جامعة بجاية
Tasdawit n'Bgayet
Université de Béjaïa

DEPARTEMENT
INFORMATIQUE

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université A/Mira de Béjaia

Faculté des Sciences Exactes

Département d'Informatique

Mémoire de Master

en Informatique

Option : Réseaux et Systèmes Distribués

Thème

Réécriture de Graphes avec Défaillances et Algorithmes Distribués

Présenté par :

M. Djamal ATMANI

Soutenu devant le jury composé de :

Promoteur **: Hamouma MOUMEN**

Président **: Kamal KABYL**

Examinateur **: Abderrahmane BAADACHE**

Promotion 2011

Résumé :

En algorithmique distribuée, chaque système peut être représenté par un graphe étiqueté, où les sommets correspondent aux différents terminaux, les arêtes aux liens de communication et les étiquettes associées aux sommets codent les états des processeurs. Dans un mécanisme de réétiquetage local, un algorithme distribué est décrit par un système de règles de transition locale où la nouvelle étiquette d'un sommet est fonction de son étiquette précédente et de celles de ses voisins. Dans le cadre de ce mémoire, nous étudions la réalisabilité et non-réalisabilité des tâches distribuées. Nous illustrons notre méthode en nous intéressant en particulier à certains problèmes spécifiques aux systèmes distribués (élection d'un nœud, énumération de graphes, problème de consensus et découverte de topologie dans les réseaux Ad Hoc). Dans tous ces cas, nous caractérisons ce que n'est pas réalisable par calcul distribué en fonction de la topologie du graphe sous-jacent et de la connaissance structurelle de ce graphe. Les différents cas d'impossibilité de calcul d'une manière distribuée sont dus aux « similarités » de familles de réseaux. Ces « similarités » sont décrites à l'aide de morphismes de graphes particuliers : les revêtements et les quasi-revêtements. Les preuves d'impossibilité emploient des techniques de simulation à base de ces morphismes.

Abstract:

In distributed algorithmic each system can be modeled by a labelled graph, where nodes correspond to different processors, the edges to communication links and labels associated with the nodes encode the states of processors. In a local relabeling mechanism, a distributed algorithm is described as a system of local transition rules, where the new label of a node is a function of its previous label and those of its neighbours. As part of this paper, we study the feasibility and infeasibility of distributed tasks. We illustrate our method by focusing in particular on specific problems of distributed systems (election of a node, enumeration of graphs, and the problem of consensus and topology discovery in Ad Hoc networks). In all these cases, we characterize what is not feasible for distributed computing based on the topology of the underlying graph and the structural knowledge of this graph. The impossibility cases of distributed computing usually due to the "similarities" of families of networks. These "similarities" are described with special morphisms of graphs: coverings and quasi- coverings. The impossibility proofs employ simulation techniques based on these morphisms.

Remerciements

Je tiens à remercier tout particulièrement et à témoigner ma grande reconnaissance à M. Hamouma MOUMEN, mon encadreur, pour son accompagnement exemplaire tout au long de mes recherches et l'intention qu'il a porté à mon travail.

Je remercie également M. Nadir SALHI pour ses conseils et son soutien au long de mon cursus de Master.

Un grand merci à M. Kamal KABYL d'avoir accepté d'être le président du jury.

Je remercie aussi M. Abderrahmane BAADACHE de m'avoir fait l'honneur d'être examinateur de mon travail.

Toute ma reconnaissance à M. TARI A/Kamel, chef de département Informatique, et M. ALOUI Abdouahab, adjoint chef de département Informatique, qui m'ont beaucoup soutenu durant ma formation.

Mes très vifs remerciements vont à mon frère Merouane, ma sœur Abla, Fadila et tous ceux qui ont contribué de près ou de loin à la réalisation de ce travail.

Je remercie finalement et sincèrement M. Salim BEDJIL pour son orientation.

Dédicaces

A la mémoire de mon grand-père tombé pour une Algérie socialiste, digne et prospère.

Je dédie ce travail :

En premier lieu à mes très chers parents qui m'ont toujours soutenu et encouragé tout au long de mes études.

A ma grande mère, toujours présente à mes côtés.

A mon frère Merouane, à mes sœurs Abla, Fadila et la petite Widad.

A ma très chère petite-amie Baya.

A mon très cher ami Yacine.

A tous mes proches …

A tous mes amis …

Djamal ATMANI

Table des matières

Liste des figures

Introduction

Plusieurs structures de données sont souvent représentées à l'aide de structures de graphes. Cela, pour la raison de faciliter le partage de sous-structures, ce qui induit des calculs efficaces, mais aussi pour le fait qu'ils permettent de considérer des structures de données plus complexes.

Un système distribué est un environnement où différents processus se partagent les ressources et collaborent pour réaliser un objectif commun. La modélisation de tels systèmes par des graphes permet de résoudre une multitude de problèmes, cependant la présence de défaillances exige plus de robustesse et de performance dans les algorithmes distribués.

Dans ce mémoire, nous avons réalisé une étude sur le problème de réécriture de graphes avec défaillances dans les systèmes distribués, en particulier : le réétiquetage, l'utilisation des revêtements, l'élection et l'énumération dans les graphes et les cas d'impossibilité de résolution de quelques problèmes distribués.

Nous commencerons ce mémoire par le rappel de quelques généralités concernant les graphes, les connaissances structurelles et les défaillances dans les systèmes distribués. Ceci pour une meilleure compréhension de problèmes exposés dans ce mémoire ainsi les différentes solutions proposées. Ensuite, nous décrivons, dans le deuxième chapitre, le mécanisme du réétiquetage de graphes et la sérialisation, mais surtout nous nous concentrons sur l'utilisation des revêtements de graphes dans les systèmes distribués, dont le but est de fournir une approche permettant de caractériser ce qui est calculable de manière distribuée et ce qui ne l'est pas.

Dans le 3ème chapitre, nous donnerons quelques exemples d'utilisation de revêtements dans les systèmes distribués, nous étudierons les trois exemples suivants : algorithme de Mazurkiewicz, problème d'élection dans un réseau et preuves d'impossibilité de résolution de problèmes distribués de consensus.

Enfin, dans le dernier chapitre, nous traiterons le problème de découverte de topologie dans les réseaux ad hoc en présence de défaillances byzantines. D'abord en donnant deux algorithmes de découverte de topologie forte et faible respectivement, puis, en donnant une application du problème d'accord byzantin pour les réseaux ad hoc, à savoir l'impossibilité de distinguer un changement de topologie d'une défaillance byzantine.

I. Généralités sur les graphes et les défaillances dans les systèmes distribués

Pour une meilleure compréhension des autres chapitres, il est évident de rappeler quelques généralités sur la théorie des graphes et la notion de défaillance dans les systèmes distribués.

I.1 Généralités sur les graphes

Nous commencerons par rappeler certaines définitions élémentaires sur les graphes, ainsi en profiter pour expliciter les notations que l'on retrouvera fréquemment par la suite. Nous définirons ensuite ce que nous entendons par graphes étiquetés.

I.1.1 Définitions

Les définitions et les notations utilisées sont standards [1, 2, 3, 4]

Définition 1 : Graphe

Un *graphe simple non orienté* ***G*** est défini par un ensemble de sommets, noté ***V(G)***, et un ensemble de paires de sommets distincts ayant un lien, noté ***E(G)***. Les éléments de ***E(G)*** sont des *arêtes*.

Dans le cas d'un graphe non orienté ces liens sont symétriques.

Une arête ayant deux sommets *u* et *v* pour extrémités sera notée *{u, v}*. Les sommets *u* et *v* sont dits *adjacents* ou *voisins*.

Un graphe est dit *fini* si l'ensemble de ses sommets et de ses arêtes est fini.

Par la suite, le terme *graphe* désigne, sauf précision contraire, *un graphe fini simple* (*i.e.* non orienté, sans boucles et sans arêtes multiples).

Définition 2 : Arbre

Un arbre est un graphe connexe non orienté dont le nombre de sommets excède le nombre d'arêtes d'une unité exactement.

Étant donné deux graphes ***G*** et ***H***, on définit les notions suivantes :

Définition 3 : Sous-graphe partiel

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

Le graphe ***H*** est un *sous-graphe induit* de ***G*** si et seulement si ***H*** est un sous-graphe partiel de ***G*** et ***H*** contient toutes les arêtes de ***G*** dont les extrémités sont dans *V(H)*

i.e. $\forall x, y \in V(H), \{x, y\} \in E(H) \Leftrightarrow \{x, y\} \in E(G)$.

Définition 4 : Chemin dans un graphe

Un *chemin* dans un graphe est une suite de sommets voisins, c'est-à-dire une suite de la forme $\boldsymbol{\Gamma} = (u_0, u_1, \ldots, u_{n-1})$ telle que :

$$\{u_i , u_{i+1}\} \in E(G) \qquad 0 \leq i < n-1$$

Les sommets u_0 et u_{n-1} sont les extrémités de ce chemin. On dit que $\boldsymbol{\Gamma}$ est un chemin de u_0 à u_n. De plus, si tous les sommets sont distincts, excepté peut-être pour les extrémités, alors le chemin est dit *simple*.

De plus, si pour tout i : $u_{i-1} \neq u_{i+1}$ alors le chemin est dit *non bègue* [5].

La longueur d'un chemin est égale au nombre d'arêtes parcourues, ici n. On note $\Gamma_G(u)$ l'ensemble des chemins sur G issus du sommet u.

Lorsque les extrémités d'un chemin sont confondues, ce chemin est appelé *cycle*.

Définition 5 : Graphe connexe

Un graphe ***G*** est *connexe* si quels que soient u et v, deux sommets de ***G***, il existe un chemin entre u et v.

Un *arbre* est un graphe connexe et sans cycle. Un *anneau* est un graphe constitué d'un cycle simple.

Définition 6 : Distance

Soient u et v deux sommets d'un graphe connexe ***G*** :

On appelle *distance* de u à v, notée $d_G(u , v)$, la longueur du plus court chemin de u à v. Le diamètre de ***G*** est

$$\Delta(G) = max\ (d_G(u , v)) \qquad \text{avec} \quad u, v \in V(G)$$

Définition 7 : Voisinage

Le *voisinage* d'un sommet u dans un graphe $\boldsymbol{G}$, noté $N_G(u)$, est l'ensemble des sommets adjacents à u dans $\boldsymbol{G}$:

$$N_G(u) = \{v \in V(G) \mid \{u , v\} \in E(G) \}$$

En notant le *cardinal* d'un ensemble $\boldsymbol{S}$ par $card(S)$ ou par $|S|$, alors le degré de u dans $\boldsymbol{G}$, noté $deg_G(u)$, est le nombre de voisins de u dans $\boldsymbol{G}$ i.e. le cardinal du voisinage de u :

$$deg_G(u) = |N_G(u)|$$

Un graphe dont tous les sommets ont le même degré d est dit *d-régulier*.

Définition 8 : Boule

On appelle *boule* de centre u qu'on note $B_G(u)$, le graphe constitué des sommets $\{u\} \cup N_G(u)$ et des arêtes issues de u.

Soit un graphe $\boldsymbol{G}$ et u un de ses sommets, soit $k \in \mathbb{N}$; la boule de centre v et de rayon k est le sous-graphe constitué des chemins issus de v de longueur inférieure ou égale à k. Elle est également appelée *k-boule* de centre u et notée $B_G(u, k)$. Elle contient les sommets et arêtes suivants :

$$V(B_G(u , k)) = \{ v \in V(G) \mid d_G(u , v) \leq k \}$$

$$E(B_G(u , k)) = \{ \{v , w\} \in E(G) \mid d(u , v) \leq k-1, d(u , w) \leq k \}$$

Si l'on ajoute les arêtes reliant les sommets à une distance exactement de $k>1$, on obtient la boule fermée de centre u et de rayon k, notée $\overline{B_G}(u , k)$:

$$V(\overline{B_G}(u , k)) = \{ v \in V(G) \mid d_G(u , v) \leq k \}$$

$$E(\overline{B_G}(u , k)) = \{ \{v , w\} \in E(G) \mid v , w \in V(\overline{B_G}(u , k)) \}$$

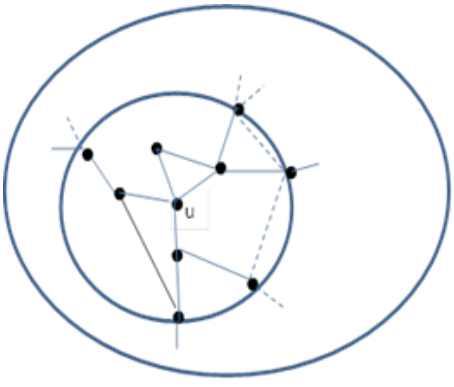
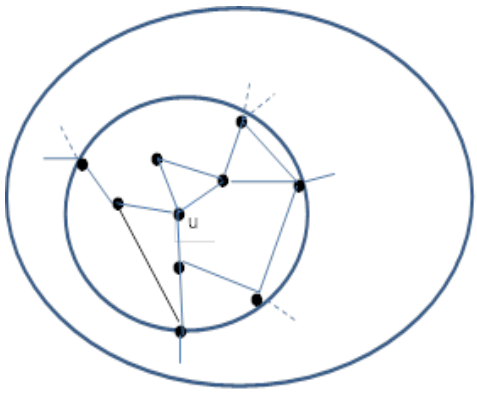


Figure 1 : boule ouverte de centre u



boule fermée de centre u

Définition 9 : Matrice d'adjacence

Une *matrice d'adjacence* pour un graphe fini G à n sommets est une matrice de dimension $n \times n$ dont l'élément non-diagonal a_{ij} est le nombre d'arêtes liant le sommet i au sommet j. L'élément diagonal a_{ii} est le nombre de boucles au sommet i.

Définition 10 : Homomorphisme

Un *homomorphisme* entre ***G*** et ***H*** est une application ϕ de $V(G)$ dans $V(H)$ telle que :

si $\{x , y\} \in E(G)$ alors $\{ \phi(x) , \phi(y)\} \in E(H)$

Définition 11 : Isomorphisme

De plus, si ϕ est une application *bijective* et si son inverse ϕ^{-1} est un homomorphisme de ***H*** vers ***G***, alors ϕ définit un *isomorphisme* entre les deux graphes ***G*** et ***H***. On dit alors que ***G*** et ***H*** sont isomorphes.

Définition 12 : Famille de graphes

On appelle *classe* ou *famille* de graphes, toute famille ***F***, au sens de la théorie des ensembles, close par isomorphisme.

On considère essentiellement des graphes finis et connexes.

Exemple :

Un anneau de taille $n \in \mathbb{N}$, noté A_n est le graphe suivant :

$$V(A_n) = \{1, \dots ,n\}$$

$$E(A_n) = \{ \{i , j\} \mid i - j = \pm 1 \bmod n\}$$

($k \bmod n$: dénote le reste de la division euclidienne de k par n)

Définition 13 : Fermeture transitive

La fermeture transitive $C(G)$ d'un graphe G est un graphe tel qu'il existe un arc entre toute paire de sommets entre lesquels il existe un chemin.

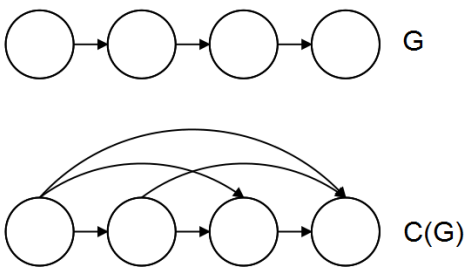


Figure 2 : Exemple fermeture transitive

I.1.2 Graphes étiquetés

Dans ce mémoire, nous travaillons principalement sur des graphes dont les sommets et les arêtes sont étiquetés par un ensemble d'étiquettes ***L***, qu'on peut définir par :

Un *graphe étiqueté* sera noté de la manière suivante ***(G**, λ)*, où **G** est un graphe (dit *graphe sous-jacent*) et λ une application (dite *étiquetage*) de $V(G) \cup E(G)$ dans ***L***.

Lorsqu'il n'est pas nécessaire d'expliciter la fonction d'étiquetage, nous adopterons la notation ***G, H,*** . . . pour désigner des graphes étiquetés.

On note $\lambda(G)$ l'ensemble des étiquettes présentes sur ***G***.

Un exemple simple de tels graphes que nous utiliserons fréquemment est le suivant : Soit ***G*** un graphe et α une *étiquette* de ***L***. Alors Λ_α^G est l'étiquetage de ***G*** tel que chaque arête et sommet sont étiquetés par α. Dans ce cas, on dit que l'étiquetage est *uniforme* en α. On abrégera fréquemment en (G, Λ_α).

Soient (G, λ) et (G', λ') deux graphes étiquetés. Alors :

(G, λ) est un sous-graphe de (G', λ'), noté $(G, \lambda) \subseteq (G', \lambda')$, si seulement si ***G*** est un sous-graphe de ***G′*** et λ est la restriction de l'étiquetage λ' à $V(G) \cup E(G)$.

Définition 14 : Homomorphisme de graphes étiquetés

Une application $\phi : V(G) \cup E(G) \rightarrow V(G') \cup E(G')$ est un homomorphisme de (G , λ) sur (G', λ') si ϕ est un homomorphisme de graphe de ***G*** sur ***G′*** qui conserve l'étiquetage, i.e. tel que $\lambda'(\phi(x)) = \lambda(x)$ pour tout $x \in V(G) \cup E(G)$.

L'application ϕ est un isomorphisme, si elle définit, sur les graphes sous-jacents, un isomorphisme de graphes. On écrit alors $(G, \lambda) \simeq (G', \lambda')$.

Une occurrence de (G, λ) dans (G', λ') est un isomorphisme ϕ entre (G, λ) et un sous-graphe (H, η) de (G', λ').

Définition 15 : Famille de graphes étiquetés

On appelle *classe* ou *famille* de graphes étiquetés, toute famille ***F***, au sens de la théorie des ensembles, close par isomorphisme de graphes étiquetés.

On note $\mathcal{G}_L$ la classe des graphes *L-étiquetés*.

On sera amené à utiliser des sur-étiquetages de graphes étiquetés, ou encore étiquetage produit :

Définition 16 : Sur-étiquetage

Soit $\boldsymbol{G} = (G, \lambda)$ un graphe étiqueté. On appelle *sur-étiquetage :*

$\mu : V(\boldsymbol{G}) \rightarrow \boldsymbol{L}$ *de* $\boldsymbol{G}$**,** le graphe étiqueté $(\boldsymbol{G}, \mu) = (G, \lambda \times \mu)$. Dans ce cas on appelle registre, les composantes λ ou μ des étiquettes.

Par la suite, en assimilant tout graphe non étiqueté $\boldsymbol{G}$ à $(G , \Lambda_\varepsilon)$, où ε dénote le mot vide, on se ramènera à ne manipuler que des graphes étiquetés. Par conséquent, sauf mention contraire, le terme "graphe" désignera indistinctement un graphe non-étiqueté ou étiqueté.

I.1.3 Connaissances Structurelles

Lors de l'élaboration des algorithmes distribués, il est fréquent de poser certaines hypothèses simplificatrices sur la topologie du réseau sous-jacent comme le fait qu'il possède une topologie particulière (anneau, grille, ...), que chaque sommet dispose d'une identité distincte, ou encore que chaque sommet ait connaissance de certaines propriétés structurelles du réseau (sa topologie, sa taille, une borne sur sa taille,...). Souvent, en termes de calculabilité, le fait de poser certaines hypothèses simplifie de manière conséquente le problème. Par exemple, on sait que dans un réseau anonyme (i.e. sans identité), on ne peut pas toujours élire, alors que si l'on suppose l'existence d'identités distinctes, on peut élire quelle que soit la topologie de graphe, la difficulté devenant alors de concevoir les algorithmes les plus rapides. Puisque l'objectif est d'étudier ce qui est ou n'est pas réalisable par calculs locaux alors, il semble judicieux de pouvoir définir rigoureusement quelles sont exactement les hypothèses nécessaires. Par exemple, en ce qui concerne l'élection, entre aucune hypothèse et l'existence d'identités distinctes (qui est une spécification exigeante et coûteuse à maintenir sur un réseau), y a-t-il une hypothèse intermédiaire qui permet de pouvoir élire.

Une connaissance structurelle est simplement une fonction qui associe à tout graphe une étiquette.

Définition 17 : Connaissance structurelle

On appelle *connaissance structurelle* toute fonction récursive $\iota : \mathcal{G}_L \rightarrow L$

Une connaissance structurelle peut "*encoder*" toute connaissance dépendant de la structure globale du graphe. On étiquettera initialement et uniformément $\boldsymbol{G}$ par $\Lambda_{\iota(G)}$ pour exprimer formellement cette connaissance. On s'intéressera plus particulièrement ici à certaines connaissances de la liste, non exhaustive, ci-dessous :

- Connaissances "***Métriques***"
  - Une borne supérieure *b* sur la taille du graphe : il s'agit d'une fonction *b* telle que :

    $b(G) \geq |V(G)|$ pour tout $\boldsymbol{G}$.
  - Une bonne borne : il s'agit d'une borne *b* telle que pour tout graphe $\boldsymbol{G}$,

    $|V(G)| \leq b(G) < 2|V(G)|$
  - La taille, i.e. le nombre de sommets *card(V(G))*.
  - Le diamètre $\Delta(G)$ du graphe $\boldsymbol{G}$. Une borne sur le diamètre.
- "***Caractéristique***" ou ***connaissance booléenne*** : savoir si le graphe est ou n'est pas dans une classe donnée $\boldsymbol{C}$. On note $\boldsymbol{X_C}$ la connaissance structurelle associant chaque graphe de $\boldsymbol{C}$ à l'étiquette **VRAI** et chaque graphe de $\mathcal{G}\backslash\boldsymbol{C}$ à l'étiquette **FAUX**.

L'utilisation de telles fonctions nous permet, par exemple, de calculer la taille du graphe dans une classe particulière.

- Connaissance de la ***topologie*** (par exemple, la matrice d'adjacence du graphe sous-jacent est donnée).

I.2 Les défaillances

I.2.1 Défaillances des processus

Un processus est défaillant lorsqu'il ne respecte pas ses spécifications durant l'exécution du système. On distingue plusieurs catégories de défaillances [6, 7] :

I.2.1.1 Défaillance par arrêt définitif (*Crash-fault*)

Dans ce type de défaillances, un processus s'arrête prématurément et ne fait rien à partir de ce point et il ne peut pas reprendre son exécution (à cause de la perte de tout ou d'une partie de son code par exemple).

I.2.1.2 Défaillance par omission

Dans ce cas, un processus défaillant peut omettre certaines actions. Ces actions concernent, par exemple, l'envoi ou la réception de messages.

I.2.1.3 Défaillance de performances

Cette défaillance se caractérise par un non-respect des contraintes temporelles par exemple, les délais d'exécution des tâches.

I.2.1.4 Défaillance arbitraire (*Byzantine*)

Les défaillances byzantines sont définies en termes de déviation d'un processus de l'algorithme qu'il exécute. Par exemple, il diffuse un message M à un sous ensemble de processus. Pour d'autres processus, il envoie M' ou bien il envoie un message qu'il n'est pas prévu dans l'algorithme.

Remarque : la défaillance par arrêt définitif est un cas particulier de la défaillance par omission. Cette dernière est un cas particulier de la défaillance de performances, alors que les défaillances arbitraires incluent tous les types de défaillances [8].

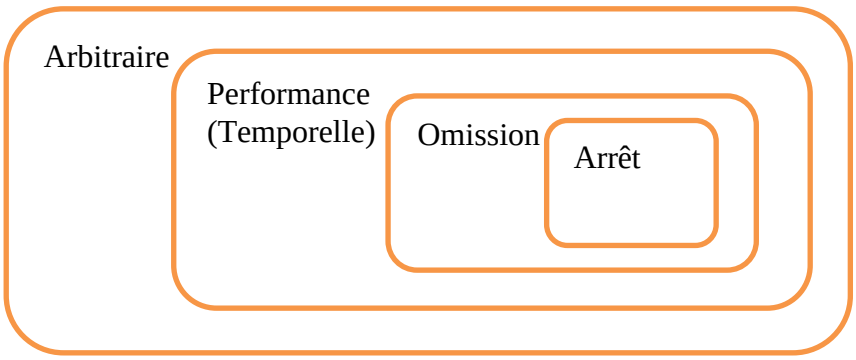


Figure 3 : classification des défaillances de processus

I.2.2 défaillances des liens de communication

Les liens de communication qui relient les différents processus peuvent subir des défaillances de plusieurs types :

- la perte de messages.
- la duplication de messages.
- l'altération de messages.

etc.

Maintenant qu'on a rappelé les différentes généralités sur les graphes et les défaillances des systèmes on va s'intéresser par la suite au mécanisme de réétiquetage et à l'utilisation des revêtements de graphes dans les systèmes distribués.

II. Réétiquetages des graphes et revêtements

II.1 Réétiquetages

Les calculs locaux considérés ici peuvent être décrits de la manière suivante. Lors d'un calcul distribué, les nœuds changent d'état en fonction de leur état et de ceux de leurs voisins.

II.1.1 Définitions

Soit $\mathcal{R} \subseteq \mathcal{G}_L \times \mathcal{G}_L$ une relation binaire sur $\mathcal{G}$. Alors $\mathcal{R}$ est un système de réécriture de graphes. On imposera que $\mathcal{R}$ soit fermé par isomorphisme, i.e. si $(G, \lambda)\mathcal{R}(G', \lambda')$, et si $(G_1, \lambda_1) \simeq (G, \lambda)$ alors $(G_1, \lambda_1)\mathcal{R}(G_1', \lambda_1')$ pour un certain graphe étiqueté $(G_1', \lambda_1') \approx (G', \lambda')$. Par la suite $\mathcal{R}^*$ sera la fermeture transitive de $\mathcal{R}$. Le graphe étiqueté (G, λ) est $\mathcal{R}$-irréductible (ou irréductible si $\mathcal{R}$ est fixé) s'il n'existe aucun étiquetage λ' de G tel que $(G, \lambda)\mathcal{R}(G, \lambda')$.

Soit $(G , \lambda) \in \mathcal{G}_L$, alors $Irred_{\mathcal{R}}((G , \lambda))$ est l'ensemble des graphes $\mathcal{R}$-irréductibles en relation $\mathcal{R}^*$ avec (G, λ). La relation $\mathcal{R}$ est dite noethérienne s'il n'existe pas de chaîne infinie $(G, \lambda_1)\mathcal{R}(G , \lambda_2)\mathcal{R}\ldots.$

Définition 1 : Système de réétiquetage de graphes

Soit $\mathcal{R} \subseteq \mathcal{G}_L \times \mathcal{G}_L$ un système de réécriture de graphes.

1. $\mathcal{R}$ est un système de réétiquetage de graphes si pour tout couple de graphes en relation, les graphes sous-jacents sont égaux (égaux, et pas seulement isomorphes), i.e. :

$$(G, \lambda)\mathcal{R}(H, \lambda') \Rightarrow G = H.$$

2. $\mathcal{R}$ est local si $\mathcal{R}$ ne modifie les étiquettes que dans une boule de rayon 1, i.e. $(G, \lambda)\mathcal{R}(G, \lambda')$ implique qu'il existe un sommet $v \in V(G)$ tel que :

$$\lambda(x) = \lambda'(x) \text{ pour tout } x \notin V(B_G(v)) \cup E(B_G(v)).$$

La prochaine définition exprimera qu'un système de réétiquetage de graphes local $\mathcal{R}$ est localement engendré si sa restriction aux boules de rayon 1 détermine son comportement sur tout graphe.

Définition 2 : Système de réétiquetage localement engendré

Soit $\mathcal{R}$ un système de réétiquetage de graphes. Alors $\mathcal{R}$ est localement engendré si les conditions suivantes sont satisfaites :

Pour tous graphes étiquetés (G, λ), (G, λ'), (H, η), (H, η') pour tous sommets $v \in V(G)$, $w \in V(H)$ tels que les boules $B_G(v)$ et $B_H(w)$ sont isomorphes via $\varphi : V(B_G(v)) \rightarrow V(B_H(w))$, avec $\varphi(v) = w$,

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

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

Nous n'utiliserons que des systèmes de réétiquetage de graphes localement engendrés. Le comportement de tels systèmes étant déterminé par leur comportement sur les boules, un système de réétiquetage sera donc décrit par la donnée d'un ensemble de couples de boules étiquetées, de graphe non étiqueté sous-jacent identique. La première composante indique la condition préalable et la seconde composante le réétiquetage.

Étant donné un système de réétiquetage de graphes $\mathcal{R}$, un pas de calcul sur le graphe G est l'application en une boule de G d'une règle $\mathcal{R}_i=(C_i, \mathcal{R}_i)$ de $\mathcal{R}$, c'est-à-dire la substitution, dans G, d'une boule de rayon 1 isomorphe à la condition préalable C_i par le réétiquetage $\mathcal{R}_i$. Une exécution de $\mathcal{R}$ sur le graphe G est alors une suite $G = G_0 \mathcal{R} G_1 \mathcal{R} G_2 \mathcal{R} \ldots \mathcal{R} G_i \mathcal{R} \ldots$ de pas de calcul tels que pour tout i, $G_i \mathcal{R} G_{i+1}$.

On utilisera indistinctement les termes réécritures et réétiquetages, étant entendu qu'ici il s'agit toujours, à strictement parler, de réétiquetages.

Remarques :

Il faut souligner que les définitions précédentes ne précisent pas a priori les conditions de terminaison, un calcul se termine lorsque plus aucune règle ne peut s'appliquer, il s'agit d'une terminaison implicite. Un sommet v peut également rechercher à détecter localement la

terminaison locale (son étiquette $\lambda(v)$ ne sera plus modifiée) ou globale (le réseau a atteint un état irréductible).

Il faut également noter que puisque nous raisonnerons sur les chaînes de réétiquetages, et en particulier sur leur dernier terme -irréductible-, les exécutions seront (implicitement) supposées "équitables" (fair), c'est à dire que le système atteindra toujours un état irréductible.

Notations

Une exécution de l'algorithme est ainsi déterminée par une succession de règles de $\mathcal{R}$ appliquée en une boule particulière. On adoptera donc la notation suivante pour une exécution ρ de $\mathcal{R} = \{ \mathcal{R}_j \mid j \in J \}$:

$$\rho = \mathcal{R}_{i1} B_{i1} \mathcal{R}_{i2} B_{i2} \ldots$$

Où $\mathcal{R}_{ij} \in \mathcal{R}$ et B_{ij} est la boule de rayon 1 où le pas de calcul est effectué.

Lorsqu'il n'y a pas d'ambiguïté dans le réétiquetage des voisins, on indiquera uniquement le centre de la boule et on notera :

$$\rho = \mathcal{R}_{i1} u_{i1} \mathcal{R}_{i2} u_{i2} \ldots$$

Où $\mathcal{R}_{ij} \in \mathcal{R}$ et u_{ij} est le centre de la boule de rayon 1 où le pas de calcul est effectué.

L'ensemble des exécutions partielles possibles sur un graphe G est notée $E_G(\mathcal{R})$.

Si l'on s'intéresse à la complexité, c'est-à-dire ici, en première approximation, au nombre de pas de calcul, on notera la chaîne $\mathcal{R}_{i1} \mathcal{R}_{i2} \ldots$ et $\mid \mathcal{R}_{i1} \mathcal{R}_{i2} \ldots \mid$ correspondra à sa longueur, c'est-à-dire au nombre de termes R_{ij}.

II.1.2 Sérialisation

Les notions et notations de suites de réétiquetages définies ci-dessus correspondent de manière évidente à une notion de calcul séquentiel. Il est important de noter qu'un système de réétiquetage de graphes permet également des réécritures parallèles, puisque des boules qui ne se recouvrent pas peuvent être réétiquetées indépendamment.

On peut donc définir une exécution distribuée en disant que deux pas de réétiquetages consécutifs appliqués à des boules disjointes peuvent être effectués dans n'importe quel ordre. On dit que de tels pas commutent et peuvent être exécutés de manière concurrente.

Plus généralement, deux suites de réétiquetages partant du même graphe étiqueté et telles que l'on peut passer de l'une à l'autre par de telles commutations aboutiront au même résultat. Donc notre notion de suite de réécritures peut être vue comme une *sérialisation* [9] d'une exécution distribuée donnée.

Ce modèle est clairement asynchrone : plusieurs pas de réétiquetage peuvent être effectués en même temps (mais ne doivent pas nécessairement l'être). Par la suite, on présentera essentiellement les réécritures comme étant séquentielles, mais il sera important de se rappeler qu'elles pourraient être effectuées de manière distribuée.

II.2 Revêtements

Le but est de fournir une approche permettant de caractériser ce qui est calculable de manière distribuée et ce qui ne l'est pas. Et comme le souligne N. Lynch dans [10], ce qui est impossible en algorithmique distribuée l'est en règle générale pour des raisons de "similarités", la définition des similarités étant à chaque fois spécifique au problème considéré. Ici, les réseaux "similaires" se comportent de manière similaire car l'on peut simuler ce que l'on fait avec l'un sur l'autre. Les propriétés que l'on cherchera à prouver dans le modèle de réétiquetage précédemment défini utiliseront, implicitement, la notion de simulation. Les outils permettant de manipuler ces similarités et simulations sont des morphismes de graphes conservant les propriétés locales sur lesquelles s'appuient les règles de réétiquetages, *i.e.* conservant les boules : les revêtements.

Nous commencerons par donner les définitions et quelques propriétés immédiates. Puis nous donnerons le lien avec les systèmes de réétiquetages.

II.2.1 Définitions et premières propriétés

Définition 3 : revêtement

Soit γ un homomorphisme surjectif de G sur H. Le morphisme γ est un *revêtement* si pour tout sommet $u \in V(G)$, γ induit un isomorphisme de $B_G(u)$ sur $B_H(\gamma(u))$.

Un revêtement de *G* sur *H* est dit *propre* si *G* et *H* ne sont pas isomorphes.

Remarque : Un revêtement est exactement un revêtement au sens classique de la topologie, voir [11]. A noter que pour parler rigoureusement, le revêtement est le morphisme. Par extension et abus de langage, on parlera d'un graphe comme étant un revêtement d'un graphe donné.

Définition 4 : graphe minimal

Un graphe est *minimal* s'il n'est revêtement propre d'aucun autre graphe. La famille des graphes minimaux sera noté $\mathcal{G}_{min}$.

On étend la notion de revêtement aux graphes étiquetés de la manière suivante. Le morphisme γ est un revêtement de *(G, λ)* sur (*H, λ'*), si γ est un homomorphisme de (*G*, *λ*) sur (*H*, *λ'*) dont la restriction à $B_G(u)$ est un isomorphisme de $(B_G(u), \lambda_{BG(u)})$ sur $(B_H(\gamma(u)), \lambda'_{(BH(\gamma(u))})$.

Exemple 1 : revêtement

Un premier exemple simple de revêtement est donné par sur la figure 4 . L'image sur *H*, par le revêtement, de chacun des sommets du graphe *G* est donnée par la lettre romane correspondante. En outre, on remarquera que l'image de chaque sommet de *G* est également donné par sa position sur le motif de *H*.

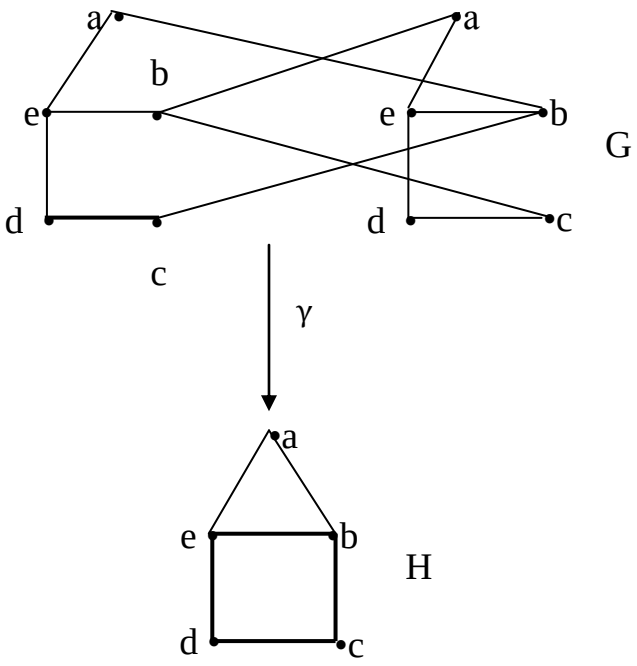


Figure 4 : *G* est un revêtement de *H*

Définition 5 : revêtement à q feuillets

Précisons maintenant l'intuition que l'on peut avoir d'un revêtement :

- Soit γ un revêtement de G sur H. Soit u un sommet de H. Alors $\gamma^{-1}(B_H(u))$ est une collection de boules disjointes isomorphes à $B_H(u)$.
- Soit H un graphe connexe et G un revêtement fini de H via γ. Alors il existe un entier q tel que, pour tout sommet u de $V(H)$, on a $card(\gamma^{-1}(u)) = q$

➔ On dit alors que γ est un revêtement à q feuillets.

L'exemple 1 est un revêtement à 2 feuillets. On peut décrire *tous* les revêtements d'un graphe d'un nombre de feuillets q donné. Ce sera l'objet de la section II.2.4.

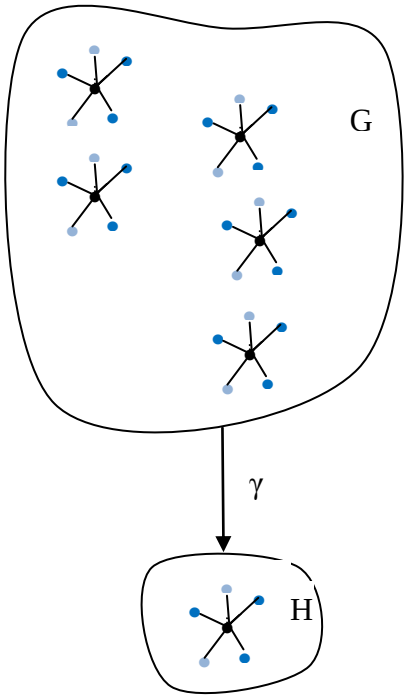


Figure 5 : les antécédents d'une boule de H forment une collection de boules disjointes de G

Les propriétés suivantes sont des illustrations de ce qui précède.

Propriétés :

- Soit n, m deux entiers. Alors l'anneau A_n est un revêtement de l'anneau A_m si et seulement si $m \mid n$. Dans ce cas, $\frac{n}{m}$ est le nombre de feuillets.
- Soit G un graphe connexe, si $/V(G)/$ et $/E(G)/$ sont premiers entre eux, alors G est minimal. En particulier, les graphes de taille premières sont minimaux, et un anneau A_n est minimal si et seulement si n est premier.

II.2.2 Revêtements et réétiquetages locaux

Nous allons voir immédiatement comment les revêtements permettent d'étudier les systèmes de réétiquetages. Le lemme suivant fait le lien entre l'existence d'un revêtement et le réétiquetage de deux graphes.

Lemme de Relèvement

([12]) : Soit $\mathcal{R}$ un système de réétiquetage de graphes et soit $\gamma : G \rightarrow H$ un revêtement. Supposons qu'il existe un graphe H' tel que $H \; \mathcal{R}^* \; H'$. Alors il existe un graphe étiqueté G' (de même graphe sous-jacent que G) tel que :

- $G \; \mathcal{R}^* \; G'$
- G' soit un revêtement de H'

La figure suivante résume ce lemme qui signifie intuitivement que *pour tout système de réétiquetage de graphes $\mathcal{R}$, on peut simuler sur G toute exécution de $\mathcal{R}$ sur H.*

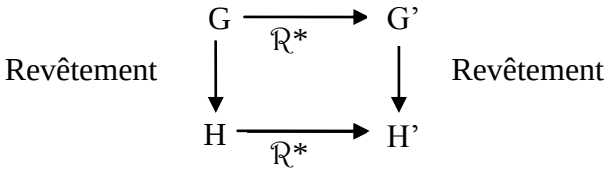


Figure 6 : Revêtement et réétiquetage

II.2.3 Quasi-revêtements

Les quasi-revêtements ont été introduits dans [13] pour étudier la détection de la terminaison des systèmes de réétiquetage. Ces sont des applications qui se comportent partiellement comme des revêtements.

Définition 6 : Quasi-revêtement

Soit K et H deux graphes, et soit δ une fonction (partielle) de $V(K)$ dans $V(H)$. δ est un *quasi-revêtement* de rayon r s'il existe un graphe G (éventuellement infini) qui soit revêtement de H via γ, et s'il existe $z_K \in V(K)$, $z_G \in V(G)$ tels que :

- il existe un isomorphisme $\varphi' : B_K(z_K, r) \rightarrow B_G(z_G, r)$
- Le domaine de définition de δ contient $B_K(z_K, r)$
- $\delta \,|V(B_K(z_k, r)) = \delta \; o \; \varphi \,|V(B_K(z_K, r))$

L'entier $s = |B_K(z_K, r)|$ est appelé l'*étendue* du quasi-revêtement et le graphe G le revêtement associé.

Par abus de langage, nous dirons que K est un *quasi-revêtement de H de rayon r*. On notera qu'en particulier, tout revêtement de H est également un quasi-revêtement pour n'importe quel rayon $r \in \mathbb{N}$. Et que tout quasi-revêtement de rayon r est aussi quasi-revêtement de rayon $r' \leq r$.

Définition 7 : Nombre de feuillets d'un quasi-revêtement

En utilisant les notations précédentes :

On définit le *nombre de feuillets* q d'un quasi-revêtement comme étant la cardinalité minimale des ensembles d'antécédents des sommets de H :

$$q = \min_{w \in H} \{card(\{v \in B_K(z_K, r) | \delta(v) = w\})\}$$

Définition 8 : Quasi-revêtement strict

Un quasi-revêtement est dit *strict* si $B_K(z_K, r-1) \not\subseteq K$

Un quasi-revêtement *non strict* est un revêtement, et dans ce cas les deux définitions du nombre de feuillets coïncident.

Exemple 2 : Quasi-revêtement

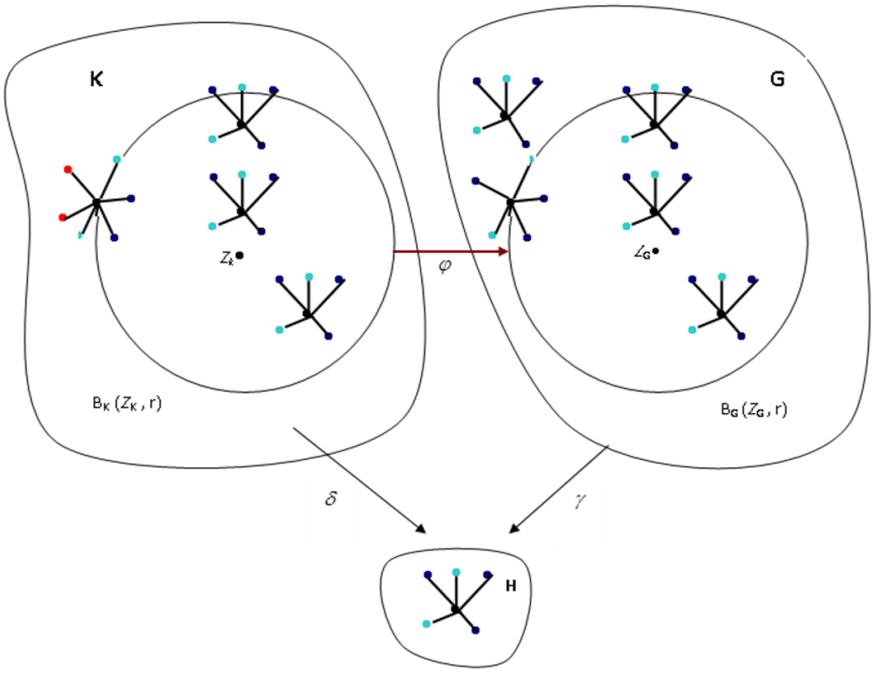


Figure 7 : δ : $K \rightarrow H$ est un quasi-revêtement de rayon r et revêtement associé γ : G→H

Le lemme suivant est une extension du lemme de Relèvement, si un graphe est un quasi-revêtement, on pourra simuler, mais le rayon de simulation diminuera à chaque simulation.

Lemme de Quasi-relèvement

Soit $\mathcal{R}$ un système de réétiquetage de graphes et soit K un quasi-revêtement de H _ via γ de rayon $r \geq 2$. Supposons qu'il existe un graphe H' tel que $H \mathcal{R} H'$

Alors il existe un graphe K′ tel que

- $K \mathcal{R}^* K'$
- K' est un quasi-revêtement de H' de rayon *r-2*

Ainsi, dans le cas d'une relation de quasi-revêtement, on peut, comme dans le cas des revêtements, effectuer des simulations mais seulement un nombre fini de fois.

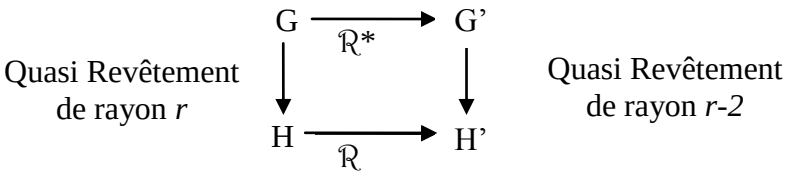


Figure 8 : Quasi-revêtement et réétiquetage

II.2.4 Description des revêtements à *q* feuillets : méthode de Reidemeister

K. Reidemeister a présenté dans [14] une méthode de construction de tous les revêtements d'un graphe donné. Il a en effet montré que tout revêtement d'un graphe *G* donné se construit à l'aide d'un arbre recouvrant de *G* dont on fait plusieurs copies et que l'on relie à l'aide des arêtes n'appartenant à l'arbre couvrant. La façon de connecter les copies de l'arbre couvrant est décrite par un ensemble de permutations.

Soit G un graphe connexe, A un arbre recouvrant de G et q un entier. On note F l'ensemble des arêtes de G qui ne sont pas dans A et on associe à chaque arête de F une permutation de l'ensemble $\{1, ..., q\}$

Nous allons relier q copies de l'arbre recouvrant entre elles grâce aux q copies de chaque arête de F obtenues en permutant leurs extrémités selon la permutation associée à l'arête. Le graphe que l'on obtient de cette façon est un revêtement de G à q feuillets.

La figure 9 présente une telle construction, les permutations sont décrites par leur décomposition en cycles.

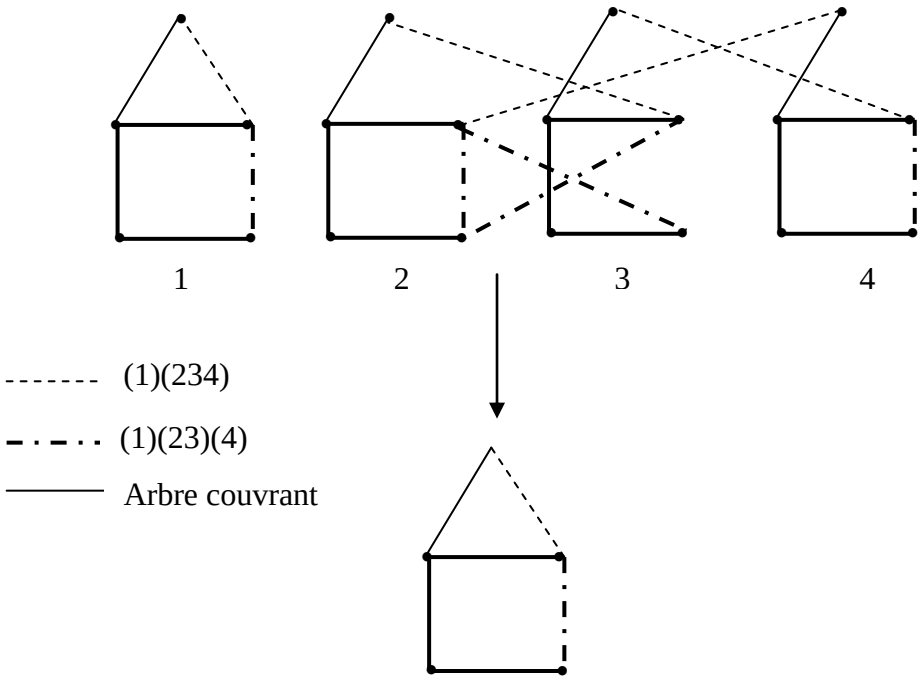


Figure 9 : La construction de Reidemeister pour 4 feuillets. Les permutations sont données comme produit de permutations circulaires.

III. Quelques présentations sur l'utilisation de revêtements dans les systèmes distribués.

Dans ce chapitre, nous donnerons quelques exemples d'utilisation des revêtements dans les systèmes distribués, nous étudierons les trois exemples suivants :

- Algorithme de Mazurkiewicz
- Problème d'élection dans un réseau
- Preuves d'impossibilité de résolution de problèmes distribués de consensus

III.1 Algorithme de Mazurkiewicz

Pour comprendre l'utilisation de l'algorithme de ***Mazurkiewicz*** [15] dans des problèmes de revêtement dans les systèmes distribués, nous allons décrire d'abord son modèle d'utilisation.

III.1.1 Modèlee

Dans le modèle considéré, un pas de calcul permet à un sommet de modifier son état et celui de ses voisins en fonction de son propre état et de l'état de ses voisins. Dans ce modèle, cet algorithme peut être décrit par un ensemble de règles de réétiquetage dont le support est une étoile. Ces règles sont de la forme de la règle présentée sur la figure ci-dessous :

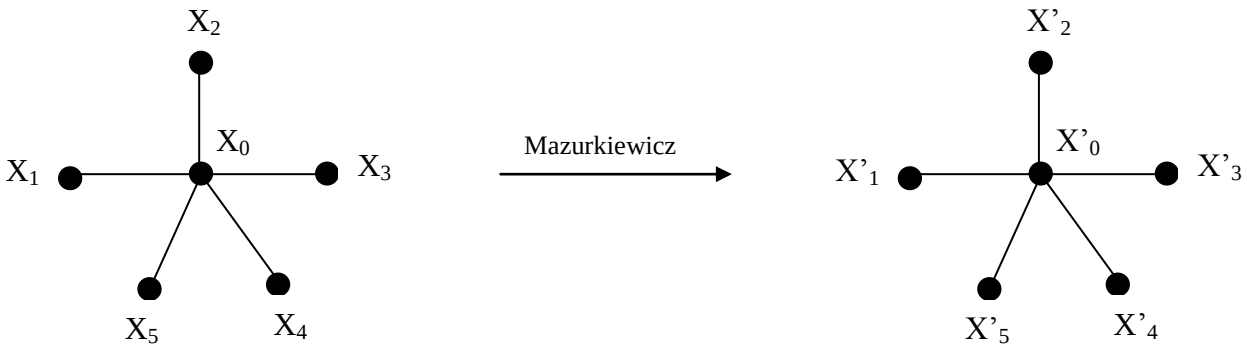


Figure 10 : La forme d'une règle de réétiquetage dans le modèle de Mazurkiewicz

.

Cependant, comme pour l'algorithme d'élection dans les arbres, on utilisera des règles décrites de manière générique.

III.1.2 Rappels sur les revêtements

III.1.2.1 Rappel 1 : homomorphisme

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

Un exemple d'homomorphisme est présenté sur la figure ci-dessous.

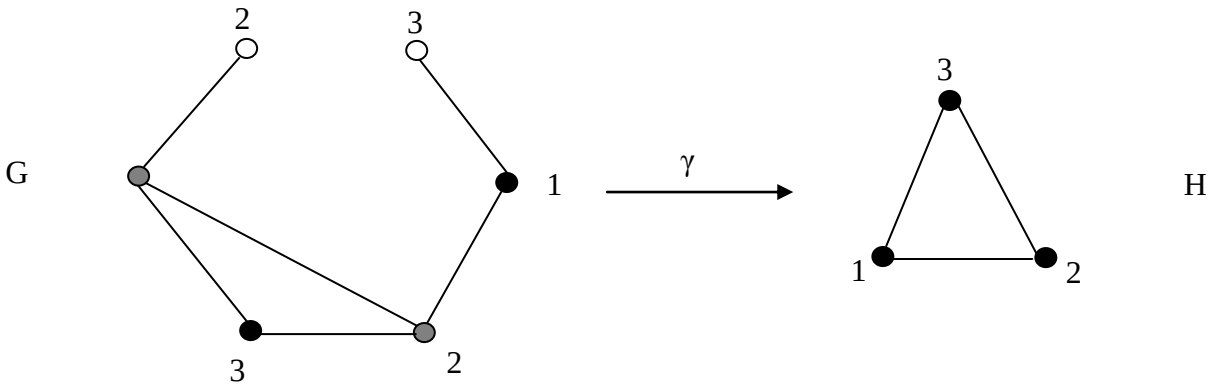


Figure 11 : La fonction γ qui envoie chaque sommet de G étiqueté i sur l'unique sommet de H dont l'étiquette est i est un homomorphisme de G dans H.

III.1.2.2 Rappel 2 : isomorphisme

Un homomorphisme φ d'un graphe ***G*** dans un graphe ***H*** est un isomorphisme si φ est une fonction bijective alors φ^{-1} est aussi un isomorphisme.

Les homomorphismes localement bijectifs, i.e., les revêtements, sont les homomorphismes qui permettent de donner des conditions nécessaires que doivent vérifier les graphes admettant un algorithme d'élection dans le modèle de Mazurkiewicz.

III.1.2.3 Rappel 3 : revêtement

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

- $/ N_G(u) / = / N_H((u)) /$
- $\gamma(N_G(u)) = N_H(\gamma(u))$

On dit alors que l'homomorphisme γ est localement bijectif

Un graphe ***G*** est un revêtement propre de ***H*** si γ n'est pas un isomorphisme et ***G*** est ***minimal*** pour les revêtements si ***G*** n'est un revêtement propre d'aucun autre graphe.

III.1.2.4 Exemples

Le graphe ***G*** de la figure 12 est un revêtement de ***H*** à travers l'homomorphisme γ.

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

Le graphe ***G*** de la figure 11 n'est pas un revêtement de ***H*** à travers γ, car les sommets blancs et gris de G n'ont pas autant de voisins que leurs images (les blancs en ont moins, les gris en ont plus).

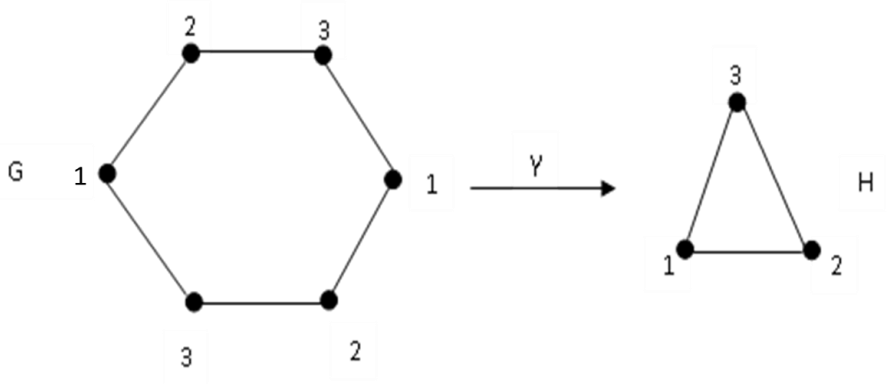


Figure 12 : Le graphe G est un revêtement de H à travers l'homomorphisme γ qui envoie chaque sommet de G étiqueté i sur l'unique sommet de H dont l'étiquette est i

III.1.3 Propriétés

- Si ***G*** est un revêtement 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***

- Si un ***G*** est un revêtement d'un graphe connexe ***H*** à travers γ, alors γ est surjectif, c'est-à-dire, pour des graphes connexes, tout homomorphisme localement bijectif est surjectif.

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

Alors si un graphe ***G*** est un revêtement d'un graphe connexe ***H*** à travers γ, 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.

Exemples : Voici quelques exemples de graphes connexes qui sont minimaux pour les revêtements.

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

III.1.4 Résultat d'impossibilité

On présente maintenant le lemme qui met en évidence le lien entre les calculs locaux et les revêtements. Ce lemme a été prouvé par Angluin en 1980 [12] :

On dit qu'un graphe étiqueté $G = (G, \lambda)$ est un revêtement de $H = (H, \eta)$ à travers γ si G est un revêtement de H à travers γ et si γ préserve l'étiquetage, i.e., pour tout $v \in V(G)$, $\lambda(v) = \eta(\gamma(v))$.

Relèvement: On considère un graphe ***G*** qui est un revêtement d'un graphe ***H*** à travers un homomorphisme γ et un algorithme ***A*** utilisant des calculs locaux. S'il existe une exécution de ***A*** sur ***H*** qui permet d'atteindre une configuration ***H'***, alors il existe une exécution de ***A*** sur ***G*** qui permet d'atteindre une configuration ***G'*** tel que ***G'*** est un revêtement de ***H'*** à travers γ.

On déduit de ce lemme de relèvement le résultat d'impossibilité suivant qui a aussi été prouvé par Angluin en 1980 :

Soit G un graphe qui n'est pas minimal pour les revêtements. Il n'existe pas d'algorithme d''élection pour le graphe G utilisant des calculs locaux.

III.1.5 Algorithme d'énumération

III.1.5.1 Description

On va maintenant décrire l'algorithme d'énumération ***M*** de ***Mazurkiewicz*** qui permet de résoudre le problème de l'énumération sur les graphes minimaux pour les revêtements [15].

Durant l'exécution de l'algorithme, chaque sommet v essaie d'obtenir une identité qui est un numéro entre 1 et $|V(G)|$. A 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, alors chaque sommet a un numéro unique.

III.1.5.2 Etiquettes

On considère un graphe ***G***. Lors de l'exécution, chaque sommet va obtenir une étiquette de la forme $(n(v), N(v), M(v))$ qui représente les informations suivantes :

- $n(v) \in \mathbb{N}$ est le numéro courant du sommet v qui est modifié lors de l'exécution de l'algorithme,
- $N(v) \in P_{fin}(\mathbb{N})$ ($P_{fin}(\mathbb{N})$ est l'ensemble des ensembles finis d'entiers.) est *la vue locale* du sommet v qui contient des informations sur les voisins de v ; c'est un ensemble fini d'entiers. A tout moment de l'exécution, pour chaque voisin v' de v' tel que $n(v') \neq 0$, la vue locale de v contient $n(v')$.
- $M(v) \subseteq \mathbb{N} \times P_{fin}(\mathbb{N})$ 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 $(0, \emptyset, \emptyset)$ 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.

III.1.5.3 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.

Ensuite, étant données deux ensembles $N1, N2 \in P_{fin}(\mathbb{N})$ distincts, on dit que $N1 \prec N2$ si le maximum de la différence symétrique $N1 \Delta N2 = (N1 \setminus N2) \cup (N2 \setminus N1)$ appartient à N2.

On peut aussi voir cet ordre comme l'ordre lexicographique usuel sur les ensembles ordonnés *N1* et *N2*. On note $n_1, n_2, \ldots, n_k$ et $n'_1, n'_2, \ldots, n'_l$ les éléments respectifs de *N1* et de *N2* respectivement avec : $n_1 \geq n_2 \geq \ldots \geq n_k$ et $n'_1 \geq n'_2 \geq \ldots \geq n'_l$. Alors N1 $\prec$ N2 si l'une des conditions suivantes est vérifiée :

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

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 $N(v)$. On note $N(u) \leq N(v)$ lorsque $N(u)=N(v)$ ou $N(u) \prec N(v)$.

III.1.5.4 Les Règles de réetiquetage

On décrit maintenant l'algorithme d'énumération grâce à des règles de réetiquetage.

Les règles sont décrites pour une étoile (boule) $B(v_0)$ 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 $(n(v), N(v), M(v))$ et on note $(n'(v), N'(v), M'(v))$ l'étiquette de v après l'application de la règle de réetiquetage. 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

Pré-condition :

- $\exists\, v \in N_G(v_0)$ 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 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 $(n'(v),N'(v))$ pour tous les sommets v de l'étoile $B(v_0)$.

$\mathcal{M}2$: Règle de Renommage

Pré-condition :

- $\forall v \in N_G(v_0),\ M(v)=M(v_0)$
- $n(v_0) = 0$ ou $\exists (n(v_O, N) \in M(v_0)$ tel que $N(v_0) \prec N$

Réétiquetage:

- $n'(v_O) := 1 + max\{n' \mid \exists (n', N') \in M(v_O)\}$
- $\forall v \in N_G(v_O),\ N'(v) := N(v) \setminus \{n(v_O)\} \cup \{n'(v_O)\}$
- $\forall v \in V(B(v_O)),\ M'(v) := M(v) \cup \bigcup_{w \in V(B(v_0))} \{(n'(w), N'(w))\}$

III.1.5.5 Correction de l'Algorithme d'Enumération

On considère un graphe ***G***. Pour tout sommet $v \in V(G)$, on note $(n_i(v), N_i(v), M_i(v))$ 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 [15].

Propriétés satisfaites lors de l'exécution :

Les propriétés suivantes, qui peuvent être facilement prouvées par une récurrence sur le nombre d'étapes, rappelle quelques propriétés qui sont toujours satisfaites par l'étiquetage : Pour tout sommet $v \in V(G)$, et pour toute étape *i*

1. $n_i(v) \neq 0 \Rightarrow (n_i(v), N_i(v)) \in M_i(v)$
2. $\exists n \in N_i(v) \Leftrightarrow \exists v' \in N_G(v)$ tel que $N_i(v) = n > 0$
3. $\forall n \in N_i(v)$, $n \neq n_i(v)$ et $\exists (n,N) \in M_i(v)$
4. $\forall v', v'' \in N_G(v)$, $n_i(v)$, $n_i(v') > 0 \Rightarrow n_i(v') \neq n_i(v'')$

L'algorithme $\mathcal{M}$ a des propriétés de monotonie intéressantes :

Pour chaque sommet v et chaque étape i,

- $n_i(v) \leq n_{i+1}(v)$,
- $N_i(v) \prec N_{i+1}(v)$,
- $M_i(v) \subseteq M_{i+1}(v)$.

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.

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 propriétés suivantes permettent de prouver que si un sommet v connaît un numéro m à une étape i (i.e., il existe N tels que $(m,N) \in Mi(v)$), 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 $n_i(w) = m$

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

- Pour chaque sommet v et chaque étape i, pour tout $(m,N) \in M_i(v)$, pour tout $m' \in [1,m]$, il existe $(m',N') \in M_i(v)$.

(C'est-à-dire, si un sommet v connaît un numéro m, alors il connaît tous les numéros inférieurs à m)

On va maintenant montrer que toute exécution de l'algorithme $\mathcal{M}$ termine sur $\boldsymbol{G}$. D'après les propriétés précédentes, 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 la propriété de monotonie, 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 n', alors il existe $v' \in N_G(v)$ tel que $n_i(v') = n'$. 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.

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 :

- Toute exécution ρ de l'algorithme $\mathcal{M}$ sur un graphe $\boldsymbol{G}$ termine et l'étiquetage final (n_ρ, N_ρ, M_ρ) vérifie les propriétés suivantes :
- 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' :

- $M_\rho(v) = M_\rho(v')$
- $(n_\rho(v), N_\rho(v)) \in M_\rho(v')$
- si $n_\rho(v) = n_\rho(v')$, alors $N_\rho(v) = N_\rho(v')$
- $\forall w,w' \in N_G(v)$, $n_\rho(w) \neq n_\rho(w')$
- $n \in N_\rho(v)$ si et seulement s'il existe $w \in N_G(v)$ tel que $n_\rho(w) = n$; auquel cas, $n_\rho(v) \in N_\rho(w)$.

On peut alors prouver que l'étiquetage final permet de construire un graphe $\boldsymbol{H}$ tel que $\boldsymbol{G}$ est un revêtement de $\boldsymbol{H}$: *On peut construire, à partir de l'étiquetage final obtenu après une exécution ρ de $\mathcal{M}$, un graphe $\boldsymbol{H}$ tel qu'il existe un homomorphisme localement bijectif de G dans H.*

On considère maintenant un graphe $\boldsymbol{G}$ qui est minimal pour les revêtements. Pour chaque exécution ρ de $\mathcal{M}$ sur $\boldsymbol{G}$, le graphe obtenu à partir de l'étiquetage final est isomorphe à $\boldsymbol{G}$. Par conséquent, l'ensemble des numéros des sommets est exactement $[1,|V(G)|]$: chaque sommet a un identifiant unique.

De plus, une fois qu'un sommet a obtenu le numéro $|V(G)|$, il sait que tous les sommets de $\boldsymbol{G}$ ont un numéro unique qui ne va plus être modifié. Dans ce cas-là, ce sommet

peut prendre l'étiquette «*élu*» et diffuser ensuite l'information qu'un sommet a été élu. On a donc obtenu un algorithme d'élection pour le graphe ***G***.

Par ailleurs, on sait que pour tout graphe ***G*** qui n'est pas minimal pour les revêtements, il n'existe aucun algorithme d'élection utilisant des calculs locaux pour le graphe ***G***. On a donc prouvé le théorème suivant :

*Pour tout graphe **G**, il existe un algorithme d'élection pour **G** si et seulement si **G** est minimal pour les revêtements.*

III.2 Problème d'élection dans un réseau

Dans ce chapitre nous allons étudier le problème de l'élection dans un réseau. Ce problème, qu'aucun considère comme un véritable paradigme de l'algorithmique distribuée, sera étudié et résolu.

III.2.1 Élection

L'élection dans un réseau consiste à distinguer par calcul distribué, parmi l'ensemble des nœuds un unique nœud. Celui-ci pourra ensuite effectuer une opération n'autorisant qu'un seul accès exclusif, ou encore centraliser et coordonner un comportement général du réseau, par exemple pour un redémarrage après une panne. Un grand nombre de problèmes se réduise souvent à "élire" un coordinateur pour ensuite faire effectuer au réseau, sous sa coordination, les opérations idoines.

Ce problème a été étudié très largement, et un nombre très important d'algorithmes est disponible dans la littérature. Les premiers résultats théoriques où il est apparu que ce problème n'avait parfois pas de solution sont ceux d'Angluin. Nous allons rappeler ces résultats ainsi que ceux qui ont suivi.

III.2.2 Définitions

On va utiliser deux étiquettes particulières *ÉLU* et *PAS ÉLU*. Celles-ci seront des étiquettes terminales, *i.e.*, au cours de toute chaîne de réétiquetages, si l'une de ces étiquettes apparaît sur un sommet u, alors ce sommet conservera cette étiquette.

III.2.2.1 Définition 1

Soit ***G*** un graphe. On dit qu'un système de réétiquetage de graphes $\mathcal{R}$ élit sur ***G*** si pour toute chaîne de réétiquetage, on a :

- La chaîne est finie
- L'étiquetage final est de la forme *(G, λ)* avec :
  - il existe un unique sommet u_0 tel que $\lambda(u_0) = \acute{E}LU$,
  - pour tout $u \neq u_0$, $\lambda(u) = PAS\ \acute{E}LU$.
- Les étiquettes *ÉLU* et *PAS ÉLU* sont terminales : pour tout sommet v, si à un moment donné, l'étiquette de v est ÉLU (resp. *PAS ÉLU*) alors, pour tous les réétiquetages suivants, l'étiquette de v demeure *ÉLU* (resp. *PAS ÉLU*).

III.2.2.2 Définition 2

Soit $\mathcal{F}$ une famille de graphe. On dit qu'un système de réétiquetage de graphe $\mathcal{R}$ est un algorithme d'élection *universel* pour $\mathcal{F}$ si, pour tout graphe ***G*** de $\mathcal{F}$, $\mathcal{R}$ élit sur G.

L'étude de l'élection effective, c'est-à-dire en fonction de la connaissance structurelle, sera réalisée par la suite dans la section III.2.6.

Nous allons maintenant présenter un système de réétiquetage, ELECTION_ARBRE, qui est un algorithme d'élection universel pour la famille des arbres. Il consiste en deux règles, l'une *"élague"* l'arbre, c'est-à-dire que chaque feuille peut devenir *PAS ÉLU* ; et l'autre qui élit le dernier sommet étiqueté ε. La correction de l'algorithme repose sur le fait, qu'à tout moment, le sous-graphe des sommets étiquetés ε est connexe.

ELECTION_ARBRE_1 : **Élagage**

Condition préalable :

- $\lambda(v_0) = \varepsilon$
- $\exists\, !\, v \in B(v_0), \lambda(v) = \varepsilon$

Réétiquetage :

- $\lambda'(v_0) = PAS\ \acute{E}LU$

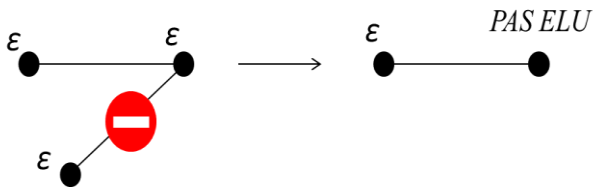


Figure 13 : l'Elagage

ELECTION_ARBRE_2 : **Élection finale**

Condition préalable :

- $\lambda(v_0) = \varepsilon$
- $\forall v \in B(v_0), \lambda(v) \neq \varepsilon$

Réétiquetage :

- $\lambda'(v_0) = ÉLU$

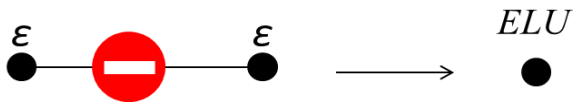


Figure 14 : Election du sommet restant

Le schéma suivant présente un exemple d'exécution du système ELECTION_ARBRE.

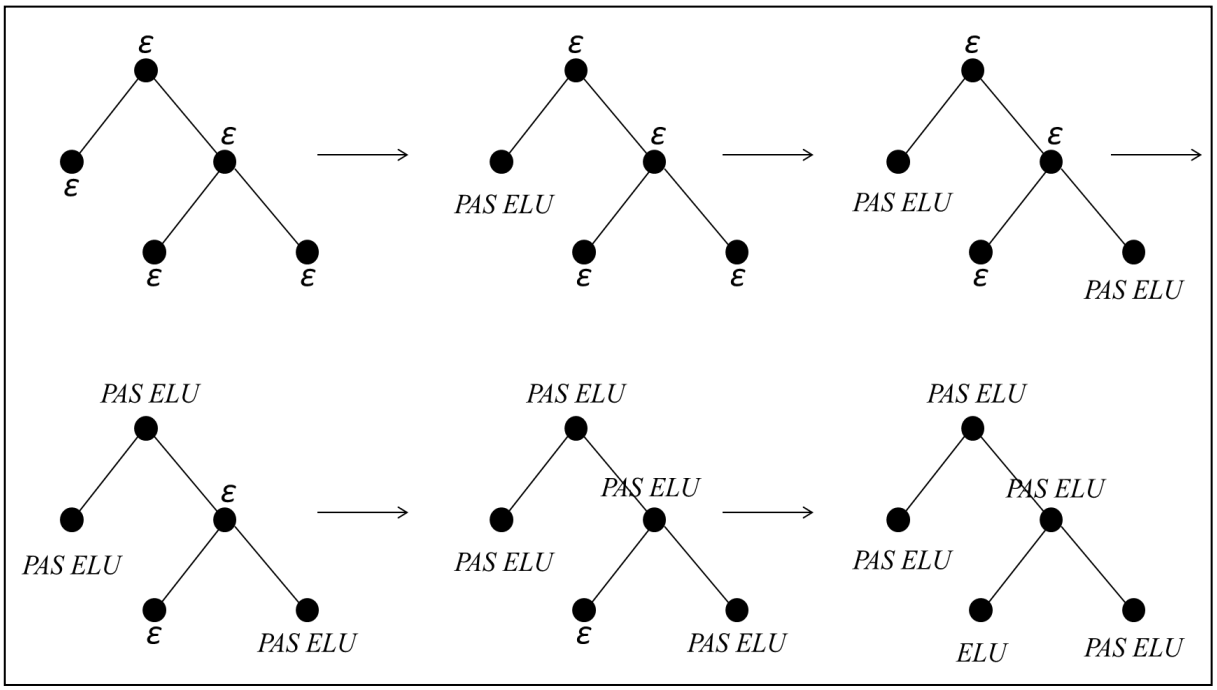


Figure 15 : Exemple d'élection

Il faut noter que l'on connaît un certain nombre d'autres résultats concernant l'élection. Outre pour les arbres, il existe des algorithmes d'élection pour des réseaux anonymes tels que les anneaux orientés premiers [16], les graphes T-premiers [17], les graphes complets...

III.2.3 La condition nécessaire d'Angluin

Dans [12], Angluin a présenté une preuve d'impossibilité de l'existence d'algorithme d'élection pour certains types de réseaux. C'est à cette occasion que le concept de revêtement fut appliqué à l'algorithmique distribuée pour la première fois.

Le théorème d'Angluin dit que : ***si G* est *un graphe non minimal pour la relation revêtement. Alors il n'existe pas d'algorithme d'élection pour G***

Pendant longtemps la réciproque de ce théorème, c'est-à-dire la question de savoir si l'on pouvait élire dans tous les graphes minimaux est restée non résolue, Mazurkiewicz l'a, cependant, démontré en 1997

III.2.4 Election avec l'algorithme de Mazurkiewicz

L'algorithme d'énumération de Mazurkiewicz se révèle être en fait un algorithme d'élection dès que l'on connaît la taille du réseau.

(Mazurkiewicz, 1997): *Soit **G** un graphe minimal. Il existe un algorithme qui élit sur **G**.*

III.2.5 Algorithmes universels d'élection

L'algorithme $\mathcal{M}_t$ est universel pour la famille des graphes de taille ***t*** et l'existence d'un algorithme universel, élisant sur tous les réseaux minimaux reste une question ouverte à ce stade.

III.2.5.1 Un théorème d'impossibilité

Un arrêt fut mis à la recherche de cet éventuel algorithme universel en 1997 qui prouvait qu'en particulier il n'y avait pas d'algorithme universel d'élection pour les anneaux minimaux (ceux de taille première), et donc *a fortiori* pour l'ensemble des graphes minimaux.

Théorème : (Métivier Muscholl Wacrenier, 1997).

Soit $\mathcal{F}$ une famille de graphes. S'il existe un graphe ***G*** de $\mathcal{F}$ admettant des quasi-revêtements d'étendue arbitrairement grande dans $\mathcal{F}$, alors il n'existe pas d'algorithme universel d'élection pour $\mathcal{F}$.

C'est alors qu'on déduit les propriétés suivantes :

- Il n'existe pas d'algorithme universel d'élection pour la famille des anneaux.
- L'ensemble des familles de graphes admettant un algorithme universel d'élection n'est pas stable pour l'union.

En fait, on note, puisque les troncatures du revêtement universel de tout graphe sont des arbres, que l'ajout de la famille des arbres à toute famille contenant un graphe qui ne soit pas un arbre provoque l'impossibilité d'avoir un algorithme universel d'élection pour l'union de ces deux familles.

III.2.5.2 Entre possibilité et impossibilité

On a remarqué précédemment que l'algorithme de Mazurkiewicz permet d'élire pour la classe des graphes minimaux d'une taille donnée. Cependant le théorème présenté précédemment (**Métivier Muscholl Wacrenier, 1997)** a pour corollaire qu'il n'existe pas d'algorithme universel d'élection pour la famille des graphes minimaux, la famille des graphes pour lesquels l'élection est possible.

La question de savoir comment améliorer cet algorithme, par exemple en arrivant à élire pour une famille de graphes minimaux de diamètre borné (on peut légitimement espérer détecter la terminaison de $\mathcal{M}$ si l'on a une telle borne) se pose alors. De même que se pose la question des familles de graphes ne correspondant pas au critère du théorème précédent. Nous allons en fait montrer dans les sections suivantes la réciproque de ce théorème. Pour cela nous allons de nouveau exploiter $\mathcal{M}$ pour lui soutirer davantage d'informations.

III.2.5.3 Caractérisation des algorithmes universels d'élection

Théorème : Soit une famille de graphes. Alors il existe un système de réétiquetage de graphes universel d'élection pour si et seulement si :

- $\mathcal{F}$ est une famille de graphes minimaux pour les revêtements,
- Il existe une fonction récursive $r : \mathcal{F} \to \mathbb{N}$ telle que tout graphe G n'admette aucun quasi-revêtement de rayon $r(G)$ dans $\mathcal{F}$, excepté lui-même.

III.2.5.4 Quelques résultats connus

Un certain nombre de résultats connus apparaissent comme des corollaires directs du théorème précédent. En particulier, les familles suivantes admettent un algorithme universel d'élection :

- Les arbres, les grilles, les réseaux avec identités, les graphes complets : ces réseaux sont minimaux et n'admettent aucun quasi-revêtement à plus de 2 feuillets. La fonction ***r*** peut donc être 2 fois la taille du graphe.

- Les graphes minimaux de taille donnée : en prenant $r(G) = |V(G)|$

III.5.5.5 Nouveaux résultats

On obtient également de nouveaux résultats en termes d'existence ou d'absence d'existence d'algorithme d'élection universel.

Possibilité

- Soit ***d***, un entier. La famille des graphes minimaux de diamètre au plus ***d*** admet un algorithme d'élection.

- Soit ***k*** un entier. La famille des graphes dont les sommets sont étiquetés par 0 ou 1, minimaux (en tant que graphes étiquetés) et possédant au moins un et au plus ***k*** sommets étiquetés par 1 admet un algorithme d'élection.

Impossibilité

En termes d'impossibilité, on retrouve la non-existence d'algorithme universel d'élection pour les anneaux de taille première et plus généralement pour les tores *d-dimensionnels* minimaux. On note $\mathbb{Z}_n$ l'anneau des entiers modulo ***n.***

Définition : Soit $d \in \mathbb{N}$, Le tore d-dimensionnel de longueurs $n_1, \ldots, n_d \in \mathbb{N}^*$ est le graphe $T_{n_1,\ldots n_d}$ défini par :

$$V(T_{n_1,\ldots n_d}) = \mathbb{Z}_{n1} \times \ldots. \times \mathbb{Z}_{nd}$$

$$V(T_{n_1,\ldots n_d}) = \left\{ \{(x_1,\ldots,x_d),(y_1,\ldots,y_d)\} \mid \exists i \forall j \neq i \; x_j = y_j, x_i - y_i = \pm 1 \bmod n_i \right\}$$

Résultat : Soit $d \in \mathbb{N}$. Il n'existe pas d'algorithme d'élection universel dans la famille des tores d-dimensionnel minimaux.

III.2.6 Élection avec connaissance structurelle

Comme indiqué auparavant, l'algorithme de Mazurkiewicz permet d'élire dans tout graphe minimal *si l'on connaît sa taille*. En effet, on peut utiliser cet algorithme et l'information concernant la taille comme *paramètre* : $\mathcal{M}$. Ou encore, si le graphe a connaissance du fait qu'il est un arbre, il peut utiliser le système d'élection d'arbre. Ceci nous amène à étudier quelles connaissances structurelles permettent d'élire dans la famille des graphes minimaux.

Définition (voir I.1.3) : Soit $k : \mathcal{G}_{min} \rightarrow L$ une information structurelle. Un système de réétiquetage de graphes $\mathcal{R}$ élit avec connaissance structurelle k si pour tout graphe minimal $\boldsymbol{G}$, $\mathcal{R}$ élit sur (G, $\Lambda_{k(\boldsymbol{G})}$)

Revenons sur [15], l'énoncé complet est en fait :

Théorème de Mazurkiewicz : $\mathcal{R}$ élit avec connaissance de la taille.

Le théorème suivant donne la caractérisation des connaissances structurelles permettant d'élire avec un algorithme "paramétré" par cette connaissance.

Théorème : Soit $k : \mathcal{G}_{min} \rightarrow L$ une information structurelle. Il existe un système de réétiquetage de graphes $\mathcal{R}$ élisant avec connaissance structurelle k si et seulement si il existe une application récursive $r : \mathcal{G}_{min} \rightarrow L$ telle que pour tout graphe G $\in \mathcal{G}_{min}$, il n'existe dans $k^{-1}(k(G))$ aucun quasi-revêtement de G de rayon r(G), excepté $\boldsymbol{G}$ lui-même.

III.3 Preuves d'impossibilité de résolution de problèmes distribués de consensus

Dans cet exemple, nous présenterons quelques preuves faciles montrant l'impossibilité de résoudre plusieurs problèmes de consensus, en particulier dans les graphes de communication. Nous prouvons des résultats d'impossibilité pour l'accord Byzantin et l'accord faible. Les bornes sont toutes les mêmes: tolérer *m* défauts nécessite au moins *3m+1* nœuds, et nécessite au moins *2m+1* connectivités dans le graphe de communication. (La connectivité d'un graphe est le nombre minimum de nœuds dont le retrait déconnecte le graphe).

III.3.1 Un modèle des systèmes distribués

Afin de rendre les résultats d'impossibilité : claires, concis et généraux, nous présenterons un modèle très simple des systèmes distribués.

Un graphe de communication est un graphe orienté ***G*** avec un ensemble de nœuds *nœuds(G)* et ensemble d'arêtes *Arêtes(G)*. Nous appelons l'arête *(u, v)* une arête sortante de *u*, et une arête entrante vers *v*. Soit ***U*** un sous ensemble de *nœuds(G)*, le sous-graphe U_G induit par *U* est le graphe contenant tous les nœuds de *U* et toutes les arêtes entre les nœuds dans *U*. La frontière des arêtes entrantes U_G est l'ensemble des arêtes de nœuds en dehors de *U* vers *U*, G_U = *arêtes(G)* ∩ *((nœuds(G)\U) × U)*

Un système $\mathcal{G}$ est un graphe de communication ***G*** avec un assignement d'un *dispositif* et une *entrée* pour chaque sommet de ***G***. Ces dispositifs ne sont pas définis comme objets primitifs. Les entrées spécifiques que nous considérons sont des encodages booléens, des nombres réels ou fonctions à valeurs réelles de temps (par exemple, les horloges locales). Le type particulier de l'entrée dépend du problème d'accord adressé. Si un sommet est affecté au dispositif ***A*** dans le système $\mathcal{G}$, nous disons que le sommet *exécute* ***A***. Un sous-système $\mathcal{U}$ de $\mathcal{G}$ est un n'importe quel sous-graphe G_U de G avec les dispositifs associés et les entrées.

Chaque système $\mathcal{G}$ a un comportement du système, ***δ***, qui est un tuple contenant le comportement de chaque sommet et chaque arête de ***G***. (Nous décrirons également ***δ*** comme un comportement du graphe de communication ***G***. Notons qu'un système a exactement un seul comportement, tandis qu'un graphe peut avoir plusieurs, selon les dispositifs et les entrée affectées aux nœuds). La restriction du comportement d'un système ***δ*** vers les comportements des nœuds et des arêtes d'un sous-graphe de $\boldsymbol{G_U}$ de ***G*** est le scénario $\boldsymbol{\delta_U}$ de $\boldsymbol{G_U}$ dans ***δ***.

Pour l'instant, nous allons prendre les comportements des nœuds et des arêtes comme primitifs. En plus de des modèles concrets et familiers, un comportement d'un sommet ou d'une arête pourrait être une suite finie ou infinie d'états, ou une cartographie des réels positifs vers une certain ensemble de d'états, on désigne un état comme une fonction du temps. Les modèles les moins familières pourraient interpréter les comportements comme des cartographies des réels vers les états, ou depuis des transfinis ordinaux vers des états. Pour obtenir nos premiers résultats, l'interprétation précise de comportements de nœuds et des arêtes est sans importance, il suffit de restreindre notre modèle de sorte que les deux axiomes suivants détiennent.

III.3.1.1 Axiome de la localité

Soient $\mathcal{G}$ et $\mathcal{G}'$ des systèmes avec des comportements $\boldsymbol{\delta}$ et $\boldsymbol{\delta}'$ respectivement, et des sous-systèmes isomorphes $\mathcal{U}$ et $\mathcal{U}'$, (avec des ensemble vertex $\boldsymbol{U}$ et $\boldsymbol{U}'$). Si les comportements correspondants aux arêtes entrantes limites de $\boldsymbol{U}$ et $\boldsymbol{U}'$ dans $\boldsymbol{\delta}$ et $\boldsymbol{\delta}'$ sont identiques, et les scénarios $\boldsymbol{\delta}_U$ et $\boldsymbol{\delta}_U'$, sont identiques

Au fond, l'axiome de localité affirme que la communication ne se déroule que sur les arêtes du graphe de communication. En particulier, il exprime la propriété suivante: Les seuls paramètres qui affectent le comportement de n'importe quelle partie locale d'un système sont les dispositifs et les entrées à chaque sommet local, ainsi que toute information entrante sur les arêtes depuis le reste du système. Si ces paramètres sont les mêmes dans les deux comportements, les comportements locaux (scénarios) sont les mêmes. De toute évidence, une telle propriété de localité doit détenir, ou l'accord est trivialement réalisable en ayant des dispositifs lisant directement des entrées d'autres dispositifs.

III.3.1.2 Axiome de défaillance

Soit **A** un dispositif quelconque. Soient $\boldsymbol{E_i,\ldots, E_d}$ $\boldsymbol{d}$ comportements d'arêtes, tels que chaque $\boldsymbol{E_i}$ est le comportement de la i-ème arête sortante, dans un certain comportement de système $\boldsymbol{\delta}^i$, d'un sommet exécutant **A**. Soit *u* un sommet avec $\boldsymbol{d}$ arêtes sortantes $(u, v_1),\ldots, (u, v_d)$. Il existe un dispositif $\boldsymbol{F}$ tel que dans tout système dans lequel *u* exécute $\boldsymbol{F}$, le comportement de chaque arête sortante (u, v_i) est $\boldsymbol{E_i}$.

Dans ce cas, nous allons écrire $\boldsymbol{F_A(E_1,\ldots, E_d)}$ pour $\boldsymbol{F}$. Cet axiome exprime une capacité puissante des dispositifs défaillant. Tout comportement manifesté par un dispositif sur

différentes arêtes dans différents comportements de systèmes peut être manifesté par un dispositif défaillant dans un comportement d'un système unique. Lorsque cet axiome est considérablement affaibli (par exemple, en ajoutant une hypothèse de signature infalsifiable), les résultats d'impossibilité ne tiennent pas [18].

Afin d'établir la pertinence des résultats d'impossibilité pour plus de modèles concrets des systèmes distribués, il suffit d'interpréter les définitions dans le modèle particulier et ensuite de prouver les axiomes de localité et de défaillance.

Nos preuves utilisent la notion de théorie des graphes : ***revêtement***. Pour tout graphe ***G***, soient *voisins = {(u, V) | u est un* sommet *de* ***G*** *et V est l'ensemble de tous les nœuds v tels qu'il existe une arête de v vers u dans G}*. Un graphe ***S*** couvre ***G*** s'il y a une cartographie des nœuds de ***S*** vers les nœuds de ***G*** qui préserve "voisin". Sous une telle cartographie, ***S*** se comporte localement comme ***G***.

Les revêtements de graphes jouent un rôle important dans la compréhension de l'interaction de la topologie du réseau et le calcul distribué.

III.3.2 Accord Byzantin

Définition: Nous dirons que l'accord Byzantin est possible dans un graphe ***G*** (avec *n* nœuds) s'il existe *n* dispositifs A_1,…, A_n (que nous appellerons *les dispositifs d'accord*), avec les propriétés suivantes :

Chaque dispositif d'accord A_u prend une entrée booléenne, et choisit **'1'** ou **'0'** comme résultat. (Pour modéliser le choix d'un résultat, supposons qu'il existe une fonction **CHOISIR** depuis les comportements des nœuds exécutant des dispositifs accord vers l'ensemble ***{0,1}***). Un sommet *u* de ***G*** est correct dans un comportement ***δ*** de ***G*** si le sommet *u* exécute A_u dans ***δ***. Tout comportement d'un système ***δ*** de ***G*** dans lequel au moins *n - m* nœuds sont corrects est un comportement correct du système. Les comportements corrects d'un système doivent satisfaire les conditions suivantes :

Accord : Chaque sommet correct choisit la même valeur

Validité : Si le sommet correct a la même entrée, cette entrée doit être la valeur choisie.

Théorème : *L'accord Byzantin n'est pas possible dans les graphes inadéquats.*

III.3.2.1 Le nombre de nœuds

Nous commençons avec la borne inférieure de *3m+1* pour le nombre de nœuds nécessaires pour un accord Byzantin. Considérons d'abord le cas où $|\boldsymbol{G}| = n = 3$ et $m = 1$. Supposons que le problème peut être résolu pour le graphe de communication ***G*** composé de trois nœuds entièrement connecté par des arêtes de la communication. Soient A,B et C les trois nœuds de *G* et supposons qu'ils exécutent le dispositif d'accord A, B et C respectivement. (Ici, et plus tard nous utiliserons souvent les mêmes noms pour les nœuds dans ***G*** et les dispositifs qu'ils exécutent)

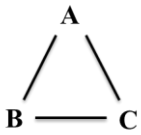


Figure 16 : Le graphe *G*

Le graphe de revêtement S se fera comme suit :

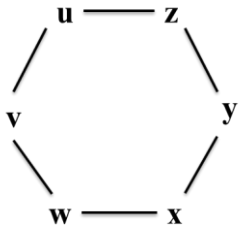


Figure 17 : Le revêtement de G: S

Ce graphe se comporte localement comme *G* (avec chaque sommet relié à deux autres par des bords de la communication).

Maintenant nous spécifions le système en attribuant des dispositifs et des entrées pour les nœuds de S comme suit:

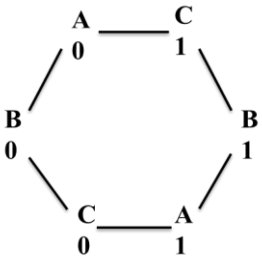


Figure 18 : Dispositifs et entrées de S

Nous entendons par là que le sommet *u* exécute l'appareil **A** avec l'entrée **0**, le sommet *v* exécute **B** avec l'entrée **0**, et ainsi de suite. Soit $\mathscr{S}$ le comportement résultant du système; $\mathscr{S}$ intègre un comportement pour chaque sommet et arête de **S**.

Considérons maintenant les scénarios $\mathscr{S}_{vw}$, $\mathscr{S}_{wx}$ et $\mathscr{S}_{xy}$ dans $\mathscr{S}$, où chacun comporte les comportements des deux nœuds indiqués dans **S**, avec l'activité au cours des deux arêtes de connexion. Nous ferons valoir que chacun de ces scénarios est identique à un scénario dans un comportement correct de ***G***.

- Le scénario $\mathscr{S}_{vw}$, premier scénario dans la chaîne:

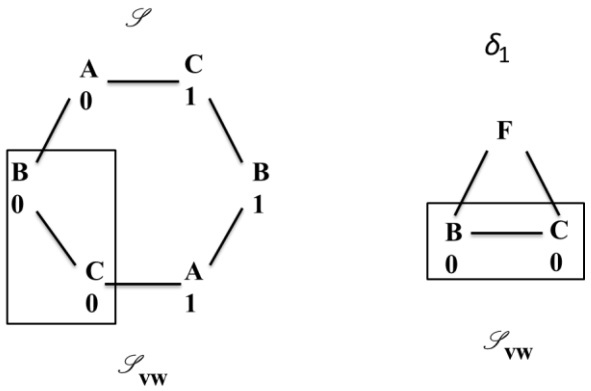


Figure 19 : Le 1er scénario $\mathscr{S}_{vw}$

Ce scénario est le comportement, dans $\mathscr{S}$, des nœuds *v* et *w*, ainsi que celui des arêtes de communication entre *v* et *w*. Considérons maintenant le de comportement $\boldsymbol{\delta_1}$ de ***G*** dans lequel B exécute B sur l'entrée 0, C exécute C sur l'entrée 0 et A exécute un dispositif qui imite *u* en communiquant avec B, et imite *x* en communiquant avec C. formellement, si $\mathbf{E}_{(u,v)}$ et $\mathbf{E}_{(x,w)}$ sont les comportements des arêtes indiquées dans $\mathscr{S}$, A exécute le dispositif $\mathbf{F}_A(\mathbf{E}_{(u,v)}, \mathbf{E}_{(x,w)})$ (nous avons écrit juste **F** dans la figure). Ce dispositif existe, par l'axiome de défaillance, et dans le comportement résultant, les arêtes de A vers B et C ont les comportements $\mathbf{E}_{(u,v)}$ et $\mathbf{E}_{(x,w)}$, respectivement. Par l'axiome de localité, le scénario contenant le comportement de B et C dans $\boldsymbol{\delta_1}$ est identique à $\mathscr{S}_{vw}$. Les conditions de validité assurent que B et C doivent choisir 0 dans $\boldsymbol{\delta_1}$. Puisque leur comportement est identique dans $\mathscr{S}$, *w* et *x* choisissent 0 dans le $\mathscr{S}$.

- Le scénario $\mathscr{S}_{wx}$, le deuxième scénario dans la chaîne :

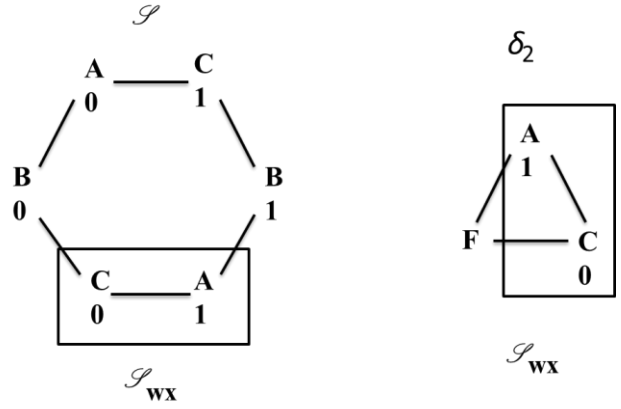


Figure 20 : Le 2ème scénario $\mathscr{S}_{wx}$

Ce scénario inclut le comportement de *w* et *x* dans $\mathscr{S}$. C'est également le comportement de A et C dans un comportement $\boldsymbol{\delta_2}$ de ***G*** qui se résulte quand les deux exécutent leurs dispositifs (sur les entrées 1 et 0, respectivement), et B est défaillant, exhibant le même comportement à C que *v* exhibe à *w* dans $\mathscr{S}$, et le comportement à A que *y* exhibe à *x* dans $\mathscr{S}$.

Le comportement de C dans $\boldsymbol{g_2}$ est identique à celui de *w* dans $\mathscr{S}$, ainsi C choisit 0 dans $\boldsymbol{\delta_2}$, de l'argument ci-dessus. Par accord, A décide 0 dans $\boldsymbol{\delta_2}$. Ainsi *x* décide 0 dans $\mathscr{S}$.

- Troisième scénario $\mathscr{S}_{xy}$:

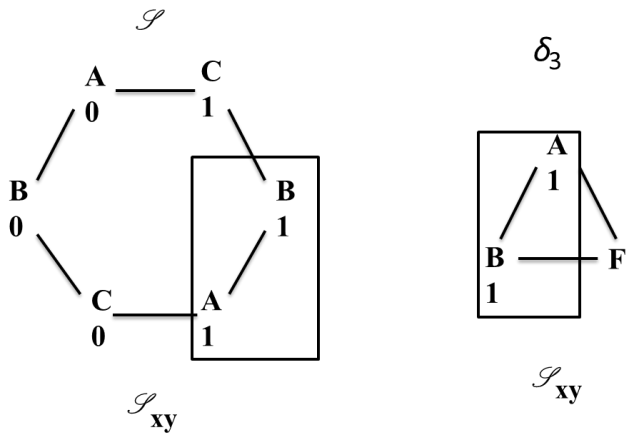


Figure 21 : Le 3ème scénario $\mathscr{S}_{xy}$

Ce scénario est le comportement de *x* et *y* dans $\mathscr{S}$. C'est également le comportement de A et B dans un comportement $\boldsymbol{\delta_3}$ de G qui se résulte quand les deux exécutent leurs dispositifs sur l'entrée 1, et C est défaillant, exhibant le même comportement à A que *w* exhibe à *x* dans $\mathscr{S}$, et le même comportement à B que *z* exhibe à *y* dans le $\mathscr{S}$. Les conditions de validité assurent que A et B doivent choisir 1. Ainsi *x* et *y* choisissent 1. Mais nous avons déjà établi que *x* doit choisir 0, une contradiction !

Considérons maintenant le cas général de $|G| = n \leq 3m$. Classant les nœuds de ***G*** dans trois groupes, A, B et C, chacun avec au moins 1 et au plus *m* nœuds. Ceci signifie que tous deux groupes quelconques contiennent ensemble au moins *n-m* nœuds. Les nœuds dans chaque groupe exécutent des dispositifs d'accord, et nous identifierons la collection de dispositifs s'exécutant au sein de chaque groupe avec le nom du groupe, comme avant. Soient le graphe de revêtement **S** et les dispositifs assignés exactement comme ci-dessus, où chaque sommet dans **S** représente un ensemble de nœuds dans G, avec leurs arêtes de connexion, et les arêtes entre deux nœuds de **S**, alors A et B, sont maintenant une représentation de sténographie pour toutes les arêtes dans ***G*** entre les nœuds dans A et les nœuds dans B. Les

entrées assignées aux symboles A, B et C sont maintenant assignés à tous les nœuds dans les groupes respectifs dans **S**. Les arguments procèdent exactement comme dans les images précédentes. Nous considérons seulement un en détail.

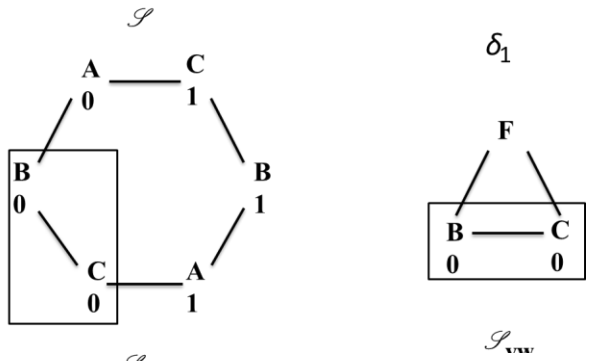


Figure 22 : Le 1er scénario $\mathscr{S}_{vw}$

Ce scénario est maintenant le comportement *des ensembles* de nœuds dans *v* et *w* dans le comportement $\mathscr{S}$. C'est le même comportement que celui des ensembles B et C dans le comportement $\boldsymbol{\delta_1}$ de ***G*** dans lequel tous les nœuds dans les deux ensembles exécutent leurs dispositifs avec l'entrée 0 et les nœuds dans A montrent le même comportement aux nœuds dans B que les nœuds correspondants dans *u* montrent aux membres de *v* dans $\mathscr{S}$, et le même comportement aux nœuds dans C que les nœuds correspondants dans *y* montrent aux membres de *x* dans $\mathscr{S}$. Puisque B et C contiennent ensemble au moins *n-m* nœuds corrects, $\boldsymbol{\delta_1}$ est un comportement correct de ***G***. Ainsi, tous les nœuds dans B et C doivent décider 0, par la condition de validité.

III.3.2.1 La connectivité

Maintenant nous donnons la preuve de limite inférieure de connectivité de *2m+1*. Soit *c(G) = connectivité de* ***G***. Nous supposerons que nous pouvons réaliser l'accord Byzantin dans un graph ***G*** avec $c(G) \leq 2m$, et tire une contradiction.

Pour l'instant, nous considérons le cas *m=1*, le graphe de communication ***G*** et les dispositifs indiqués ci-dessous.

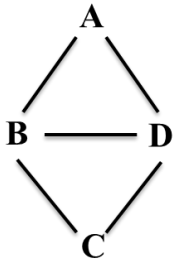


Figure 23 : Le graphe de communication G

La connectivité de ***G*** est 2; les 2 nœuds B et D séparent ***G*** en deux morceaux, les nœuds A et C.

Nous considérons le système suivant, avec le graphe à huit-nœuds **S**, des dispositifs et des entrées comme indiqué.

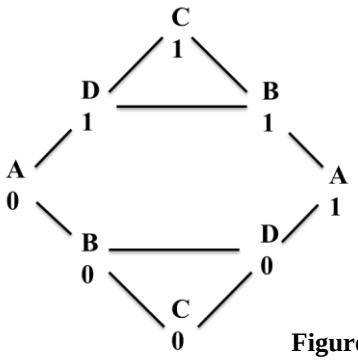


Figure 24 : Dispositifs et entrées de S

Le comportement résultant du système est $\mathscr{S}$. Nous considérons trois scénarios dans $\mathscr{S}$: $\mathscr{S}_1$, $\mathscr{S}_2$ et $\mathscr{S}_3$.

Le premier scénario, $\mathscr{S}_1$ est montré ci-dessous.

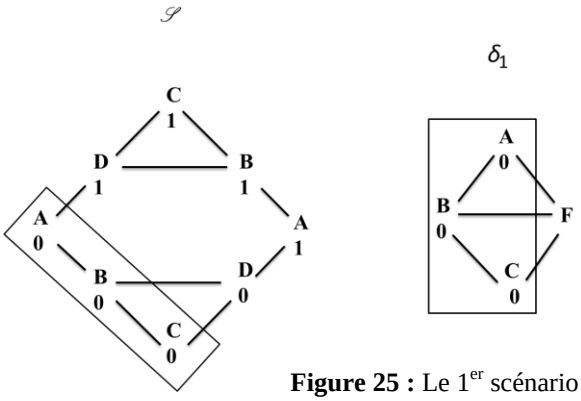


Figure 25 : Le 1er scénario

C'est également un scénario dans un comportement correct $\boldsymbol{\delta_1}$ de ***G***. Dans $\boldsymbol{\delta_1}$, A, B et C sont corrects. Le dispositif D est défaillant, exhibant le même comportement à A en tant que le premier D dans le graphe de revêtement, et le même comportement à B et à C que l'autre D exhibe dans le revêtement. Puis A, B et C doivent choisir 0 dans $\boldsymbol{\delta_1}$, comme c'est le cas pour A, B et C dans $\mathscr{S}_1$.

Le deuxième scénario, $\mathscr{S}_2$

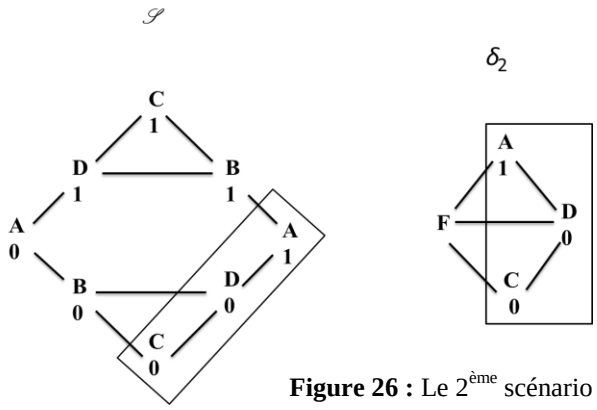


Figure 26 : Le 2ème scénario

Ce scénario dans $\mathscr{S}$ est également un scénario dans un comportement correct $\boldsymbol{\delta_2}$ de $\boldsymbol{G}$. Cette fois, B est défaillant. Le dispositif défaillant exhibe le même comportement à C et D en tant que le premier B dans le revêtement, et le même comportement à A que l'autre B. Ainsi A, C et D doivent convenir dans $\boldsymbol{\delta_2}$; et ainsi les nœuds correspondants dans $\mathscr{S}_2$. Puisque ce C choisit 0 de l'argument ci-dessus, le D et A dedans $\mathscr{S}_2$ choisissent 0, aussi.

Le dernier scénario, $\mathscr{S}_3$

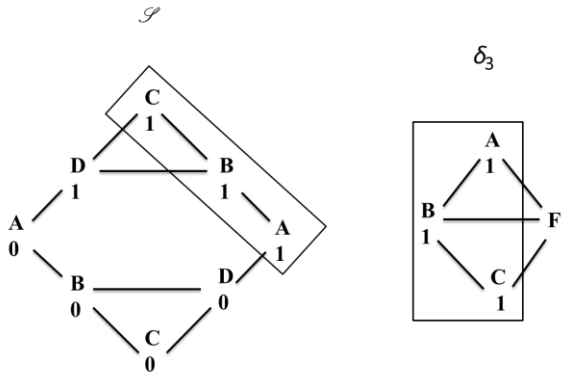


Figure 27 : Le 3ème scénario

Ce scénario est encore identique au scénario dans un comportement $\boldsymbol{\delta_3}$ de $\boldsymbol{G}$ dans lequel A, B et C sont non-défaillants, mais possède une entré 1. Le dispositif à D est défaillant, et exhibe le même comportement à A que le premier D dans le graphe de revêtement exhibe à A, et le même comportement à B et à C que l'autre D dans le revêtement. Puis A, B et C choisissent 1 dans $\boldsymbol{\delta_3}$, et ainsi doivent A, B et C dans $\mathscr{S}_3$, contredisant l'argument au-dessus qui dit que A choisit 0.

Le cas général pour des arbitraires $c(G) \leq 2m$ est une généralisation facile du cas $m=1$. Les images utilisées sont les mêmes. Il suffit de choisir B et D se composant d'au plus m nœuds chacun, telles que la suppression des nœuds dans B et D de $\boldsymbol{G}$ sépare $\boldsymbol{G}$ en deux

ensembles non vides A et C. Les arêtes de ***G*** représentent désormais toutes les arêtes possibles entre A, B, C et D.

Ceci complète la preuve du théorème posé auparavant.

Comme nous l'avons indiqué dans l'introduction, le théorème étudié a été précédemment connu, et la structure de notre preuve est très semblable à celle des preuves plus tôt [19], [20]. Notre preuve diffère dans la construction des comportements pathologiques $\boldsymbol{\delta_1}$, $\boldsymbol{\delta_2}$ et $\boldsymbol{\delta_3}$. Les premières preuves ont inclus un choix d'un modèle détaillé pour des dispositifs, et la construction inductive de ces comportements dans le modèle. Nous évitons cette construction par l'examen du comportement des dispositifs d'accord dans le graphe de revêtement. Les conditions de validité et d'accord n'imposent aucune restriction directe à ce comportement, car ils se réfèrent seulement aux comportements dans le graphe original. Cependant, les axiomes de localité et de défaillance imposent des restrictions indirectement au comportement dans le graphe de revêtement, car ils impliquent que les scénarios dans le graphe de revêtement sont également trouvés dans des comportements corrects du graphe original inadéquat.

Tandis que le modèle employé pour obtenir ces résultats est extrêmement général, mais il suppose que les systèmes se comportent déterministe. (Pour chaque ensemble d'entrées, un système a un seul comportement). Cette prétention de simplification a été faite pour garder l'exposition aussi claire que possible. En considérant un système et des entrées en tant que détermination d'un ensemble de comportements, le non déterministe et la probabilité peuvent être présentés d'une manière directe. Avec les altérations appropriées aux axiomes de localité et de défaillance, *les mêmes preuves suffisent pour montrer que les algorithmes non déterministes ne peuvent pas garantir* ***l'accord Byzantin.***

III.3.3 Accord faible

Maintenant nous donnons les résultats d'impossibilité pour le problème d'accord faible. Comme dans le cas de l'accord Byzantin, les nœuds ont des entrées booléennes, et doivent choisir une sortie booléenne. La condition d'accord est la même que pour l'accord Byzantin: tous les nœuds corrects doivent choisir la même sortie. Cependant, la condition de validité est plus faible.

Accord : Chaque sommet correct choisit la même valeur.

Validité : Si tous les nœuds sont corrects et ont la même entrée, cette entrée doit être la valeur choisie.

L'état plus faible validité a un impact intéressant sur problème d'accord. Si n'importe quel sommet correct observe un désaccord ou un comportement défaillant, ils sont tous libres de choisir une valeur par défaut, à condition qu'ils soient toujours en accord.

Lamport *[19]* note qu'il y a des dispositifs pour atteindre une forme de consensus faible et approximatif, qui s'exécutent quand $|G| \leq 3m$. Exécutant ceux-ci pendant un temps infini, produit le consensus exact (à la limite) [21]. Dans de tels comportements infinis, si n'importe quel sommet correct observe un désaccord ou un comportement défaillant, il a un temps abondant pour informer les autres avant qu'elles choisissent une valeur. Ainsi, le renforcement de la condition de choix, pour interdire telles solutions infinies, est nécessaire pour obtenir la limite inférieure

Nous devons également bondir des délais de communication à partir de zéro, ou un type semblable de comportement infini est possible. En fait, si nous supposons qu'il n'y a aucune limite inférieure dans le délai de transmission, et que les dispositifs peuvent commander le retard et ont des horloges synchronisées, on a trouvé un algorithme pour atteinte le consensus faible.

Cet algorithme exige tout au plus deux émissions par nœud, le tout avec un délai de transmission différent de zéro, et fonctionne avec n'importe quel nombre de défauts. Encore, c'est parce que n'importe quel sommet correct qui observe un désaccord ou un comportement défaillant a temps abondant pour informer les autres avant qu'elles choisissent une valeur. Comme nous l'observerons, dans plus de modèles réalistes, il est impossible d'atteindre un consensus faible dans des graphes inadéquats. Pour montrer ceci, la sémantique minimale présentée dans les sections précédentes doit être étendue pour exclure ces solutions d'infinité. Nous faisons ceci comme suit :

Précédemment, les comportements des nœuds et les arêtes étaient des éléments d'un certain ensemble arbitraire. Dorénavant, nous les considérerons comme étant des fonctions de temps (de $[0, \infty]$) aux ensembles arbitraires d'état. Ainsi, si **E** est un comportement dans le sommet *u*, alors *u* est dans l'état *E(t)* à un temps *t*.

Nous ajoutons la condition suivante au problème faible d'accord.

Choix : Un sommet correct doit choisir 0 ou 1 après un temps fini.

Ceci signifie qu'il existe une fonction CHOISIR des comportements des nœuds exécutant les dispositifs faibles d'accord à *{0.1}*, avec la propriété suivante:

Chaque comportement E a un préfixe fini $\boldsymbol{E_t}$ *(**E** réduit à l'intervalle [0, t]) tel que tous les comportements* $\boldsymbol{E'}$ *élargissant* $\boldsymbol{E_t}$ *ont CHOISIR(E) = CHOISIR(E')*

Cette condition de choix interdit la solution infinie de ***Lamport.*** Pour interdire la deuxième solution, nous bondissent le taux auquel l'information peut traverser le réseau. Pour faire ainsi, nous remplaçons l'axiome de localité par le suivant :

III.3.3.1 Axiome de localité pour un délai borné

Il existe une constante positive $\boldsymbol{\delta}$ tels que ce que suit est vrai : soient $\mathcal{G}$ et $\mathcal{G}'$ des systèmes avec des comportements $\boldsymbol{g}$ et $\boldsymbol{g}'$, respectivement, et sous-systèmes $\mathcal{U}$ et $\mathcal{U}'$, (avec le les ensembles vertex *U* et *U'*). Si les comportements correspondants aux arêtes entrantes de limites de U et U' dans $\boldsymbol{g}$ et $\boldsymbol{g}'$ sont identiques au moment *t*, puis les scénarios $\boldsymbol{g_U}$ et $\boldsymbol{g_{U'}}$ sont identique à $\boldsymbol{t + \delta}$.

Théorème: *L'accord faible n'est pas possible dans les graphes inadéquats.*

Encore, nous esquisserons d'abord les *3m+1* nœuds limites. Dans ce cas-ci, la preuve précédemment éditée [21] était très difficile. Comme avant, nous limitons notre attention au cas $|G| = n = 3, m = 1$. (Le cas général pour tout *m* suit juste comme avant)

Supposons qu'il y a des dispositifs faibles d'accord pour le graph triangulaire ***G*** contenant les nœuds A, B et C. Considérons les deux comportements de ***G*** dans lesquels tous les nœuds sont corrects, et tous ont l'entrée 0 ou tous ont l'entrée 1. Soit *t'*, une borne supérieure dans le temps où elle prend tous les nœuds pour choisir 0 ou 1 dans les deux comportements. Choisissons $\boldsymbol{k \geq t'/\delta}$ comme un multiple de 3.

Le graphe de revêtement **S** se compose de ***4k*** nœuds, disposés sur un cercle, dispositifs assignés et les entrées comme suit :

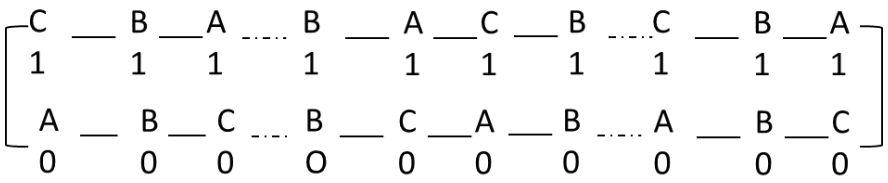


Figure 28 : Le graphe de revêtement S

Considérons le comportement résultant $\mathscr{S}$, et chaque scénario successif de deux-nœuds, tel que les deux ci-dessous.

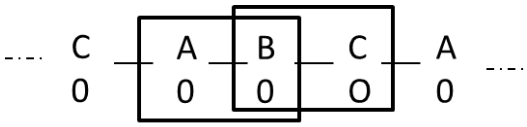


Figure 29 : Scénario de deux nœuds

Ce scénario est identique à un scénario d'un comportement dans ***G*** de deux dispositifs de consensus faibles et appropriés. Depuis chaque paire de scénarios successifs se recouvre dans un seul comportement de sommet (ici B), tous les nœuds dans les deux scénarios doivent choisir la même valeur dans ***G*** et dans ***S***. Par induction, chaque sommet dans S doit choisir la même valeur. Sans perte de généralité, supposez qu'ils choisissent 1.

Considérons les ***k*** scénarios indiqués ci-dessous.

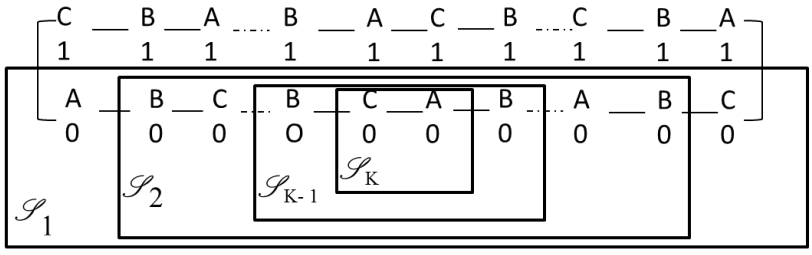


Figure 30 : *k* scénarios

Soit ***g*** le comportement de ***G*** dans lequel A, B et C sont corrects et chacun a l'entrée 0, et dénote les comportements résultant de A, B et C par $\boldsymbol{E}_A$, $\boldsymbol{E}_B$ et $\boldsymbol{E}_C$, respectivement.

Lemme: Le comportement dans le scénario $\mathscr{S}_i$ d'un sommet exécutant le dispositif A (ou B ou C) est identique à $\boldsymbol{E}_A$ (or $\boldsymbol{E}_B$ ou $\boldsymbol{E}_C$) à un temps $i\boldsymbol{\delta}$

D'après ce lemme, les nœuds exécutant les dispositifs C et A dans le scénario $\mathscr{S}_k$ ont des comportements identiques à $\boldsymbol{E}_C$ à $\boldsymbol{E}_A$ à un temps $k\boldsymbol{\delta}$**.** Puisque les dispositifs C et A dans G

ont choisi la sortie 0 en ce temps-ci, donc ils ont les dispositifs correspondants dans $\mathscr{A}_k$, une contradiction !

Le cas général $|G| \leq 3m$ et la limite de connectivité suivent le cas général présenté dans l'accord Byzantin.

Il y a des fortes similitudes entre cet argument et une preuve par **Angluin**, concernant l'élection dans les anneaux et les longues files arbitraires de processeurs [22]. Les deux résultats dépendent crucialement de l'existence d'une limite inférieure sur le taux de circulation de l'information. Dans cette prétention, des dispositifs dans différents réseaux de communication peuvent être montrés pour voir le même comportement local pour un certain temps fixé.

IV. Application du résultat d'impossibilité pour un réseau Ad Hoc en présence de défaillances Byzantines

Dans ce chapitre, nous traitons le problème de découverte de topologie dans les réseaux ad hoc en présence de défaillances byzantines. La difficulté majeure présente dans ce type de réseaux est le fait que le voisinage des nœuds change aléatoirement d'une manière imprévisible, souvent, en cause, les défaillances par arrêt ou les défaillances byzantines.

Pour cela, nous allons traiter en un premier lieu deux algorithmes de découverte de la topologie pour les réseaux ad hoc. Le premier est un algorithme de découverte de la topologie forte en présence de défaillances Crash-Reprise et le deuxième est un algorithme de découverte de la topologie faible en présence de défaillances byzantines à savoir : modification, non envoi ou non retransmission des messages et l'envoi des messages différents à des nœuds différents. En un second lieu, nous donnerons une application du problème d'accord byzantin présenté dans « III.3.2 » pour les réseaux ad hoc, à savoir l'impossibilité de distinguer un changement de topologie d'une défaillance byzantine.

IV.1 Modèle du système

Dans un réseau ad hoc, composé de n nœuds, le seul moyen de communication entre les nœuds est l'échange des messages via des canaux de communication. Les canaux de communications sont fiables, c'est à dire aucun canal peut créer, perdre ou altérer un message. Notons qu'un canal fiable peut être implémenté au-dessus d'un canal avec des pertes de message. Nous supposons que les communications sont synchrones, c'est-à-dire, il existe des bornes connues sur les délais de transmission des messages, les vitesses relatives des processeurs et les dérives des horloges.

Un sommet P_i peut être crashé par arrêt définitif, peut être crashé puis reprendre son exécution et peut être Byzantin. Un sommet byzantin peut envoyer des messages qui ne doivent pas être envoyés ou peut ne pas envoyer des messages qui doivent être envoyés, il peut aussi modifier des messages qui ne doivent pas être modifiés, ou même envoyer des messages différents à des nœuds différents. Nous supposons que la connectivité des nœuds est supérieure ou égale à $k + 1$ (k est le nombre maximum des nœuds Byzantins).

IV.2 Algorithme de découverte de la topologie forte (STDP) avec des défaillances «Crash-Reprise »

Dans cette section, nous abordons un algorithme résolvant le problème de découverte de la topologie forte en présence des défaillances Crash-Reprise. Ce type de défaillances est un cas particulier des défaillances byzantines.

Contrairement au résultat de [23] de la connectivité nécessaire pour la découverte de topologie forte, si un sommet possède seulement $k + 1$ connexions alors, ce même problème peut être résolu, mais avec un type de défaillances moins fort, qui est le Crash-Reprise. Nous prouvons également que cette connectivité est minimale pour la découverte de topologie forte dans ce contexte. C'est à dire, il est impossible de faire la même chose si nous avons même un seul sommet avec moins de $k + 1$ connexions.

Dans l'algorithme présenté, chaque sommet du réseau manipule les variables locales du tableau suivant :

Variable	Description
TOP_i	Topologie actuelle du sommet P_i
Detect	Booléen qui indique la présence d'une défaillance
$CONNECT_i$	Ensemble de nœuds avec lesquels P_i est connecté
Vi	Voisins du sommet P_i

Tableau: Variables locales d'un sommet P_i

Algorithme: Chaque sommet P_i exécute l'algorithme suivant :

Initialisation :

Detect ← *False* ; $TOP_i \leftarrow \{(P_i, V_i)\}$; $CONNECTi \leftarrow \emptyset$;

Tâche :

Envoyer *DECOUVRIR*(P_i , V_i) à tous les nœuds dans V_i ;

ECHO((P_i ,V_i), V_i) ;

Attendre la réception de *DECOUVRIR*(P_r , V_r), message de P_j

$TOP_i \leftarrow TOP_i \cup \{(P_r, P_r)\}$;

Envoyer *DECOUVRIR*(P_r , V_r) à tous les nœuds dans V_i sauf P_j;

ECHO((P_r ,V_r),$V_i - \{P_j\}$;

Soit $CONNECT_i = \{(P_x, V_x) / (P_x, V_x) \in TOP_i$ et $P_x \in V_i\}$.

Si (|*CONNECTi*| < *k+1*) **Alors**

Detect ← *True*;

Fin Si.

Procédure *ECHO*(P_r ,V_r),*Q*

Attendre la réception de *DECOUVRIR*(P_r ,V_r) du sommet $P_k \in Q$;

Si (*! DECOUVRIR*(P_r ,V_r) d'aucun sommet dans *Q*) **Alors**

Detect ← *True* ;

Fin Si

Fin Procédure

Pour la découverte de la topologie du réseau, chaque sommet exécute une copie de l'algorithme. Après l'initialisation des variables locales, le sommet P_i envoie un message *DECOUVRIR*(P_i ,V_i) à tous ses voisins et il invoque la procédure *ECHO*((P_i ,V_i), V_i) pour vérifier la relayage de ce message par tous ses voisins. Puis le sommet P_i reste en écoute pour s'assurer que tous les nœuds corrects ont relayé le message DECOUVRIR ou pour détecter les nœuds crashés. Comme le système est synchrone et les canaux de communications sont fiables, chaque voisin non correct de P_i est détecté.

Lors de la réception d'un message *DECOUVRIR*(P_r ,V_r), à partir d'un sommet P_r, P_i met à jour sa topologie. Par la suite, il relaye ce message à tous ses voisins à l'exception de son émetteur (pour éviter la réception de ce message à partir du même émetteur une infinité de fois) et il invoque la procédure *ECHO*. A partir de son ensemble TOP_i, P_i calcule l'ensemble $CONNECT_i$. Ensuite, il vérifie s'il est connecté. Si la cardinalité de $CONNECT_i$

est supérieure à *k+1*, alors P_i ne détecte aucune défaillance, c'est à dire, il a reçu des messages DECOUVRIR à partir de tous ses voisins. Dans le cas contraire, il détecte une défaillance, c'est à dire, il existe au moins un de ses voisin qui est crashé.

Cet algorithme est alors une solution au problème de la découverte de la topologie forte.

La complexité de cet algorithme est $\mathcal{O}(\delta n^3)$, tel que ***n*** indique le nombre de nœuds dans le réseau et ***δ*** représente le voisinage maximum.

Propriétés :

1. **Terminaison** : Tous les nœuds corrects du système déterminent la topologie du réseau.

2. Sûreté : Pour chaque sommet correct, la topologie déterminée est un sous ensemble de la topologie actuelle du réseau.

IV.3 Algorithme de découverte de la topologie faible (WTDP) avec des défaillances byzantines

Dans cette section, nous abordons un algorithme résolvant le problème de découverte de la topologie faible en présence des défaillances byzantines.

Les nœuds byzantins, peuvent modifier ou ne pas relayer des messages de type DECOUVRIR. Leur rôle est d'empêcher les nœuds corrects à découvrir la topologie du réseau avec des informations correctes. Contrairement à l'algorithme proposé dans [23] dont la détection de défaillances est faite juste par les nœuds destinataires, l'algorithme présenté ici permet à tous les nœuds (source et destination) de détecter des défaillances si elles existent.

Algorithme: Chaque sommet P_i exécute l'algorithme suivant :

Initialisation :

Detect ← *False* ; $TOP_i \leftarrow \{(P_i, V_i)\}$; $CONNECTi \leftarrow \emptyset$;

Tâche :

Envoyer *DECOUVRIR*(P_i, V_i) à tous les nœuds dans V_i ;

ECHO$((P_i, V_i), V_i)$;

Attendre la réception de *DECOUVRIR*(P_r, V_r), message de P_j

Si $(\exists (P_k, V_k) \in TOP_i : (P_k = P_r) \wedge (V_k \neq V_r)$ **Alors**

Detect ← *True* ;

Sinon

$TOP_i \leftarrow TOP_i \cup \{(P_r, P_r)\}$;

Fin Si

Envoyer *DECOUVRIR*(P_r, V_r) à tous les nœuds dans V_i sauf P_j;

ECHO$((P_r, V_r), V_i - \{P_j\}$;

Soit $CONNECT_i = \{(P_x, V_x) / (P_x, V_x) \in TOP_i$ et $P_x \in V_i\}$.

Si $(|CONNECTi| < k+1)$ **Alors**

Detect ← *True*;

Fin Si.

Procédure *ECHO*$(P_r, V_r), Q$

Attendre la réception de *DECOUVRIR*(P_r, V_r) à partir du sommet $P_k \in Q$;

Si (*!* *DECOUVRIR*(P_r, V_r) d'aucun sommet dans Q) **Alors**

Detect ← *True* ;

Fin Si

Fin Procédure

Pour la découverte de la topologie du réseau, chaque sommet exécute une copie de cet algorithme. Tout comme l'algorithme précédent, après l'initialisation des variables locales, le sommet P_i envoie un message *DECOUVRIR*(P_i, V_i) à tous ses voisins et il invoque la procédure *ECHO*$((P_i, V_i), V_i)$ pour vérifier si tous ces voisins ont relayé ce même message. Puis, le sommet P_i reste en écoute pour s'assurer que tous les nœuds corrects ont relayé le même message *DECOUVRIR* ou pour détecter les nœuds byzantins qui n'ont pas relayé ou qui ont modifié le message. Chaque voisin byzantin de P_i sera détecté, car le système est synchrone et les canaux de communications sont fiables.

Lors de la réception d'un message *DECOUVRIR(P_r , V_r)*, à partir d'un sommet P_r, P_i vérifie si l'information reçue est la même que celle existante dans TOP_i. Si elle est différente, alors il détecte une défaillance. Dans le cas contraire il met à jour sa topologie, il relaye ce message à tous ses voisins à l'exception de son émetteur et il invoque la procédure *ECHO*. A partir de son ensemble TOP_i, P_i calcule l'ensemble $CONNECT_i$ pour vérifier s'il est connecté ou non. Si la cardinalité de $CONNECT_i$ est supérieure à $k+1$, alors P_i ne détecte aucune défaillance, dans le cas contraire, il détecte une défaillance, c'est à dire, il existe au moins un de ses voisin qui est byzantin.

C'est alors que cet algorithme est une solution au problème de la découverte de la topologie faible.

La complexité de cet algorithme est $\mathcal{O}(\delta n^3)$, tel que ***n*** indique le nombre de nœuds dans le réseau et ***δ*** représente le voisinage maximum.

Propriétés :

1. Terminaison : Tous les nœuds corrects du système déterminent la topologie du réseau ou au moins un sommet correct détecte une défaillance.

2. Sûreté : Pour chaque sommet correct, la topologie d´déterminée est un sous-ensemble de la topologie actuelle du réseau.

3. Validité : Une défaillance est détectée si et seulement si elle existe dans le système.

IV.4 Résultat d'impossibilité appliqué aux réseaux Ad Hoc

Nous allons maintenant utiliser le résultat d'impossibilité pour un accord byzantin, présenté dans III.3.2, pour montrer l'impossibilité de distinction entre la mobilité d'un sommet et un comportement byzantin dans les réseaux dans le cas de découverte de la topologie faible. En fait, l'ensemble de voisinage d'un sommet peut être modifié à cause de la mobilité ou à cause d'un comportement byzantin. On pourra alors se référer aux deux axiomes (axiomes de localité et axiome de défaillance) présentés dans III.3.1. En se basant sur ces deux axiomes, on a le théorème suivant :

Théorème : *Dans un réseau mobile ad hoc où chaque sommet possède une connectivité de (k+1) il est impossible de distinguer entre la mobilité d'un sommet et un comportement byzantin.*

Nous allons prouver ce théorème en illustrant un exemple. Nous supposons que le message de découverte de la topologie avant la mobilité est $m = 0$ et après la mobilité est $m_0 = 1$.

On étudie d'abord le cas $k = 1$.

Soit G un graphe de communication formé de quatre nœuds *a, b, c* et *d* exécutant les dispositifs A, B, C et D comme indiqué sur la figure:

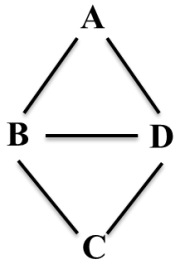


Figure 31: Le graphe de communication ***G***

La connectivité de G est égale à 2. Les deux nœuds *b* et *d* décomposent G en deux morceaux, le sommet *a* et le sommet *c*. On considère un graphe S composé de huit nœuds exécutants huit dispositifs avec des entrées différentes comme indiqué sur la figure 32. S est un revêtement de graphe G qui illustre les différents scénarios possibles. Le comportement résultat du système est $\mathscr{S}$.

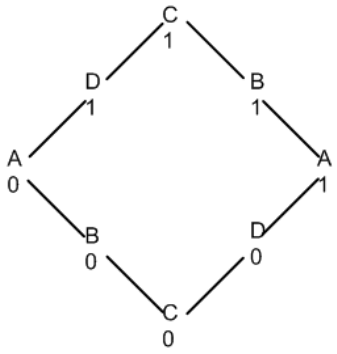


Figure 32: Le graphe S

On considère alors les trois scénarios dans $\mathscr{S}$: $\mathscr{S}_1$, $\mathscr{S}_2$ et $\mathscr{S}_3$.

Premier scénario : Le premier scénario, $\mathscr{S}_1$, est illustré sur la figure 33:

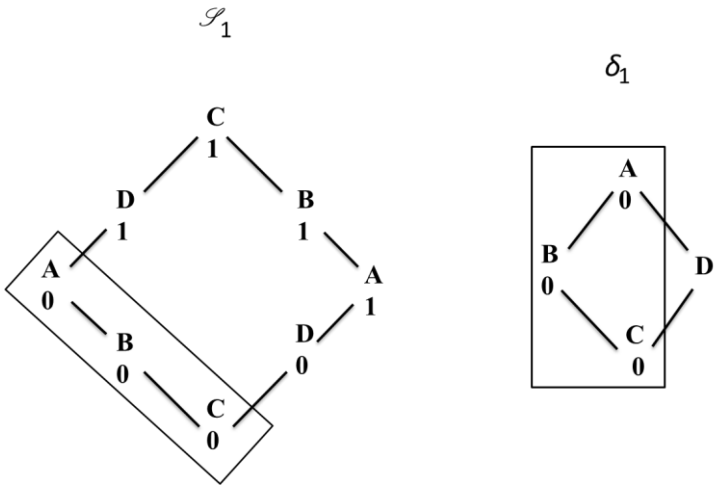


Figure 33: Le premier scénario $\mathscr{S}_1$

$\mathscr{S}_1$ est un scénario d'un comportement correct, δ_1, de G avant la mobilité. Dans δ_1, les nœuds *a*, *b* et *c* sont corrects et le sommet *d* est byzantin. Par l'axiome de défaillance, le sommet *d* peut exhiber, au sommet *a*, le même comportement exhibé par un sommet exécutant *D* dans le graphe revêtement ; et exhiber au sommet *c*, le même comportement exhibé par un autre sommet exécutant *D* dans le graphe revêtement. Dans ce scénario, nous supposons que le sommet byzantin n'exécute aucune action, par conséquent et d'après l'axiome de localité, le sommet *a* accepte le message *m*. En effet, le sommet *c* envoie le message *m* et le sommet *a* reçoit une seule copie de ce message via le sommet correct *b*.

Deuxième scénario : Ce scénario, $\mathscr{S}_2$, est illustré sur la figure 34 :

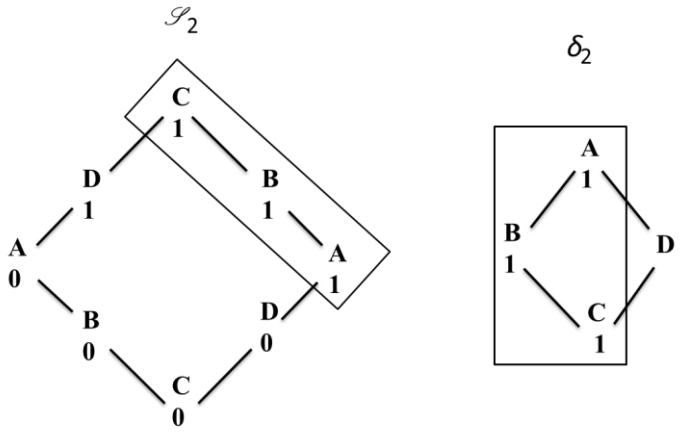


Figure 34: Le deuxième scénario $\mathscr{S}_2$

$\mathscr{S}_2$ est un scénario d'un comportement correct, δ_2, de *G* après la mobilité, où les nœuds *c*, *b* et a sont corrects et le sommet *d* est byzantin. Dans ce scénario, nous supposons aussi que le sommet byzantin n'exécute aucune action. Après la mobilité, le sommet *c* envoie le nouveau message *m'* qui porte le nouveau voisinage. D'après l'axiome de localité, le sommet *a* accepté directement ce message car il reçoit seulement une copie via le sommet correct *b*.

Troisième scénario : On considère le dernier scénario $\mathscr{S}_3$, illustré sur la figure 35 :

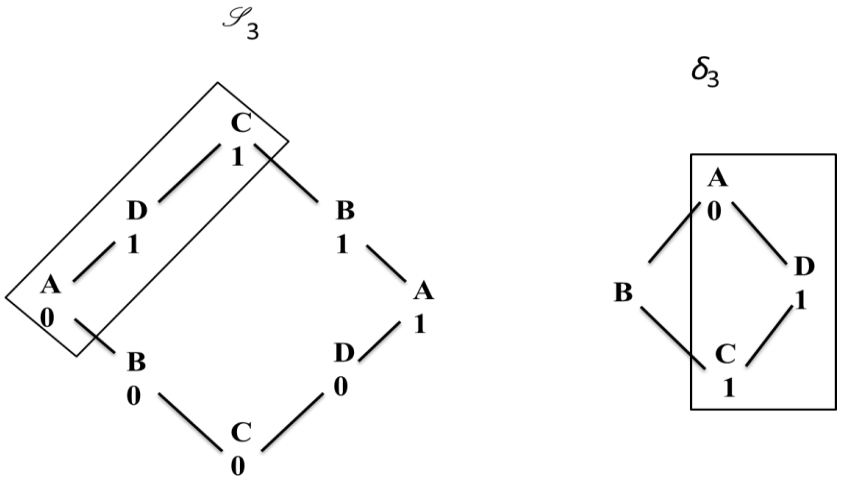


Figure 35: Le dernier scénario $\mathscr{S}_3$

Ce scénario est un scénario d'un comportement correct, δ_3, de G dont les sommets a, d et c sont corrects et le sommet b est byzantin. Le sommet b exhibe au sommet a, le même comportement exhibé par un sommet exécutant D dans le graphe revêtement et exhibe au sommet c, le même comportement exhibé par un autre sommet exécutant D dans le graphe revêtement. Après la mobilité, le sommet c envoie un message m'. Le sommet a reçoit deux copies de ce message ; une copie m', retransmise par le sommet correct d et une autre copie, m, retransmise par le sommet byzantin b. Pour le sommet a, ces deux copies ne lui permettent pas de distinguer entre la mobilité et un comportement byzantin. En effet, a ne peut pas confirmer que le sommet c a envoyé m' et non pas m, car il ne connait pas qui est le sommet byzantin, b ou d.

Le cas général pour une connectivité égale à $k+1$ est une simple généralisation de $k=1$. En effet, si $k > 1$, le sommet a reçoit $k + 1$ copies et s'il existe au moins une copie différentes de reste des copies, alors le sommet a ne peut toujours pas distinguer entre la mobilité et un comportement byzantin. Cela complète la preuve du théorème.

Dans ce chapitre, nous avons proposé deux algorithmes pour la découverte de la topologie dans les réseaux ad hoc. Le premier algorithme (STDP) est proposé pour tolérer les défaillances Crash-Reprise, et le deuxième (WTDP) est proposé pour détecter les défaillances byzantines. STDP permet de découvrir la topologie malgré la présence des nœuds crashés dans le réseau et la connectivité nécessaire pour cet algorithme est $k + 1$. WTDP permet aux différents nœuds de découvrir la topologie du réseau ou de détecter les défaillances que les nœuds peuvent subir, à savoir la modification des messages, la non retransmission et le non envoi des messages, et l'envoi des messages différents aux nœuds différents. La connectivité nécessaire pour cet algorithme est aussi $k + 1$. A la fin, nous avons exploité la preuve d'impossibilité pour l'accord byzantin pour prouver l'impossibilité de distinguer entre la mobilité d'un sommet et un comportement byzantin dans un réseau ad hoc si la connectivité de ce réseau est $k+1$, k est le nombre maximum des nœuds Byzantins dans le réseau.

Conclusion

Dans ce mémoire, nous avons traité le problème de réécriture de graphes, en particulier, le réétiquetage dans les algorithmes distribués. Et ceci en présence de défaillances. Nous avons basé notre étude sur une collection de résultats déjà connus, pour ensuite arriver à expliciter quelques exemples d'utilisation de revêtement de graphes dans l'algorithmique distribuée. Nous avons, à cette occasion, traité le problème de présence de défaillances dans les graphes en question en étudiant les cas d'impossibilité de résolution de problèmes distribués de consensus, et ceci à travers l'étude du problème d'accord Byzantin et d'accord faible.

Après avoir exposé le problème de l'élection dans les graphes et l'énumération à travers l'algorithme de Mazurkiewicz, nous nous sommes intéressés, plus en particulier, à l'étude de ce qui est impossible à réaliser dans le modèle de réécriture de graphes défaillants. De manière générale, ce qui est impossible en algorithmique distribuée l'est pour des raisons de similarités. La manipulation de ces similarités par les revêtements a permis de consolider les preuves et résultats d'impossibilité.

Dans l'idée d'application de ces résultats d'impossibilité, nous avons choisi les réseaux Ad Hoc, où on a souvent recourt à l'algorithmique distribuée ; c'est alors que nous avons exposé deux algorithmes de découverte de topologie à la fois forte et faible. Puis nous avons appliqué le résultat d'impossibilité, dans le cas d'accord Byzantin, pour ce type de réseau.

Comme perspectives futures à ce travail, il est possible d'envisager des simulations des algorithmes de découverte de topologie dans les réseaux Ad Hoc pour arriver à montrer, en pratique, les résultats d'impossibilité, et ceci dans les conditions de présence de défaillances. Aussi, nous pourrions envisager d'élargir l'application des résultats d'impossibilité à d'autres applications, en particulier, d'autres types de réseaux.

Bibliographie

[1] C. Berge. *"Graphe"s*. Gauthier Villars, 1983.

[2] A. Gibbons. *"Algorithmic graph theory"*.
Cambridge University Press, 1985.

[3] K.H. Rosen, editor. *"Handbook of discrete and combinatorial mathematics"*.
CRC Press, 2000.

[4] Emmanuel GODARD. *" Réécritures de graphes et algorithmique distribuée"*.
Thèse doctorale, Université Bordeaux I, 2002.

[5] Paolo Boldi and Sebastiano Vigna. *"Fibrations of graphs"*.
Discrete Math, 243, 2002.

[6] E. Anceaume and E. Mourgaya. *Unreliable distributed timing scrutinizer : Adapting asynchronous algorithms to the environment. In Proceedings of the 5th IEEE International Symposium on Object-oriented Real–time distributed Computing (ISORC 2002)*, Pages 70-77, Washington, DC, USA, April 2002. IEEE Computer Society.

[7] S. Djoulane et N. Zaidi. *"Le renommage dans les systèmes distribués"*.
Université Abderrahmane Mira de Béjaia, 2009.

[8] Salim. *"La duplication des données dans un système distribué"*.
Université Abderrahmane Mira- Béjaia, 2007.

[9] A. Mazurkiewicz. Trace theory. In W. Brauer et al., editor,
"Petri nets, applications and relationship to other models of concurrency".
Volume 255 of Lecture notes in computer science, Pages 279–324. Spinger-Verlag, 1987.

[10] N. Lynch. *"A hundred impossibility proofs for distributed computing"*.
In 8th International conference on distributed computing systems, pages 1–28, 1989.

[11] W. S. Massey. *"A basic course in algebraic topology"*.
Springer-Verlag, 1991. Graduate texts in mathematics.

[12] D. Angluin. *"Local and global properties in networks of processors"*.
In Proceedings of the 12th Symposium on Theory of Computing, pages 82–93, 1980.

[13] Yves Métivier, Anca Muscholl, and Pierre-André Wacrenier.
"About the local detection of termination of local computations in graphs".
In D. Krizanc and P.Widmayer, editors, SIROCCO 97 - 4th International Colloquium on Structural Information & Communication Complexity, Proceedings in Informatics, Pages 188–200. Carleton Scientific, 1997.

[14] K. Reidemeister. *"Einführung in die Kombinatorische Topologie"*.
Vieweg, Brunswick, 1932.

[15] A. Mazurkiewicz. *"Distributed enumeration"*.
Inf. Processing Letters, 61 :233–239, 1997.

[16] A. Mazurkiewicz. *"Solvability of the asynchronous ranking problem"*.
Inf. Processing Letters, 28 :221–224, 1988.

[17] S. Gruner, Y.Métivier,M.Mosbah, and P.-A.Wacrenier.
"Distributed Algorithm for Computing a Spanning Tree in Anonymous T-prime Graphs".
In OPODIS'2001, International Conference On Principles of DIstributed Systems, Manzanillo, Mexico, 2001.

[18] M. Pease, R. Shostak, L. Lamport, *"Reaching Agreement in the Presence of Faults"*.
JACM 27:2 1980, 228-234.

[19] L. Lamport, R. Shostak, M. Pease, *"The Byzantine Gencrals Problem"*
ACM Trans. on Programming Lang. andSystems 4, 3 (July 1982), 382-401.

[20] *D. Dolev. "The Byzantine Generals Strike Again," Journal of Algorithms,*
3, 1982, pages. 14-30.

[21] L. Lamport. *"The Weak Byzantine Generals Problem"*.
JACM, 30, 1983, pp. 668-676.

[22] D. Propriétés Angluin , *"locaux et mondiaux Des réseaux de processeurs"*.
Proc. STOC de l'12ème, 30 avril au 2 mai 1980, Los Angeles, CA., Pp 82-93.

[23] M. Nesterenko ; S. Tixeuil.
"Discovering Network Topology in the Presence of Byzantine Faults".
IEEE Transactions on Parallel and Distributed Systems,
20(12) :1777– 1789, December 2009.

Résumé :

En algorithmique distribuée, chaque système peut être représenté par un graphe étiqueté, où les sommets correspondent aux différents terminaux, les arêtes aux liens de communication et les étiquettes associées aux sommets codent les états des processeurs. Dans un mécanisme de réétiquetage local, un algorithme distribué est décrit par un système de règles de transition locale où la nouvelle étiquette d'un sommet est fonction de son étiquette précédente et de celles de ses voisins. Dans le cadre de ce mémoire, nous étudions la réalisabilité et non-réalisabilité des tâches distribuées. Nous illustrons notre méthode en nous intéressant en particulier à certains problèmes spécifiques aux systèmes distribués (élection d'un nœud, énumération de graphes, problème de consensus et découverte de topologie dans les réseaux Ad Hoc). Dans tous ces cas, nous caractérisons ce que n'est pas réalisable par calcul distribué en fonction de la topologie du graphe sous-jacent et de la connaissance structurelle de ce graphe. Les différents cas d'impossibilité de calcul d'une manière distribuée sont dus aux « similarités » de familles de réseaux. Ces « similarités » sont décrites à l'aide de morphismes de graphes particuliers : les revêtements et les quasi-revêtements. Les preuves d'impossibilité emploient des techniques de simulation à base de ces morphismes.

Abstract:

In distributed algorithmic each system can be modeled by a labelled graph, where nodes correspond to different processors, the edges to communication links and labels associated with the nodes encode the states of processors. In a local relabeling mechanism, a distributed algorithm is described as a system of local transition rules, where the new label of a node is a function of its previous label and those of its neighbours. As part of this paper, we study the feasibility and infeasibility of distributed tasks. We illustrate our method by focusing in particular on specific problems of distributed systems (election of a node, enumeration of graphs, and the problem of consensus and topology discovery in Ad Hoc networks). In all these cases, we characterize what is not feasible for distributed computing based on the topology of the underlying graph and the structural knowledge of this graph. The impossibility cases of distributed computing usually due to the "similarities" of families of networks. These "similarities" are described with special morphisms of graphs: coverings and quasi- coverings. The impossibility proofs employ simulation techniques based on these morphisms.