighbours except through port p
end
```

Par ailleurs, tous les sommets du graphe utilisent la même distribution pour choisir les étiquettes, donc pour toute paire de sommets $(u, w)$, on a :

$$\begin{aligned} \mathbb{P}r\left(max_u > max_w\right) &= \mathbb{P}r\left(max_w > max_u\right) \\ &= \mathbb{P}r\left(max_u \neq max_w\right)/2 \\ &\leq 1/2. \end{aligned} \tag{20.18}$$

Pour tout $i \geq 0$, soit $Y_i$ la v.a. telle que $Y_i = 1$ si $v$ envoie un message à la ronde $i$ et 0 sinon.

On définit également la suite $(k_j)_{j\geq 0}$ par $k_0 = 1$ et pour tout $j \geq 0$, $k_{j+1} = i - i'$ où $i' = \sum_{x=0}^{j} k_x$, $Y_i = Y_{i'} = 1$ et pour tout $i' < l < i$, $Y_l = 0$.

Pour tout $j > 0$, en utilisant (20.18), on obtient :

$$\mathbb{E}\left(X_{k_0+\cdots+k_j+k_{j+1}} \mid L_{k_0+\cdots+k_j}\right) \leq X_{k_0+\cdots+k_j}/2. \tag{20.19}$$

Maintenant, en définissant la v.a.. $(Z_j)_{j\geq 0}$ par $Z_j = 2^j X_{k_0+\cdots+k_j}$ pour tout $j \geq 0$, on a :

$$\mathbb{E}\left(Z_{j+1} \mid L_{k_0+\cdots+k_j}\right) = 2^{j+1}\mathbb{E}\left(X_{k_0+\cdots+k_j+k_{j+1}} \mid L_{k_0+\cdots+k_j}\right). \tag{20.20}$$

En utilisant (20.19) on obtient :

$$\mathbb{E}\left(Z_{j+1} \mid L_{k_0+\cdots+k_j}\right) \leq 2^j X_{k_0+\cdots+k_j} = Z_j. \tag{20.21}$$

Donc la v.a. $Z_j$ est une super-martingale, donc :

$$\mathbb{E}\left(Z_{j+1}\right) = \mathbb{E}\left(\mathbb{E}\left(Z_{j+1} \mid L_{k_0+\cdots+k_j}\right)\right) \leq \mathbb{E}\left(Z_j\right). \tag{20.22}$$

En conséquence : $\mathbb{E}\left(Z_i\right) \leq Z_0 \leq n$. Donc :

$$\mathbb{E}\left(X_{k_0+\cdots+k_j}\right) = \frac{1}{2^j}\mathbb{E}\left(Z_j\right) \leq \frac{n}{2^j}. \tag{20.23}$$

En prenant $j > j_0 = (3+\varepsilon)\log_2 n$ pour tout $\varepsilon > 0$, on obtient :

$$\mathbb{E}\left(X_{k_0+\cdots+k_j}\right) \leq \tfrac{1}{n^{2+\varepsilon}}. \tag{20.24}$$

L'inégalité de Markov implique :

$$\mathbb{P}r\left(X_{k_0+\cdots+k_j} > 1\right) \quad \leq \quad \frac{1}{n^{2+\varepsilon}}. \tag{20.25}$$

Donc la probabilité que certains sommets envoient plus que $(3+\varepsilon)\log n$ messages est $\frac{1}{n^{1+\varepsilon}}$ qui est $o\left(\frac{1}{n}\right)$. Ce qui termine la preuve.

□

**Corollaire 20.12** *Soit $G$ un graphe de degré borné ayant $n$ sommets et soit $L$ un ensemble d'étiquettes totalement ordonné. Chaque sommet $v$ de $G$ choisit une étiquette dans $L$ suivant une distribution donnée. La procédure Broadcasting-Max a une complexité en nombre de messages égale à $O(n\log n)$.*

## 20.6 Un algorithme Monte Carlo pour calculer un arbre recouvrant correct a.f.p. (resp. a.t.f.p.)

Chaque sommet $v$ est initialement étiqueté par l'étiquette $(t_v, id_v)$ engendrée par la procédure Splitting-naming-whp (resp. Splitting-naming-wvhp). Chaque sommet essaie de construire un arbre dont il est la racine. Si deux sommets sont en compétition pour capturer un troisième sommet $w$ alors $w$ sera attribué à l'arbre dont la racine a l'étiquette la plus forte. Cette étiquette est mémorisée dans $max_v$. Le père de $v$ est noté dans le numéro de port $father_v$ (par convention, si $root_v = true$ alors $father_v = 0$). Les fils de $v$ correspondent à l'ensemble $children_v$, les voisins de $v$ qui ne sont ni fils ni père de $v$ sont mémorisés dans l'ensemble $other_v$.

**Fait. 11** *Soit $G$ un graphe. Toute exécution de Spanning-Tree-whp termine, le réseau atteint alors une configuration terminale dans laquelle il n'y a plus de message en transit et aucune action ne peut se produire. La complexité en temps est $O(D)$, où $D$ est le diamètre du graphe.*

*Soit $(t_{max}, id_{max})$ le couple maximal parmi les étiquettes de $G$. Lorsque le réseau atteint une configuration terminale, l'ensemble des numéros $father_v$ définit une forêt couvrante, la racine de chaque arbre est étiqueté $(t_{max}, id_{max})$. Si il n'y a qu'un seul sommet $v$ tel que $(t_v, id_v) = (t_{max}, id_{max})$ alors la forêt est un arbre recouvrant de $G$ dont $v$ est la racine.*

**Remarque 20.13** *L'algorithme Spanning-Tree-wvhp est obtenu en substituant Splitting-Naming-wvhp à Splitting-Naming-whp dans l'algorithme Spanning-Tree-whp. Le résultat précédent est encore vrai pour l'algorithme Spanning-Tree-wvhp.*

Concernant les complexités, les sections précédentes impliquent :

**Proposition 20.14** *soit $G$ un graphe ayant $n$ sommets et $m$ arêtes. L'algorithme Spanning-Tree-whp (resp. l'algorithme Spanning-Tree-wvhp) calcule un arbre recouvrant a.f.p. (resp. a.t.f.p.). La taille des messages de l'algorithme Spanning-Tree-whp (resp. Spanning-Tree-wvhp) est $O(\log n)$ a.f.p., c'est également la valeur de l'espérance de la taille (resp. $O\left((\log n)(\log^* n)^2\right)$ a.t.f.p.) et la valeur de l'espérance est $O\left((\log n)(\log^* n)\right)$). La complexité en nombre de messages de ces deux algorithmes est a.f.p. $O(m\log n)$.*

**Algorithme 29:** Spanning-Tree-whp.

```
I : {If (t_v, id_v) is not defined}
begin
    call Splitting-Naming-whp(v) ;
    max_v := (t_v, id_v);
    root_v := true;
    father_v := 0; other_v := ∅; children_v := ∅;
    send < (t_v, id_v) > to all neighbours
end
D : {A Message (t, id) has arrived at v through port l}
begin
    if (t_v, id_v) is not defined then
        call Splitting-Naming-whp(v) ;
        max_v := (t_v, id_v);
        root_v := true;
        father_v := 0; other_v := ∅; children_v := ∅;
        send < (t_v, id_v) > to all neighbours
    if (t, id) > max_v then
        max_v := (t, id);
        root_v := false;
        father_v := l; children := ∅; other := ∅;
        send < (parent, max_v) > through l ;
        send < max_v > to all neighbours except through l
    else
        if (t, id) = max_v then
            send < (already, max_v) > through l
end
P : {A Message < (parent, m) > has arrived at v through port l}
begin
    if m = max_v then
        add l to children_v
end
A : {A Message < (already, m) > has arrived at v through port l}
begin
    if m = max_v then
        add l to other_v
end
```

## 20.7 Un algorithme Monte Carlo pour compter le nombre de sommets d'un anneau correcte a.f.p. (resp. a.t.f.p.)

L'idée principale est très simple : chaque sommet $v$ génère une étiquette $label_v$ en utilisant la procédure Splitting-naming-whp. Puis $v$ émet sur l'anneau un message contenant cette étiquette et un compteur égal à 1 initialement. Chaque sommet mémorise dans $max_v$ la plus grande valeur d'étiquette qu'il a reçu. Un message est rejeté si il est reçu par un sommet $u$ tel que $max_u$ est plus grand que lui. Si ce n'est pas le cas alors le compteur est incrémenté jusqu'à ce que le message soit reçu par un sommet $w$ ayant la même étiquette que $v$. Dans ce cas $w$ considère que le message est son propre message et que la valeur du compteur est le nombre de sommets de l'anneau ; en conséquence il émet dans l'anneau un nouveau message pour indiquer ce fait à chaque sommet $v$ qui mémorise la taille et l'étiquette si celle-ci est supérieure ou égale à son $max_v$.

**Fait. 12** *Soit $G$ un anneau. Chaque exécution de Counting-Ring-whp se termine : l'anneau atteint une configuration terminale sans message en transition et sans action possible. La complexité en temps est $O(n)$, où $n$ est le nombre de sommets de l'anneau.*

**Algorithme 30:** Counting-Ring-whp.

```
I : {If label_v is not defined}
begin
    call Splitting-Naming-whp(v) ;
    label_v := (t_v, id_v);
    max_v := label_v;
    send < (label_v, 1) > to Next_v
end
D : {A Message ((t, id), s) has arrived at v}
begin
    if label_v is not defined then
        call Splitting-Naming-whp(v) ;
        label_v := (t_v, id_v);
        max_v := label_v ;
        send < (label_v, 1) > to Next_v
    if (t, id) > max_v then
        max_v := (t, id);
        s := s + 1;
        send < ((t, id), s) > to Next_v
    else
        if (t, id) = max_v = (t_v, id_v) then
            size_v := s;
            send < (RES, max_v, s) > to Next_v
end
R : {A Message (RES, m, s) has arrived at v}
begin
    if label_v is not defined then
        call Splitting-Naming-whp(v) ;
        label_v := (t_v, id_v);
        max_v := label_v
        send < (label_v, 1) > to Next_v
    if m > max_v then
        size_v := s;
        max_v := m;
        send < (RES, m, s) > to Next_v
end
```

*Soit $(t_{max}, id_{max})$ le couple maximal parmi les étiquettes de $G$. Si il y a un unique sommet $v$ tel que $(t_v, id_v) = (t_{max}, id_{max})$ alors pour chaque sommet $w$ de $G$, $size_w$ est égal au nombre de sommets de $G$ et $max_w = (t_{max}, id_{max})$.*

**Remarque 20.15** *L'algorithme Counting-Ring-wvhp est obtenu en substituant Splitting-Naming-wvhp à Splitting-Naming-whp dans l'algorithme Counting-Ring-whp. Le fait 12 est encore vrai pour l'algorithme Counting-Ring-wvhp.*

Comme pour le calcul de l'arbre recouvrant, les complexités concernant l'algorithme Counting-Ring-whp (resp. l'algorithme Counting-Ring-wvhp) sont déduites immédiatement des sections précédentes et résumées par :

**Proposition 20.16** *Soit $G$ un anneau ayant $n$ sommets. L'algorithme Counting-Ring-whp (resp. l'algorithme Counting-Ring-wvhp) est correcte a.f.p. (resp. a.t.f.p.). La taille des messages de l'algorithme Counting-Ring-whp (resp. Counting-Ring-wvhp) est $O(\log n)$ a.f.p., c'est également la valeur de l'espérance de la taille (resp. $O\left((\log n)(\log^* n)^2\right)$ a.t.f.p.) et la valeur de son espérance est $O\left((\log n)(\log^* n)\right)$). La complexité en nombre de messages de ces deux algorithmes est a.f.p. $O(n \log n)$.*

## 20.8 Un algorithme d'élection du type Monte Carlo correcte a.f.p. (resp. a.t.f.p.)

### 20.8.1 Un algorithme d'élection du type Monte Carlo

Le but de l'algorithme *Elect-whp* (voir l'algorithme 31) est de désigner comme élu l'unique sommet $v$ (si il en existe un unique), tel que :

$$\forall \quad w \neq v \quad (t_w, id_w) < (t_v, id_v).$$

**Algorithme 31:** Elect-whp.

```
I : {If (t_v, id_v) is not defined}
begin
    call Splitting-Naming-whp(v) ;
    max_v := (t_v, id_v);
    leader_v := true;
    send < (t_v, id_v) > to all neighbours
end
D : {A Message (t, id) has arrived at v through port l}
begin
    if (t_v, id_v) is not defined then
        call Splitting-Naming-whp(v) ;
        max_v := (t_v, id_v);
        leader_v := true;
        send < (t_v, id_v) > to all neighbours
    if (t, id) > max_v then
        max_v := (t, id);
        leader_v := false;
        send < (t, id) > to all neighbours except through l
end
```

Les propriétés et complexités de cet algorithme sont déduites directement des sections précédentes.

## 20.9 Un anneau avec identités + la procédure de fractionnement = un algorithme d'élection avec une complexité en messages de $O(n \log n)$

Cette section donne un exemple d'utilisation de la procédure de fractionnement et illustre sa puissance grâce en particulier à la proposition 20.11.

Soit $R$ un anneau ayant $n$ sommets. On suppose que les sommets de $R$ connaissent $n$ et chaque sommet $v$ a une identité unique $ident_v$. On considère l'ordre sur les couples défini plus haut. Pour élire un sommet chaque sommet $v$ commence par exécuter la procédure Splitting obtenant $t_v$, puis il diffuse à travers l'anneau le triplet $(t_v, ident_v, hop)$, où $ident_v$ est l'identité de $v$ et $hop$ est un entier (sa valeur initiale est 1) ; cet entier est incrémenté lors de son déplacement d'un sommet à un autre. Chaque sommet mémorise la plus grande valeur qu'il voit passer. Un message est arrêté dès qu'il atteint un sommet $w$ ayant une valeur maximale supérieure au couple porté par le message. Le sommet élu est le sommet $u$ qui reçoit un message $(t, ident, h)$ avec $t = t_u$, $ident = ident_u$ et $h = n$. Initialement, pour chaque sommet $v$ $leader_v$ est is indéfini.

**Algorithme 32:** Procedure Splitting($v$).

```
1: t_v := 0
2: repeat
3:    draw uniformly at random a bit b(v) ;
4:    t_v := t_v + 1
5: until b(v) = 1
```

**Algorithme 33:** Elect-ring.

```
I : {If t_v is not defined}
begin
    call Splitting(v) ;
    max_v := (t_v, ident_v);
    send < (t_v, ident_v, 1) > to Next
end
D : {A Message (t, ident, h) has arrived at v}
begin
    if t_v is not defined then
        call Splitting(v) ;
        max_v := (t_v, ident_v);
        send < (t_v, ident_v, 1) > to Next
    if (t, ident) > max_v then
        max_v := (t, ident);
        send < (t, ident, h + 1) > to Next
    else
        if (t, id) = (t_v, ident_v) and h = n then
            Leader_v := true
end
```

**Proposition 20.17** *Soit $R$ un anneau ayant $n$ sommets tel que chaque sommet a une identité unique et a connaissance de $n$. L'algorithme Elect-ring termine et élit un sommet avec des messages de taille $O(\log n)$ a.f.p. et le nombre total de messages est a.f.p. $O(n \log n)$.*

**Preuve :** Deux sommets $v$ et $w$ de $R$ choisissent au hasard et avec la même distribution les éléments $t_v$ et $t_w$ de leur couple. La proposition 20.11 et l'ordre sur les couples impliquent que chaque sommet émet $O(\log n)$ messages relativement au premier élément du couple. La proposition est une conséquence directe de la proposition 20.5. □

## 20.10 Conclusion

On peut se demander si il est possible d'obtenir un algorithme du type Monte Carlo qui résoud le problème du comptage dans un anneau a.f.p. et assurer une probabilité d'erreur bornée par une constante $\varepsilon$. La réponse est positive.

Soit $R$ un anneau, soit $v$ un sommet de $R$ et soit $\varepsilon_v$ une constante ($\varepsilon_v < 1$) connue de $v$. Le sommet $v$ veut que la taille calculer par l'algorithme soit correcte avec probabilité $1 - \varepsilon_v$. La proposition 20.6 fournit une taille $K$ d'un anneau telle que la probabilité qu'une exécution de l'algorithme de comptage vu précédemment ne fournisse pas une étiquette maximale unique soit bornée par $\varepsilon_v$. Maintenant, au lieu de tirer un bit jusqu'à ce qu'il obtienne 1, le sommet $v$ répète

cette opération $K$ fois et il mémorise le temps de vie maximale, noté $t_v^{(K)}$, par les $K$ temps de vie obtenus lors des tirages. La suite de l'algorithme est la même que pour Counting-Ring-whp : le sommet $v$ tire au hasard un nombre $id_v$ dans l'ensemble $\{0, ..., 2^{t_v^{(K)}+3\log_2(t_v^{(K)})} - 1\}$ etc.

On obtient ainsi un algorithme du type Monte Carlo qui résoud le problème du comptage dans un anneau a.f.p. et avec une probabilité d'erreur bornée par la probabilité d'erreur requise par le sommet le plus exigeant.

On peut faire de même pour l'arbre recouvrant et l'élection.

# Chapitre 21

# Calcul des plus courts chemins entre tous les couples de sommets d'un graphe et applications

Ce chapitre présente et analyse un algorithme distribué permettant de calculer les plus courtes distances entre tout couple de sommets d'un graphe $G$ en un temps linéaire en le nombre de sommets avec des messages de taille constante. On en déduit des algorithmes optimaux pour calculer le diamètre de $G$, sa maille ou bien pour décider si une arête est un isthme etc. [MRZ16].

## 21.1 Le modèle

### 21.1.1 Le réseau

On considère le modèle à base de passage de messages. Il consiste en un réseau de communication en mode point à point décrit par un graphe $G = (V(G), E(G))$ $(= (V, E))$ où les sommets $V$ représentent les processus du réseau et les arêtes $E$ les canaux de communication bi-directionnels. On suppose le système synchrone : les processus démarrent en même temps et procèdent par rondes synchronisées.

### 21.1.2 La complexité en temps

Une ronde (un cycle) pour un processus est composée de trois étapes : 1. envoyer un message à certains voisins, 2. recevoir un message de certains voisins, 3. effectuer un calcul localement. La complexité en temps est le nombre maximal de rondes pour que tous les processus aient terminés leurs calculs.

### 21.1.3 La complexité en bits

Par définition, dans une b-ronde chaque sommet peut envoyer/recevoir au plus 1 bit à/de chaque voisin. La complexité en b-rondes d'un algorithme $\mathcal{A}$ est le nombre de b-rondes nécessaires pour l'exécuter.

Une ronde d'un algorithme se décompose en 1 ou plusieurs b-rondes. la complexité en b-rondes d'un algorithme distribué est une borne supérieure du nombre total de bits qui transitent par chaque canal de communication pendant son exécution. C'est également une borne supérieure

de sa complexité en temps. Si on considère un algorithme distribué dont les messages ont une taille en $O(1)$ alors la complexité en temps et la complexité en b-rondes sont égales modulo une constante multiplicative.

### 21.1.4 Connaissance du réseau et des processus

Le réseau $G = (V, E)$ est anonyme : on ne peut pas garantir l'unicité des identificateurs des processus. On suppose qu'un sommet est distingué, il sera noté Leader. On n'a aucune connaissance globale. On suppose que les processus peuvent distinguer leurs voisins (par des numéros de port par exemple).

### 21.1.5 Plus courts chemins entre deux sommets quelconques, diamètre, maille, isthme et point d'articulation

Soit $G = (V, E)$ un graphe, soient $u, v \in V$. La distance entre $u$ et $v$ dans $G$, notée $dist_G(u, v)$, est la longueur d'un plus court chemin entre $u$ et $v$ dans $G$.

Le problème des plus courts chemins entre deux sommets quelconques (PCC en abrégé) de $G$ est de calculer la longueur des plus courts chemins entre tous les couples de sommets de $G$.

Le diamètre de $G$, noté $D(G)$, est la distance maximale entre deux sommets quelconques de $G$, i.e., $D(G) = \max\{dist_G(u, v) \mid u, v \in V\}$.

La maille d'un graphe $G$ est la longueur d'un plus court cycle de $G$.

Un point d'articulation de $G$ est un sommet dont la suppression augmente le nombre de composantes connexes.

Un isthme de $G$ est une arête dont la suppression augmente le nombre de composantes connexes.

## 21.2 Plus courts chemins entre deux sommets quelconques

Cette section décrit les étapes de l'algorithme *PCC* qui calcule tous les plus courts chemins ; ces étapes sont :

1. calcul d'un arbre recouvrant de $G$ du type BFS dont la racine est *Leader* ;
2. chaque sommet calcule sa distance à *Leader* ;
3. énumération des sommets suivant un parcourt du BFS ;
4. chaque sommet calcule sa distance maximale à un sommet quelconque de $G$ à l'aide d'une onde anonyme dont l'émission se fait suivant l'ordre défini par l'énumération calculée précédemment.

Un arbre $T$ est un ensemble fini non vide de sommets tel que :

1. il existe un sommet particulier appelé racine de l'arbre, noté $root(T)$,
2. les autres sommets sont partitionnés en $m \geq 0$ sous-ensembles disjoints $T_1$,...,$T_m$ chacun de ces sous-ensembles étant un arbre. Les arbres $T_1$,...,$T_m$ sont les sous-arbres de la racine.

Soit $T$ un arbre, soit $v$ un sommet de $T$, le niveau de $v$ relativement à $T$ est défini récursivement par : le niveau de $root(T)$ est 0 et le niveau d'un autre sommet $v$, noté $level(v)$, est 1 de plus que le niveau de $v$ dans le sous-arbre le contenant.

On considère des arbres ordonnés (appelés également arbres plans) ; cela signifie que dans la définition ci-dessus l'ensemble des sous-arbres de chaque sommet est totalement ordonné par

l'ordre induit par la numérotation des ports. Ainsi, si on considère un sommet $v$ ayant $k$ sous-arbres on peut parler du premier sous-arbre, du second sous-arbre etc.

### 21.2.1 Calcul du BFS enraciné sur le sommet Leader

La première étape de l'algorithme $PCC$ est le calcul du BFS enraciné sur le sommet distingué Leader noté $BFS\text{-}ST$. Le calcul se fait niveau par niveau. Initialement chaque sommet est dans l'état *waiting*, puis :

1. Leader envoie le signal *Start* à ses voisins ;
2. un sommet dans l'état *waiting* recevant le signal *Start* d'au moins un voisin à l'instant $t$ fait :
   (a) il choisit pour père le premier voisin qui lui a envoyé le signal (le premier dans l'ordre des numéros de port par exemple) ;
   (b) à l'instant $t+1$, il envoie *Accept* à son père et *Reject* aux autres voisins ayant envoyé le signal *Start* ;
   (c) à l'instant $t+1$ il envoie *Start* à tous les autres voisins ;
   (d) à l'instant $t+2$ il enregistre comme fils tout voisin ayant envoyé le signal *Accept* et il envoie *Reject* aux voisins ayant envoyé le signal *Start* à l'instant $t+1$ ;
3. dès qu'un sommet différent de Leader a envoyé *Start* et a reçu $OK$ de tous ses fils (une feuille n'a pas de fils), il envoie $OK$ à son père ;
4. dès que Leader a envoyé *Start* et a reçu $OK$ de ses fils (c'est à dire de tous ses voisins), il sait que le calcul du BFS est terminé.

**Fait. 13** *Soit $G$ un graphe ayant $n$ sommets et un sommet distingué Leader. La procédure BFS calcule un arbre recouvrant de $G$ en largeur en un temps $O(n)$. Sa complexité en bits est également $O(n)$.*

On rappelle que les arêtes de $G$ qui ne figurent pas dans le BFS connectent des sommets dont la différence de niveau est au plus 1.

### 21.2.2 Calcul de la distance de Leader à chaque sommet

Dès que Leader détecte la terminaison du calcul de l'arbre, il met en oeuvre un algorithme permettant à chaque sommet de connaître sa distance à Leader.

Leader envoie un signal à chaque voisin lui indiquant que sa distance est 1 puis chaque sommet $v$ envoie à chaque fils un message indiquant que sa distance est 1 de plus que la sienne. Pour ce faire, on utilise une reprèsentation unaire des distances. Ce calcul est décrit par la procédure *Dist-Cal* definie par :

1. Leader envoie les messages 1 et *End* à chaque fils en deux étapes ;
2. un sommet recevant 1 envoie 1 à chaque fils à l'étape suivante ;
3. un sommet recevant *End* envoie 1 et *End* à ses fils dans les deux étapes suivantes ;
4. la distance d'un sommet à Leader est le nombre de 1 qu'il reçoit ;
5. dès qu'un sommet différent de Leader a envoyé *End* et reçu OK de tous ses fils (une feuille ayant zéro fils), il envoie OK à son père ;
6. dès que Leader a reçu OK de tous ses fils il sait que le calcul des distances est fini.

**Fait. 14** *Soit $G$ un graphe ayant $n$ sommets et un sommet distingué Leader. La procédure Dist-Cal permet à chaque sommet de connaître sa distance à Leader; le temps et le nombre de bits sont $O(n)$.*

**Remarque 21.1** *On considère le BFS enraciné en Leader donc le niveau d'un sommet est sa distance à la racine (Leader).*

### 21.2.3 une énumération des sommets

Une fois que Leader détecte la terminaison du calcul des distances, il met en oeuvre une phase au cours de laquelle chaque sommet va initier une onde qui va se propager à travers tout le graphe. Pour définir l'ordre dans lequel les ondes vont être émises on définit un parcours du BFS et une numérotation des sommets.

On définit tout d'abord un parcours du BFS, noté $Trav$, permettant de numéroter chaque sommet. Ce parcours et cette numérotation peuvent être définis itérativement comme suit.

On ajoute une boucle sur chaque feuille du BFS et les sommets sont visités deux fois dans un parcours du type DFS (une feuille est visitée en y arrivant puis en utilisant la boucle). Le parcours $Trav$ débute sur Leader et est défini par :

- si un sommet est visité pour la première fois alors aller à son premier fils si il a un fils ; si il n'a pas de fils (i.e., c'est une feuille) aller de la feuille à la feuille en utilisant la boucle ;
- si un sommet est visité pour la deuxième fois aller au frère suivant du sommet, si il a un frère ; si il n'a pas de frère aller au père du sommet si il a un père sinon on est sur la racine et le parcours est terminé.

Ainsi une visite de $v$ est immédiatement suivie d'une visite au fils de $v$ ou au frère de $v$ ou au père de $v$ ou à $v$ lui même (si $v$ est une feuille).

Une simple induction sur la taille des arbres montre :

**Lemme 21.2** *Le nombre de pas lors du parcours d'un arbre est impair.*

Le lemme suivant est obtenu par une induction sur le niveau des sommets :

**Lemme 21.3** *Un sommet est visité après un nombre pair de pas si et seulement si son niveau est pair.*

La position d'un sommet visité est le nombre de pas effectués avant sa visite. Les deux lemmes précédents impliquent :

**Corollaire 21.4** *Chaque exécution de $Trav$ visite chaque sommet $v$ deux fois, une fois en position paire (notée $s^e(v)$) et une fois en position impaire (notée $s^o(v)$).*

Le numéro d'un sommet $v$ est obtenu en comptant le nombre de pas effectués pendant le parcours. Plus précisément :

**Définition 21.5** *Le sommet numéroté $k$ est le $k$ième sommet visité en position paire ; il est noté $v_k$.*

Un exemple d'exécution de $Trav$ et la numérotation associée sont donnés dans Fig. 1.

Une induction sur le niveau d'un sommet permet de montrer :

**Lemme 21.6** *Le sommet $v_k$ est le $k$ième sommet $v$ tel que $k = s^e(v)/2 + 1$.*

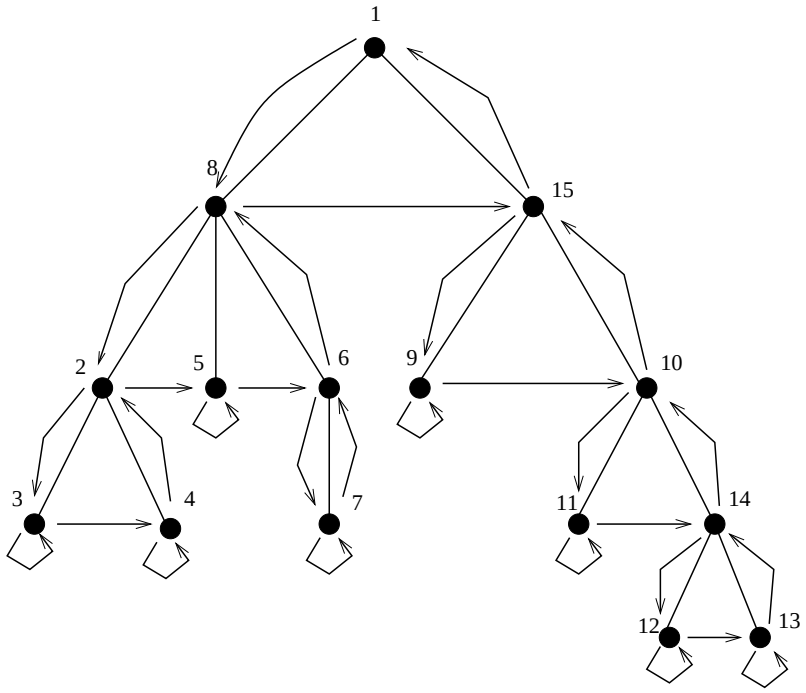



FIGURE 21.1 – Un exemple d'exécution de $Trav$ avec la numérotation asociée.

Nous pouvons maintenant énoncer et prouver la propriété fondamentale de la numérotation que nous utiliserons plus loin :

**Lemme 21.7** *La distance entre $v_i$ et $v_{i+1}$ est au plus* 3 *dans le BFS et donc dans $G$.*

**Preuve :** Deux sommets ayant des numéros consécutifs sont séparés par deux pas dans le parcours de l'arbre. De plus une visite de $v$ est immédiatement suivie d'une visite d'un fils ou d'un frère ou du père de $v$ ou de $v$ lui-même (si $v$ est une feuille). Donc la distance entre un sommet $v$ et le sommet $w$ atteint après deux pas est au plus 3. Ce qui prouve le lemme. □

La numérotation des sommets peut être réalisée en combinant le parcours DFS et l'envoie de messages 1 (utilisant la représentation unaire des nombres) et *End*. Plus précisément :

1. Leader envoie les messages 1 et *End* en deux étapes successives à son 1er fils ; le numéro de Leader est 1 ;
2. un sommet qui reçoit 1 envoie 1 à son successeur relativement au parcours *trav* (i.e., à un fils, à un frère, à son père ou à lui-même (pour une feuille)) à l'étape suivante ;
3. un sommet recevant *End* envoie 1 et *End* à son successeur relativement au parcours de l'arbre dans les deux étapes suivantes ;
4. un sommet $v$ reçoit 1 de 2 prédécesseurs ; soit $p_1$ et $p_2$ le nombre de 1 reçus par $v$ de ces deux prédécesseurs pour ce parcours ; un est pair et l'autre est impair. Supposons que $p_1$ est pair, le numéro de $v$ est alors $p_1/2 + 1$ ;
5. dès que Leader reçoit *End* de son dernier fils il en déduit que la numérotation est terminée.

Comme dans les sections précédentes on a :

**Fait. 15** *L'énumération des sommets a une complexité en temps et en bits égale à $O(n)$.*

**Remarque 21.8** *La complexité en temps et en bits pour calculer la taille de $G$ et en informer les sommets est $O(n)$.*

### 21.2.4 Calcul des distances à l'aide d'ondes

Dès que Leader détecte la terminaison de l'énumération des sommets, il initialise une phase au cours de laquelle chaque sommet, dans l'ordre de l'énumération, va émettre une onde anonyme. La propagation d'une onde suit les règles suivantes :

1. la source de l'onde envoie *wave* à chaque voisin ;
2. un sommet $v$ qui reçoit *wave* à l'instant $t$ d'un ou plusieurs voisins envoie *wave* à l'instant $t+1$ aux autres voisins ;
3. $v$ ignore tout signal *wave* reçu à l'instant $t+1$.

**Fait. 16** *Chaque sommet $v$ reçoit le signal wave à l'instant $d$ et éventuellement à l'instant $d+1$ après l'initialisation de l'onde où $d$ est la distance de la source à $v$.*

**Lemme 21.9** *Chaque sommet calcule sa distance à n'importe quel autre sommet.*

**Preuve :** Soit $t_1$ l'instant auquel la racine émet son onde. Chaque sommet connaît sa distance à la racine et peut ainsi calculer $t_1$ dès qu'il reçoit le signal *wave* pour la première fois.

Ensuite, chaque sommet $v_i$ émet une nouvelle onde (notée $w_i$) à l'instant $t_i = t_1 + 5(i-1)$.

Sachant que l'onde $w_{i+1}$ démarre 5 rondes après l'onde $w_i$ et que la distance entre $v_i$ et $v_{i+1}$ est au plus 3, ces deux ondes arrivent sur n'importe quel sommet $v$ séparément et dans l'ordre $w_i$ suivi de $w_{i+1}$ et à une distance au moins 2.

Soient $\tau_1$,...,$\tau_n$ les instants d'arrivée des $n$ ondes sur $v$. On a : $\tau_1 = t_1 + d(v, root)$. Or, $\tau_i = t_i + d(v, v_i)$ donc $d(v, v_i) = \tau_i - t_i = \tau_i - t_1 - 5(i-1)$, et $v$ peut calculer sa distance à $v_i$.

Finalement, quand aucune onde n'arrive sur un sommet $v$ (i.e., aucune nouvelle onde n'arrive après huit étapes suivant la dernière) $v$ connaît sa distance maximale à n'importe quel sommet. □

En résumé, le calcul des plus courts chemins entre deux sommets quelconques (PCC) se fait de la façon suivante :

1. Leader émet une onde à l'instant $t_1$ ;
2. chaque sommet calcule $t_1$ lorsqu'il reçoit la première onde ;
3. chaque sommet $v_i$ émet une onde à l'instant $t_i = t_1 + 5(i-1)$ ;
4. chaque sommet $v_i$ calcule sa distance à n'importe quel autre sommet $v_j$ lorsqu'il reçoit la $j$ième onde ;
5. chaque sommet $v$ calcule la distance maximale à n'importe quel autre sommet quand plus aucune onde n'arrive 8 étapes après l'arrivée de la dernière.

Finalement :

**Théorème 21.10** *Soit $G$un graphe ayant $n$ sommets et un sommet distingué. Il existe un algorithme synchrone qui calcule les longueurs des plus courts entre deux sommets quelconques de $G$ en $O(n)$ rondes avec une complexité en bits de $O(n)$.*

## 21.3 Deux applications simples de PCC

Cette section présente deux applications simples de l'algorithme $PCC$ : implémentation d'un schéma de routage suivant les plus courts chemins dans $G$ et calcul du diamètre de $G$.

### 21.3.1 Implémentation d'un schéma de routage suivant les plus courts chemins

On souhaite construire les schémas de routage de telle sorte que lorsqu'un sommet $v_i$ doit envoyer un message $M$ à une destination $v_j$ il sache à travers quel port de sortie il doit expédier $M$.

Ce port de sortie est connu de $v_i$ quand il reçoit l'onde issue de $v_j$ pour la première fois. Si $v_i$ reçoit simultanément des ondes à travers différents ports alors il en choisit un. Ainsi $G$ peut construire ses tables pendant l'exécution de $PCC$.

### 21.3.2 Calcul du diamètre

Cette section indique comment calculer le diamètre de $G$ en centralisant la distance maximale et en diffusant le résultat.

**Théorème 21.11** *Soit $G$ un graphe ayant $n$ sommets et un sommet distingué. Il existe un algorithme distribué synchrone qui calcule le diamètre de $G$ en $O(n)$ rondes et une complexité en bits de $O(n)$. De plus chaque sommet connaît la valeur du diamètre à la fin de l'algorithme.*

**Preuve :** Les distances maximales calculées dans la section précédente sont envoyées à leader via le BFS.

Étant donné un sommet $v$, $m_v$ désigne la distance maximale de $v$ à n'importe quel sommet.

Chaque sommet $v$ différent de la racine envoit sa distance maximale, notée $M_v$, de n'importe quel sommet dans le sous arbre ayant $v$ pour racine ; cette valeur est envoyée en $M_v + 2$ rondes consécutives sous la forme d'un signal $max$ suivi de $M_v$ "1" et d'un signal $endmax$. Une feuille peut démarrer ce processus dès qu'elle connaît sa distance maximale $m_v$ puisque $M_v = m_v$.

Un sommet $v$ différent d'une feuille attend d'avoir reçu le signal $max$ de chacun de ses fils $w$. Puis il envoie le signal $max$ à son père et continue d'envoyer 1 jusqu'à ce qu'il ait reçu le signal $endmax$ de chacun de ses fils. À ce moment il connaît $M_w$ pour chacun de ses sous-arbres et $m_v$ et peut donc calculer $M_v$ puis envoyer le nombre supplémentaire nécessaire de 1 à son père. On peut noter que le nombre de 1 envoyer à ce moment est au plus $M_w$ pour tout fils $w$ qui a envoyé le dernier $endmax$ et par conséquent ne peut pas être plus grand que $M_v$.

Ainsi la racine connaît le maximum global après un nombre linéaire de rondes.

Pour terminer, la racine diffuse le maximum global (il suffit d'utiliser le format unaire vu plus haut) à ses fils qui font de même etc. Ceci prend de nouveau un temps linéaire. □ En résumé, le

diamètre est obtenu après les étapes suivantes :

1. Calcul du BFS initialisé par Leader ;
2. Calcul de la distance entre Leader et chaque sommet de $G$ ;
3. Numérotation des sommets ;
4. Génération d'ondes et calcul des plus courtes distances entre deux sommets quelconques ;
5. Centralisation de la distance maximale et diffusion du diamètre.

## 21.4 Autres applications de la numérotation des sommets

### 21.4.1 Calcul de la maille

**Théorème 21.12** *Soit $G$ un graphe ayant $n$ sommets et un sommet distingué. La maille de $G$ peut être calculée par un algorithme distribué et connu de tous les sommets avec une complexité en temps et en bits égale à $O(n)$.*

**Preuve :** Comme pour le calcul du diamètre, dès que Leader détecte la terminaison de l'énumération des sommets il démarre une phase au cours de laquelle chaque sommet émet une onde anonyme.

Soit $v$ un sommet, si $v$ reçoit le signal *wave* de deux voisins à l'instant $d$ alors il conclut qu'il appartient à un cycle de longueur $2d$. Si $v$ reçoit le signal *wave* à l'instant $d$ et le signal *wave* à l'instant $d+1$ d'un autre voisin alors il en déduit qu'il appartient à un cycle de longueur $2d+1$.

Si $v$ appartient à un cycle alors il existe au moins un sommet $u$ de ce cycle qui émet une onde qui atteindra $v$ simultanément ou avec un décalage d'un top par deux ports différents. Ainsi la longueur de ce cycle sera calculée par $v$.

Quand $v$ sait qu'aucune onde supplémentaire n'arrivera (i.e., aucune onde n'arrive pendant 8 rondes) il calcule la longueur minimale des cycles auquels il appartient (cette longueur est notée $c_v$).

Si $v$ n'appartient à aucun cycle alors par convention $c_v = 0$.

Pour terminer, chaque sommet envoie à Leader $(n - c_v)$ en suivant la même procédure que dans les sections précédentes. La centralisation de ces valeurs permet à Leader de calculer la maille et de diffuser ce résultat. □

### 21.4.2 Détection des isthmes

**Théorème 21.13** *Soit $G$ un graphe connexe ayant $n$ sommets et un sommet distingué. Les isthmes de $G$ peuvent être détectés par un algorithme distribué et connus par leurs extrémités en $O(n)$ rondes avec un complexité en bits de $O(n)$.*

**Preuve :** Comme pour le calcul du diamètre, dès que Leader détecte la terminaison de l'énumération des sommets il démarre une phase au cours de laquelle chaque sommet va émettre dans l'ordre de l'énumération une onde.

Soit $v$ un sommet. Soit $e$ une arête incidente à $v$. Le théorème est une conséquence directe du fait suivant : l'arête $e$ est un isthme si et seulement si, quand $v$ reçoit le signal *wave* à travers $e$ à l'instant $t$ il n'en reçoit pas à travers une autre arête à l'instant $t$ ou $t+1$. □

### 21.4.3 Détection des points d'articulation, reconnaissance des graphes bi-connexes

**Théorème 21.14** *Soit $G$ un graphe connexe ayant $n$ sommets et un sommet distingué. Les points d'articulation peuvent être détectés par un algorithme distribué en $O(n)$ rondes avec une complexité en bits de $O(n)$.*

**Preuve :** Comme pour le calcul du diamètre, dès que Leader détecte la terminaison de l'énumération des sommets il démarre une phase au cours de laquelle chaque sommet va émettre, dans l'ordre de l'énumération, une onde.

Soit $u$ un sommet ayant au moins 2 voisins. Par définition, deux sommets $v_1$ et $v_2$ sont en relation modulo la relation $R_u$ si le sommet $u$ reçoit le signal *wave* de $v_1$ à l'instant $t$ et le signal *wave* de $v_2$ à l'instant $t$ ou l'instant $t+1$ pour un certain $t$.

Le théorème est une conséquence directe du fait suivant :

le sommet $u$ n'est pas un point d'articulation si et seulement si chaque paire de voisins de $u$ est en relation modulo la clôture transitive de $R_u$. □

Sachant qu'un graphe est bi-connexe si et seulement si il n'a pas de points d'articulation, ce théorème admet le corollaire suivant :

**Corollaire 21.15** *Soit $G$ un graphe connexe ayant $n$ sommets et un sommet distingué. Savoir si $G$ est bi-connexe peut être déterminé par un algorithme distribué en $O(n)$ rondes avec une complexité en bits de $O(n)$.*

# Chapitre 22

# Algorithmique des Bips

Parmi les paramètres qui conditionnent un algorithme distribué la communication est un élément majeur. Dans ce cadre, la taille des messages ou bien la signature de l'origine d'un message influencent directement l'existence ou la complexité d'un algorithme distribué.

Dans une série de travaux récents, de nouveaux modèles à base de bips ont été explorés. Dans ces modèles, le seul moyen de communication dont dispose un sommet est le bip : un sommet peut biper ou écouter. Plusieurs variantes sont définies : un sommet qui bipe peut détecter qu'au moins un voisin bipe en même temps (collision interne), notée $B_{cd}$ ; de la même façon un sommet qui écoute peut faire la différence entre aucun voisin ne bipe, un voisin bipe et au moins deux voisins bipent (collision périphérique), notée $L_{cd}$. La non détection d'une collision interne est notée $B$ et la non détection d'une collision périphérique est notée $L$. Finalement la combinaison de ces différents cas permet de définir 4 modèles basés sur des bips : $BL$, $B_{cd}L$, $BL_{cd}$ et $B_{cd}L_{cd}$.

Ce chapitre présente tout d'abord "des blocs" ou des "motifs" qui semblent revenir systématiquement dans l'écriture de nombreux algorithmes à base de bips. Ensuite, on présente des algorithmes pour calculer un MIS, un $2-$hop MIS, une coloration, une $2-$hop coloration et enfin le calcul des degrés des sommets. Ces algorithmes illustrent les blocs précédents. On introduit également des procédures d'émulation qui permettent de traduire automatiquement des algorithmes exprimés dans $B_{cd}$ et/ou $L_{cd}$ dans $BL$. la suite du chapitre est consacrée à l'analyse de la complexité en temps de ces différents algorithmes.

Le temps est divisé en intervalle de temps synchronisés appelés slots (ce qui correspond aux rondes). Tous les sommets se réveillent et démarrent les calculs au même slot. À chaque slot, tous les sommets agissent en parallèle, et chaque sommet bipe ou écoute. De plus, chaque sommet peut exécuter une quantité quelconque d'actions entre deux slots.

**Remarque 22.1** *En général, les sommets sont dans l'état* actif *ou l'état* passif*, suivant qu'ils participent ou non aux calculs. Quand ils sont dans l'état actif ils bipent ou ils écoutent. Dans la description des algorithmes on dit explicitement quand un sommet bipe, ce qui signifie qu'un sommet qui ne bipe pas écoute.*

La complexité en temps, appelée également la complexité en slots, est le nombre maximum de slots nécessaires pour que tous les sommets aient fini.

**Remarque 22.2** *Un algorithme exprimé avec des bips induit un algorithme dans le modèle à base de messages. Ainsi, étant donné un problème, toute borne inférieure sur la complexité en rondes dans le modèle à base de messages implique une borne inférieure pour la complexité en slots dans un modèle à base de bips.*

Une présentation plus développée avec les preuves peut être consultée [CMRZ16].

## 22.1 Shémas (patrons, modèles de conception) pour la description d'algorithmes à base de bips

Cette section présente plusieurs shémas d'utilisation des bips qui permettent de décrire différents algorithmes utilisant des bips. Ces shémas sont utilisés par la suite pour résoudre différents problèmes : calcul d'un MIS, coloriage etc.

**Biper seul.** Comme le titre l'indique le but est de permettre à un sommet de biper seul en excluant ainsi ses voisins.

**Algorithme 34:** Bip exclusif (en utilisant $B_{cd}$).

```
repeat
    beep with some probability ;
    if I beeped alone then
        do something exclusive ;
    ...
until finished;
```

**Biper seul à distance** 2**.** Comme précédemment mais cette fois à distance 2.

**Algorithme 35:** 2-hop exclusif bip (en utilisant $B_{cd}L_{cd}$).

```
repeat
    switch slot do
        1 beep with some probability ;
        2 if several neighbours beeped in slot 1 then
            beep
    if I beeped alone in slot 1 and no neighbour beeped in slot 2 then
        do something 2-hop exclusive
    ...
until finished;
```

**Phase.** L'algorithme 35 illustre un autre aspect des algorithmes à base de bips : un shéma basé sur la combinaison de plusieurs slots, formant une phase, permettant d'augmenter la puissance d'un seul slot.

**Détection de la terminaison locale.** Un slot, appelé slot de fin, peut être ajouté dans une phase : un noeud qui a atteint sont état final reste silencieux. Ainsi un sommet dont le voisinage reste silencieux à ce slot sait que ses voisins ont terminés.

**Probabilités adaptatives.** Certains algorithmes utilisent des choix probabilistes, adapter les probabilités consiste à faire évoluer lors d'une phase les probabilités attachées à ces choix en fonctions des résultats de la phase précédente.

**Algorithme 36:** Probabilité de biper adaptative (utilisant $B_{cd}L_{cd}$).

*Float* $p \leftarrow 1/2$ say **repeat**
  beep with probability $p$;
  **if** *I beeped alone* **then**
    `do something exclusive;`
  **else**
    **if** *no one beeped* **then**
      increase $p$;
    **else**
      decrease $p$;
**until** $finished$;

## 22.2 Quelques algorithmes sur les graphes

### 22.2.1 Colorier sans connaissance initiale

Le but est d'associer à chaque noeud du réseau une couleur de telle sorte que 2 noeuds voisins aient des couleurs différentes.

L'algorithme 37 procède de la façon suivante. À chaque phase chaque noeud incrémente un compteur. Un noeud qui réussit à bipper seul prend comme couleur la valeur du compteur et le fait savoir à ses voisins en bippant. La probabilité de bipper est une probabilité adaptative : la probabilité de bipper est divisée par 2 ou multipliée par 2 (sans dépasser 1/2). Si l'on veut qu'un noeud sache quand tous ses voisins sont coloriés alors il suffit d'ajouter un slot où un noeud non colorié bippe.

### 22.2.2 2-hop coloriage

Un 2−hop coloriage d'un graphe $G$ est un coloriage de $G$ tel que 2 sommets à distance au plus 2 ont des couleurs différentes. De fait, c'est un coloriage du carré de $G$ : le graphe obtenu à partir de $G$ tel que deux sommets sont liés par une arête s'ils sont voisins ou ont un voisin en commun.

**2-hop coloriage sans connaissance initiale.** Le 2−hop coloriage est obtenu à partir du coloriage en remplaçant un bip exclusif de l'algorithme 37 pour le coloriage par un 2−hop bip exclusif. Dès qu'un noeud produit un tel bip, il prend pour couleur la valeur du compteur.

De plus contrairement au coloriage, un sommet colorié continue d'être actif pour transmettre d'éventuelles collisions périphériques à des sommets voisins. La 2−hop coloration d'un graphe

**Algorithme 37:** Coloriage dans $B_{cd}L$ (sans connaissance).

```
Float p ← 1/2 ;
Integer colour ← nil ;
Integer counter ← 0 ;
repeat
    beep with probability p ;
    if I beeped alone then
        colour ← counter
    else
        if no one beeped then
            increase p ;
        else
            decrease p ;
        counter ← counter + 1 ;
until colour ≠ nil;
```

$G$ est la coloration du carré de $G$.

### 22.2.3 Calcul du degré de chaque sommet

Le calcul du degré passe pour chaque sommet par le comptage de ses voisins, et donc par un bip exclusif d'un sommet parmi ses voisins à distance au plus 2. Donc le calcul du degré reprend l'algorithme du 2−hop coloriage en ajoutant un bip dès lors qu'un sommet a bippé tout seul parmi ses voisins à distance au plus 2 ; ses voisins incrémentent alors leur degré.

**Algorithme 38:** 2-hop coloriage en utilisant $B_{cd}L_{cd}$ (sans connaissance).

```
Float p ← 1/2 ;
Integer colour ← nil ;
Integer counter ← 0 ;
repeat
    switch slot do
        1 if colour = nil then
            beep with probability p ;
        2 if several neighbours beeped in slot 1 then
            beep
        if I beeped alone in slot 1 and heard no beep in slot 2 then
            colour ← counter
        3 if someone beeped in slot 1 then
            beep
        if colour = nil then
            if no beep heard in slot 1 nor 3 then
                increase p
            else
                decrease p
        4 if colour = nil then
            beep
    counter ← counter + 1
until no beep heard in slot 4;
```

### 22.2.4 MIS et 2-hop MIS

Un MIS est un ensemble de sommet tel que 2 quelconques ne sont pas voisins et qui est maximal pour cette propriété.

L'algorithme permet à un sommet d'être dans cet ensemble dès lors qu'il bippe tout seul. Si un sommet est sélectionné tous ses voisins en sont automatiquement exclus.

**Algorithme 39:** Calcul d'un MIS dans $B_{cd}L$.

```
Var :
    state ∈ {active, InMIS, NotInMIS} Init active ;
    candidate, bh : Boolean ;
    p : real Init 1/2 ;
    slot : Integer ;
repeat
    switch slot do
        1 if state = active then
            set candidate to true with probability p else false;
            if candidate then
                beep
        2 bh := (internal collision at slot 1) or (at least one beep heard at slot 1) ;
        if candidate and no internal collision at slot 1 then
            state := InMIS; beep
        else
            if state = active and not candidate and no beep heard at slot 1 then
                if p < 1/2 then
                    p := p * 2
    if beep heard at slot 2 then
        state := NotInMis
    else
        if bh then
            p := p/2
until state ∈ {InMIS, NotInMIS};
```

**Calcul d'un 2-hop MIS.** Le calcul d'un 2-hop MIS d'un graphe $G$ se ramène au calcul d'un MIS dans le carré de $G$.

## 22.3 Détection de collisions et techniques d'émulation

Cette section présente un algorithme (du type Monte Carlo) pour détecter des collisions dans $BL$. La technique est simple : on remplace un slot où une collision est possible par une série de 2 slots où un sommet décide aléatoirement de bipper ou d'écouter.

**Algorithme 40:** Détection de collisions dans $BL$ (paramétré par $k$)

```
Boolean collision ← false;
Integer i ← 0;
while i < k do
    if v wishes to beep then
        Flip a coin;
        if heads then
            beep in slot 1;
            listen in slot 2;
        else
            listen in slot 1;
            beep in slot 2;
        if another beep was heard then
            collision ← true
    else
        listen in both slots;
        if beeps are heard in both slots then
            collision ← true;
    i ← i + 1;
return collision;
```

**Lemme 22.3** *Soit $v$ un sommet. Si une collision intervient dans le voisinage de $v$ alors $v$ la détecte en $O\left(\log(\frac{1}{\varepsilon})\right)$ slots avec une probabilité au moins égale à $1-\varepsilon$, et en $O(\log n)$ slots avec une probabilité $1-o\left(\frac{1}{n^2}\right)$.*

**Corollaire 22.4** *Soit $G$ un graphe. Si des collisions interviennent dans le voisinage d'un nombre quelconque de sommets, alors ils détectent tous une collision après au plus $O(\log(\varepsilon))$ slots avec une probabilité au moins égale à $1-\varepsilon$, et après au plus $O(\log n)$ slots avec forte probabilité.*

### 22.3.1 Procédures d'émulation

Cette section présente deux procédures probabilistes qui remplacent les instructions pour biper ou écouter et qui détectent les éventuelles collisions dans $BL$. Les deux sont du type Monte Carlo. La première procédure, `EmulateB`$_{cd}$`inBL()`, correspond à l'algorithme 41 et la seconde, `EmulateL`$_{cd}$`inBL()`, à l'algorithme 42. Les deux procédures sont paramétrisées par un entier $k > 1$, qui compte le nombre de phases utilisées à chaque appel de la procédure ($k$ contrôle la borne sur l'erreur possible). Elles retournent `true` si une collision a été détectée, `false` sinon.

Avant une exécution, chaque sommet génère une suite $s$ de $k$ bits tirés au hasard, chacun correspond à une phase. Ainsi, si deux noeuds génèrent des suites différentes, toutes les collisions seront détectées quelque soit la longueur du calcul.

La même suite est utilisée pour chaque appel de la procédure (i.e., elle est fixée pour chaque sommet).

**Algorithme 41:** Une Procédure pour émuler $B_{cd}$ dans $BL$.

**Procedure** `EmulateB`$_{cd}$`inBL`($in$ : *Integer* $k$, *Array* $s$; $out$ : *Boolean collision*)
*Boolean collision* $\leftarrow$ *false*;
*Integer* $i \leftarrow 0$;
**repeat**
  **if** $s[i]$ **then** beep in slot 1 ; listen in slot 2
  **else** listen in slot 1 ; beep in slot 2
  **if** *another beep was heard* **then** *collision* $\leftarrow$ *true* $i \leftarrow i + 1$
**until** $i = k$ ;
**End Procedure**

**Algorithme 42:** Une procédure pour émuler $L_{cd}$ dans $BL$.

**Procedure** `EmulateL`$_{cd}$`inBL`($in$ : *Integer* $k$; $out$ : *Boolean beep*, *Boolean collision*)
*Boolean beep* $\leftarrow$ *false*;
*Boolean collision* $\leftarrow$ *false*;
*Integer* $i \leftarrow 0$;
**repeat**
  **switch** *slot* **do**
    1 and 2 listen **if** *a beep was heard in any slot* **then**
      *beep* $\leftarrow$ *true*
    **if** *a beep was heard in both slots* **then**
      *collision* $\leftarrow$ *true*
  $i \leftarrow i + 1$
**until** $i = k$ ;
**End Procedure**

La valeur de $k$ dépend de la borne que l'on souhaite sur l'erreur ; on obtient :

**Lemme 22.5** *Pour tout $\varepsilon > 0$, tout $n > 0$ et tout $c > 2$ :*

1. *si $k = \lceil \log\left(\frac{1}{\varepsilon}\right) \rceil$, les procédures sont correctes pour un noeud donné avec probabilité $1 - \varepsilon$,*
2. *si $k = \lceil \log\left(\frac{n}{\varepsilon}\right) \rceil$, les procédures sont correctes pour tous les noeuds avec probabilité $1 - \varepsilon$,*
3. *si $k = \lceil c \log(n) \rceil$, les procédures sont correctes pour tous les noeuds avec forte probabilité.*

**Lemme 22.6** *Pour tout $\varepsilon > 0$, pour tout $n > 0$, pour tout $c > 2$ et pour toute exécution :*

1. *si $k = \lceil \log\left(\frac{\Delta}{\varepsilon}\right) \rceil$, les procédures sont correctes pour un noeud donné avec probabilité $1 - \varepsilon$,*
2. *si $k = \lfloor \log\left(\frac{n\Delta}{\varepsilon}\right) \rfloor$, les procédures sont correctes pour tous les noeuds avec probabilité $1 - \varepsilon$,*

3. *si* $k = \lfloor c \log(n) + \log \Delta \rfloor$, *les procédures sont correctes pour tous les noeuds avec forte probabilité.*

Tous les algorithmes présentés dans la section 22.2 peuvent être adaptés au modèle $BL$ en remplaçant les appels à biper ou à écouter par des appels à $\texttt{EmulateB}_{cd}\texttt{inBL}$ ou à $\texttt{EmulateL}_{cd}\texttt{inBL}$. Ces substitutions entraînent dans le pire des cas un ralentissement logarithmique.

**Théorème 22.7** *Soit $G$ un graphe ayant $n$ sommets. La complexité d'un algorithme du type Monte Carlo qui détecte une collision avec forte probabilité dans le modèle $BL$ est $\Omega(\log n)$.*

## 22.4 Complexité

Cette section présente quatre résultats de complexité relatifs aux algorithmes présentés.

### 22.4.1 L'algorithme MIS

**Théorème 22.8** *Le nombre de phases de l'algorithme de calcul d'un MIS est borné supérieurement par $76 \log n$ avec forte probabilité.*

### 22.4.2 Coloriage

**Théorème 22.9** *Soit $G = (V, E)$ un graphe avec $|V| = n$ et de degré maximum $\Delta$, l'algorithme 37 colorie tous les sommets de $G$ en un temps $O(\Delta + \log n)$ avec une probabilité $1 - o\left(n^{-1}\right)$.*

Les complexités des variantes pour le 2−hop MIS ou le 2−hop coloriage sont $O(\log n)$ et de $O(\Delta^2 + \log n)$.

# Bibliographie


[Ang80] D. Angluin. Local and global properties in networks of processors. In *Proc. of the 12th Symposium on Theory of Computing (STOC 1980)*, pages 82–93, 1980.

[AW04] H. Attiya and J. Welch. *Distributed computing : fundamentals, simulations, and advanced topics*. John Wiley and Sons, 2004.

[Ber83] C. Berge. *Graphes*. Gauthier Villars, 1983.

[BGM+01] M. Bauderon, S. Gruner, Y. Métivier, M. Mosbah, and A. Sellami. visualization of distributed algorithms based on labeled rewriting systems. In *Proc. of the 2nd International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2001)*, volume 50 of *ENTCS*, pages 229–239, 2001.

[CGM12] J. Chalopin, E. Godard, and Y. Métivier. Election in partially anonymous networks with arbitrary knowledge in message passing systems. *Distributed Computing*, 25(4) :297–311, 2012.

[CGMO06] J. Chalopin, E. Godard, Y. Métivier, and R. Ossamy. Mobile agent algorithms versus message passing algorithms. In *Proc. of the 10th International Conference on Principles of Distributed Systems (OPODIS 2006)*, volume 4305 of *Lecture Notes in Computer Science*, pages 187–201. Springer-Verlag, 2006.

[Cha06] J. Chalopin. *calculs locaux et homomorphismes de graphes*. PhD thesis, Université Bordeaux 1, 2006.

[CM07] J. Chalopin and Y. Métivier. An efficient message passing election algorithm based on Mazurkiewicz's algorithm. *Fundam. Inform.*, 80(1-3) :221–246, 2007.

[CM10] J. Chalopin and Y. Métivier. On the power of synchronization between two adjacent processes. *Distributed Computing*, 23(3) :177–196, 2010.

[CMRZ16] A. Casteigts, Y. Métivier, J. Michael Robson, and A. Zemmari. Design patterns in beeping algorithms. *CoRR*, abs/1607.02951, 2016.

[Dij74] E.W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11) :643–644, 1974.

[DM97] V. Diekert and Y. Métivier. Partial commutation and traces. In G. Rozenberg and A. Salomaa, editors, *Handbook of formal languages*, volume 3, pages 457–533. Springer-Verlag, 1997.

[Dol00] S. Dolev. *Self-Stabilization*. MIT Press, 2000.

[God02] E. Godard. *Réécritures de Graphes et Algorithmique Distribuée*. PhD thesis, Université Bordeaux 1, 2002.

[Hoa78] C.A.R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8) :666–677, 1978.

[KS08] A. D. Kshemkalyani and M. Singhal. *Distributed computing*. Cambridge, 2008.

[Lav95] C. Lavault. *Evaluation des algorithmes distribués*. Hermès, Paris, 1995.

[LM92] I. Litovsky and Y. Métivier. Computing trees with graph rewriting systems with priorities. In *Tree Automata and Languages*, pages 115–140. 1992.

[LMS95] I. Litovsky, Y. Métivier, and E. Sopena. Different local controls for graph relabelling systems. *Math. Syst. Theory*, 28(1) :41–65, 1995.

[LMS99] I. Litovsky, Y. Métivier, and E. Sopena. Graph relabelling systems and distributed algorithms. In H. Ehrig, H.J. Kreowski, U. Montanari, and G. Rozenberg, editors, *Handbook of graph grammars and computing by graph transformation*, volume 3, pages 1–56. World Scientific, 1999.

[Lyn96] N. Lynch. *Distributed algorithms*. Morgan Kaufman, 1996.

[Mas91] W.S. Massey. *A basic course in algebraic topology*. Springer-Verlag, 1991. Graduate texts in mathematics.

[MRSDZ10] Y. Métivier, J. M. Robson, N. Saheb-Djahromi, and A. Zemmari. About randomised distributed graph colouring and graph partition algorithms. *Inf. Comput.*, 208(11) :1296–1304, 2010.

[MRSDZ11] Y. Métivier, J. Michael Robson, N. Saheb-Djahromi, and A. Zemmari. An optimal bit complexity randomized distributed mis algorithm. *Distributed Computing*, 23(5-6) :331–340, 2011.

[MRZ15] Y. Métivier, J. M. Robson, and A. Zemmari. Analysis of fully distributed splitting and naming probabilistic procedures and applications. *Theor. Comput. Sci.*, 584 :115–130, 2015.

[MRZ16] Y. Métivier, J. Michael Robson, and A. Zemmari. A distributed enumeration algorithm and applications to all pairs shortest paths, diameter.. *Inf. Comput.*, 247 :141–151, 2016.

[MSZ02] Y. Métivier, N. Saheb, and A. Zemmari. Randomized local elections. *Information Processing Letters*, 82(6) :313–320, 2002.

[MSZ03] Y. Métivier, N. Saheb, and A. Zemmari. Analysis of a randomized rendezvous algorithm. *Inf. Comput.*, 184(1) :109–128, 2003.

[Ray85] M. Raynal. *Algorithmes distribués et protocoles*. Eyrolles, 1985.

[Ray87] M. Raynal. *Systèmes répartis et réseaux - concepts, outils et algorithmes*. Eyrolles, 1987.

[Ray91] M. Raynal. *La communication et le temps dans les réseaux et les systèmes répartis*. Eyrolles, 1991.

[San06] N. Santoro. *Design and analysis of distributed algorithms*. Wiley, 2006.

[Sel04] A. Sellami. *Des calculs locaux aux algorithmes distribués*. PhD thesis, Université Bordeaux 1, 2004.

[SSP85] B. Szymanski, Y. Shy, and N. Prywes. Terminating iterative solutions of simultaneous equations in distributed message passing systems. In *Proc. of the 4th Annual ACM Symposium on Principles of Distributed Computing (PODC 1985)*, pages 287–292. ACM Press, 1985.

[Tel00] G. Tel. *Introduction to distributed algorithms*. Cambridge University Press, 2000.

[TvS02] A. Tanenbaum and M. van Steen. *Distributed Systems - Principles and Paradigms*. Prentice Hall, 2002.

[YK96] M. Yamashita and T. Kameda. Computing on anonymous networks : Part i - characterizing the solvable cases. *IEEE Transactions on parallel and distributed systems*, 7(1) :69–89, 1996.

[Zem00] A. Zemmari. *Contribution à l'analyse d'algorithmes distribués*. PhD thesis, Université Bordeaux 1, 2000.