Présentation et étude de quelques problèmes d'algorithmique distribuée

Thomas Morsellino

Numéro d'ordre : 4586 **École Doctorale de Mathémathiques et Informatique**

Présentation et Étude de Quelques Problèmes d'Algorithmique Distribuée

THÈSE

présentée et soutenue publiquement le 25 septembre 2012

pour l'obtention du

Doctorat de l'université Bordeaux I

(spécialité informatique)

par

Thomas MORSELLINO

Devant la commission d'examen composée de :

Président :	Ralf Klasing	Directeur de recherche CNRS
Rapporteurs :	Joffroy Beauquier	Professeur à l'université Paris-Sud
	Franck Petit	Professeur à l'université Pierre et Marie Curie
Examinateur :	Mohamed Mosbah	Professeur à l'Institut Polytechnique de Bordeaux
Directeurs :	Jérémie Chalopin	Chargé de recherche CNRS
	Yves Métivier	Professeur à l'Institut Polytechnique de Bordeaux

Laboratoire Bordelais de Recherche en Informatique — UMR 5800

Remerciements

Généralement, les remerciements effectués dans un manuscrit de thèse se ressemblent beaucoup, mais sont, néanmoins, un reflet de la qualité du travail réalisé.

Tout d'abord, Yves Métivier a su me présenter les bases nécessaires pour la résolution des problèmes traités dans ce manuscrit. Humainement, sa disponibilité (je suis débiteur de quelques cafés), son expérience, son intégrité (ainsi que sa sensibilité pour la cause doctorante) en font un directeur de thèse plus qu'exemplaire. D'un autre côté, Jérémie Chalopin est une personne patiente et dotée d'un (super) pouvoir lui permettant de traiter des preuves de propositions par téléphone. C'est une grande chance d'avoir pu travailler à leurs côtés et je tiens à leur exprimer ma gratitude la plus profonde.

Je tiens à remercier chaleureusement Joffroy Beauquier et Franck Petit qui ont accepté de relire mon manuscrit et qui ont fait le déplacement, malgré leur emploi du temps chargé, pour assister à ma soutenance. Sans oublier Ralf Klasing qui m'a fait l'honneur de présider la soutenance.

L'accomplissement de ces trois années de thèse ne peut pas se réduire qu'aux remerciements de mes deux directeurs. Je tiens donc à remercier Mohamed Mosbah qui m'a fait confiance et m'a intégré pleinement au sein du projet ViSiDiA. La conception et la réalisation des fonctionnalités décrites dans le chapitre 4 de ce manuscrit ont été réalisées grâce à sa disponibilité et en collaboration avec Cédric Aguerre sans qui le projet ne serait pas ce qu'il est aujourd'hui.

Au début de ma thèse, Yves m'a fait la confidence que l'enseignement est une notion importante dans l'équilibre d'un enseignant-chercheur. C'est pourquoi je tiens à remercier toutes les personnes que j'ai côtoyées dans le département informatique de l'IUT Bordeaux 1. Mon passage à l'IUT en tant qu'étudiant m'a donné l'envie d'enseigner au même endroit. Isabelle Dutour, Patrick Félix, Pierre Ramet, Colette Johnen, Arnaud Pêcher, Nicholas Journet m'ont fait confiance et ont su m'intégrer dans cette structure pédagogique à taille humaine tournée autour de la réussite des étudiants avant tout.

Toutefois l'équilibre n'aurait pas été le même sans la présence d'une équipe administrative efficace. Je pense en premier lieu à Cathy Roubineau qui m'a aidé dans tant de démarches, mais aussi à Brigitte Cudeville, Lebna Mizani, Auriane Dantes et Philippe Biais. Parallèlement, de par mes activités au laboratoire, j'ai pu échanger et discuter avec beaucoup de personnes comme Nicolas Hanusse ou encore Maylis Delest.

Parmi les membres non permanents du laboratoire, beaucoup d'entre eux ont participé de près ou de loin à la réussite de cette thèse et sont devenus des amis depuis : Tegawende Bissyande que je côtoie depuis le stage de Master, Julien Castet sans qui le bureau aurait été bien triste, Florent Foucaud et Reza Naserasr pour leur culture de la théorie des graphes, Vincent Rabeux ami depuis l'IUT, Pierre Halftermeyer qui m'aura appris à avoir de la repartie, Anaïs Lefeuvre pour son intégrité et la mise en place des JSD, Ève Garnaud pour sa fraicheur, Benjamin Martin qui m'a fait découvrir les joies des OS mobiles alternatifs, Allyx Fontaine qui m'a appris à relativiser, Hervé Hocquard mon premier collègue de bureau ou bien encore Petru Valicov, Eleonora Guerini, Matjaz Kovse, Srivathsan Balaguru, Rémi Laplace, Anne-Laure Gaillard, Émilie Diot, Nicolas Aucouturier, Razanne Issa, Lorijn van Rooijen et tant d'autres...

Et parmi les membres permanents, je remercie Akka Zemarri, Nicolas Bonichon, Jean-François Marckert, Philippe Duchon, Cyril Gavoille, David Ilcinkas, Pierre Castéran, Damien Magoni, Adrien Boussicault, Valentin Feray, Mathilde Bouvel, Aurélie Bugeau, Antoine Rollet...

Enfin, comme à l'accoutumée, je finirai par remercier Laure, mon amie qui me suit, soutient, supporte, aime, dispute, console depuis quelques années déjà. Cette thèse ne se serait pas passée aussi bien sans elle. Et pour tout et bien plus encore, je la remercie énormément...

Pour clôturer ces remerciements, je dédie cette thèse à mes parents et à ma soeur. Ils m'ont fait confiance depuis 8 ans (et avant, bien évidemment). J'ai toujours voulu en arriver là où j'en suis actuellement et tout ce travail, même s'ils ne le comprendront très probablement pas, représente l'aboutissement de ce que j'ai construit grâce à eux... Je vous aime.

Merci à tous ! (et veuillez m'excuser des éventuels oublis)

Présentation et Étude de Quelques Problèmes d'Algorithmique Distribuée

Résumé : Nous proposons tout d'abord une étude de plusieurs problèmes de l'algorithmique distribuée. Nous fournissons un modèle formel appliqué aux réseaux de diffusion anonymes. Dans ce modèle, nous caractérisons les graphes dans lesquels il est possible de résoudre l'énumération et l'élection. Cette caractérisation se base sur la notion d'homomorphisme de graphes. Nous proposons deux algorithmes dont la complexité est polynomiale et qui améliorent les complexités exponentielles connues jusqu'à présent. Dans un second temps, nous étudions le problème du calcul de l'état global et nous introduisons la notion de weak snapshot. Nous montrons qu'il existe des solutions pour ce problème dans les réseaux anonymes. Nous présentons plusieurs résultats concernant le calcul de l'état global en liaison avec des applications telles que le calcul de points de reprise, la détection de la terminaison ou encore le calcul d'une cartographie du réseau. Dans un cadre plus pratique, nous présentons la conception, le développement et l'implémentation des algorithmes proposés pour le calcul de l'état global au sein du logiciel de simulation et de visualisation ViSiDiA.

Mots-clés : Algorithme distribué, modélisation, graphe, réseau anonyme, état global, élection, énumération, nommage, propriété stable, débogage, visualisation.

Presentation and Study of Some Distributed Algorithm Problems

Abstract : In this thesis, we first present a study of several problems in the field of distributed algorithms. We provide a formal model that relies on anonymous networks. In this model, we characterize graphs in which it is possible to solve enumeration and leader election problems. This characterization is based on graph homomorphism. We introduce two algorithms with polynomial complexities that improve existing works with exponential complexities. On the other hand, we study the snapshot problem and we introduce the notion of weak snapshot. We show that there exist solutions for this problem in the context of anonymous networks. We present several results about distributed snapshots that deal with checkpoint and rollback recovery, termination detection or the cartography computation of a network. In a practical aspect, we present the conception, the development process and the implementation of these distributed snapshot algorithms within the simulation and visualization software ViSiDiA.

Keywords : Distributed algorithm, modelization, graph, anonymous network, global state, leader election, enumeration, naming, stable property, debugging, visualization.

Laboratoire Bordelais de Recherche en Informatique (LaBRI)
Université Bordeaux 1,
351 cours de la Libération,
33405 Talence Cedex (FRANCE)

Introduction

Le nombre et la variété des supports communicants tels que les ordinateurs, les téléphones mobiles ou encore les capteurs et les détecteurs augmentent considérablement. Une fois interconnectées à travers un réseau, ces entités, ou processus, forment un environnement distribué et collaborent dans le but de réaliser un objectif commun. Dans de tels environnements, il est difficile de définir une entité plus importante ayant la mainmise et une vision sur la globalité du réseau. Bien au contraire, chacun des processus de réseau n'a qu'une vision limitée de celui-ci. Plus précisément, il n'est possible pour un processus que d'interagir localement avec ses voisins proches. L'algorithmique distribuée se définit donc naturellement comme l'action "de penser globalement et d'agir localement"[1]. En conséquence, le développement de modèles théoriques et de protocoles à destination de ces environnements est un défi qui nécessite bien souvent l'utilisation d'outils combinatoires avancés. Nous proposons différentes contributions originales pour ces systèmes.

Quelques Problématiques de l'Algorithmique Distribuée

Dans une première partie, ce mémoire propose d'étudier plusieurs modèles et problèmes de l'algorithmique distribuée. Là où dans certains réseaux, tels que le réseau Internet, il existe des supports centralisés permettant à chaque entité d'effectuer des actions influençant la globalité du réseau comme pour l'attribution d'un identifiant unique (adresse IP sur Internet), il n'en est rien dans les environnements distribués présentés précédemment. Parallèlement, il peut être intéressant comme dans le cas des réseaux de capteurs, de pouvoir définir une entité spéciale jouant le rôle de *leader* pour ce réseau. Ces problèmes rentrent dans la catégorie des problèmes où la « symétrie » initiale du réseau doit être brisée. Le problème de l'*élection* est un problème de ce type, qui a été étudié pour la première fois par Lelann [LeL77]. Le but d'un algorithme d'élection est de distinguer un noeud du réseau qui est dans l'état ÉLU à la fin de l'exécution de l'algorithme, alors que tous les autres noeuds sont dans l'état NON-ÉLU. Si les noeuds possèdent des identifiants uniques, alors on peut élire le noeud qui a le plus petit identifiant. Si de tels identifiants n'existent pas, le réseau est dit *anonyme* et on va voir qu'il est impossible de résoudre l'élection dans certains réseaux anonymes qui sont trop « symétriques ». Un autre problème qui nécessite de briser la symétrie initiale du réseau est le problème du *nommage*. Le but d'un algorithme de *nommage* est d'arriver dans une configuration finale où un identifiant unique est associé à chaque noeud. Les problèmes du nommage et de l'élection sont équivalents dans certains modèles (si on sait résoudre l'un, on sait résoudre l'autre), mais ce n'est pas

1. Citation de René Dubos (1901-1982), chercheur et biologiste français.

toujours le cas. D'autres problèmes classiques en algorithmique distribuée sont aussi basés sur la rupture de la « symétrie » tels que le calcul d'un arbre couvrant ou la reconstruction d'une carte du réseau où chaque noeud doit localiser sa position dans le réseau.

Ces problèmes sont très importants en algorithmique distribuée puisque de nombreux algorithmes existants ne fonctionnent correctement que sous l'hypothèse qu'il existe un noeud distingué ou que les processus ont des identifiants uniques. Disposer d'un noeud distingué permet, par exemple, de centraliser ou de diffuser de l'information, d'initialiser l'exécution d'un algorithme, de prendre une décision de manière centralisée, etc. De même, disposer d'une carte du réseau où chaque processeur connaît sa position dans le réseau permet de diffuser de l'information à partir d'un noeud en minimisant le nombre de messages utilisés (voir [TvS02] p. 262 pour plus de détails).

Ces problèmes ont été très étudiés dans la littérature, et de nombreux algorithmes ont été proposés pour des topologies particulières : les arbres, les anneaux, les tores, etc. On pourra se référer aux livres de Tel [Tel00], d'Attiya et Welch [AW04] ou de Lynch [Lyn96] pour plus de détails sur ces résultats. Dans les sections suivantes, on présente les résultats existants pour les réseaux de topologies arbitraires obtenus par Yamashita et Kameda [YK96b], par Boldi, Codenotti, Gemmell, Shammah, Simon et Vigna [BCG+96] et par Mazurkiewicz [Maz97a].

Dans une seconde partie, nous proposons de résoudre les problèmes de calculs d'états globaux et de détection de propriétés stables dans les systèmes distribués et anonymes. L'état d'un système distribué, aussi appelé son *état global*, est défini par l'état de chacun des processus et l'état de chacun des canaux de communication. L'algorithme permettant d'effectuer ce calcul est un algorithme de *snapshot*. Ainsi, l'état global d'un système peut être vu comme une carte de celui-ci dans laquelle chaque sommet (resp. chaque arête) est étiqueté par son état. Par définition, dans un système entièrement distribué et asynchrone, il n'y a pas d'horloge globale. En conséquence, aucun processus n'a la connaissance de l'état global. Chaque processus connaît seulement, a priori, son propre état. Il ne connaît ni l'état des autres processus ni l'état des autres liens de communication.

Étant donné un système distribué, le but d'un algorithme de snapshot est de calculer un tel état global. Comme expliqué par Tel [Tel00] (p. 335-336), la construction d'un état global est motivée par :

- la détection de propriétés stables d'un système distribué (propriétés qui, lorsqu'elles deviennent vraies, restent vraies par la suite),
- si le système doit être relancé (p. ex., *crashs* d'un ou plusieurs éléments du système), il peut être relancé depuis le dernier état global calculé (et non depuis le début de l'exécution),
- c'est une notion très importante dans le cas du débogage de systèmes distribués. Cette notion est mise en application dans un des chapitres de ce mémoire.

Défis de l'Algorithmique Distribuée

Nous considérons uniquement les réseaux qualifiés d'*anonymes*. L'anonymat dans le domaine de l'algorithmique distribuée trouve une multitude de définitions dans beaucoup de travaux [GR05, AAER07] et dont l'inspiration originale provient de l'article d'Angluin [Ang80]. La motivation principale, d'actualité de nos jours, réside dans la protection de

la vie privée et du contrôle du partage de données sensibles. Pour diverses raisons qui lui sont propres, un utilisateur peut ne pas souhaiter partager des informations au sujet de son identité lors d'un échange quelconque ou, dans un souci de sécurité, il peut être souhaitable de ne pas autoriser l'échange de données personnelles.

Outre le défi imposé par l'anonymat, pour le développement de solutions distribuées, la connaissance initiale requise par chacune des entités du réseau est le fil rouge de nos contributions. Intuitivement, on appelle connaissance initiale une propriété inhérente du réseau telle que le nombre d'appareils qui le composent ou encore sa topologie. Il est à noter toutefois qu'un algorithme peut être totalement spécifique à un réseau particulier, mais il est préférable de proposer un algorithme *universel* pour telle ou telle famille de graphes non restreinte à un unique graphe.

Dans la littérature, une grande partie des modèles de systèmes distribués considèrent la numérotation de ports, c.-à-d., l'identification numérotée de chacun des voisins d'un sommet, comme une connaissance initiale implicite. Néanmoins, il peut être intéressant de ne pas considérer ce type de connaissance. Ainsi, le modèle associé peut s'apparenter à des situations réalistes beaucoup plus proches de la situation d'anonymat : ne pas utiliser l'identité des entités du réseau va de pair avec le fait de ne pas connaître les entités voisines.

Proposer des modèles pour lesquels les connaissances préalables sur le réseau sont les plus faibles permet de développer et de proposer des algorithmes considérés comme "fiables" : que ces hypothèses soient connues, corrompues ou inexistantes, les algorithmes fournissent un résultat correct attendu.

Nous savons que tous les réseaux anonymes n'autorisent pas d'algorithmes d'élection ou de nommage. Dans ce manuscrit, nous tentons de répondre à la question de savoir quelles sont les conditions nécessaires et suffisantes requises par un réseau pour résoudre ce problème. L'étape suivante consiste à proposer une solution distribuée pour laquelle la connaissance initiale est la plus faible possible. Cette connaissance permet, entre autres, de résoudre effectivement le problème, mais peut aussi aider, a priori, sur l'existence ou non d'une solution pour ce réseau. Lorsque l'on considère d'autres problèmes comme le calcul de l'état global, d'autres hypothèses sont à prendre en compte telles que le nombre d'initiateurs. Encore une fois, relaxer au maximum cette hypothèse permet de proposer des algorithmes beaucoup plus fiables.

La Symétrie, c'est l'Ennui [2]

Depuis le travail original d'Angluin [Ang80], il est connu que certains réseaux n'admettent pas d'algorithme d'élection en raison du phénomène lié à la « symétrie ». Intuitivement, un réseau G est « symétrique » s'il existe un autre réseau H, d'ordre (ou de taille) inférieur en nombre de processus, tel que G est schématiquement constitué de plusieurs copies entrelacées de H de telle sorte qu'un processus ne puisse, à l'aide des informations *locales* dont il dispose, savoir s'il se trouve au sein du réseau G ou du réseau H. Il existe deux outils principaux pour l'expression de ces symétries.

Dans le modèle d'Angluin, l'outil proposé pour exprimer les symétries est donné par

2. Victor Hugo (1802-1885), extrait "Les Misérables"

la notion de *revêtements simples* qui sont des homomorphismes de graphes simples localement bijectifs. En conséquence, si l'on considère le problème de l'élection pour le réseau H, le résultat final d'une exécution peut effectivement produire un résultat correct en élisant un unique sommet. Toutefois, puisque le réseau G est constitué de plusieurs copies de H, une même exécution sur le graphe G aurait alors un comportement incorrect au regard des exigences du problème de l'élection. Plusieurs processus se comporteraient de la même façon et plusieurs seraient désignés comme *élus*. Il est à noter cependant qu'il est tout à fait probable qu'une exécution sur G donne une configuration finale correcte, mais il n'est pas possible de le décider avant cette exécution. Ainsi, il n'y a pas de solution au problème de l'élection pour le réseau G.

Ce schéma de preuve, aussi appelé résultat d'impossibilité, s'adapte à tout autre modèle intermédiaire considéré, notamment, par Mazurkiewicz [Maz97a] et dans le modèle étudié par Boldi et al. [BCG+96]. Dans le modèle de Mazurkiewicz [Maz97a], un pas de calcul permet à un sommet de modifier son étiquette et l'étiquette de ses voisins en fonction de sa propre étiquette et des étiquettes de ses voisins. Il donne ainsi un algorithme qui permet de résoudre nommage et élection dans tous les graphes pour lesquels le raisonnement d'Angluin ne permettait pas d'obtenir un résultat d'impossibilité. L'algorithme de Mazurkiewicz repose seulement sur les connaissances locales que chaque sommet peut collecter à propos de ses voisins et permet de résoudre les problèmes de l'élection et du nommage sur tout graphe qui n'est un revêtement d'aucun autre graphe que lui-même.

Une autre façon de définir les symétries d'un réseau est proposée par la caractérisation obtenue par Yamashita et Kameda [YK96b] dans un modèle où les processus communiquent par échange de messages. Cette caractérisation est différente de celle de Mazurkiewicz et les techniques utilisées sont elles aussi totalement différentes puisqu'exprimées en termes de « vues ». La vue d'un sommet v est un arbre infini dont les sommets sont les chemins issus de v dans le graphe. Yamashita et Kameda montrent que pour tout algorithme, si deux sommets ont la même vue, il existe une exécution de l'algorithme où ces deux sommets restent toujours dans le même état. L'algorithme de Yamashita et Kameda repose sur un résultat de Norris [Nor95] qui a montré que deux sommets d'un graphe de taille n avaient la même vue si leurs vues tronquées à la hauteur n étaient les mêmes. Ainsi, dans l'algorithme de Yamashita et Kameda, chaque sommet construit sa vue tronquée à la hauteur n et la compare ensuite avec celle des autres sommets du graphe ; le sommet ayant la plus « petite » vue est ensuite élu.

Boldi et al. [BCG+96] utilisent la combinaison des deux outils dans leur caractérisation. Ils s'appuient sur le schéma de preuve d'Angluin pour établir des résultats d'impossibilité tandis que les algorithmes qu'ils développent s'appuient entièrement sur les vues des processus. Leur modèle est une variante du modèle de Marzurkiewicz. Ainsi, l'expression des symétries est réalisée à l'aide de *fibrations* qui sont des homomorphismes de graphes dirigés qui induisent une bijection entre les arcs entrants de chaque sommet et les arcs entrants de son image. Dans [BCG+96], Boldi et al. montrent comment obtenir des fibrations (qui ont des propriétés similaires aux revêtements) à partir de la vue de chaque sommet.

Dans ces travaux, la connaissance initiale joue un rôle important dans la puissance des modèles traités. Les algorithmes de Yamashita et Kameda et de Boldi et Vigna nécessitent des connaissances initiales sur le graphe afin de savoir jusqu'à quelle hauteur chaque

sommet doit calculer sa vue. Au contraire, l'algorithme de Mazurkiewicz ne nécessite aucune connaissance initiale et il termine toujours, même si sans connaissance initiale, il est impossible aux sommets de détecter si l'exécution de l'algorithme est terminée ou non. Cette différence est importante puisque cette propriété de l'algorithme de Mazurkiewicz est utilisée dans [GMM04] afin de caractériser les classes de graphes reconnaissables de manière distribuée dans le modèle considéré par Mazurkiewicz (dans ce cadre, les sommets ne doivent pas nécessairement détecter la terminaison de l'algorithme). Par ailleurs, pour exécuter l'algorithme de Mazurkiewicz, les sommets d'un graphe G ont besoin d'une mémoire polynomiale en la taille du graphe, alors que les algorithmes de Yamashita et Kameda et de Boldi et Vigna nécessitent que chaque sommet ait une mémoire exponentielle en la taille du graphe. Cette propriété est importante puisque dans l'algorithme de Yamashita et Kameda, la taille de la mémoire de chaque processus est liée à la taille des messages échangés, qui sont aussi de taille exponentielle.

Instantané d'un Réseau et ses Applications

Le calcul de l'état global ou *snapshot* d'un réseau est un problème important pour l'analyse, le test et la supervision d'un algorithme distribué. De par l'essence même des systèmes distribués, il n'est pas toujours évident de calculer ce genre d'instantané du réseau : il n'existe pas d'entité centralisée, d'horloge globale et les communications sont asynchrones. Depuis, le travail initial de L. Lamport [Lam78] sur la formalisation de la notion du temps dans les systèmes distribués, plusieurs corollaires ont été obtenus dont une large étude est présentée dans [KS08].

Intuitivement, l'état local d'un processus est composé de l'état de sa mémoire ainsi que de l'historique de son activité préalable. De même, l'état d'un canal de communication est composé des messages actuellement en transit sur ce canal. Par voie de conséquence, le snapshot d'un système distribué est une collection des états locaux des processus et des canaux de communication.

Chandy et Lamport [CL85] sont les précurseurs dans le calcul de l'état global. Ils décrivent didactiquement la faisabilité d'un état global suivant les hypothèses connues du réseau. Il proposent alors un algorithme, l'algorithme de Chandy-Lamport, dont les utilisations principales s'attachent aux problèmes en relation avec la détection de propriétés stables telles que les *interblocages* ou *deadlocks* ou encore la détection de la terminaison. Pour une reprise après incident, aussi appelé *rollback recovery*, un snapshot, nommé dans ce cas précis un *checkpoint*, est régulièrement calculé. Lors d'un échec, il est alors possible de forcer la reprise des processus depuis un état correct associé à l'un des checkpoints préalablement calculés. Toutefois, le domaine d'application du problème du snapshot ne se limite pas à ces notions. L'utilisation de snapshots aide pour le débogage et la supervision de systèmes distribués.

Afin de résoudre les problèmes cités précédemment, il est classique de faire des hypothèses fortes sur le réseau. Initialement, il ne peut y avoir qu'un seul processus responsable de l'exécution du calcul de l'état global. De plus, il est impossible de s'abstraire de l'identifiant de chaque processus. En ce sens, nous perdons le bénéfice apporté par l'anonymat d'un réseau. Enfin, les canaux de communication doivent, dans la plupart des travaux, respecter l'ordre dans lequel les messages sont envoyés, c.-à-d., les canaux sont

FIFO. Cette dernière hypothèse est, toutefois, relaxée dans certains travaux, mais n'est pas sans contrepartie : les algorithmes deviennent alors inhibiteurs [Hel89], c.-à-d., retardent l'exécution d'un algorithme sous-jacent, ou encore, les messages sont agrémentés d'informations telles que des couleurs identifiant certains états des processus [LY87] et ainsi de déterminer un ordre logique pour le système. Mattern [Mat89] présente aussi un algorithme non-inhibiteur pour le problème du snapshot basé sur des couleurs nécessitant une mémoire plus faible. Une variation de l'algorithme de Chandy-Lamport existe pour l'utilisation de canaux non-FIFO en utilisant la notion de marqueurs [Ahu92].

Actuellement, il n'existe pas de solution pour les problèmes précédents dans les réseaux anonymes si aucun processus n'a de connaissance sur le réseau telle qu'un arbre couvrant ou s'il y a plusieurs initiateurs. Néanmoins, nous allons montrer dans le chapitre 3 qu'il est possible pour un processus de connaître anonymement quelques propriétés du réseau. Nous introduisons ainsi la notion de *weak snapshot* étroitement liée à la notion de *revêtements de graphes* initialement utilisée pour les problèmes de symétries.

Techniques et Résultats

L'objectif de ce manuscrit ainsi que des contributions de ce mémoire consiste à étudier différents problèmes de l'algorithmique distribuée. Pour cela, nous fournissons plusieurs caractérisations complètes en lien avec les hypothèses sur la connaissance initiale.

Études et Analyses

Nous présentons tout d'abord, dans le chapitre 1, le formalisme nécessaire à nos caractérisations. Pour cela, nous exposons quelques rappels de la théorie des graphes et de combinatoires notamment autour des homomorphismes de graphes. Nous rappelons également la notion de réétiquetage de graphes ou *calculs locaux* utilisée dans le chapitre 2. Enfin, nous terminons par les définitions formelles des problèmes traités dans ce manuscrit ainsi que quelques détails supplémentaires à propos de la connaissance initiale.

Dans un premier temps, nous considérons un modèle distribué partiellement influencé par ceux proposés par Mazurkiewicz [Maz97a], Yamashita et Kameda [YK99] et Boldi *et al.* [BCG$^+$96] : le modèle de communication par diffusion. Dans ce modèle, le réseau de communication peut avoir une topologie arbitraire. Notons que les liens de communication sont fiables : les messages envoyés ne sont pas perdus ou corrompus durant leur transit et les noeuds ne sont soumis à aucune défaillance. Habituellement, ce type de réseau est représenté par un graphe étiqueté dans lequel les processus correspondent aux sommets, les liens de communication aux arêtes et les états des processus aux étiquettes. Dans le chapitre 2, nous considérons les communication de type asynchrone ou synchrone. Dans le cas asynchrone, il n'y a pas d'horloge globale et un message envoyé depuis un processus arrive en un temps arbitraire, mais fini. Dans le cas synchrone, la notion d'étape de calcul intervient durant laquelle lorsqu'un sommet envoie un message, tous ses voisins le reçoivent avant le début d'une nouvelle étape. Dans ce chapitre, nous aborderons les problèmes du nommage/énumération ainsi que de l'élection. Nous présenterons les conditions nécessaires, basées sur des résultats d'impossibilité et des *fibrations*, requises par un graphe pour admettre un algorithme de nommage ou d'élection dans les modèles asynchrone et

synchrone. Nous montrons ensuite que ces conditions sont suffisantes en proposant des algorithmes de nommage/énumération et d'élection pour les modèles considérés inspirés de l'algorithme de Mazurkiewicz. En supplément du chapitre 2, on étudie aussi la complexité de ces algorithmes et nous montrons qu'ils possèdent une complexité polynomiale contrairement aux travaux existants qui proposent des algorithmes de complexité exponentielle. Une partie de ces résultats a fait l'objet de l'article [CMM12a] en ce qui concerne le modèle de diffusion asynchrone.

Dans le chapitre 3, on étudie les problèmes de calcul d'états globaux et de détection de propriétés stables [KS08]. D'autres applications sont aussi présentées comme le calcul de points de reprise ou encore de la détection de la terminaison. Pour ce chapitre, nous considérons les réseaux anonymes et le modèle classique de communication par passage de messages. Nous présentons une démarche progressive autour des problèmes traités. Ainsi, nous proposons des solutions originales jusqu'alors inexistantes, totalement distribuées et anonymes pour la détection de propriétés stables et le calcul de points de reprise. Pour ces problèmes, nous abordons une fois de plus les problèmes liés à la connaissance initiale : notre contribution ne requiert des processus qu'une faible connaissance. Enfin, élément important de notre contribution, contrairement au travaux existants, on autorise des initiateurs multiples, c.-à-d., plusieurs processus peuvent être instigateurs des calculs. Une partie des résultats présentés dans le chapitre 3 a été publiée dans [CMM12b] et dans [CMM12c].

Parmi les domaines d'applications des algorithmes totalement distribués et anonymes pour le calcul de propriétés stables, on retrouve le débogage de systèmes distribués. De par sa nature distribuée, il est difficile de superviser le comportement d'un réseau. Toutefois, il peut être important, voire crucial, de pouvoir détecter une faille, un *crash* ou tout simplement de pouvoir surveiller un réseau. Dans le chapitre 4, nous proposons de développer et d'intégrer les algorithmes formalisés dans le chapitre 3 dans le logiciel de simulation d'algorithmes distribués ViSiDiA. Étant donné la nature distribuée de nos algorithmes, il est à noter que leur utilisation dans le cadre d'une fonctionnalité de débogage ne s'arrête pas qu'à un seul logiciel. Dans ce chapitre, nous montrons le processus de développement afin de montrer qu'il est relativement aisé d'intégrer cette fonctionnalité au sein d'un logiciel de simulation. Dans un premier temps, nous faisons une brève étude de solutions existantes et nous montrons la plus-value de notre solution par rapport aux autres. Nous exposons à l'aide d'un exemple qu'il est possible de détecter d'autres propriétés d'un réseau, non obligatoirement stables, c.-à-d., qui changent au fur et à mesure de l'exécution. L'étude, le développement ainsi que l'intégration au sein du logiciel ViSiDiA a fait l'objet de deux articles [AMM12b] et [AMM12a].

Vers le Développement d'Algorithmes "Implémentables"

L'objectif principal de la contribution du chapitre 2 consiste à proposer un nouvel algorithme de nommage et d'élection pour le modèle de diffusion. Il s'agit d'obtenir un algorithme « à la Mazurkiewicz » dans le modèle proposé par [CM98] ; le but étant d'obtenir un algorithme plus efficace, c.-à-d., de complexité au plus polynomial et aisément « implémentable ». Ainsi, notre algorithme est « totalement » polynomial, au sens où le temps d'exécution, le nombre de messages et les tailles des messages échangés sont poly-

nomiaux en la taille du graphe, alors que l'algorithme de Yamashita et Kameda nécessite un nombre polynomial de messages de tailles exponentielles. Le modèle ainsi considéré s'apparente à un modèle de communication sans fil par liaison radio. Il est alors plus que nécessaire de prendre en compte des arguments en faveur de la consommation d'énergie étroitement liée à la complexité des algorithmes utilisés.

On explique ensuite comment ces résultats peuvent être généralisés à un modèle synchrone où les processus communiquent par échange de messages qui ont été étudiés par Yamashita et Kameda dans [YK99].

Il en est de même pour le chapitre 3. Nous proposons effectivement des solutions originales pour les problèmes de calculs d'états globaux ou de détection de propriétés stables totalement distribuées et anonymes. Toutefois, lorsque l'on traite les travaux existants, on remarque régulièrement la difficulté quant au fait d'assembler les informations distribuées partout sur le réseau. Dans une grande majorité des cas, il existe un noeud central qui s'occupe de les récupérer afin de les traiter. Néanmoins, dans un réseau le coût engendré par la circulation de ces informations peut s'avérer non négligeable. C'est pourquoi les solutions que nous développons possèdent une complexité acceptable en regard des tâches réalisées à savoir la détection de propriétés globales stables.

Dans le chapitre 4, nous montrons l'importance des fondements théoriques présentés dans le chapitre 3 en développant puis en intégrant une couche de débogage au sein même d'un logiciel de simulation d'algorithmes distribués, ViSiDiA. Le fait d'avoir développé et prouvé les algorithmes de détection de propriétés stables en se basant sur le modèle à base de passage de messages permet de les implémenter plus aisément. Nous fournissons alors un système complet de débogage totalement distribué et anonyme permettant à chacun des processus d'un réseau de déterminer l'existence d'une propriété particulière en limitant au maximum le nombre et la taille des messages échangés. Cette limitation a pour but d'éviter un engorgement du réseau et, a fortiori, de limiter les erreurs.

Travaux Précédents

La contribution proposée par le chapitre 2 de ce manuscrit complète les travaux commencés initialement par Y. Métivier *et al.*. Depuis les résultats d'impossibilité exposés dans [CM94, MMW97] ainsi que plusieurs résultats sur l'étude de la puissance du réétiquetage de graphes, Godard [God02] et Chalopin [Cha06] ont étudié les modèles « intermédiaires » entre le modèle de Mazurkiewicz et le modèle de Yamashita et Kameda. Ils considèrent différents modèles utilisant des *calculs locaux* dont le modèle étudié par Mazurkiewicz [Maz97a] est le plus puissant. Ainsi, dans ce modèle, un algorithme peut être décrit à l'aide de règles de réétiquetage qui permettent de modifier les étiquettes des sommets d'une étoile du graphe (un sommet et ses voisins) en fonction seulement des étiquettes apparaissant sur cette étoile. Des corollaires de ces résultats ont été notamment obtenus par Godard et Métivier [GM02].

Le modèle entrelacé de Boldi et al. [BCG$^+$96] dans lequel un pas de calcul permet à un sommet de modifier seulement son étiquette en fonction de sa propre étiquette et de l'étiquette de ses voisins peut s'apparenter au modèle exposé dans le chapitre 2 pour le cas synchrone. Dans ce modèle et par l'intermédiaire des *calcul locaux*, Chalopin [Cha06] (chapitre 4) présente en particulier des algorithmes de nommage et d'élection qui utilisent

certaines idées de l'algorithme de Mazurkiewicz et qui nécessitent que chaque sommet ait une mémoire polynomiale en la taille du graphe, et non exponentielle comme les algorithmes de Boldi *et al.*. Cependant, l'utilisation des calculs locaux oblige une synchronisation entre voisins : bien que l'exécution soit globalement asynchrone, un pas de calcul nécessite une synchronisation entre un sommet et ses voisins. La contribution du chapitre 2 diffère dans le sens où est utilisé un modèle à base de passage de messages basé sur des communications asynchrones.

De nombreux autres modèles de calculs locaux sont étudiés pour lesquels les synchronisations prennent différentes formes et autorisent plusieurs variantes de réétiquetage de graphes [God02, Cha06, CMZ06, CM07a, CM10a]. Pour chaque modèle, on caractérise les graphes dans lesquels on peut résoudre les problèmes de l'élection et du nommage. Pour obtenir des conditions nécessaires, il faut d'abord trouver quels sont les homomorphismes qui permettent d'obtenir un lemme de relèvement « à la Angluin » pour la classe de graphes la plus large possible pour chaque modèle considéré. Pour obtenir des conditions suffisantes, il faut ensuite trouver un algorithme qui permette de nommer dans tous les graphes pour lesquels on n'a pas obtenu de résultat d'impossibilité. La recherche de ces homomorphismes et de ces algorithmes sont très liés, puisqu'il faut réussir à déterminer exactement quelles sont les informations que peut obtenir chaque sommet à propos de ses voisins. Par exemple, dans certains modèles [CMZ06], un sommet peut réussir à détecter s'il a un ou plusieurs voisins qui ont la même étiquette, alors que ce n'est pas possible dans d'autres modèles. Les algorithmes qu'on propose sont inspirés de l'algorithme de Mazurkiewicz, au sens où, un sommet essaie d'obtenir le plus d'informations possible à propos de son voisinage. Il faut toutefois noter que les algorithmes proposés ne sont pas des adaptations triviales de l'algorithme de Mazurkiewicz et qu'il faut dans chaque modèle utiliser des méthodes particulières afin de s'assurer que chaque sommet réussisse à collecter toute l'information à propos de ses voisins dont il a besoin. Les caractérisations obtenues permettent de mettre en évidence les relations entre les modèles et la différence entre leurs puissances de calcul respectives. Il reste ensuite à déterminer quelles connaissances initiales sont nécessaires pour résoudre ces problèmes. Les résultats obtenus permettent de mettre en évidence quels sont les résultats existants dans le modèle de Mazurkiewicz [MT00, GM02, GM03, GMM04] qu'il est possible d'étendre dans les différents modèles considérés.

La compréhension des caractéristiques des graphes admettant un algorithme d'élection ou de nommage dans les différents modèles est une étape importante dans la compréhension de ce qui est calculable de manière distribuée. Ainsi, les travaux de Yamashita et Kameda [YK96a, YK98] sur les fonctions qui pouvaient être calculées dans un modèle où les processus communiquent par échange de messages utilisent les concepts introduits dans [YK96b] où ils caractérisent les réseaux admettant un algorithme d'élection dans ce même modèle.

Dans [BCG$^+$96], Boldi, Codenotti, Gemmell, Shammah, Simon et Vigna caractérisent les graphes qui admettent un algorithme d'élection dans un modèle où, en un pas de calcul, un sommet peut modifier son état en fonction de l'état de ses voisins. Ils considèrent un modèle *synchrone* où, en un pas de calcul, tous les sommets du graphe appliquent une transition de ce type et un modèle *entrelacé*, où à chaque étape, il y a un unique sommet qui applique une transition. À partir des résultats présentés dans [BCG$^+$96], Boldi et

Vigna ont caractérisé les fonctions qui pouvaient être calculées de manière distribuée dans le même modèle [BV99, BV01]. Toujours en utilisant les mêmes outils, ils ont aussi caractérisé quelles tâches pouvaient être effectuées de manière auto-stabilisante [BV02b] dans un système synchrone.

Dans [Maz97a], Mazurkiewicz caractérise les graphes admettant un algorithme de nommage dans un modèle où, en un pas de calcul, un sommet peut modifier son état et l'état de ses voisins en fonction de son propre état et de l'état de ses voisins. Ce modèle est *entrelacé* puisque deux voisins ne peuvent pas appliquer simultanément une transition de ce type. Dans [GM02], Godard et Métivier utilisent l'algorithme de Mazurkiewicz ainsi que d'autres outils pour caractériser quelles sont les familles de graphes qui admettent un algorithme universel d'élection dans ce modèle. Cet algorithme est aussi utilisé dans [MT00] par Métivier et Tel afin de déterminer quelles conditions permettent de détecter la terminaison globale d'un algorithme distribué. Dans les travaux de Godard, Métivier et Muscholl [GMM04, GM03], l'algorithme de Mazurkiewicz est aussi l'un des outils utilisés afin de caractériser les classes de graphes qui peuvent être reconnues de manière distribuée dans ce modèle.

Table des matières

Chapitre 1

Préliminaires

Sommaire

Dans ce chapitre, on rappelle d'abord quelques définitions élémentaires sur les graphes non-dirigés et dirigés et on introduit les notations qu'on utilisera par la suite. On présente ensuite les relations de réétiquetage de graphes et on explique comment elles permettent de coder des algorithmes distribués. On présente finalement les problèmes que l'on considère dans ce mémoire.

1.1 Graphes Non-dirigés

1.1.1 Définitions

Définition 1.1 *Un* graphe non-dirigé sans boucle *est défini par un ensemble de sommets $V(G)$, un ensemble d'arêtes $E(G)$ et par une fonction ext qui associe à chaque arête deux éléments distincts de $V(G)$, appelés ses* extrémités.

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

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

Définition 1.2 *Un* graphe simple non-dirigé *est un graphe non-dirigé sans boucle tel qu'il y ait au plus une arête entre deux sommets, i.e., pour tous sommets $u, v \in V(G)$, $|\{e \in E(G) \mid ext(e) \in \{u, v\}\}| \leq 1$.*

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

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

Définition 1.3 *Un graphe H est un sous-graphe partiel d'un graphe G si $V(H) = V(G)$ et $E(H) \subseteq E(G)$.*

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

Un graphe H est un sous-graphe induit de G si $V(H) \subseteq V(G)$ et $E(H)$ est l'ensemble des arêtes de G dont les extrémités sont dans $V(H)$, i.e., $\forall e \in E(G), ext(e) \subseteq V(H) \iff e \in E(H)$. Pour tout graphe G et tout ensemble $S \subseteq V(G)$, on note $G[S]$ le sous-graphe induit de G dont les sommets sont les éléments de S.

Définition 1.4 *Dans un graphe G, le* voisinage *d'un sommet u, noté $N_G(u)$, est l'ensemble des voisins de u, i.e., $N_G(u) = \{v \in V(G) \mid \exists e \in E(G)$ telle que $ext(e) = \{u, v\}\}$. On note $I_G(u)$ l'ensemble des arêtes incidentes au sommet u, i.e., $I_G(u) = \{e \in E(G) \mid \exists e \in E(G)$ telle que $u \in ext(e)\}$.*

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

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

Définition 1.5 *Étant donnés un graphe simple G et un sommet $u \in V(G)$, l'*étoile *de centre u est le sous-graphe de G défini par $V(B_G(u)) = N_G(u)$ et $E(B_G(u)) = I_G(u)$.*

Définition 1.6 *Dans un graphe G, un* chemin Γ *entre deux sommets u et v de G est une suite alternée $(u_0, e_1, u_1, e_2, \ldots, e_n, u_n)$ telle que :*
- *pour tout $i \in [0, n]$, $u_i \in V(G)$,*
- *pour tout $i \in [1, n]$, $e_i \in E(G)$,*
- *pour tout $i \in [1, n]$, $ext(e_i) = \{u_{i-1}, u_i\}$,*
- *$u_0 = u$ et $u_n = v$.*

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

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

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

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

Définition 1.7 *Un graphe G est* connexe *si pour tous sommets $u, v \in V(G)$, il existe un chemin entre u et v dans G.*

Tous les graphes considérés dans ce mémoire seront connexes.

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

Définition 1.8 *Étant donnés un graphe connexe G et deux sommets $u, v \in V(G)$, la* distance *de u à v dans G, notée $\mathrm{dist}_G(u, v)$, est la longueur du plus court chemin de u à v. Le* diamètre *de G, noté $D(G)$, est la plus grande distance entre deux sommets de G, i.e., $D(G) = \max\{\mathrm{dist}_G(u, v) \mid u, v \in V(G)\}$.*

Définition 1.9 *Un* homomorphisme *φ d'un graphe G dans un graphe H est une application de $V(G)$ dans $V(H)$ et de $E(G)$ dans $E(H)$ qui préserve les relations d'incidence, i.e., pour toute arête $e \in E(G)$, $ext(\varphi(e)) = \varphi(ext(e))$.*

Définition 1.10 *Un homomorphisme φ d'un graphe G dans un graphe H est un* isomorphisme *si φ est bijectif.*

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

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

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

Définition 1.11 *Un* homomorphisme φ *d'un graphe simple* G *dans un graphe simple* H *est une application de* $V(G)$ *dans* $V(H)$ *qui préserve les relations d'adjacence entre sommets, i.e., pour toute arête* $\{v, w\} \in E(G)$, $\{\varphi(v), \varphi(w)\} \in E(G)$.

Définition 1.12 *Un homomorphisme* φ *d'un graphe simple* G *dans un graphe simple* H *est un* isomorphisme *si* φ *est bijective et si* φ^{-1} *est aussi un isomorphisme.*

1.1.2 Graphes Étiquetés

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

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

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

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

Définition 1.13 *Un graphe* $\mathbf{H} = (H, \eta)$ *est un graphe partiel (resp. sous-graphe, sous-graphe induit) d'un graphe* $\mathbf{G} = (G, \lambda)$ *si* H *est un graphe partiel (resp. sous-graphe, sous-graphe induit)de* G *et pour tout* $x \in V(H) \cup E(H)$, $\lambda(x) = \eta(x)$.

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

Définition 1.14 *Un* homomorphisme φ *d'un graphe étiqueté* $\mathbf{G} = (G, \lambda)$ *dans un graphe* $\mathbf{H} = (H, \eta)$ *est un homomorphisme de* G *dans* H *tel que pour tout* $x \in V(G) \cup E(G)$, $\lambda(x) = \eta(\varphi(x))$.

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

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

Remarque 1.15 *On peut assimiler tout graphe non-étiqueté* G *au graphe étiqueté* (G, λ_ϵ) *où* λ_ϵ *est un étiquetage tel que pour tout* $x \in V(G) \cup E(G)$, $\lambda_\epsilon(x) = \epsilon$, *où* ϵ *est le mot vide. Par conséquent, le terme «graphe» désignera indistinctement un graphe étiqueté et un graphe non-étiqueté.*

1.1.3 Étiquetages et Colorations

Nous montrons le lien qu'il existe entre la notion d'étiquetage et la notion de coloration. Les fonctions d'étiquetages considérées n'attribuent des étiquettes qu'aux sommets (les arêtes ne sont pas étiquetées). Une coloration d'un graphe simple G est un étiquetage tel que deux sommets adjacents dans G ont deux couleurs distinctes.

Définition 1.16 *Une* coloration ℓ *d'un graphe simple G est un étiquetage de G tel que pour toute arête $\{u, v\} \in V(G)$, $\ell(u) \neq \ell(v)$.*

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

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

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

Définition 1.17 *Un étiquetage ℓ d'un graphe G est un étiquetage* propre *si $|\ell(V(G))| < |V(G)|$.*

1.1.4 Étiquetage des Ports

Dans les chapitres 2 et 3 suivants, un système distribué va être représenté par un graphe étiqueté dont les sommets correspondent aux processus et les arêtes aux liens de communication. Dans le modèle utilisé par le chapitre 3, chaque processus peut distinguer ses voisins, c.-à-d., il peut distinguer les canaux de communication par lesquels il reçoit ou envoie des messages. Pour cela, on suppose que le graphe correspondant au système distribué a un étiquetage des ports défini de la manière suivante.

Dans un graphe simple, un étiquetage des ports est un ensemble de fonctions locales qui permettent à chaque sommet de distinguer ses voisins.

Définition 1.18 *Étant donné un graphe simple $\mathbf{G}$, un étiquetage des ports δ est un ensemble de fonctions $\{\delta_u \mid u \in V(G)\}$ tel que pour tout sommet u, δ_u est une bijection entre $N_G(u)$ et $[1, \deg_G(u)]$.*

On représente un graphe $\mathbf{G}$ avec un étiquetage δ en représentant le numéro $\delta_u(v)$ sur l'incidence entre le sommet u et l'arête $\{u, v\}$ comme représenté sur la figure 1.1.

1.2 Graphes Dirigés

Les définitions et notations utilisées ici proviennent du travail de Boldi et Vigna sur les fibrations [BV02a].

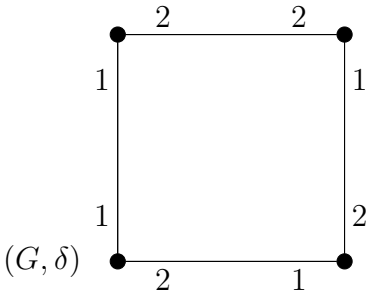

FIGURE 1.1 – Un graphe simple G avec un étiquetage de ports δ.

1.2.1 Définitions

Définition 1.19 *Un* graphe dirigé D *est défini par un ensemble de* sommets $V(D)$*, un ensemble d'*arcs $A(D)$ *et deux fonctions s_D et t_D de $A(D)$ dans $V(D)$.*

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

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

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

Définition 1.20 *Dans un graphe dirigé D, le* voisinage sortant *d'un sommet u, noté $N_D^+(u)$ est l'ensemble des successeurs de u, i.e., $N_D^+(u) = \{v \in V(D) \mid \exists a \in A(D)$ tel que $s(a) = u$ et $t(a) = v\}$. On note $I_D^+(u)$ l'ensemble des arcs sortants de u, i.e., $I_D^+(u) = \{a \in A(D) \mid s(a) = u\}$. Le* degré sortant *de u, noté $d_D^+(u)$ est le nombre d'arcs sortants de u, i.e., $d_D^+(u) = |I_D^+(u)|$.*

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

Définition 1.21 *Dans un graphe dirigé D, un* chemin *de u à v est une suite d'arcs $(a_0, a_2, \ldots, a_n)$ telle que $s(a_0) = u$, $t(a_n) = v$ et pour tout $i \in [1, n]$, $t(a_{i-1}) = s(a_i)$.*

Définition 1.22 *Un graphe dirigé D est fortement connexe si pour tous sommets u et v, il existe un chemin de u à v dans D.*

Définition 1.23 *Un* graphe dirigé symétrique *est un graphe dirigé D muni d'une involution $Sym : A(D) \to A(D)$ qui à chaque arc $a \in A(D)$ associe son arc symétrique $Sym(a)$ tel que $s_D(Sym(a)) = t_D(a)$.*

Définition 1.24 *Un* homomorphisme φ *d'un graphe dirigé D dans un graphe dirigé D' est une application de $V(D)$ dans $V(D')$ et de $A(D)$ dans $A(D)$ qui commute avec les fonctions source et cible, i.e., pour toute arc $a \in A(D)$, $s_{D'}(\varphi(a)) = \varphi(s_D(a))$ et $s_{D'}(\varphi(a)) = \varphi(s_D(a))$.*

Définition 1.25 *Un homomorphisme φ d'un graphe dirigé D dans un graphe dirigé D' est un* isomorphisme *si φ est bijective.*

On dit alors que D et D' sont isomorphes.

1.2.2 Graphes Dirigés Étiquetés

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

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

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

Définition 1.26 *Un* homomorphisme *φ d'un graphe dirigé étiqueté $\mathbf{D} = (D, \lambda)$ dans un graphe dirigé $\mathbf{D'} = (D', \lambda')$ est un homomorphisme de D dans D' tel que pour tout $x \in V(D) \cup A(D)$, $\lambda(x) = \lambda'(\varphi(x))$.*

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

Remarque 1.27 *Comme pour les graphes non-dirigés, on peut assimiler tout graphe dirigé non-étiqueté D au graphe étiqueté (D, λ_ϵ) où λ_ϵ est la fonction d'étiquetage qui associe à chaque sommet et à chaque arc de D l'étiquette ϵ qui est le mot vide.*

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

Définition 1.28 *Étant donné un graphe dirigé D, on dit qu'une fonction d'étiquetage λ est* déterministe *(resp.* codéterministe*), si pour tous arcs $a, a' \in A(D)$ tels que $s(a) = s(a')$ (resp. $t(a) = t(a')$), $\lambda(a) \neq \lambda(a')$.*

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

Définition 1.29 *Étant donné un graphe dirigé symétrique D, on dit qu'une fonction d'étiquetage λ est* symétrique*, s'il existe une fonction $Sym : L \to L$ telle que pour tout arc a, $Sym(\lambda(a)) = \lambda(Sym(a))$.*

Par la suite, lorsqu'on considérera des graphes dirigés symétriques ayant un étiquetage symétrique, on supposera que tout arc a a une étiquette de la forme (p, q) telle que $Sym(a) = (q, p)$. Étant donnés un graphe dirigé symétrique D et un étiquetage symétrique

λ de D, il est toujours possible d'obtenir un étiquetage λ' de cette forme, en posant pour tout arc $a \in A(D)$, $\lambda'(a) = (\lambda(a), Sym(\lambda(a)))$.

Nos preuves utilisent la notion de *vues*. Intuitivement, la vue d'un sommet v d'un graphe dirigé étiqueté $\mathbf{D}$ est obtenu en considérant tous les chemins étiquetés dans $\mathbf{D}$ se terminant en v. D'un point de vue algorithmique distribuée, comme expliqué dans l'introduction, la vue d'un processus dans un réseau est un arbre représentant toutes les informations qu'il peut rassembler à propos de ce réseau.

Définition 1.30 *Étant donné un graphe dirigé étiqueté $\mathbf{D}$, la vue $T_{\mathbf{D}}(v_0)$ d'un sommet v_0 est un arbre étiqueté infini qui peut être défini de façon récursive. La racine de l'arbre est le sommet x_0 qui correspond à v_0 et est étiqueté par $\lambda(v_0)$. Pour chaque voisin entrant v_i de v_0 dans $\mathbf{D}$, il y a un arc entre x_0 et la racine x_i de l'arbre $T_{\mathbf{D}}(v_i)$. Soit d un entier, la d-vue $T^d_{\mathbf{D}}(v_0)$ de $v_0/inV(D)$ est la vue infinie $T_{\mathbf{D}}(v_0)$ tronquée à une profondeur d.*

D'après cette définition, nous pouvons en déduire que l'ensemble des d-vue d'un graphe dirigé $\mathbf{D}$ est fini. Ainsi, nous pouvons définir un ordre partiel $\succeq$ sur cette ensemble comme suit :

Définition 1.31 *Pour chaque sommet $v, w \in V(D)$, si $T = T^d_{\mathbf{D}}(w)$ est un sous-arbre de $T' = T^d_{\mathbf{D}}(v)$, alors $T' \succeq T$. Il est à noter que s'il existe un isomorphisme entre T et T', nous les qualifions comme similaires, noté $T \approx T'$.*

1.2.3 Des Graphes non-dirigés aux Graphes Dirigés

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

En général, on manipulera les graphes dirigés symétriques obtenus à partir de graphes simples. Dans ce cas là, pour tout graphe simple, $\mathbf{G} = (G, \lambda)$, on peut définir $Dir(\mathbf{G}) = (Dir(G), \lambda')$ de la manière suivante. L'ensemble des sommets de $V(Dir(G))$ est l'ensemble $V(G)$ et pour chaque arête $\{u, v\} \in E(G)$, il existe deux arcs $a_{u,v}$ et $a_{v,u}$ dans $A(Dir(G))$ tels que $s(a_{u,v}) = t(a_{v,u}) = u$, $s(a_{v,u}) = t(a_{u,v}) = v$ et $Sym(a_{u,v}) = a_{v,u}$. Pour tout sommet $u \in V(G)$, $\lambda'(u) = \lambda(u)$ et pour toute arête $\{u, v\}$, $\lambda'(a_{u,v}) = \lambda'(a_{v,u}) = \lambda(\{u, v\})$. Il est facile de voir que dans le cas des graphes simples, cette construction permet d'obtenir le même graphe dirigé symétrique que la précédente. Un exemple de cette construction est présenté sur la figure 1.2.

De même, on peut représenter un graphe $(\mathbf{G}, \delta)$ où $\mathbf{G} = (G, \lambda)$ est un graphe simple dont seuls les sommets sont étiquetés et où δ est une fonction d'étiquetage des ports.

À chaque réseau $(\mathbf{G}, \delta)$, on associe le graphe dirigé symétrique $(Dir(\mathbf{G}), \delta)$ où tout sommet $u \in V(Dir(\mathbf{G}))$ est étiqueté comme dans $\mathbf{G}$ et où tout arc $a_{u,u'} \in A(Dir(G))$ tel que $s(a) = u$ et $t(a) = u'$ est étiqueté par $(\delta_u(u'), \delta_{u'}(u))$. On remarque que pour tout arc $a_{u,u'} \in A(Dir(G))$ étiqueté (p, q), l'arc symétrique $Sym(a_{u,u'})$ est étiqueté (q, p) :

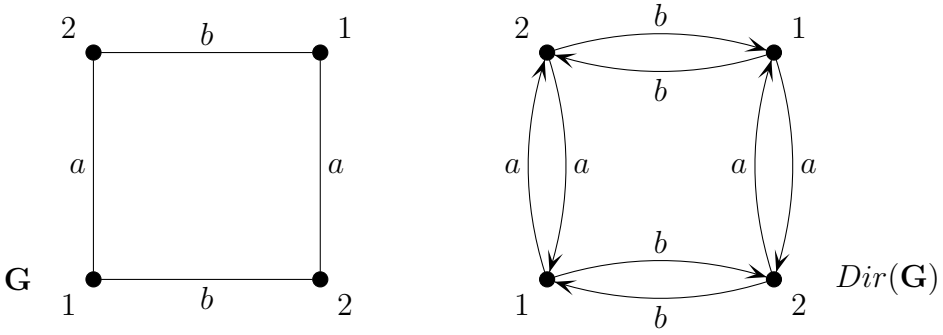

FIGURE 1.2 – Un graphe simple $\mathbf{G}$ et le graphe dirigé symétrique $Dir(\mathbf{G})$ correspondant.

l'étiquetage des arcs du graphe dirigé obtenu est symétrique. Par ailleurs, puisque δ est une fonction d'étiquetage des ports, on remarque que l'étiquetage des arcs ainsi obtenu est déterministe et codéterministe. Une telle représentation d'un réseau (G, δ) est présentée sur la figure 1.3.

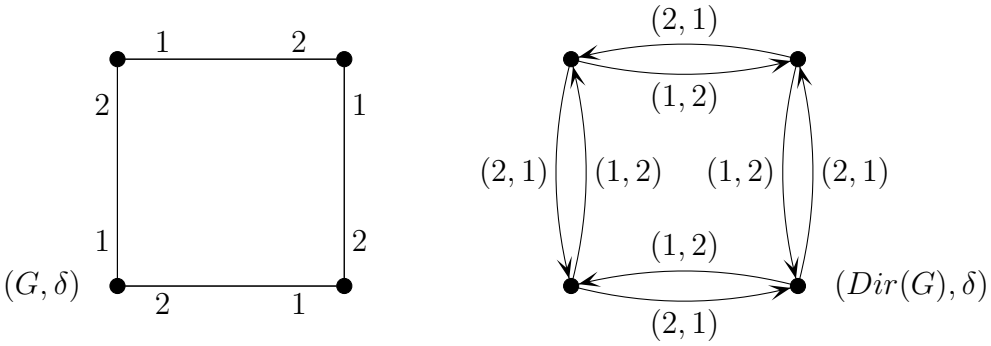

FIGURE 1.3 – Un graphe G avec un étiquetage de ports δ et sa représentation sous forme de graphe dirigé.

1.3 Réétiquetages

Dans le chapitre 2, on va modéliser des algorithmes distribués par des relations de réétiquetage de graphe. On présente ici les modèles les plus généraux qu'on considère. Les modèles plus restrictifs sont introduits et étudiés dans [Cha06].

1.3.1 Définitions

Étant donnée une relation binaire $\mathcal{R}$ sur les graphes étiquetés, $\mathcal{R}$ définit une relation de *réécriture* de graphes. On ne considère que des relations closes par isomorphisme, i.e., si $\mathbf{G}\ \mathcal{R}\ \mathbf{H}$ et $\mathbf{G} \simeq \mathbf{G}'$, alors il existe un graphe $\mathbf{H}'$ tel que $\mathbf{G}'\ \mathcal{R}\ \mathbf{H}'$ et $\mathbf{H} \simeq \mathbf{H}'$.

FIGURE 1.4 – Une règle de réétiquetage d'arête.

Par ailleurs, on ne considère que des relations de *réétiquetage* de graphes qui ne modifient pas la structure du graphe mais seulement son étiquetage.

Définition 1.32 *Une relation de réécriture de graphes $\mathcal{R}$ est une* relation de réétiquetage *de graphes si pour tout couple de graphes en relations, les graphes sous-jacents sont égaux, i.e.,*

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

On note $\mathcal{R}^*$ la fermeture réflexive et transitive de $\mathcal{R}$. La relation $\mathcal{R}$ est *noetherienne* s'il n'existe pas de chaîne infinie $(G, \lambda_1)\ \mathcal{R}\ (G, \lambda_2)\ \mathcal{R} \ldots \mathcal{R}\ (G, \lambda_i)\ \mathcal{R} \ldots$.

Dans la suite de ce mémoire, on ne considère que des relations de réétiquetage récursives et cela pour considérer des modèles de calcul ayant une puissance de calcul raisonnable.

1.3.2 Relations de Réétiquetage sur les Arêtes

Une relation de réétiquetage est localement engendrée sur les arêtes si sa restriction aux arêtes détermine son comportement sur tout le graphe.

Définition 1.33 *Une relation de réétiquetage $\mathcal{R}$ est localement engendrée sur les arêtes si la condition suivante est satisfaite. Pour tous graphes (G, λ), (G, λ'), (H, η), (H, η') et toutes arêtes $e \in E(G)$ et $f \in E(H)$ telles que $ext(e) = \{v_1, v_2\}$ et $ext(f) = \{w_1, w_2\}$, si les trois conditions suivantes sont vérifiées :*

1. *$\lambda(v_1) = \eta(w_1)$, $\lambda(v_2) = \eta(w_2)$, $\lambda(e) = \eta(f)$, $\lambda'(v_1) = \eta'(w_1)$, $\lambda'(v_2) = \eta'(w_2)$ et $\lambda'(e) = \eta'(f)$,*
2. *pour tout sommet $v \in V(G)$ différent de v_1 et de v_2, $\lambda(v) = \lambda'(v)$ et pour toute arête $e' \in E(G)$ différente de e, $\lambda(e') = \lambda'(e')$,*
3. *pour tout sommet $w \in V(H)$ différent de w_1 et de w_2, $\lambda(w) = \lambda'(w)$ et pour toute arête $f' \in E(H)$ différente de f, $\lambda(f') = \lambda'(f')$,*

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

Étant donné une relation de réétiquetage $\mathcal{R}$, un *pas de calcul* sur le graphe $\mathbf{G}$ est la modification de l'étiquetage d'une arête de $\mathbf{G}$ pour obtenir un graphe $\mathbf{G'}$ tel que $\mathbf{G}\ \mathcal{R}\ \mathbf{G'}$. Une *exécution* de $\mathcal{R}$ sur $\mathbf{G}$ est alors une suite $\mathbf{G} = \mathbf{G}_0\ \mathcal{R}\ \mathbf{G}_1\ \mathcal{R} \ldots \mathcal{R}\ \mathbf{G}_i\ \mathcal{R} \ldots$ Le graphe étiqueté $\mathbf{G}_i$ est la *configuration* du graphe à l'étape i de l'exécution. Une configuration $\mathbf{G}$ est *finale* s'il n'existe aucun $\mathbf{G'}$ tel que $\mathbf{G}\ \mathcal{R}\ \mathbf{G'}$. On remarque que si $\mathcal{R}$ est noethérienne, il n'existe pas d'exécution infinie de $\mathcal{R}$ sur $\mathbf{G}$ et toute exécution atteint donc une configuration finale.

Une relation de réétiquetage localement engendrée sur les arêtes peut être décrite par un ensemble récursif de règles de la forme présentées sur la figure 1.4. Réciproquement, un tel ensemble de règles induit une relation de réétiquetage localement engendrée sur les

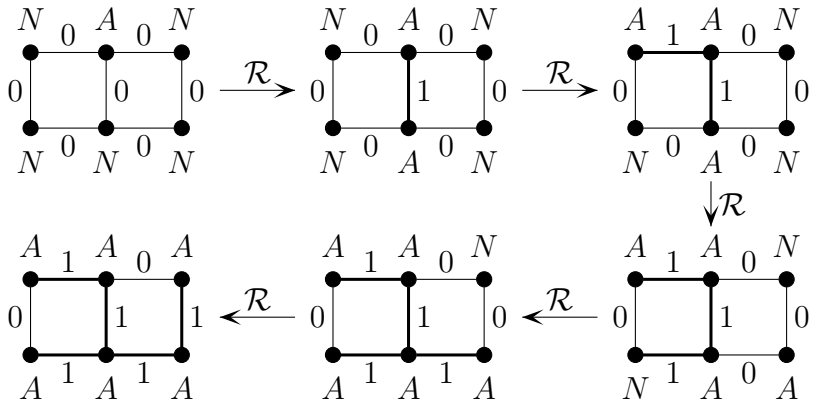

FIGURE 1.5 – Une exécution de l'algorithme décrit par la règle de la figure 1.4.

arêtes. Ainsi, on notera $\mathcal{R}$ l'ensemble de règles de réétiquetage aussi bien que la relation de réétiquetage correspondante.

Exemple 1.34 *On présente sur la figure 1.5, une exécution de l'algorithme décrit par la règle de réétiquetage présentée sur la figure 1.4. Pour tout graphe connexe* $\mathbf{G}$ *dans lequel il y a un unique sommet étiqueté* A *et où tous les autres sommets sont étiquetés* N*, toute exécution de cet algorithme sur* $\mathbf{G}$ *permet d'atteindre une configuration finale* $\mathbf{G}'$ *où l'ensemble des arêtes étiquetées* 1 *définit un arbre couvrant de* $\mathbf{G}'$*.*

Remarque 1.35 *On sait que toute exécution de l'algorithme défini par la règle de la figure 1.4 termine sur tout graphe* $\mathbf{G}$ *où un seul sommet est étiqueté* A *et où tous les autres sommets sont étiquetés* N*. Cependant, on remarque que dans la configuration finale, aucun sommet ne peut déduire à partir de son état que l'exécution est terminée : la terminaison de l'algorithme est dite* implicite*.*

Lorsqu'un sommet peut détecter à partir de son état que le résultat global de l'exécution a été calculé, on parle de terminaison explicite*. On reviendra sur ces différentes notions de terminaison dans la Section 1.5.*

1.3.3 Relations de Réétiquetage sur les Étoiles

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

Définition 1.36 *Une relation de réétiquetage* $\mathcal{R}$ *est localement engendrée sur les étoiles si la condition suivante est satisfaite. Pour tous graphes* (G,λ)*,* (G,λ')*,* (H,η)*,* (H,η')*, pour tous sommets* $v \in V(G)$ *et* $w \in V(H)$ *tels qu'il existe un isomorphisme* $\varphi : B_G(v) \to B_G(w)$*, si les conditions suivantes sont vérifiées :*

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

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

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

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

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

Par exemple, on considère l'algorithme décrit par les deux règles $\mathcal{E}_1$ et $\mathcal{E}_2$ présentées ci-dessous. Cet algorithme est un algorithme d'élection pour les arbres où tous les sommet sont initialement étiquetés A.

$\mathcal{E}_1$: **Règle d'Élagage**

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

Réétiquetage :
- $\lambda'(v_0) :=$ NON-ÉLU.

$\mathcal{E}_2$: **Règle d'Élection**

Précondition :
- $\lambda(v_0) = A$,
- $\nexists v \in V(B(v_0)) \setminus \{v_0\}$ tel que $\lambda(v) = A$.

Réétiquetage :
- $\lambda'(v_0) :=$ ÉLU.

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

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

Une exécution de cet algorithme est présentée sur la figure 1.6. On va montrer que pour tout arbre $\mathbf{T} = (T, \lambda)$ où tous les sommets sont étiquetés A (i.e., $\forall v \in V(T), \lambda(v) = A$),

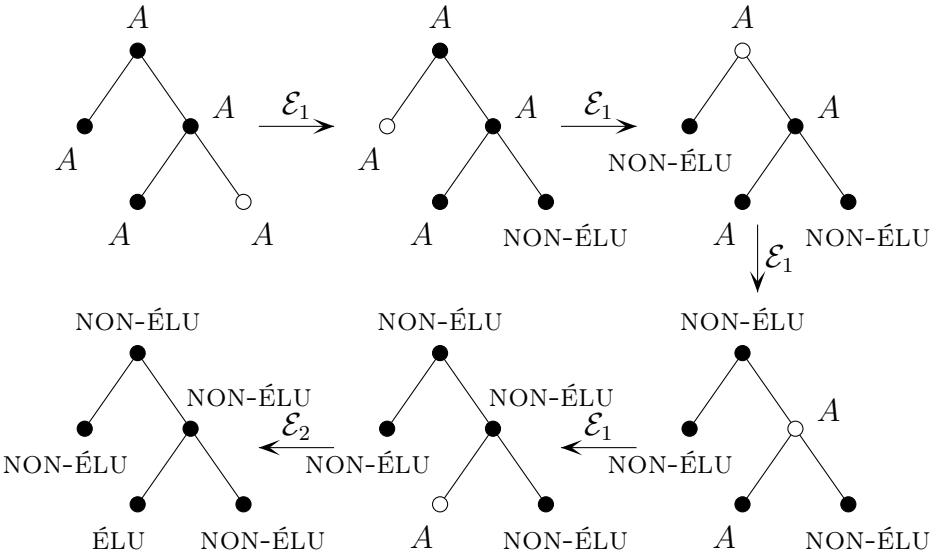

FIGURE 1.6 – Une exécution de l'algorithme décrit par les règles $\mathcal{E}_1$ et $\mathcal{E}_2$. Pour chaque configuration, le centre de l'étoile sur laquelle est appliquée la règle $\mathcal{E}_i$ pour passer à la configuration suivante est le sommet blanc.

toute exécution de l'algorithme décrit par les règles $\mathcal{E}_1$ et $\mathcal{E}_2$ permet de résoudre l'élection dans $\mathbf{T}$.

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

On observe qu'à chaque application d'une des deux règles, le nombre de sommets qui n'ont pas d'étiquettes finales diminue strictement. On est donc assuré que toute exécution de l'algorithme termine. On montre dans le lemme suivant un invariant qui permet de prouver la correction de l'algorithme.

Lemme 1.37 *Pour toute étape i, le graphe induit par l'ensemble des sommets étiquetés A est un arbre et s'il existe un sommet étiqueté A alors aucun sommet n'a l'étiquette* ÉLU.

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

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

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

On considère la configuration finale d'une exécution de l'algorithme sur **T**. S'il existe encore des sommets étiquetés A, alors d'après le lemme 1.37, le graphe induit par les sommets étiquetés A est un arbre et ou bien, cet arbre est réduit à un sommet v et la règle $\mathcal{E}_2$ peut être appliquée sur l'étoile $B_T(v)$ de centre v, ou bien il existe un sommet v qui est une feuille dans cet arbre et la règle $\mathcal{T}_1$ peut être appliquée sur l'étoile $B_T(v)$ de centre v. Par conséquent, dans la configuration finale, tous les sommets ont l'étiquette ÉLU ou NON-ÉLU. De plus, puisqu'à chaque étape, l'étiquette d'un seul sommet est modifié, le dernier sommet qui a changé d'étiquette a nécessairement pris l'étiquette ÉLU, et d'après le lemme 1.37, les autres sommets ont l'étiquette NON-ÉLU.

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

1.3.4 Sérialisation

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

Plus généralement, deux suites de réétiquetage partant du même graphe étiqueté et telles qu'on peut obtenir l'une à partir de l'autre par ce type de commutation aboutiront au même résultat, c.-à-d., au même graphe étiqueté final. Ainsi, la notion d'exécution définie par une suite de réétiquetages peut être vue comme une *sérialisation* [Maz87] d'une exécution distribuée donnée.

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

1.3.5 ViSiDiA

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

Cet outil permet d'implémenter des algorithmes distribués dans différents modèles. Un algorithme est généralement implémenté dans un système asynchrone où les processus communiquent en échangeant des messages. Cependant, il est aussi possible d'implémenter des algorithmes codés par des relations de réétiquetage de graphes ; pour cela, les méthodes de synchronisation locale probabilistes présentées dans [MSZ02, MSZ03] sont utilisées. L'algorithme d'énumération présenté dans le chapitre 2 a en particulier été implémenté sous ViSiDiA.

Le chapitre 4 est consacré à la modification du logiciel ViSiDiA pour y ajouter des fonctionnalités permettant de deboguer les algorithmes développés au sein de ViSiDiA.

Pour d'autres approches sur l'outil ViSiDiA, on peut se référer au chapitre 5 de la thèse d'Afif Sellami [Sel04] et au chapitre 6 de la thèse Bilel Derbel [Der06].

1.4 Élection et Nommage

Le problème de l'*élection* est un problème fondamental en algorithmique distribuée qui a été pour la première fois étudié par Lelann [LeL77]. Un algorithme distribué permet de résoudre le problème de l'élection sur un graphe $\mathbf{G}$ si toute exécution de l'algorithme sur $\mathbf{G}$ termine et si dans la configuration finale, il existe exactement un sommet $v \in V(G)$ qui est dans l'état ÉLU, alors que tous les autres sommets de $\mathbf{G}$ sont dans l'état NON-ÉLU. On suppose par ailleurs, que les états ÉLU et NON-ÉLU sont *finaux*, c.-à-d., une fois qu'un sommet est dans l'un de ces états, alors son état n'est plus modifié jusqu'à la fin de l'exécution de l'algorithme. Le problème de l'élection est important en algorithmique distribuée puisqu'une fois qu'un sommet a été élu, ce sommet peut être utilisé pour centraliser ou diffuser de l'information, pour initialiser l'exécution d'un autre algorithme qui nécessite un sommet distingué (comme par exemple, l'algorithme de calcul d'un arbre couvrant présenté dans la Section 1.3.2), pour prendre une décision de manière centralisée, etc.

Le but d'un algorithme de *nommage* est d'arriver dans une configuration finale où un identifiant unique est associé à chaque sommet. Ce problème aussi est très important en algorithmique distribuée, puisque de nombreux algorithmes distribués fonctionnent correctement sous l'hypothèse que tous les sommets ont des identifiants uniques. Le problème de l'*énumération* est une variante du problème de nommage. Le but d'un algorithme d'énumération pour un graphe $\mathbf{G}$ est d'attribuer à chaque sommet de $\mathbf{G}$ un entier de telle sorte que dans la configuration finale, l'ensemble des numéros des sommets de $\mathbf{G}$ soit exactement $[1, |V(G)|]$. L'algorithme de Mazurkiewicz [Maz97a], dont plusieurs adaptations sont présentées dans le chapitre 2, est un algorithme d'énumération codé par une relation de réétiquetage localement engendrée sur les étoiles.

Les problèmes du nommage et de l'élection sont équivalents dans certains modèles [CM10b]. Il existe cependant des modèles où il existe strictement plus de graphes admettant un algorithme d'élection que de graphes admettant un algorithme de nommage ; c'est le cas pour les modèles étudiés dans le chapitre 2.

1.5 Terminaison Implicite et Explicite

On considère un algorithme distribué $\mathcal{A}$ et un graphe $\mathbf{G}$ tel que toute exécution de $\mathcal{A}$ sur $\mathbf{G}$ termine. On va expliciter les différents types de terminaison qu'on considère par la suite.

La terminaison est *implicite* si toute exécution de l'algorithme $\mathcal{A}$ termine, c.-à-d., aucun pas de calcul ne peut plus être effectué sur le graphe $\mathbf{G}$. Dans ce cas là, les sommets du graphe ne peuvent pas forcément détecter à partir de leurs états que l'exécution de l'algorithme est globalement terminée.

Cependant, si on dispose d'un algorithme de nommage $\mathcal{A}$ et qu'on veut utiliser les identités assignées à chaque sommet par $\mathcal{A}$ pour pouvoir exécuter un algorithme $\mathcal{A}'$ qui

nécessite que tous les sommets aient des identifiants distincts, il faut que les sommets puissent détecter qu'ils ont obtenus leurs numéros finaux afin de pouvoir les utiliser pour exécuter l'algorithme $\mathcal{A}'$.

En général, lorsqu'un algorithme distribué est exécuté sur un graphe $\mathbf{G}$, chaque sommet $v \in V(G)$ utilise des variables pour stocker des informations qui sont nécessaires à l'exécution de l'algorithme, mais qui ne font pas partie du « résultat » de l'algorithme (ni de ses spécifications). Dans les modèles qu'on considère, l'étiquette de chaque sommet représente son état et les valeurs de ces variables supplémentaires apparaissent dans son étiquette.

Ainsi, si un algorithme $\mathcal{A}$ utilisant un ensemble d'étiquettes L résout un problème $\mathcal{P}$ sur un graphe $\mathbf{G}$, on suppose qu'il existe un ensemble S d'étiquettes et une fonction $\mathbf{res} : L \to S$ tels que pour toute exécution ρ de $\mathcal{A}$ sur $\mathbf{G}$ termine et l'étiquetage final λ_ρ de G est tel que l'étiquetage $\mathbf{res} \circ \lambda_\rho$ est une solution de $\mathcal{P}$ sur $\mathbf{G}$.

Parfois, il n'est pas possible de détecter que les étiquettes de tous les sommets ne vont plus être modifiées, mais un sommet peut détecter que pour tout sommet v, $\mathbf{res}(v)$ ne sera plus modifiée dans la suite de l'exécution. Dans ce cas là, on dit qu'on peut détecter la terminaison de l'algorithme et on parle de terminaison *explicite*.

Dans les chapitres suivants, on va présenter des algorithmes qui permettent de résoudre des problèmes (avec détection de la terminaison) sur des familles de graphes, c.-à-d., l'algorithme permet de résoudre (avec détection de la terminaison) le problème sur tous les graphes de la famille. La définition suivante présente une définition formelle de la notion de *détection de la terminaison* étendue aux familles de graphes.

Définition 1.38 *Un algorithme $\mathcal{A}$ utilisant un ensemble d'étiquettes L permet de résoudre un problème $\mathcal{P}$ sur une famille de graphes $\mathcal{F}$ avec détection de la terminaison s'il existe un ensemble S, une fonction $\mathbf{res} : L \to S$ et un ensemble $L_f \subseteq L$ tels que les propriétés suivantes sont toujours vérifiées pour tout graphe $\mathbf{G} \in \mathcal{F}$.*

- *Toute exécution ρ de $\mathcal{A}$ sur $\mathbf{G}$ termine et l'étiquetage final λ_ρ de G est tel que l'étiquetage $\mathbf{res} \circ \lambda_\rho$ est une solution de $\mathcal{P}$ sur $\mathbf{G}$.*
- *Pour toute exécution ρ de $\mathcal{A}$ sur $\mathbf{G}$, il existe un sommet v et une étape i tels que $\lambda_i(v) \in L_f$ (où $\lambda_i(v)$ est l'étiquette de v après la $i$ème étape de l'exécution ρ).*
- *Pour toute exécution ρ de $\mathcal{A}$ sur $\mathbf{G}$, s'il existe un sommet $v \in V(G)$ et une étape i telle que $\lambda_i(v) \in L_f$, alors pour tout $i' > i$, $\mathbf{res} \circ \lambda_{i'} = \mathbf{res} \circ \lambda_i$.*

Remarque 1.39 *Si un algorithme $\mathcal{A}$ permet de résoudre un problème $\mathcal{P}$ sur une famille de graphes $\mathcal{F}$ avec détection de la terminaison, alors pour toute exécution de $\mathcal{A}$ sur un graphe $\mathbf{G} \in \mathcal{F}$, un sommet peut détecter que tous les sommets ont calculé leurs valeurs finales, mais il ne peut pas forcément détecter que l'exécution est terminée. On assure toutefois dans la définition 1.38 que toute exécution de $\mathcal{A}$ sur un graphe $\mathbf{G} \in \mathcal{F}$ termine. Par ailleurs, si on veut exécuter un second algorithme qui utilise le résultat du calcul de $\mathcal{A}$, un sommet peut commencer à exécuter le second algorithme une fois qu'il sait que tous les sommets ont calculé leurs valeurs finales dans l'exécution de $\mathcal{A}$.*

1.6 Connaissances Initiales

Dans le chapitre suivant, on va caractériser les graphes qui admettent des algorithmes d'élection ou de nommage dans un modèle particulier. On étudie, de plus, les problèmes du calcul de l'état global et de détection de propriétés stables pour lesquels nous déterminons quelles sont les connaissances sur le graphe que l'on doit disposer afin de pouvoir obtenir des algorithmes qui permettent de résoudre ces problèmes.

Exemple 1.40 *Si on sait qu'un graphe* $\mathbf{G}$ *est un arbre, alors il existe un algorithme codé par une relation de réétiquetage localement engendrée sur les étoiles qui permet de résoudre l'élection dans* $\mathbf{G}$ *: c'est l'algorithme présenté dans la Section 1.3.3.*

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

Dans [Sak99], Sakamoto étudie différents types de connaissances initiales globales et étudie quelles sont les informations qui peuvent être déduites à partir d'autres connaissances initiales. Dans les chapitres suivants, on va plus particulièrement étudier les connaissances globales suivantes et étudier ce qu'elles permettent de calculer :

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

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

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

ou qu'on peut résoudre $\mathcal{P}$ sur la famille des graphes disposant de la même connaissance initiale (les graphes de taille donnée, par exemple).

Chapitre 2

Énumération et Élection dans les Réseaux de Diffusions Multi-sauts Anonymes

Sommaire

Dans ce second chapitre, nous proposons de résoudre les problèmes de l'énumération et de l'élection dans les modèles de diffusions asynchrones et synchrones. Pour cela, nous introduisons ces modèles, les étapes de calculs associés et les outils combinatoires permettant de fournir une caractérisation complète.

2.1 Introduction

Un réseau ad hoc de diffusion multi-saut est une collection de processus qui communiquent en *broadcastant* des messages et qui doivent fonctionner en l'absence de toute infrastructure préexistante (p. ex., réseau sans fil ad hoc). Parmi les défis importants que nous retrouvons dans ce genre de réseau, les problèmes de l'énumération ou, plus généralement du nommage, ainsi que l'élection sont les plus connus dans le domaine des systèmes distribués [HR88, Maz97a, San06, Sto05, Tel00].

L'existence de processus identifiés dans un réseau peut ainsi permettre un meilleur routage des informations, une meilleure gestion des ressources et des performances [Sto05]. Toutefois, la complexité et les hypothèses initiales des solutions existantes sont, pour la plupart, non réalistes et font d'elles de mauvais prétendants pour une implémentation du monde réel. Nous sommes, ici, intéressés par la caractérisation des réseaux anonymes pour lesquels il existe un algorithme qui résout les problèmes de nommage et d'énumération ou qui résout le problème de l'élection tout en réduisant les complexités et hypothèses existantes.

2.1.1 Une Étude de Cas : Surveillance à l'Aide de Capteurs Sans Fil

Dans [Sto05], Stojmenovic propose une étude complète de la variété des modèles ainsi que des algorithmes qui ont été développés pour répondre à des problèmes importants dans le domaine des réseaux de capteurs. Comme exemples de problèmes, on peut citer notamment le nommage et l'élection. Notre contribution peut être utilisée pour répondre à ces questions dont la portée devient apparente dès lors qu'il est nécessaire de déployer un réseau collaboratif d'une manière spontanée.

Lorsque nous considérons des situations de catastrophes naturelles telles que les tremblements de terre ou les inondations, de vastes zones, inaccessibles pour la plupart, doivent être couvertes avec un nombre considérable de capteurs afin d'assister les secours ou les scientifiques. Cette surveillance impose la connaissance du nombre de capteurs et de désigner une entité particulière qui se charge de gérer cette collection de capteurs. Le caractère spontané de ce type de réseau rend la tâche d'identification de chaque capteur difficile.

De plus, puisque dans cette situation, les capteurs ne peuvent pas s'appuyer sur le support d'une architecture existante et centralisée – la catastrophe peut avoir détruit les serveurs DHCP, par exemple – attribuer une identité (adresse) à chaque capteur devient fastidieux. Accomplir cette procédure manuellement est quasiment impossible dans cette configuration.

Nous proposons ici une approche totalement distribuée pour ce type de situation dont la seule hypothèse porte sur le nombre de capteurs déployés. Il est à noter que les capteurs communiquent entre eux en *full-duplex* et que chacun des capteurs n'est pas informé du nombre de voisins qu'il possède (et n'a pas les moyens de le calculer). Ainsi, le réseau est initialement qualifié comme *anonyme* en ce sens qu'aucun capteur n'a une identité distincte des autres. Par conséquent, les capteurs doivent exécuter le même algorithme. Cet algorithme attribue ensuite une identité unique à chaque capteur pour interagir sur le réseau. À la fin de cette phase, lorsque le réseau n'est plus anonyme, les secours peuvent progressivement collecter les données de surveillance depuis chaque capteur, un par un. Cependant, en raison des contraintes inhérentes des réseaux sans fil telles que la consommation d'énergie, la faible capacité de calcul des capteurs et les configurations atypiques de ce réseau comme sa position géographique, il est beaucoup plus intéressant de déterminer dynamiquement un capteur particulier, le *leader*, qui fera la passerelle entre le réseau et les secours. Une élection de leader peut être réalisée en s'appuyant sur un algorithme basé sur les identités de chaque capteur : il suffit d'élire le capteur avec l'identité la plus élevée (resp. la moins élevée). Néanmoins, la propriété asynchrone des communications dans un réseau sans fil rend la détection de la terminaison difficile (p. ex., plus d'un capteur élu). Nous prenons cette configuration en compte en proposant un algorithme d'élection qui garantit que dans le réseau entier le même leader est reconnu comme tel.

2.1.2 Le Modèle

Nous considérons un modèle de diffusion asynchrone aussi présenté dans [CM98, Chl01]. Un réseau est représenté par un graphe simple connecté $G = (V(G), E(G)) = (V, E)$ où les sommets correspondent aux processus et les arêtes aux liens de communication directs. L'état de chaque processus est représenté par une étiquette $\lambda(v)$ associée au sommet $v \in V(G)$ correspondant ; nous dénotons par $\mathbf{G} = (G, \lambda)$ un tel graphe étiqueté.

Remarque 2.1 *Les étiquettes (états) sont attachées aux sommets. Elles permettent de coder différentes situations. Si le réseau est* anonyme *alors tous les sommets du graphe ont la même étiquette ; les sommets ayant des identités distinctes, un sommet distingué ou toute autre configuration intermédiaire, qualifiée comme* partiellement anonyme*, sont d'autres exemples d'étiquettes associées aux sommets.*

Nous considérons un modèle robuste dans lequel les graphes sont partiellement anonymes, c.-à-d., les processus ont des identités qui ne sont pas nécessairement distinctes. La question de l'anonymat n'est pas nouvelle et est souvent considérée quand les processus ne doivent pas divulguer leurs identités pendant les exécutions par souci de vie privée ou de politique de sécurité [GR05]. De plus, les processus peuvent avoir été fabriqués à grande échelle et il est pratiquement infaisable de garantir l'unicité des identités attribuées. Par

conséquent, chaque processus exécute le même algorithme de la même manière, peu importe son identité (voir [Ang80, AAER07] pour des travaux connexes sur l'anonymat des systèmes distribués).

Les messages émis par un processus sont simplement entendus par les processus à portée dans son voisinage immédiat. Nous considérons les réseaux ad hoc qui fonctionnent en l'absence de toute infrastructure et qui s'appuient sur le système à base de passage de messages et de communications de diffusions asynchrones : les processus n'ont pas accès à une horloge globale et exécutent les étapes de calculs (émissions, écoutes, calculs internes atomiques) en un temps arbitraire, mais fini. Les liens de communication sont fiables, mais asynchrones, c.-à-d., un message émis depuis un processus à l'ensemble de ses voisins arrive en un temps fini, mais non prédictible. Il faut noter que les communications ne sont pas nécessairement FIFO (First In First Out), c.-à-d., les messages peuvent arriver dans un ordre différent que celui dans lequel ils ont été envoyés.

2.1.3 Résultats

Dans ce chapitre nous donnons une caractérisation complète des réseaux de diffusion multi-sauts pour lesquels il existe un algorithme d'énumération ou un algorithme d'élection (théorème 2.17 et théorème 2.28). Dans ce modèle, les problèmes de l'énumération et de l'élection ne sont pas équivalents, ce qui signifie que même s'il est possible d'élire, il n'est pas toujours possible de donner un identifiant unique à tous les processus.

Soit $\mathbf{G} = (G, \lambda)$, un graphe simple étiqueté. Nous rappelons que nous notons par $Dir(\mathbf{G})$ le graphe dirigé étiqueté symétrique $(Dir(G), \lambda)$ construit de la façon suivante. Les sommets de $Dir(G)$ sont les sommets de G et ils ont la même étiquette aussi bien dans $\mathbf{G}$ que dans $Dir(\mathbf{G})$. Chaque arête $\{u, v\}$ de G est remplacée dans $Dir(\mathbf{G})$ par deux arcs $a_{(u,v)}, a_{(v,u)} \in A(Dir(G))$ tels que $s(a_{(u,v)}) = t(a_{(v,u)}) = u$, $t(a_{(u,v)}) = s(a_{(v,u)}) = v$. Noter que ce graphe dirigé ne contient pas d'arcs multiples ou de boucles. L'objet combinatoire que nous utilisons pour notre étude est $(Dir(G), \lambda)$ et les résultats sont prouvés avec des graphes dirigés étiquetés symétriques.

Une fibration d'un digraphe $\mathbf{D}$ vers un digraphe $\mathbf{D}'$ est un homomorphisme de $\mathbf{D}$ vers $\mathbf{D}'$ qui induit un isomorphisme entre les arcs entrants de chaque sommet de D et les arcs entrants de son image.

D'abord, nous prouvons que, dans le modèle de diffusion asynchrone, il existe un algorithme d'énumération si et seulement si $Dir(\mathbf{G})$ est minimal pour les fibrations, c.-à-d., s'il existe une fibration de $Dir(\mathbf{G})$ vers $\mathbf{D}'$ alors c'est un isomorphisme.

Pour le problème de l'élection, nous prouvons qu'il existe un algorithme d'élection si et seulement si dès lors qu'il existe une fibration φ de $Dir(\mathbf{G})$ vers $\mathbf{D}'$ alors nécessairement il existe un sommet v de D' tel que $\varphi^{-1}(v)$ est un singleton.

Pour les deux problèmes, nos algorithmes ne nécessitent pas que chaque processus connaisse son degré. Pour le problème de l'énumération, les processus ne connaissent que la taille du réseau. Cependant, nous montrons que cette connaissance initiale n'est pas suffisante dès que nous considérons le problème de l'élection. Ainsi, notre algorithme d'élection suppose que chaque processus connaisse une carte du réseau sans, toutefois, être informé de sa position dans cette carte.

De plus, nos algorithmes ont une complexité polynomiale : mémoire locale, nombre de messages échangés et taille des messages sont polynomialement bornés par la taille du réseau.

Remarque 2.2 (Connaissance Initiale) *Pour l'algorithme d'énumération, il suffit que chaque processus connaisse la taille du réseau pour détecter la terminaison de l'algorithme. Cette hypothèse est classique lorsque l'on considère les problèmes de nommage et d'élection [Ang80, BCG+96, YK96b, YK99, Maz97a].*

Concernant l'algorithme d'élection, afin de détecter la terminaison, nous supposons que chaque processus connaît une carte du graphe complet (voir la section 2.4.2 pour une discussion plus complète) ; nous prouvons aussi qu'il suffit que chaque processus connaisse la taille du graphe et la taille de son voisinage (voir la section 2.4.5)

Les résultats présentés dans ce chapitre ont été obtenus en collaboration avec Jérémie Chalopin et Yves Métivier et une partie de ces résultats a été publiée dans [CMM12a]. Nous rappelons que l'algorithme d'énumération 1 a été implémenté sous ViSiDiA.

2.1.4 État de l'Art

Les graphes pour lesquels les problèmes de l'élection ou de nommage sont solvables ont déjà été étudiés sous différents modèles. Les solutions proposées dépendent du modèle de calcul considéré, de la topologie du réseau ou encore de la connaissance initiale.

Étude des Modèles Existants

Lorsque nous considérons les réseaux de capteurs sans fil, *Wireless Sensor Networks* (WSN), et plus généralement les réseaux ad hoc sans fil, une multitude de modèles différents existe sous différentes hypothèses : détection ou non de collisions (CD/no-CD). Les communications sans fil sont, par essence, non fiables : lorsque plus de deux processus tentent de communiquer avec un autre processus, il y a risque de collision et aucun message n'arrive à destination. Dans le cas des réseaux filaires (p. ex., réseau Ethernet), la détection de collision est rendue possible par l'utilisation de protocoles situés dans la sous-couche MAC (*Medium Access Control*) de la couche Liaison du modèle OSI. Parmi les protocoles les plus utilisés, on retrouve le protocole CSMA/CD [Tan02] (Chap. 4). Pour les réseaux sans fil dans lesquels les processus ne partagent pas le même support de communication, il n'est pas toujours simple de détecter les collisions ou de les différencier du bruit et des interférences. Gallager [Gal85] présente une étude des différentes approches possibles pour la résolution des problèmes de collisions. Il souligne que les problèmes de collisions ne se situent pas exclusivement au niveau des interférences, du bruit ou de l'arrivée des messages, mais provient aussi de la façon dont est codée l'information qui transite à travers le canal ou de la façon dont sont modélisés les canaux de transmission. En conséquence, l'utilisation d'un mécanisme de détection de collision n'est pas toujours justifiée, en particulier pour les réseaux sans fil dans lesquels une collision est très difficilement distinguable du bruit ambiant. Un algorithme n'utilisant pas de mécanismes de détection de collision est, par définition, plus fiable qu'un algorithme en utilisant [BYGI92]. Toutefois, la bonne réception des messages envoyés peut être émulée par l'utilisation d'un protocole de niveau

OSI inférieur probabiliste [BYGI91, BYGI92]. Pour notre caractérisation, nous proposons des algorithmes déterministes appartenant aux couches supérieures. Le modèle que nous utilisons dans ce chapitre est dérivé du modèle proposé par Cidon et Mokryn [CM98] qui inclue les environnements de diffusions multi-sauts tels que les réseaux radio [Sto05], les réseaux utilisant plusieurs fréquences de transmission [MR83], ou encore les réseaux meshs [AWW05]. Les messages émis par un processus à ses voisins arrivent en un temps fini, mais arbitraire. Ce délai de temps dépend essentiellement des protocoles utilisés pour assurer la bonne réception (p. ex., absence de collisions, d'interférences). Toutefois, nous rappelons que notre modèle ne requiert pas que les communications respectent l'ordre des messages, c.-à-d., les communications ne sont pas FIFO (les messages peuvent se doubler). Nous supposons que les problèmes liés à l'accès au support de communication sont résolus par des protocoles largement utilisés tels que FDMA ou CSMA/CA (IEEE 802.11).

Travaux Liés

Angluin [Ang80] a présenté les techniques de preuves classiques utilisées pour montrer la non-existence d'un algorithme d'élection basée sur la notion de revêtements de graphes, qui est une notion provenant de la topologie algébrique [Mas91]. Jusqu'ici, plusieurs caractérisations des graphes pour lesquels il existe un algorithme d'élection ont été obtenues [BCG$^+$96, YK96b, YK99, Maz97a].

Le modèle étudié dans ce chapitre correspond au modèle de communication *diffusion-à-boîte* proposé par Yamashita et Kameda [YK99] et au modèle sans étiquetage des ports entrants/sortants défini par Boldi *et al.* [BCG$^+$96]. Nous utilisons les fibrations introduites pour la première fois dans [BCG$^+$96] comme outil combinatoire pour l'algorithmique distribuée et étudiées dans [BV02a]. Les fibrations, à l'instar des revêtements, proviennent de la topologie algébrique. Toutefois, l'outil fondamental utilisé dans [YK99, BCG$^+$96] est la notion de vues. La vue depuis un sommet v d'un graphe étiqueté (G, λ) est un arbre étiqueté infini enraciné en v et obtenu en considérant tous les chemins étiquetés dans (G, λ) et commençant depuis v.

La caractérisation des graphes où l'élection est possible, obtenue dans [YK99], est formulée en utilisant les vues tandis que dans le travail de Boldi *et al.*, ce sont les fibrations qui sont utilisées. Dans les deux cas, les algorithmes d'élection sont basés sur les vues et les algorithmes génèrent des messages de taille exponentielle. Ils obligent chaque sommet à connaître la taille du graphe ainsi que la taille de son voisinage ; cette connaissance est utilisée par les algorithmes et contraint toutes les exécutions à être pseudo-synchrones et les liens de communication à être FIFO.

Les techniques utilisées dans ce chapitre sont inspirées du travail de Mazurkiewicz [Maz97a]. Il considère le modèle de calcul asynchrone dans lequel en une étape de calcul l'étiquette des sommets est modifiée dans un sous-graphe constitué du sommet lui-même et de ses voisins, suivant des règles dépendantes de ce sous-graphe seulement. La caractérisation des graphes où il existe une solution pour les problèmes de nommage ou d'élection donnée par Mazurkiewcz est basée sur la notion de graphes non ambigus et peut être formulée de façon équivalente en utilisant les revêtements de graphes simples (voir [GMM04], p. 256). Un graphe G est un revêtement d'un autre graphe G' s'il y a un homomorphisme surjectif φ de G vers G' qui est localement bijectif. Il propose un algorithme d'énuméra-

tion simple et élégant pour les graphes qui sont minimaux pour les revêtements, c.-à-d., qui ne peuvent revêtir qu'eux-mêmes. L'outil fondamental est un ordre total associé aux vues locales composées du sommet et de son voisinage.

Ces techniques ont aussi été utilisées dans [CM07a, CM10a]. Le modèle de [CM07a] (qui est le même que celui de [YK96b]) est tel que durant chaque étape, un des sommets, en fonction de son étiquette courante, soit change son état, soit reçoit ou envoie un message par l'intermédiaire d'un de ses ports. Le modèle de [CM10a] est défini par l'utilisation de calculs locaux sur les arêtes étiquetées de graphes. Dans les deux cas, les problèmes de l'élection et de l'énumération sont équivalents.

Cidon et Mokryn présentent dans [CM98] un algorithme d'élection dans les réseaux radio multi-sauts. Cet algorithme partitionne le réseau en fragments de collections de processus où un seul processus est identifié comme un candidat et marqué initialement comme actif. Ils considèrent des réseaux qui ne sont pas anonymes : chaque sommet a un identifiant unique. Durant le calcul, un candidat peut devenir inactif et rejoindre le fragment d'un autre candidat et ainsi de suite jusqu'à la présence d'un seul candidat.

2.1.5 Résumé du Chapitre

Tout d'abord, nous présentons dans la section 2.2 les outils combinatoires utilisés dans notre caractérisation. Puis dans la section 2.3, nous montrons les conditions nécessaires et suffisantes pour caractériser les graphes qui admettent un algorithme d'énumération (ou de nommage). Nous montrons la complexité de la solution obtenue. Dans la section 2.4 , nous étudions le cas du problème de l'élection pour lequel nous soulignons l'importance liée à la connaissance initiale. Nous montrons ainsi que l'élection et l'énumération sont deux problèmes qui ne sont pas équivalents dans notre modèle. Enfin, nous terminons par la section 2.5 dans laquelle nous montrons l'extension de notre modèle asynchrone au modèle synchrone.

2.2 Homomorphismes et Fibrations

Afin de décrire nos caractérisations, nous avons besoin de considérer les graphes dirigés pouvant avoir des arcs multiples ou des boucles. Dans cette section, nous présentons diverses définitions à propos des fibrations qui sont une forme particulière d'homomorphismes de graphes. À la suite de ces définitions, nous donnons le lemme fondamental qui fait le lien entre les fibrations et le modèle de communication que nous utilisons dans ce chapitre.

2.2.1 Fibrations de Graphes

Fibrations, t-fibrations et nt-fibrations sont des outils combinatoires importants pour ce travail (voir [Bod89, BV02a] pour quelques définitions et propriétés).

Intuitivement, une fibration est un homomorphisme de graphes dirigés qui préservent le voisinage sortant de chaque sommet. Plus précisément, une fibration est un homomorphisme qui induit un isomorphisme entre les arcs entrants d'un sommet et les arcs entrants

de son image.

Définition 2.3 *Un graphe dirigé D est* fibré *sur un graphe dirigé D' à travers un homomorphisme $\varphi : D \to D'$ si pour tout arc $a' \in A(D')$ et pour tout sommet $v \in \varphi^{-1}(t(a'))$, il existe un unique arc $a \in A(D)$ tel que $t(a) = v$ et $\varphi(a) = a'$; Cet arc a est appelé le* relèvement *de a' en v.*

On dit alors que l'homomorphisme φ est une fibration *de D dans D'. Le digraphe D est le* graphe total *de φ et le digraphe D' est la* base *de φ.*

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

Le digraphe $\mathbf{D}$ est minimal *si pour tout digraphe $\mathbf{D'}$ tel que $\mathbf{D}$ est fibré sur $\mathbf{D'}$, $\mathbf{D}$ et $\mathbf{D'}$ sont isomorphes.*

Remarque 2.4 *A contrario, un isomorphisme entre les arcs sortants de chaque sommet et les arcs sortants de leurs images est appelé* opfibration.

Si un digraphe $\mathbf{D}$ est *fibré* sur un digraphe $\mathbf{D'}$ via un homomorphisme φ, et si $\mathbf{D}$ et $\mathbf{D'}$ ne sont pas isomorphes, on dit que $\mathbf{D}$ est *proprement* fibré sur $\mathbf{D'}$ et que φ est une fibration *propre*.

D'après [BV02a], nous savons qu'il existe un unique digraphe $\mathbf{B_G}$ tel que $Dir(\mathbf{G})$ est fibré sur $\mathbf{B_G}$, et pour tout $\mathbf{D}$ tel que $Dir(\mathbf{G})$ est fibré sur $\mathbf{D}$, $\mathbf{D}$ est fibré sur $\mathbf{B_G}$. Ce digraphe est appelé la *base minimale* de $\mathbf{D}$.

Dans ce travail, nous utilisons les t-fibrations et les nt-fibrations.

Définition 2.5 *La fibre d'un sommet v est qualifiée de* triviale *si $|\varphi^{-1}(v)| = 1$, sinon, elle est* non-triviale.

Une fibration φ est une t-fibration s'il existe au moins un sommet dont la fibre est triviale ; c'est une nt-fibration si toutes les fibres sont non-triviales.

Un digraphe D est t-fibré (resp. nt-fibré) sur un digraphe D' via φ si et seulement si φ est une t-fibration (resp. nt-fibration).

Le digraphe $\mathbf{D}$ est nt-minimal si pour tout digraphe $\mathbf{D'}$ tel que $\mathbf{D}$ est fibré sur $\mathbf{D'}$ via une fibration φ, φ est une t-fibration.

Pour la section 2.5, nous définissons les graphes minimaux pour les fibrations discrètes. Les fibrations discrètes sont les fibrations sans boucle :

Définition 2.6 *Une fibration φ d'un graphe dirigé étiqueté $\mathbf{D}$ vers un graphe dirigé étiqueté $\mathbf{D'}$ est* discrète *si $\mathbf{D'}$ ne contient pas de boucle.*

Un graphe dirigé fortement connexe sans boucle $\mathbf{D}$ est minimal pour les fibrations discrètes *si $\mathbf{D}$ n'est fibré proprement sur aucun autre graphe dirigé sans boucle fortement connexe.*

Un graphe simple $\mathbf{G}$ est minimal si $Dir(G)$ est minimal. Pareillement, un graphe simple $\mathbf{G}$ est nt-minimal si $Dir(\mathbf{G})$ est nt-minimal. Des exemples de fibrations sont donnés dans la figure 2.1.

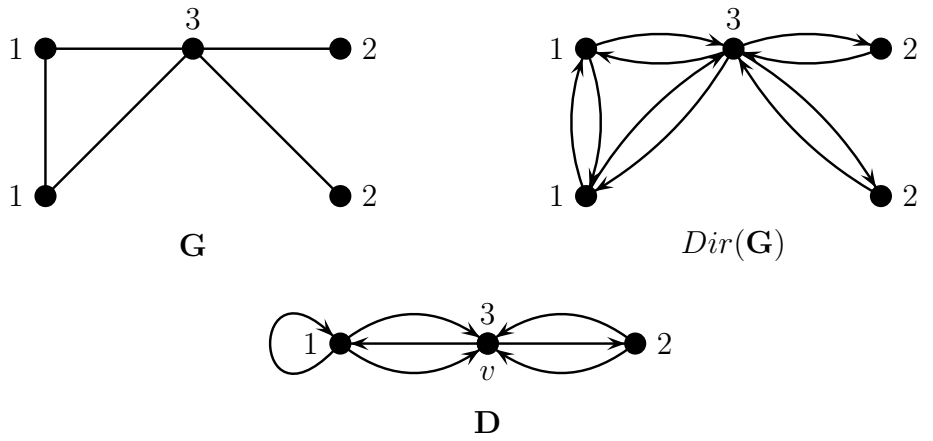

FIGURE 2.1 – Le digraphe étiqueté $Dir(\mathbf{G})$ est fibré sur le digraphe $\mathbf{D}$. Par conséquent, $Dir(\mathbf{G})$ n'est pas minimal. Puisque $Dir(\mathbf{G})$ a un unique sommet de degré 4, $Dir(\mathbf{G})$ est nt-minimal. Le digraphe $\mathbf{D}$ est minimal et aussi nt-minimal.

Remarque 2.7 *Comme conséquence de la définition 1.31, nous obtenons : soit* $\mathbf{H}$ *une sous-digraphe de* $Dir(\mathbf{G})$*, pour tout sommet* $v \in Dir(\mathbf{G})$*,* $T^d_{Dir(\mathbf{G})}(v) \succeq T^d_{\mathbf{H}}(v)$*.*

De plus, soient $\mathbf{D}$ *et* $\mathbf{D}'$ *deux digraphes. Si* $\mathbf{D}$ *est fibré sur* $\mathbf{D}'$ *via* φ*, alors* $T_{\mathbf{D}}(v) \approx T_{\mathbf{D}'}(\varphi(v))$*, i.e, la vue de* v *dans* $\mathbf{D}$ *est isomorphe à la vue de* $\varphi(v)$ *dans* $\mathbf{D}'$*.*

Noter que les sommets de la base minimale $\mathbf{B}$ de $\mathbf{D}$ peuvent être identifiés par leurs vues dans $\mathbf{B}$: cela définit un homomorphisme unique de $\mathbf{G}$ dans $\mathbf{B}$. En ce sens, nous définissons la notion de candidat pour un digraphe $\mathbf{D}$ tel que $Dir(\mathbf{G})$ est fibré sur $\mathbf{D}$.

Définition 2.8 *On considère un graphe* nt*-minimal* $\mathbf{G}$*, soit* $\mathbf{B}$ *la base minimale de* $Dir(\mathbf{G})$ *et soit* φ *l'unique fibration de* $Dir(\mathbf{G})$ *dans* $\mathbf{B}$*. Un sommet* $v \in V(B)$ *est un candidat de* $\mathbf{B}$ *si* $|\varphi^{-1}(v)| = 1$*, i.e., si il existe un unique sommet* $w \in V(G)$ *tel que* $\mathbf{T}_{\mathbf{G}}(w) \approx \mathbf{T}_{\mathbf{B}}(v)$*.*

Étant donné un digraphe $\mathbf{D}$ *tel que* $Dir(\mathbf{G})$ *est fibré sur* $\mathbf{D}$*, on sait que* $\mathbf{D}$ *est fibré sur* $\mathbf{B}$ *via un unique homomorphisme* φ'*. Un sommet* v *est un candidat de* $\mathbf{D}$ *si et seulement si* $\varphi'(v)$ *est un candidat de* $\mathbf{B}$*.*

On note par $C_{\mathbf{G},\mathbf{D}}$ *l'ensemble des candidats de* $\mathbf{D}$*.*

Nous remarquons que si un digraphe $Dir(\mathbf{G})$ nt-minimal est fibré sur un digraphe $\mathbf{D}$ via un homomorphisme φ, alors pour tout sommet $v \in C_{\mathbf{G},\mathbf{D}}$, $|\varphi^{-1}(v)| = 1$.

Intuitivement, un algorithme d'élection pour un graphe $\mathbf{G}$ fibré sur $\mathbf{D}$ ne pourra pas déclarer comme étant un élu un sommet n'appartenant pas à l'ensemble des candidats $C_{\mathbf{G},\mathbf{D}}$ (voir la section 2.4).

2.2.2 Fibrations et Communications par Diffusion

Afin d'étendre le lemme de relèvement proposé par Angluin [Ang80] et Boldi *et al.* [BCG+96] aux communications par diffusions asynchrones, nous présentons la corrélation qu'il existe entre les fibrations et ces communications.

Les problèmes d'élection et d'énumération imposent au réseau d'atteindre une configuration finale *non symétrique*. L'état d'un réseau est qualifié comme étant symétrique

s'il contient plusieurs processus qui sont exactement dans la même situation ; seulement leurs états locaux, mais aussi les états de leurs voisins, des voisins de leurs voisins, etc. C'est-à-dire qu'il existe une "similarité locale" entre les processus sur un rayon de distance infini.

L'argument de répétition montre que différents processus qui sont dans un état localement similaire sur un rayon de distance infinie exhiberont les mêmes comportements en un certain calcul infini. Ainsi, il n'y a pas d'algorithme qui garantit la suppression de la symétrie dans tout calcul fini.

Il n'est pas difficile de voir que les similarités locales sur un rayon de distance infini peuvent exister dans la famille des graphes finis. C'est précisément capturé par la notion de revêtements de graphes utilisée par Angluin et cela constitue l'outil mathématique pour prouver l'existence de symétries sur un rayon de distance infinie.

Dans notre modèle, lorsqu'un processus émet un message, il modifie son état en fonction de son état précédent seulement. Tandis que l'ensemble de ses voisins qui reçoivent ce message change leurs états en fonction de leur état précédent, mais aussi en fonction de l'état du processus depuis lequel est envoyé le message.

Ainsi, les réseaux de diffusion multi-sauts dans lesquels il existe des symétries sont précisément les réseaux qui ne sont pas minimaux et l'impossibilité de briser la symétrie peut être montrée pour ces graphes. Le lemme suivant établit le lien entre les fibrations et les étapes des communications par diffusions asynchrones.

Une exécution maximale ρ d'un algorithme est soit une exécution infinie (c.-à-d., qui ne termine jamais), soit une exécution finie telle que dans la configuration finale, il n'y a pas de message en transit et aucun processus ne souhaite émettre un message.

Lemme 2.9 (Lemme de Relèvement Asynchrone) *Soit $\mathbf{D}_1$ un digraphe fibré sur un digraphe $\mathbf{D}_2$ via une fibration φ. Considérons $\mathcal{A}$, un algorithme basé sur le modèle de diffusion asynchrone. S'il existe une exécution maximale ρ_2 de $\mathcal{A}$ sur $\mathbf{D}_2$ qui donne comme résultat $\mathbf{D}'_2$ alors il existe une exécution maximale ρ_1 de $\mathcal{A}$ sur $\mathbf{D}_1$ qui donne comme résultat $\mathbf{D}'_1$ de telle sorte que $\mathbf{D}'_1$ est fibré sur $\mathbf{D}'_2$ via φ.*

Preuve : Soient $\mathbf{D}_1 = (D_1, \lambda_1), \mathbf{D}_2 = (D_2, \lambda_1)$, deux digraphes tels que (D_1, λ_1) est fibré sur (D_2, λ_2) via φ.

Considérons un ensemble d'exécutions particulières Π sur $\mathbf{D}_2$ dans lequel les messages émis depuis un processus v sont suivis par la réception de ces mêmes messages par tous ses voisins immédiats. Soit une étape $\rho \in \Pi$: le processus v émet un message dans $\mathbf{D}_2$ et tous ses voisins l'entendent aussitôt après sont émission. Soit λ'_2 le nouvel étiquetage de $\mathbf{D}_2$ après cette étape. On peut relever cette exécution sur $\mathbf{D}_1$ dans laquelle tout sommet dans $\varphi^{-1}(v)$ émet le même message (par forcément simultanément et dans un ordre quelconque). Alors, tous les messages émis sont entendus. Notons λ'_1, le nouvel étiquetage de $\mathbf{D}_1$. Chaque sommet $w \in N_{\mathbf{D}_2}(v)$ entend k messages, avec k dépendant du nombre d'arcs $a \in A(\mathbf{D}_2)$ tels que $s(a) = v$ et $t(a) = w$. Puisque φ est une fibration, pour tout sommet $w' \in \varphi^{-1}(w)$, w' a k processus voisins dans $\varphi^{-1}(v)$ et entend donc k messages identiques. Par conséquent, $\lambda'_1(w') = \lambda'_2(w)$ et les étiquettes de tous les autres sommets ne sont pas modifiées. Notons que s'il existe des boucles sur v, alors il existe des arcs $a \in A(\mathbf{D}_2)$ tels que $s(a) = t(a) = v$. Une fois que v a émis un message, $\lambda'_1(v) = \lambda'_2(\varphi^{-1}(v))$. Par la

suite, une fois que v a entendu ce message, nous en déduisons que $\lambda'_1(v) = \lambda'_2(\varphi^{-1}(v))$. Ainsi, le digraphe (D_1, λ'_1) est fibré sur (D_2, λ'_2) via φ. Si l'exécution ρ est infinie sur $\mathbf{D}_2$, l'exécution correspondante relevée sur $\mathbf{D}_1$ est aussi infinie. Si l'exécution maximale ρ on $\mathbf{D}_2$ est finie, alors tous les messages sont arrivés et aucun processus n'a besoin d'émettre un message de nouveau. Par conséquent, après l'exécution relevée depuis ρ sur $\mathbf{D}_1$, celui-ci est fibré sur $\mathbf{D}_2$ et tous les messages sont arrivés et aucun processus n'a besoin d'émettre de messages : l'exécution relevée est maximale. □

Le schéma de cette preuve est illustrée par le diagramme suivant :

$$\begin{array}{ccc} \mathbf{D}_1 & \xrightarrow{p^*} & \mathbf{D}'_1 \\ {\scriptstyle \text{fibrations}}\downarrow & & \downarrow{\scriptstyle \text{fibrations}} \\ \mathbf{D}_2 & \xrightarrow[p^*]{} & \mathbf{D}'_2 \end{array}$$

2.3 Algorithme d'Énumération pour les Réseaux de Diffusion

Dans cette section, nous donnons la condition nécessaire basée sur un résultat d'impossibilité qui montre qu'il n'existe pas d'algorithme d'énumération (et a fortiori de nommage) pour un graphe étiqueté $\mathbf{G}$ tel que $Dir(\mathbf{G})$ n'est pas minimal pour les fibrations. Ensuite, nous prouvons que cette condition est suffisante en présentant un algorithme d'énumération (voir l'algorithme 1) qui s'appuie sur les communications de diffusion asynchrones et qui est inspirée par le travail précurseur de Mazurkiewicz [Maz97a].

2.3.1 Résultat d'Impossibilité

Étant donné un réseau représenté par un graphe $\mathbf{G}$, nous donnons une condition nécessaire que doit satisfaire $\mathbf{G}$ pour autoriser un algorithme d'énumération dans notre modèle. C'est un résultat d'impossibilité qui s'appuie sur la notion de fibrations présentée précédemment pour les communications asynchrones. Suivant la preuve du lemme 2.9, nous montrons que deux processus qui appartiennent à la même fibre ne peuvent pas avoir des noms différents.

Proposition 2.10 *Soit* $\mathbf{G}$ *un graphe étiqueté tel que* $Dir(\mathbf{G})$ *n'est pas minimal pour les fibrations. Il n'y a pas d'algorithme d'énumération pour* $\mathbf{G}$ *dans le modèle de diffusion asynchrone.*

Preuve : Considérons un graphe simple $\mathbf{G} = (G, \lambda)$ et un digraphe fortement connexe $\mathbf{D} = (D, \eta)$ tel que $Dir(\mathbf{G})$ est proprement fibré sur $\mathbf{D}$ via une fibration φ. Étant donné un algorithme $\mathcal{A}$ basé sur les communications de diffusions asynchrones, considérons une exécution de $\mathcal{A}$ sur $\mathbf{D}$ comme décrite dans le lemme 2.9. Notons que, si cette exécution de $\mathcal{A}$ sur $\mathbf{D}$ est infinie, alors suivant le lemme 2.9, il existe une exécution infinie de $\mathcal{A}$ sur $\mathbf{G}$. Enfin, $\mathcal{A}$ n'est pas un algorithme d'énumération pour $\mathbf{G}$.

Supposons maintenant que cette exécution de $\mathcal{A}$ sur $\mathbf{D}$ soit finie et donne une configuration $\mathbf{D}'$. Dans la configuration finale, tous les messages sont arrivés et aucun processus n'a besoin d'émettre. Ainsi, chaque sommet possède son étiquette finale. D'après le lemme 2.9, il existe une exécution relevée de $\mathcal{A}$ sur $Dir(\mathbf{G})$ qui donne une configuration $\mathbf{G}'$ telle que $\mathbf{G}'$ est proprement fibré sur $\mathbf{D}'$ via φ. Puisque $\mathbf{G}'$ est fibré sur $\mathbf{D}'$, cela implique qu'il existe au moins deux sommets qui ont la même étiquette dans $\mathbf{G}'$. Par conséquent, l'algorithme $\mathcal{A}$ ne donne pas d'identifiant unique à chacun des sommets du graphe et ce n'est donc pas un algorithme d'énumération pour $\mathbf{G}$. □

2.3.2 Description Informelle de l'Algorithme d'Énumération

Nous donnons tout d'abord une description générale de notre algorithme, dénoté $\mathcal{M}$ par la suite, lorsque celui-ci est exécuté sur un graphe simple connexe étiqueté $\mathbf{G}$.

Pendant l'exécution de l'algorithme d'énumération, chaque sommet v tente d'obtenir une identité (étiquette) unique : un nombre compris entre 1 et $|V(G)|$, la taille du graphe. Une fois qu'un sommet v a choisi son numéro $n(v)$, il l'émet à l'ensemble de son voisinage.

Quand un sommet v entend un message depuis un voisin u, il stocke le numéro $n(u)$. Toutes les identités ainsi rassemblées par un sommet v depuis ses voisins permettent à celui-ci de mettre à jour une liste qui sera appelée la *vue locale* de v. Plus précisément, la vue locale de v est un multi-ensemble, noté $N(v)$, des numéros attribués qui apparaissent dans son voisinage. Ensuite, v diffuse son numéro accompagné de son étiquette et de sa vue locale. Lorsqu'un sommet u découvre qu'il existe un autre sommet v avec le même numéro que lui, alors le sommet u doit décider s'il modifie son identité : il compare son étiquette $\lambda(u)$ et sa vue locale $N(u)$ avec l'étiquette $\lambda(v)$ et la vue locale $N(v)$ 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 que nous définirons par la suite), alors le sommet u choisit un nouveau numéro (sa nouvelle identité temporaire) et informe ses voisins de ce changement de numéro en l'émettant avec sa vue locale. Peu à peu, les numéros et les vues locales ainsi modifiées vont se diffuser dans le graphe. Lorsque l'exécution est terminée, si le digraphe $Dir(\mathbf{G})$ est minimal pour les fibrations, alors chaque sommet a un numéro unique : l'algorithme permet de résoudre le problème de l'énumération.

Étiquettes

On considère un graphe $\mathbf{G}$ où $\mathbf{G} = (G, \lambda)$ est un graphe simple étiqueté. La fonction $\lambda : V(G) \to L$ est l'étiquetage initial. Nous supposons qu'il existe un ordre total $<_L$ sur L. Lors de l'exécution, les étiquettes des arêtes ne seront pas modifiées et chaque sommet va obtenir une étiquette de la forme $(\lambda(v), n(v), N(v), M(v))$ correspondant aux informations suivantes :

- $\lambda(v) \in L$ est l'étiquette initiale de v et n'est pas modifiée par l'algorithme,
- $n(v) \in \mathbb{N}$ est le *numéro* courant du sommet v calculé par l'algorithme,
- $N(v) \in \mathcal{P}_{\text{fin}}(\mathbb{N} \times \mathbb{Z})$ [3] est la *vue locale* de v. Intuitivement, une fois que v a mis à

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

jour sa vue locale, (n, p) appartient à $N(v)$ si v a entendu p voisins ayant n pour numéro,
- $M(v) \in \mathbb{N} \times L \times \mathcal{P}_{\text{fin}}(\mathbb{N}^2)$ est la *boîte-aux-lettres* de v. La boîte-aux-lettres contient toutes les informations entendues par v pendant l'exécution de l'algorithme. Si $(m, \ell, \mathcal{N}) \in M(v)$, cela signifie qu'à une étape précédente de l'exécution, il y avait un sommet u tel que $n(u) = m$, $\lambda(u) = \ell$ et $N(u) = \mathcal{N}$.

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

Afin de mettre à jour la vue locale d'un sommet $v_0 \in V(G)$, on définit une fonction $update(n, n_{old})$ regroupant les opérations suivantes. Tout d'abord, si $n_{old} \neq 0$, nous appliquons la règle suivante :
- s'il existe $(n_{old}, 1) \in N(v_0)$ alors $N(v_0) := N(v_0) \setminus \{(n_{old}, 1)\}$,
- s'il existe $(n_{old}, p) \in N(v_0)$ avec $p \neq 1$ alors $N(v_0) := N(v_0) \setminus \{(n_{old}, p)\} \cup \{(n_{old}, p-1)\}$,
- sinon, $N(v_0) := N(v_0) \cup \{(n_{old}, -1)\}$.

De même, symétriquement, nous appliquons les opérations suivantes :
- s'il existe $(n, -1) \in N(v_0)$ alors $N(v_0) := N(v_0) \setminus \{(n, -1)\}$,
- s'il existe $(n, p) \in N(v_0)$ avec $p \neq -1$ alors $N(v_0) := N(v_0) \setminus \{(n, p)\} \cup \{(n, p+1)\}$,
- sinon, $N(v_0) := N(v_0) \cup \{(n, 1)\}$.

Messages

Dans notre algorithme, les processus échangent des messages de la forme $< (m, n_{old}, M) >$. Si un processus u émet un message $< (m, n_{old}, M) >$, alors :
- m est le numéro courant $n(u)$ de u,
- n_{old} est le numéro précédent de u, c.-à-d., le numéro que u a émis à v dans un précédent message ; si dans l'intervalle, u n'a pas modifié sont numéro, alors $n_{old} = m$,
- M la boîte-aux-lettres de u.

Remarque 2.11 *S'il existe $(n, p) \in N(v)$ avec $p < 0$, alors cela signifie que parmi tous les messages $< (m, n_{old}, M) >$ que v a entendu, il y a plus de messages où $n_{old} = n$ que de messages où $m = n$. Cependant, chaque fois qu'un processus w émet un message $< (m, n_{old}, M) >$ avec $m \neq n_{old}$, on sait que w a précédemment émis un message $< (n_{old}, n'_{old}, M) >$ avec $n_{old} > n'_{old}$.*

Par conséquent, s'il existe $(n, p) \in N(v)$ avec $p < 0$, cela implique que v n'a pas encore entendu de messages de ses voisins, et ainsi il peut attendre jusqu'à ce qu'il entende un message de la forme $< (m, n, M) >$.

Un Ordre sur les Vues Locales

Comme pour l'algorithme de Mazurkiewicz [Maz97a], les bonnes propriétés de l'algorithme reposent sur un ordre total sur les vues locales. Cet ordre permet à un sommet u de modifier son numéro s'il découvre l'existence d'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 a lui-même envoyé lors d'une étape précédente. L'algorithme que nous décrivons ici est tel que la vue locale de chaque sommet ne peut pas diminuer durant l'exécution.

Afin de comparer deux éléments de $\mathbb{N}^2$, nous considérons l'ordre lexicographique usuel sur $\mathbb{N}^2$: $(n,p) < (n',p')$ si $n < n'$, ou si $n = n'$ et $p < p'$.

Ensuite, soient $N_1, N_2 \in \mathcal{P}_{\text{fin}}(\mathbb{N}^2)$, $N_1 \neq N_2$. Considérons (n,p) comme étant l'élément maximal de la différence symétrique $N_1 \triangle N_2 = (N_1 \setminus N_2) \cup (N_2 \setminus N_1)$. Alors $N_1 \prec N_2$ si et seulement si une des conditions suivantes est satisfaites :

– $(n,p) \in N_1$ et $p < 0$,
– $(n,p) \in N_2$ et $p > 0$.

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 donc que $N(u)$ est plus *faible* que $N(v)$). Noter qu'en particulier l'ensemble vide est minimal pour $\prec$. Nous rappelons que l'ensemble des étiquettes initiales L est totalement ordonné par $<_L$. Nous étendons $\prec$ à un ordre total sur $L \times \mathcal{P}_{\text{fin}}(L \times \mathbb{N})$: $(\ell, N) \prec (\ell', N')$ si soit $\ell <_L \ell'$, ou soit $\ell = \ell'$ et $N \prec N'$. Nous notons $\preceq$ la clôture réflexive de $\prec$.

2.3.3 L'Algorithme d'Énumération $\mathcal{M}$

L'algorithme pour le sommet v_0, décrit par l'algorithme 1 est présenté suivant un modèle dirigé par les événements (voir Tel [Tel00] p. 553). Nous rappelons que l'algorithme que nous décrivons ne nécessite pas de communication FIFO, c.-à-d., l'ordre des messages émis n'est pas obligatoirement le même que celui des messages entendus.

La première action **I** peut être exécutée seulement par un processus qui se réveille si celui-ci n'a pas entendu de messages préalablement. Dans ce cas, il choisit le numéro 1, met à jour sa boîte-aux-lettres et informe son voisinage.

L'action **R** regroupe la suite d'instructions que le processus v_0 doit suivre lorsqu'il entend un message de la forme $< (n', n'_{old}, M') >$ depuis un de ses voisins. D'abord, il met à jour sa boîte-aux-lettres en y ajoutant le contenu de M'. Ensuite, il modifie son numéro s'il existe un élément $(n(v_0), \ell, \mathcal{N}) \in M(v_0)$ tel que $(\lambda(v_0), N(v_0)) \prec (\ell, \mathcal{N})$, c.-à-d., s'il existe un autre processus dans le réseau qui a le même numéro avec une vue locale plus forte. Il met alors sa vue locale suivant la fonction $update(n, n_{old})$ décrite plus haut. Il ajoute son nouvel état calculé $(n(v_0), \lambda(v_0), N(v_0))$ dans sa boîte-aux-lettres. Enfin, si sa boîte-aux-lettres a été modifiée par l'exécution de ces instructions, il émet son numéro et sa nouvelle boîte-aux-lettres.

Si la boîte-aux-lettres de v_0 n'est pas modifiée par l'exécution de l'action **R**, cela signifie que l'information qu'a v_0 à propos de son voisin (c.-à-d., son numéro) est correcte, que tous les éléments contenus dans M' sont déjà présents dans $M(v_0)$ et, que pour chaque $(n(v_0), \ell, \mathcal{N}) \in M(v_0)$, $(\ell, \mathcal{N}) \preceq (\lambda(v_0), N(v_0))$.

L'action **S** est invoquée lorsque la variable locale booléenne *emit* est mise à *vrai* par les actions **I** ou **R**. Cela signifie que le processus a besoin d'émettre un message à l'ensemble de son voisinage. Noter que l'utilisation de ce booléen permet de limiter le nombre d'émissions de messages, et ainsi de réduire la complexité en message.

Algorithme 1: L'algorithme $\mathcal{M}$ dans le modèle de diffusion asynchrone.

var : $emit$: booléen **init** faux;
n_{old} : entier **init** 0 ;
I : $\{n(v_0) = 0$ et aucun message n'est arrivé à $v_0\}$
début
  $M_{old} := \emptyset$;
  $n(v_0) := 1$;
  $M(v_0) := \{(n(v_0), \lambda(v_0), \emptyset)\}$;
  $emit := vrai$
fin
S : $\{emit = vrai\}$
début
  emit $< (n(v_0), n_{old}, M(v_0)) >$;
  $n_{old} := n(v_0)$;
  $emit := faux$
fin
R : $\{$Un message $< (n', n'_{old}, M') >$ est arrivé à $v_0\}$
début
  $M_{old} := M(v_0)$;
  $M(v_0) := M(v_0) \cup M'$;
  si $n(v_0) = 0$ *ou* $\exists (n(v_0), \ell, \mathcal{N}) \in M(v_0)$ *tel que* $(\lambda(v_0), N(v_0)) \prec (\ell, \mathcal{N})$ **alors**
    $n(v_0) := 1 + \max\{n \mid \exists (n, \ell, \mathcal{N}) \in M(v_0)\}$;
  $N(v_0) := update(n', n'_{old})$;
  $M(v_0) := M(v_0) \cup \{(n(v_0), \lambda(v_0), N(v_0))\}$;
  si $\forall (n, p) \in N(v_0), p > 0$ *et* $M(v_0) \neq M_{old}$ **alors**
    $emit := vrai$
fin

2.3.4 Correction de l'Algorithme $\mathcal{M}$

Soit **G** un graphe simple étiqueté. Dans la suite, i est un entier représentant une étape de calcul. Étant donnée $(\lambda(v), (n_i(v), N_i(v), M_i(v))$ une étiquette du sommet v après la $i$ème étape de calcul de l'algorithme $\mathcal{M}$ (algorithme 1). Nous présentons quelques propriétés satisfaites par chaque exécution de l'algorithme dans le modèle que nous considérons.

Le lemme suivant, qui peut être facilement prouvé par une récurrence sur la longueur de l'exécution, rappelle quelques propriétés simples qui sont toujours satisfaites par l'état de chaque sommet.

Lemme 2.12 *Pour tout sommet $v \in V(G')$, et pour toute transition i,*

1. $n_i(v) \neq 0 \implies (n_i(v), \lambda(v), N_i(v)) \in M_i(v)$,
2. $\forall n' \in N_i(v)$ *alors* $n' > 0$ *and* $\exists \ell' \in L, \exists N' \in \mathcal{P}_{\text{fin}}(\mathbb{N}^2)$ *tel que* $(n', \ell', N') \in M_i(v)$.

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

Lemme 2.13 *Pour chaque transition i et chaque sommet v,*
- $M_i(v) \subseteq M_{i+1}(v)$,
- $n_i(v) \leq n_{i+1}(v)$,
- $N_i(v) \preceq N_{i+1}(v)$.

De plus, si le sommet v applique l'action **S** *à l'étape i et j avec $i \neq j$, alors $M_i(v) \neq M_j(v)$.*

Preuve : La propriété est trivialement vraie pour tous les sommets qui ne sont pas actifs à l'étape i. Il est aisé de voir que pour chaque sommet v, nous avons toujours $M_i(v) \subseteq M_{i+1}(v)$.

Pour chaque sommet v et chaque transition i tels que $n_i(v) \neq n_{i+1}(v)$, $n_{i+1}(v) = 1 + \max\{n_1; (n_1, \ell_1, N_1) \in M_i(v)\}$ et, comme montré par le lemme 2.12, soit $n_i(v) = 0 < n_{i+1}(v)$ ou soit $(n_i(v), \lambda(v), N_i(v)) \in M_i(v)$. Par conséquent, $n_i(v) < n_{i+1}(v)$.

Lorsque v entend un message de la forme suivante : $mess =< (n', n', M') >$, $N_{i+1}(v) = update(n', n') = N_i(v)$. Si $N_i(v) \neq N_{i+1}(v)$ alors v a entendu un message $mess =< (n', n'_{old}, M') >$ avec $n' > n'_{old}$ et ainsi $N_i(v) \prec N_{i+1}(v)$.

De plus, la condition imposée par l'action **S** est satisfaite quand la valeur de la variable *emit* devient *vrai*, c.-à-d., lorsque la boîte-aux-lettres $M(v)$ de v est modifiée. □

Les informations dont dispose chaque sommet v dans sa boîte-aux-lettres permettent d'obtenir des informations vérifiées par la configuration courante et globale du graphe. Le lemme suivant permet de prouver que si un sommet v connaît le numéro m (i.e., il existe ℓ, N tels que $(m, \ell, N) \in M_i(v)$), alors pour chaque $m' \leq m$, il existe un sommet v' dans le graphe tel que $n_i(v') = m'$. Nous montrons d'abord que si v connaît un numéro m, alors il existe un sommet v' tel que $n_i(v') = m$. Nous montrons aussi que si un sommet v connaît un numéro m, alors il connaît tous les numéros attribués plus petits que m.

Lemme 2.14 *Pour chaque sommet $v \in V(G)$ et chaque étape i, soit $n_i(v) \neq 0$, étant donné $(m', \ell', N') \in M_i(v)$, pour tout $1 \leq m \leq m'$, il existe un sommet $w \in V(G)$ tel que $n_i(w) = m$ et $(m, \ell, N) \in M_i(v)$.*

Preuve : Par récurrence sur i, on montre que pour chaque sommet v avec $n_i(v) \neq 0$, étant donné $(m', \ell', N') \in M_i(v)$, pour tout $1 \leq m \leq m'$, il existe $(m, \ell, N) \in M_i(v)$. Nous supposons que l'affirmation précédente est vraie pour toute étape $i \geq 0$. Si l'action I est appliquée par v, alors, $M_i(v) = (1, \lambda(v_0), \emptyset)$ et la propriété est trivialement vraie.

Si l'action R est appliquée par v, alors, v a entendu un message $mess =< (n', n'_{old}, M') >$ depuis un autre sommet voisin v'. Soit j l'étape durant laquelle v' a émit ce message. On sait que $M' = M_j(v')$. Si v garde son numéro à l'étape $i+1$, alors, $M_{i+1}(v) = M_i(v) \cup M_j(v')$ et, par hypothèse de récurrence, la propriété est vérifiée. Parallèlement, si v' modifie son numéro, alors, $n_{i+1}(v) = 1 + max\{n \mid \exists (n, l, N) \in M_i(v) \cup M_j(v')\}$ et $M_{i+1}(v) = M_i(v) \cup M_j(v') \cup \{(n_{i+1}(v), \lambda(v), N_{i+1}(v))\}$. Par conséquent, la propriété est aussi vérifiée.

Supposons que le nombre m soit connu par v et soit $U = \{(u, j) \in V(G) \times \mathbb{N} \mid j \leq i, n_j(u) = m\}$. Considérons l'ensemble $U' = \{(u, j) \in U \mid \forall (u', j') \in U, N_{j'}(u') \prec$

$N_j(u)$ ou $N_{j'}(u') = N_j(u)$ et $j' \leq j\}$. Il est facile de voir qu'il existe i_0 tel que pour chaque $(u, j) \in U', j = i_0$. Puisque $(m, \ell, N) \in M_i(v)$, ni U et ni U' sont vides.

Si $i_0 < i$, le numéro $n_{i_0}(u) = m$ de u a été modifié à l'étape $i_0 + 1$ mais par maximalité de $(\lambda(u), N_{i_0}(u))$, le sommet u ne peut pas modifier son numéro. Par conséquent, $i_0 = i$ et il existe un sommet $w \in V(G)$ tel que $n_i(w) = m$. □

D'après le lemme 2.14, on déduit qu'après chaque étape, les identifiants attribués à chaque sommet forment soit un ensemble $[1, k]$, soit un ensemble $[0, k]$ avec $k \leq V(G)$.

Pour chaque étape i et chaque sommet v, s'il existe $n' \in N_i(v)$, d'après le lemme 2.12, il existe v' tel que $n_i(v') = n'$ et donc la vue locale $N(v)$ peut seulement contenir un nombre fini de valeurs. Il en est de même pour $M(v)$. Durant l'exécution de l'algorithme, l'étiquetage consécutif de chaque sommet v forme une séquence croissante, $(n_i(v), N_i(v), M_i(v))$, $i = 1, 2, \ldots$ et, chaque sommet peut émettre un message seulement s'il modifie sa boîte-aux-lettres. Puisque le nombre d'étiquettes disponibles est fini (mais dépendant de la taille du graphe), l'algorithme termine toujours.

Nous rappelons que nous avons fait l'hypothèse que tous les sommets connaissent la taille du graphe. Par conséquent, une fois qu'un sommet se voit attribuer l'identifiant correspondant à $|V(G)|$, d'après le lemme 2.14, il sait que tous les autres sommets ont des identifiants distincts qui ne changeront plus jusqu'à la fin de l'exécution. De plus, s'il possède $|V(G)|$ éléments dans sa boîte-aux-lettres, le sommet peut détecter localement la terminaison de l'algorithme.

2.3.5 Propriétés Satisfaites par l'Étiquetage Final

Puisqu'on sait que l'algorithme $\mathcal{M}$ termine toujours, on s'intéresse maintenant aux propriétés satisfaites par l'étiquetage final. On sait que lorsque l'algorithme est terminé, tous les messages émis sont arrivés à destination.

Lemme 2.15 *Toute exécution ρ de l'algorithme $\mathcal{M}$ sur un graphe simple étiqueté $\mathbf{G} = (G, \lambda)$ termine et l'étiquetage final (λ, n_p, N_p, M_p) des sommets vérifie les propriétés suivantes :*

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

et pour tous sommets v, v' :

2. *$M_p(v) = M_p(v')$,*
3. *$(n_p(v), \lambda(v), N_p(v)) \in M_p(v')$,*
4. *$n_p(v) = n_p(v')$ implique que $\lambda(v) = \lambda(v')$ et $N_p(v) = N_p(v')$,*
5. *$(n, p) \in N_p(v)$ si et seulement s'il existe des sommets $w_1, \ldots, w_p \in N_G(v)$ tels que pour chaque i, $n_p(w_i) = n$; dans ce cas, il existe $(n_p(v), p') \in N_p(w_i)$ avec $p' \geq 1$.*

Preuve :

1. D'après le lemme 2.14 appliqué à l'étiquetage final.

2. Sinon, deux sommets voisins v, v' tels que $M(v) = M(v')$. Toutefois, la configuration est finale, aussi bien v que v' ont envoyé leurs boîtes-aux-lettres à leurs voisins et ainsi $M(v) = M(v')$.
3. Un corollaire du point précédent en utilisant le lemme 2.12.
4. C'est une conséquence directe du point précédent et puisque ni v, ni v' ont besoin de changer leurs numéros.
5. Puisque chaque voisin de v qui a le même numéro n a émis un message avec un son numéro, et puisque tous les messages ont été entendus, nous savons qu'il existe $(n', p') \in N_p(v)$ avec $p' > p$. De plus, en raison de la fonction *update*, on sait que $\sum_{(n,p)\in N_p(v), p>0} p$ est bornée par le degré du sommet v. Par conséquent la propriété est vérifiée.

□

Grâce au lemme 2.15, on prouve, dans la proposition suivante, que l'étiquetage final de $\mathbf{G}$ permet de construire un graphe dirigé $\mathbf{D}$ tel que $Dir(\mathbf{G})$ est fibré sur $\mathbf{D}$.

Proposition 2.16 *Étant donné un graphe $\mathbf{G} = (G, \lambda)$, on peut associer, à partir de l'étiquetage final obtenu après une exécution ρ de l'algorithme $\mathcal{M}$, un digraphe $\mathbf{D}$ tel que $Dir(\mathbf{G})$ est fibré sur $\mathbf{D}$ et $V(D)$ est l'ensemble des identifiants apparaissant sur les sommets de $\mathbf{G}$ à la fin de l'exécution ρ.*

Preuve : Nous utilisons les notation du lemme 2.15. Soit le digraphe $\mathbf{G} = (G, \lambda)$.

On considère le graphe dirigé $\mathbf{D}$ défini par $V(D) = \{m \in \mathbb{N} \mid \exists v \in V(G), n_\rho(v) = m\}$. Pour tous $m, m' \in V(D)$, il y a p arcs $a_{m',m,1}, \ldots, a_{m',m,p}$ de m' à m s'il existe $v \in V(G)$ tel que $n_\rho(v) = m'$ et $(m, p) \in N_\rho(v)$ avec $p > 0$. D'après le lemme 2.15, c'est indépendant du choix de $v \in V(G)$. Pour tous sommets $v, v' \in V(G)$, si $n_\rho(v) = n_\rho(v')$ alors $\lambda(v) = \lambda(v')$ et on peut définir l'étiquetage η de D : pour tout $v \in V(G)$, $\eta(n_\rho(v)) = \lambda(v)$.

Rappelons que $V(Dir(\mathbf{G})) = V(G)$ et pour toute paire d'arêtes $\{v, v'\} \in E(G)$, il existe deux arcs $a_{v',v}, a_{v,v'}$ tels que $s(a_{v,v'}) = t(a_{v',v}) = v$ et $t(a_{v,v'}) = s(a_{v',v}) = v'$. De plus, pour chaque $v \in V(G)$, l'étiquette de v dans $Dir(\mathbf{G})$ est la même que dans $\mathbf{G}$.

Il reste à définir l'homomorphisme φ de $Dir(\mathbf{G})$ dans $\mathbf{D}$. Pour tout sommet $v \in V(G)$, $\varphi(v) = n_\rho(v)$. Pour tout sommet v tel que $\varphi(v) = n$, et pour chaque $(m, p) \in N_\rho(v)$ avec $p > 0$, on sait d'après le lemme 2.15 qu'il existe p arcs $a_1, \ldots, a_p \in A(Dir(\mathbf{G}))$ tels que $t(a_i) = v$ et $n_\rho(s(a_i)) = m$. Pour chaque $1 \leq i \leq p$, $\varphi(a_i) = a_{m,n,i}$.

Par définition, φ est une fibration et ainsi on en déduit que $Dir(\mathbf{G})$ est fibré sur $\mathbf{D}$. □

On considère maintenant un graphe $\mathbf{G}$ tel que $Dir(\mathbf{G})$ est minimal pour les fibrations. Pour chaque exécution ρ de l'algorithme $\mathcal{M}$ sur $\mathbf{G}$, le graphe dirigé obtenu à partir de l'étiquetage final est isomorphe à $Dir(\mathbf{G})$. Par conséquent, l'ensemble des identifiants attribués aux sommets est exactement $1, \ldots, |V(\mathbf{G})|$: chaque sommet a un identifiant unique.

L'algorithme 1 permet donc de résoudre le nommage sur tout graphe $\mathbf{G}$ tel que $Dir(\mathbf{G})$ est minimal pour les fibrations. Toutefois, nous avons fait l'hypothèse que chaque processus connaît la taille du réseau. Par conséquent, une fois qu'un processus possède $|V(G)|$

éléments différents dans sa boîtes-aux-lettres, d'après le lemme 2.14, il sait que tous les processus ont des identifiants deux à deux distincts qui ne vont plus changer jusqu'à la fin de l'exécution.

Finalement, nous avons prouvé le théorème suivant :

Théorème 2.17 *Pour tout graphe simple étiqueté* $\mathbf{G}$*, les assertions suivantes sont équivalentes :*

1. *il existe un algorithme de nommage (ou d'énumération) pour* $\mathbf{G}$ *utilisant des communications par diffusions asynchrones,*
2. *le graphe* $Dir(\mathbf{G})$ *est minimal pour les fibrations.*

2.3.6 Complexité de l'Algorithme $\mathcal{M}$

L'analyse de la complexité des algorithmes distribués représente un point crucial de beaucoup de propriétés telles que la consommation d'énergie lorsque l'on considère, par exemple, les réseaux de capteur sans fil. Nous nous intéressons ici à la complexité de l'algorithme 1. Pour cela, nous étudions la complexité du nombre de messages échangés par les processus ainsi que leur taille. On s'intéresse aussi à la mémoire requise et utilisée par chaque processus.

On considère que chaque sommet ne garde pas plus d'un seul élément (n, ℓ, N) pour tout n dans sa boîtes-aux-lettres. De plus, nous supposons que l'étiquetage initial λ du graphe G est tel que chaque étiquette initiale ℓ peut être encodée avec $O(\log |V(G)|)$ bits (ce qui est suffisant pour attribuer des étiquettes différentes à tous les sommets de G).

Proposition 2.18 *Pour tout graphe simple étiqueté* $\mathbf{G}$ *à* n *sommets,* m *arêtes et de degré maximum* Δ*, toute exécution de l'algorithme* $\mathcal{M}$ *nécessite* $O(mn^2)$ *émissions de messages de taille* $O(\Delta n \log n)$ *bits. De plus, chaque processus utilise* $O(\Delta n \log n)$ *bits de mémoire.*

Preuve : On considère un graphe $\mathbf{G}$ à n sommets, m arêtes et de degré maximum Δ. On considère une exécution ρ de l'algorithme $\mathcal{M}$ sur $\mathbf{G}$. D'après le lemme 2.14, on sait que chaque sommet ne modifie pas son numéro plus de n fois.

Pour tout sommet v, puisque les numéros de v et de ses voisins ne peuvent qu'augmenter, le couple $(n(v), N(v))$ peut prendre $(d(v)+1)n$ valeurs différentes. À chaque fois que le sommet v modifie son numéro ou sa vue locale, cela peut produire au plus l'émission de $O(n)$ messages (car les sommets ayant déjà $(n(v), \delta(v), N(v))$ dans leurs boîtes-aux-lettres ne diffusent plus ces messages). Par conséquent, toute exécution de l'algorithme nécessite $O(mn^2)$ messages. Puisque nous supposons que chaque sommet ne garde que l'information utile dans sa boîte-aux-lettres, il existe au plus n éléments (n_0, ℓ, N) dans $M(v)$ et chacun de ces éléments peut être représenté $O(\Delta \log n)$ bits. Ainsi, on peut représenter la boîte-aux-lettres de chaque sommet avec $O(\Delta n \log n)$ bits et donc, la taille de chaque message est de $O(\Delta n \log n)$ bits.

D'après la preuve précédente, on sait que la boîte-aux-lettre de chaque sommet est codée avec $O(\Delta n \log n)$ bits. De plus, pour tout sommet v, $n(v)$ peut être représenté avec $\log n$ bits tandis que $N(v)$ peut être représenté avec $O(\Delta \log n)$ bits. Par conséquent, chaque sommet a besoin de $O(\Delta nlogn)$ pour stocker son état. On peut noter que

la connaissance que doit avoir tout sommet pour fonctionner, n'influe par sur le calcul précédent. □

Comme corollaire à l'analyse de la complexité, le théorème 2.17 peut être étendu de la façon suivante :

Théorème 2.19 *Pour tout graphe simple étiqueté* $\mathbf{G}$*, les assertions suivantes sont équivalentes :*

- *il existe un algorithme de nommage (ou d'énumération) de complexité polynomiale (mémoire, messages, et taille de messages) pour* $\mathbf{G}$ *utilisant des communications par diffusions asynchrones,*
- *le graphe* $Dir(\mathbf{G})$ *est minimal pour les fibrations.*

Les algorithmes proposés par Yamashita et Kameda et Boldi *et al.*, présentés dans la section 2.1.4, nécessitent $O(n^2)$ émissions de messages de taille $2^{O(n)}$ bits. De plus, chaque processus utilise $2^{O(n)}$ bits de mémoire. Ainsi, en considérant différents aspects de la complexité, l'algorithme $\mathcal{M}$ s'adapte particulièrement bien au réseaux de diffusion multi-sauts composés de processus de faible capacité comme, par exemple, les capteurs sans fil.

2.4 Algorithme d'Élection pour les Réseaux de Diffusion

Comme précisé dans l'introduction de ce manuscrit, s'il est possible de résoudre le problème de l'énumération (ou du nommage) dans un graphe $\mathbf{G}$ alors il est possible de résoudre le problème de l'élection sur ce même graphe en déclarant le sommet ayant l'identifiant $|V(G)|$ (ou le plus élevé, dans le cas du nommage) comme étant élu. Toutefois, dans notre modèle, les problèmes de l'énumération et l'élection de ne sont pas équivalents. Considérons un graphe $\mathbf{G}$ et le digraphe $Dir(\mathbf{G})$ correspondant de la figure 2.1. Puisque $Dir(\mathbf{G})$ est fibré sur $\mathbf{D}$, d'après le théorème 2.17, le problème de l'énumération ne peut pas être résolu. Néanmoins, si tout les sommets connaissent initialement le graphe $\mathbf{G}$, on peut considérer un algorithme d'élection comme suit : chaque sommet émet un message et, une fois qu'un sommet reçoit 4 messages, il peut se déclarer lui-même comme étant élu. Puisque le sommet étiqueté 3 est l'unique sommet de degré supérieur ou égal à 4 dans $\mathbf{G}$, le sommet 3 sera élu.

Dans cette section, nous allonss présenter un résultat d'impossibilité qui énonce qu'il n'existe pas d'algorithme d'élection pour un graphe $\mathbf{G}$ si $Dir(\mathbf{G})$ n'est pas $\mathcal{nt}$-minimal. Cette condition est nécessaire et nous montrons qu'elle est suffisante en donnant un extension de l'algorithme $\mathcal{M}$ (voir l'algorithme 2) pour résoudre le problème de l'élection.

2.4.1 Résultat d'Impossibilité

Étant donné un réseau représenté par un graphe simple étiqueté $\mathbf{G}$, nous présentons la condition nécessaire basée sur les $\mathcal{nt}$-fibrations que doit vérifier $\mathbf{G}$ pour admettre un algorithme d'élection.

Proposition 2.20 *Soit* $\mathbf{G}$ *un graphe étiqueté tel que* $Dir(\mathbf{G})$ *n'est pas* nt*-minimal. Il n'y a pas d'algorithme d'élection pour* $\mathbf{G}$ *dans le modèle de diffusion asynchrone.*

Preuve : Considérons un graphe simple $\mathbf{G} = (G, \lambda)$ et un digraphe fortement connexe $\mathbf{D} = (D, \eta)$ tel que $Dir(\mathbf{G})$ est nt-fibré sur $\mathbf{D}$ via une fibration φ. Étant donné un algorithme $\mathcal{A}$ basé sur les communications de diffusions asynchrones, considérons une exécution de $\mathcal{A}$ sur $\mathbf{D}$ comme décrite dans le lemme 2.9. Notons que, si cette exécution de $\mathcal{A}$ sur $\mathbf{D}$ est infinie, alors suivant le lemme 2.9, il existe une exécution infinie de $\mathcal{A}$ sur $\mathbf{G}$. Enfin, $\mathcal{A}$ n'est pas un algorithme d'élection pour $\mathbf{G}$.

Supposons maintenant que cette exécution de $\mathcal{A}$ sur $\mathbf{D}$ soit finie et donne une configuration $\mathbf{D}'$. Dans la configuration finale, tous les messages sont arrivés et aucun processus n'a besoin d'émettre. Ainsi chaque sommet possède son étiquette finale. Suivant le lemme 2.9, il existe une exécution relevée de $\mathcal{A}$ sur $Dir(\mathbf{G})$ qui donne une configuration $\mathbf{G}'$ telle que $\mathbf{G}'$ est fibré sur $\mathbf{D}'$ via φ. Puisque $\mathbf{G}'$ est nt-fibré sur $\mathbf{D}'$, cela implique que pour tout sommet $v \in V(\mathbf{G})$, il existe au moins deux sommets dans $\varphi^{-1}(\varphi(v))$ qui ont la même étiquette dans $\mathbf{G}'$. Par conséquent, il n'y a pas de sommet $v \in V(\mathbf{G})$ qui a une étiquette unique. L'algorithme $\mathcal{A}$ n'est donc pas un algorithme d'élection pour $\mathbf{G}$. □

2.4.2 Importance de la Connaissance Initiale

Nous nous intéressons ici à l'importance de la connaissance initiale. Dans l'algorithme $\mathcal{M}$ présenté dans la section précédente, chaque processus connaît seulement la taille du réseau. En utilisant cette connaissance initiale, nous assurons qu'à la fin d'une exécution de l'algorithme, chaque processus sait localement que tous les sommets ont obtenu un identifiant unique et ceci même si certains messages sont arbitrairement retardés, c.-à-d., sont envoyés, mais ne sont pas encore entendus. Boldi *et al.* [BCG+96] et Yamashita et Kameda [YK96c] montrent eux aussi que connaître la taille du graphe permet de résoudre le problème de l'élection dès lors que cela est possible. Cependant, dans leurs modèles, chaque processus connaît initialement son degré (ou bien peut être calculé aisément) et cette connaissance initiale est effectivement utilisée dans leurs algorithmes de constructions de vues.

Dans notre modèle, les sommets ne connaissent pas initialement leur degré et dans ce cas précis, la connaissance initiale sur la taille du graphe n'est pas suffisante pour résoudre le problème de l'élection dans les graphes où il peut être résolu. Par exemple, supposons qu'il existe un algorithme d'élection pour les trois graphes $\mathbf{G}_1$, $\mathbf{G}_2$ et $\mathbf{G}_3$ de la figure 2.2. Dans $\mathbf{G}_1$ (resp. $\mathbf{G}_2$, $\mathbf{G}_3$), il existe un seul et unique sommet de degré 4 (resp. 5, 5). Par conséquent, à l'instar du graphe de la figure 2.1, il est possible d'élire dans ces trois graphes dès lors que nous faisons l'hypothèse que chaque processus connaît initialement le graphe initial. Considérons le digraphe $\mathbf{B}$ tel que $Dir(\mathbf{G}_1)$ est t-fibré sur $\mathbf{B}$ via une fibration φ. Lorsqu'exécuté sur $\mathbf{B}$, un algorithme d'élection pour $\mathbf{G}_1$ se doit d'élire un processus dont la fibre est triviale. Ainsi, il existe deux sommets $a, b \in \mathbf{B}$ tels que $|\varphi^{-1}(a)| = |\varphi^{-1}(b)| = 1$ et qui peuvent être déclarés comme élu. Supposons que plusieurs messages sont arbitrairement retardés, c.-à-d., plusieurs liens de communication ne sont pas encore établis. On peut trouver deux graphes $\mathbf{G}_2$ et $\mathbf{G}_3$ et deux digraphes

correspondants $\mathbf{D}_2$ et $\mathbf{D}_3$ tels que $\mathbf{D}_2 \subseteq Dir(\mathbf{G}_2)$ et $\mathbf{D}_3 \subseteq Dir(\mathbf{G}_3)$ et tels que $\mathbf{D}_2$ et $\mathbf{D}_3$ sont eux aussi t-fibrés sur $\mathbf{B}$.

D'après le lemme 2.9, s'il existe une exécution maximale et finie d'un algorithme qui élit un sommet dans $\mathbf{B}$ alors il existe une exécution maximale et finie sur $Dir(\mathbf{G}_1)$, $\mathbf{D}_2$ et $\mathbf{D}_3$ qui élit aussi un sommet. Par conséquent, si le sommet b est marqué comme étant élu dans $\mathbf{B}$, alors il existe une exécution sur $Dir(\mathbf{G}_2)$ où les messages envoyés le long des arcs dans $Dir(\mathbf{G}_2) \setminus \mathbf{D}_2$ sont retardés d'un temps arbitrairement long. À un certain moment de l'exécution, deux sommets ont la même étiquette finale ÉLUE. De façon similaire, si le sommet a est marqué comme étant élu dans $\mathbf{B}$, alors il existe une exécution particulière sur $Dir(\mathbf{G}_3)$ telle que deux sommets sont marqués comme élus. Ainsi, nous ne pouvons pas trouver un algorithme universel d'élection pour tous les graphes d'ordre 8 où le problème de l'élection peut être résolu. Dans la suite de cette caractérisation, nous présentons un algorithme d'élection $\mathcal{M}_e$ qui fait l'hypothèse que chaque sommet connaît une carte du réseau sans toutefois connaître sa position dans cette carte.

Nous présentons maintenant comment étendre et adapter l'algorithme de la section précédente $\mathcal{M}$ pour résoudre le problème de l'élection sur les digraphes qui sont nt-minimaux.

Considérons un graphe $\mathbf{G}$ tel que $Dir(\mathbf{G})$ est t-fibré sur un digraphe $\mathbf{D}$. Notre but, ici, est de fournir une extension de l'algorithme précédent en le combinant avec l'algorithme de détection de la terminaison proposé dans [SSP85b]. L'idée est d'exécuter cet algorithme et reconstruire un graphe depuis le contenu de chaque boîte-aux-lettres des sommets (comme proposée par la proposition 2.16) et vérifier si tous les processus sont bel et bien impliqués dans l'exécution, c.-à-d., s'il n'existe pas de processus isolés.

L'Algorithme de Détection de la Terminaison SSP

Initialement, dans [SSP85b], l'algorithme a été conçu pour détecter la terminaison d'un autre algorithme distribué. Comme présenté dans la section 2.3.4, chaque processus est capable de déterminer sa propre condition de terminaison. L'algorithme SSP détecte l'instant pour lequel le calcul complet est terminé.

Soit G un graphe, nous associons à chaque processus v un prédicat $P(v)$ et un entier $a(v)$ correspondant à son *rayon de confiance*. Initialement, $P(v)$ est *faux* et la valeur de $a(v)$ est de -1. Si un sommet v a fini le calcul de l'algorithme initial, alors il change la valeur de $P(v)$ pour *vraie*. Dès lors que l'une ou l'autre des valeurs de $P(v)$ ou de $a(v)$ sont modifiées, le processus en informe ses voisins.

La modification de la valeur de $a(v_0)$ ne dépend seulement que de la valeur de $P(v_0)$ et de l'information que possède v_0 à propos des valeurs $\{a(v_1), \ldots, a(v_d)\}$ de ses voisins :

- si $P(v_0) = faux$ alors $a(v_0) = -1$,
- si $P(v_0) = vrai$ alors $a(v_0) = 1 + min\{a(v_k) \mid k \in [0; k]\}$.

Dans ce chapitre, nous adaptons cet algorithme en utilisant les idées de l'algorithme GSSP [GMM04]. Pour tout sommet v, la valeur de $P(v)$, au lieu d'être un booléen, sera un graphe reconstruit depuis le contenu de la boîte-aux-lettres v. Une propriété importante de la fonction P est telle qu'elle est constante entre deux moments où elle a la même valeur.

Dans le modèle que nous considérons, un sommet ne peut pas distinguer ses voisins. En

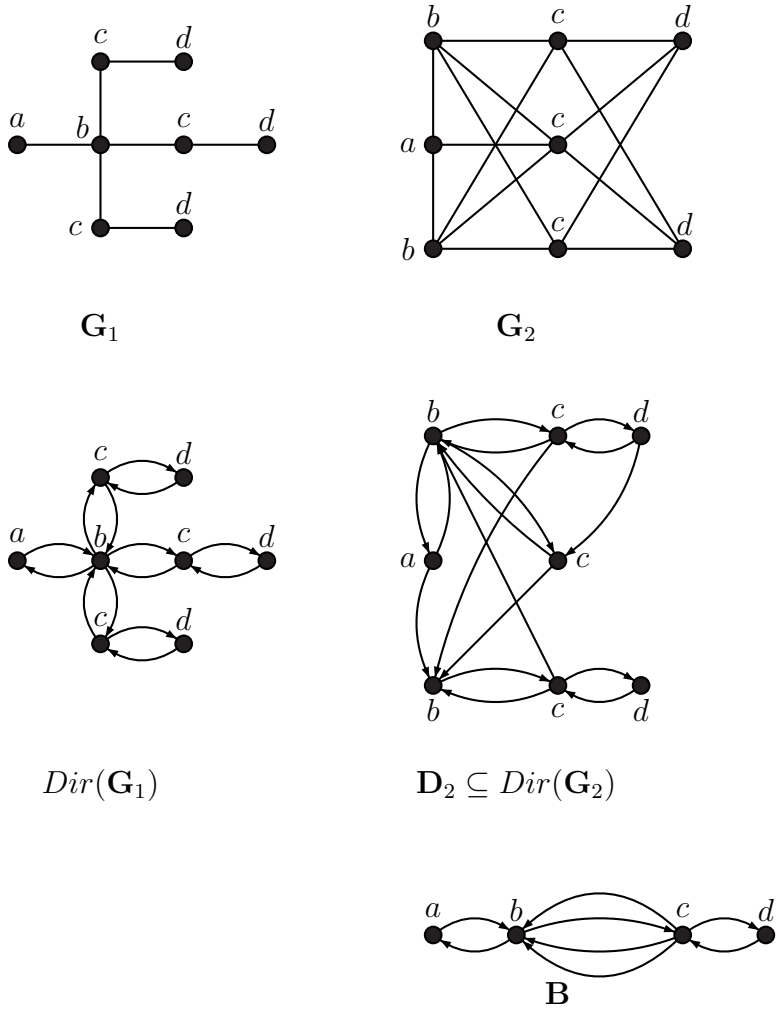

FIGURE 2.2 – Le graphe étiqueté $Dir(\mathbf{G}_1)$ est fibré sur le digraphe $\mathbf{B}$. Cette fibration est une ι-fibration et $Dir(\mathbf{G}_1)$ est $n\iota$-minimal ; les sous-digraphes $\mathbf{D}_2$ de $Dir(\mathbf{G}_2)$ et $\mathbf{D}_3$ de $Dir(\mathbf{G}_3)$ sont, eux-aussi, ι-fibrés sur la base minimale $\mathbf{B}$. Depuis le lemme 2.9, une exécution d'un algorithme d'élection sur $\mathbf{B}$ peut être relevée à une exécution sur $Dir(\mathbf{G}_1)$ et une exécution sur $\mathbf{D}_2$ et $\mathbf{D}_3$. Ainsi, le sommet a peut être marqué comme étant élu dans $\mathbf{B}$, $\mathbf{G}_1$ et $\mathbf{G}_2$ et le sommet b peut être marqué comme étant élu dans $\mathbf{B}$, $\mathbf{G}_1$ et $\mathbf{G}_3$. Si l'algorithme d'élection choisit a (resp. b), alors deux sommets dans $\mathbf{G}_2$ (resp. $\mathbf{G}_3$) sont marqués comme étant élu : ce n'est pas possible, nous arrivons à une contradiction.

conséquence, nous utiliserons les numéros qui apparaissent dans la vue locale. Un sommet v augmentera la taille du rayon de confiance $a(v)$ si et seulement si quand $|N(v)| = k$, alors v a entendu des messages provenant de k processus différents v' tels que $M(v') = M(v)$ et $a(v') \geq a(v)$.

Dans notre algorithme, chaque sommet essaie, de façon permanente, de reconstruire un digraphe $\mathbf{D}(M)$ depuis sa boîte-aux-lettres. Ce graphe dirigé est construit de la même

manière que présenté dans la proposition 2.16 de la section précédente. Étant donnée une boîte-aux-lettres M, on dit qu'un élément $(n, \ell, N) \in M$ est *maximal* si pour tout $(n, \ell', N') \in M$, $(\ell', N') \preceq (\ell, N)$; on note par $\max(M)$ l'ensemble des éléments maximaux de M ; Il est à noter que pour chaque n, il y a au plus un élément $(n, \ell, N) \in \max(M)$. S'il existe $(n, \ell, N) \in \max(M)$ tel qu'il y a $(m, p) \in N$ avec $p < 0$, ou s'il n'y a pas $(m, \ell', N') \in \max(M)$, alors $\mathbf{D}(M)$ est indéfini. Sinon, le graphe dirigé $\mathbf{D}(M)$ est défini comme suit : $V(D(M)) = \{n \mid \exists (n, \ell, N) \in \max(M)\}$, et pour chaque $(n, \ell, N) \in \max(M)$, $\lambda(n) = \ell$, et pour chaque $(m, p) \in N$, il y a exactement p arcs de m vers n dans $\mathbf{D}(M)$.

Étiquettes

Comme pour l'algorithme d'énumération $\mathcal{M}$, nous considérons un graphe étiqueté $\mathbf{G} = (G, \lambda)$. Pendant l'exécution de l'algorithme, les sommets v auront de nouvelles étiquettes de la forme $(\lambda(v), n(v), N(v), M(v), a(v), A(v))$. Ainsi, nous ajoutons deux éléments à l'étiquette de chaque sommet :

- $a(v) \in \mathbb{N}$ est le rayon de confiance du sommet v,
- $A(v) \in \mathcal{P}_{\text{fin}}(\mathbb{N} \times \mathbb{Z} \times \mathbb{Z})$ est un ensemble maintenu par chaque sommet v. Cet ensemble contient le rayon de confiance des voisins de la forme (n, p, a) où p est le nombre de voisins de v avec n comme identité et a comme rayon de confiance.

Pour des raisons de simplicité, nous définissons une fonction $rayon\_confiance(n, a)$ qui a pour but de mettre à jour l'ensemble $A(v_0)$ du processus v_0 comme suit. D'abord, si $a \geq 0$, nous appliquons les règles suivantes avec $a_{old} = a - 1$:

- s'il existe $(n, 1, a_{old}) \in A(v_0)$ alors $A(v_0) := A(v_0) \setminus \{(n, 1, a_{old})\}$,
- s'il existe $(n, p, a_{old}) \in A(v_0)$ avec $p \neq 1$ alors $A(v_0) := A(v_0) \setminus \{(n, p, a_{old})\} \cup \{(n, p - 1, a_{old})\}$,
- sinon, $A(v_0) := A(v_0) \cup \{(n, -1, a_{old})\}$.

De même, symétriquement, nous appliquons les opérations suivantes :

- s'il existe $(n, -1, a) \in A(v_0)$ alors $A(v_0) := A(v_0) \setminus \{(n, -1, a)\}$,
- s'il existe $(n, p, a) \in A(v_0)$ avec $p \neq -1$ alors $A(v_0) := A(v_0) \setminus \{(n, p, a)\} \cup \{(n, p + 1, a)\}$,
- sinon, $A(v_0) := A(v_0) \cup \{(n, 1, a)\}$.

Notons que dans l'algorithme 2, le digraphe $\mathbf{B_G}$ est la base minimale du graphe dirigé initial $Dir(\mathbf{G})$ sur lequel est exécuté l'algorithme d'élection.

Messages

Un message émis par un processus u et entendu par un processus v a la forme suivante : $< (m, n_{old}, M, a) >$, où m, n_{old} et M sont identiques aux valeurs des messages échangés dans $\mathcal{M}$. Nous ajoutons l'élément a correspondant à la valeur du rayon de confiance $a(u)$ de u.

2.4.3 L'Algorithme d'Élection $\mathcal{M}_e$

L'algorithme pour le sommet v_0 est décrit dans l'algorithme 2. Le coeur de l'action **I** reste inchangé par rapport à l'algorithme $\mathcal{M}$ à l'exception que le sommet v_0 doit initialiser

son rayon de confiance $a(v_0)$ à -1.

L'action **R** contient toutes les instructions que le sommet v_0 doit suivre lorsqu'il entend un message $< (n', n'_{old}, M', a') >$ depuis un voisin quelconque. Initialement, il réagit de la même manière que dans l'algorithme $\mathcal{M}$. Si sa boîte-aux-lettres a été modifiée, il doit initialiser de nouveau son rayon de confiance $a(v_0)$ ainsi que le rayon de confiance qu'il récupère de ses voisins dans $A(v_0)$. Inversement, si sa boîte-aux-lettre reste inchangée, il met à jour $A(v_0)$ avec la valeur a' reçue. Afin de mettre à jour son propre rayon de confiance, le sommet v_0 vérifie si tous les rayons de confiance collectés dans $A(v_0)$ sont supérieurs à $a(v_0)$ et si le graphe reconstruit depuis les boîtes-aux-lettres $M(v_0)$ est fibré sur le graphe $\mathbf{B_G}$. Cela signifie que v_0 a la même boîte-aux-lettres que ses voisins. En suivant, si la boîte-aux-lettres ou le rayon de confiance ont été modifiés par l'exécution, il émet son numéro avec sa nouvelle boîte-aux-lettres et rayon de confiance. Enfin, une exécution est terminée dès lors que le rayon de confiance est supérieur à la taille du graphe $|V(G)|$. Cela signifie que tous les processus ont reconstruit le même graphe depuis leurs boîtes-aux-lettres qui est fibré sur $\mathbf{B_G}$. Ainsi, le sommet dont la fibre est triviale est marqué comme étant élu. S'il existe plus d'un sommet satisfaisant cette condition, le sommet avec l'identité la plus faible est choisi.

2.4.4 Correction de l'Algorithme $\mathcal{M}_e$

On considère un graphe simple étiqueté **G**. Pour la suite de ce chapitre, i est un entier représentant une étape de calcul. Pour tout sommet $v \in V(G)$, on note $(\lambda(v), n_i(v), N_i(v), M_i(v), a_i(v), A_i(v))$ l'étiquette du sommet v après la $i$ème étape d'une exécution de l'algorithme $\mathcal{M}_e$. On présente d'abord quelques propriétés qui sont satisfaites par n'importe quelle exécution de l'algorithme dans le modèle de diffusion asynchrone.

Propriétés Satisfaites lors de l'Exécution

Le lemme suivant, qui peut être facilement prouvé par récurrence sur le nombre d'étapes, rappelle que si la boîte-aux-lettres de v est la même entre deux étapes, alors le rayon de confiance de v augmente.

Lemme 2.21 *Pour chaque étape i et chaque sommet v, si $M_i(v) = M_{i+1}(v)$ alors $a_{i+1}(v) \geq a_i(v)$. De plus, si v applique l'action* **S** *aux étapes i et j, alors $M_i(v) \neq M_j(v)$ ou $a_i(v) \neq a_j(v)$.*

Dans le lemme suivant, nous montrons que lorsqu'un processus émet un message, alors $\forall (n, p, a) \in A(v)$, $a \geq a(v) - 1$.

Lemme 2.22 *Pour chaque sommet $v \in V(\mathbf{G})$, et pour chaque transition i, soit $\forall (n, p, a) \in A_i(v)$, $a \geq a_i(v) - 1$, soit il existe $(n, p, a) \in A_i(v)$ tel que $p < 0$ et $\forall (n, p', a') \in A_i(v)$, $a' \geq a$.*

Preuve : On montre ce lemme par récurrence sur i. Initialement, $A(v) = \emptyset$ et la propriété est trivialement vraie. On suppose que la propriété est vérifiée pour tous les sommets à la transition i et on considère un sommet v qui entend un message $< n', n'_{old}, M', a' >$ à

Algorithme 2: L'algorithme $\mathcal{M}_e$ dans le modèle de diffusion asynchrone.

var : $emit$: booléen **init** faux;
I : $\{n(v_0) = 0$ et aucun message n'est arrivé à $v_0\}$
début
  $n(v_0) := 1$; $a(v_0) := -1$;
  $M_{old} := \emptyset$; $a_{old} := -1$; $n_{old} := 0$;
  $M(v_0) := \{(n(v_0), \lambda(v_0), \emptyset)\}$;
  $emit := vrai$
fin
S : $\{emit = vrai\}$
début
  si $\forall (n,p) \in N(v_0), p > 0$ *et* $\forall (n,p,a) \in A(v_0), p > 0$ **alors**
    si $a(v_0) = -1$ **alors**
      emit $< (n(v_0), n_{old}, M(v_0), a(v_0)) >$;
    sinon tant que $a_{old} < a(v_0)$ **faire**
      $a_{old} := a_{old} + 1$;
      emit $< (n(v_0), n_{old}, M(v_0), a_{old}) >$;
    $emit := faux$; $n_{old} := n(v_0)$; $a_{old} := a(v_0)$;
fin
R : {Un message $< (n', n'_{old}, M', a') >$ est arrivé à v_0}
début
  $M_{old} := M(v_0)$;
  $M(v_0) := M(v_0) \cup M'$;
  si $n(v_0) = 0$ *or* $\exists (n(v_0), \ell, \mathcal{N}) \in M(v_0)$ *tel que* $(\lambda(v_0), N(v_0)) \prec (\ell, \mathcal{N})$ **alors**
    $n(v_0) := 1 + \max\{n \mid \exists (n, \ell, \mathcal{N}) \in M(v_0)\}$;
  $N(v_0) := update(n', n'_{old})$;
  $M(v_0) := M(v_0) \cup \{(n(v_0), \lambda(v_0), N(v_0))\}$;
  si $M(v_0) \neq M_{old}$ **alors**
    $a(v_0) := -1$; $a_{old} := -1$;
    $A(v_0) := \{(n, p, -1) \mid (n,p) \in N(v_0)\}$;
  si $M(v_0) = M'$ *et* $a' \geq 0$ **alors**
    $A(v_0) := rayon\_confiance(n', a')$;
  si $\forall (n,p,a) \in A(v_0), a(v_0) \leq a$ **alors**
    construire $\mathbf{D}(M(v_0))$ depuis $M(v_0)$;
    si $\mathbf{D}(M(v_0))$ *est fibré sur* $\mathbf{B_G}$ **alors**
      $a(v_0) := 1 + \min\{a \mid \exists (n,p,a) \in A(v_0)\}$;
  si $a(v_0) \neq a_{old}$ *ou* $M(v_0) \neq M_{old}$ **alors**
    $emit := vrai$;
  si $a_i(v) > |V(G)|$ **alors**
    calculer $C_{\mathbf{G},\mathbf{D}(M(v_0))}$;
    si $n(v_0) = \min\{n \mid n \in C_{\mathbf{G},\mathbf{D}(M(v_0))}\}$ **alors** $statut := élu$;
    sinon $statut := non\text{-}élu$;
fin

l'étape $i+1$. Si $n'_{old} \neq n'$, ou si $M' \neq M_i(v)$, alors $a_{i+1}(v) = -1$ et pour tout $(n,p,a) \in A_{i+1}(v)$, $a = -1$. Notons que si $M_{i+1}(v) = M_i(v)$ et $a_{i+1}(v) \neq a_i(v)$, alors $a_{i+1}(v) = 1 + \min\{a \mid \exists (n,p,a) \in A_{i+1}(v)\}$ et la propriété est vérifiée.

Supposons qu'à chaque étape i, $\forall (n,p,a) \in A_i(v)$, $a \geq a_i(v)$. Si $a' \geq a_i(v)$, alors $\forall (n,p,a) \in A_{i+1}(v)$, $a \geq a_{i+1}(v) - 1$. Si $a' \leq a_i(v) - 1$, alors il existe $(n', -1, a'-1) \in A_{i+1}(v)$ et $\forall (n', p'', a'') \in A_i(v)$, $a'' \geq a' - 1$.

Maintenant, supposons qu'à chaque étape i, il existe $(n,p,a) \in A_i(v)$ tel que $p < 0$ et $\forall (n,p'',a'') \in A_i(v)$, $a'' \geq a$. Si $n' \neq n$, alors la propriété est encore une fois vérifiée. Sinon, nous avons les possibilités suivantes :

1. si $a' \leq a$, alors $(n', -1, a'-1) \in A_{i+1}(v)$ et $\forall (n,p'',a'') \in A_{i+1}(v)$, $a'' \geq a' - 1$,
2. si $a' = a+1$, alors $(n', p-1, a) \in A_{i+1}(v)$ et $\forall (n,p'',a'') \in A_{i+1}(v)$, $a'' \geq a$,
3. si $a' > a+1$, alors $(n', p, a) \in A_{i+1}(v)$ et $\forall (n,p'',a'') \in A_{i+1}(v)$, $a'' \geq a$.

□

On considère un sommet $v \in V(\mathbf{G})$ et une transition i. Pour toute valeur $a \geq 0$ donnée, et pour tout $(n,p) \in N_i(v)$, soit $\mathcal{X}_i(n,a,v) = \{p' \mid \exists (n,p',a') \in A_i(v)$ tel que $a' \geq a\}$ et $x_i(n,a,v) = \sum_{p \in \mathcal{X}_i(n,a,v)} p$.

Lemme 2.23 *Soit i une transition. Pour tout sommet $v \in V(\mathbf{G})$ et toute valeur $a \geq 0$ donnée, si $k = x_i(n,a,v) > 0$, il existe k sommets voisins $w_1, \ldots, w_k \in Dir(\mathbf{G})$ tels que pour tout $0 < l \leq k$, v a entendu un message $< (n, n', M, a) >$ de w_l à l'étape i.*

Preuve : Faisons l'hypothèse que $a = a_{max} = max\{a' \mid (n,p,a') \in A_i(v)\}$. Ainsi, nous avons $\mathcal{X}_i(n, a_{max}, v) = \{p' \mid \exists (n,p',a') \in A_i(v)$ tel que $a' = a_{max}\}$ et $x_i(n, a_{max}, v) = p'$. Cela signifie que le processus v a entendu p' messages de la forme $< (n, n_{old}, M, a) >$ avant l'étape i. D'après les lemmes 2.13 et 2.21, nous en déduisons que l'assertion est satisfaite.

Considérons $a < a_{max}$. Supposons que l'assertion est vérifiée pour $x_i(n, a+1, v)$. Par conséquent, le processus v a entendu au moins $x_i(n, a+1, v)$ messages **mess**$=<(n, n_{old}, M, a+1)>$. Ainsi, pour chaque message **mess** entendu par v, la fonction de mise à jour du rayon de confiance, $rayon\_confiance(n, a+1)$, est appelée et un élément (n,a) est retiré de l'ensemble $A_i(v)$. Cela signifie que si $(n,p',a) \in A_i(v)$, le processus v a entendu $p' + x_i(n, a+1, v)$ messages de la forme $< (n, n_{old}, M, a) >$ avant l'étape i. D'après les lemmes 2.13 et 2.21, chacun de ces messages ont été émis par des voisins différents de v. En conclusion, la propriété est vérifiée. □

Maintenant, on considère une transition i_0 et un sommet v_0 tels que $a_{i_0}(v_0) \geq 0$. On note $M = M_{i_0}(v_0)$. Pour tout sommet $v \in V(\mathbf{G})$, on définit $i(v, M, i_0)$ (ou $i(v)$ lorsque cela ne prête pas à confusion) comme suit. Si il y a un étape i telle que v émet un message $< n_i(v), n_{old}, M_i(v), a_i(v) >$ avec $M_i(v) = M$, alors $i(v)$ est la dernière étape durant laquelle v a émit un message de cette forme ; sinon $i(v) = \infty$. On définit un graphe dirigé $\mathbf{H}(M, i_0)$ comme suit. Pour tout sommet $v \in V(Dir(\mathbf{G}))$, v appartient à $V(\mathbf{H}(M, i_0))$ si $i(v) < \infty$. Pour chaque sommet $v \in V(\mathbf{H}(M, i_0))$, pour tout $(n,p) \in N_{i_0}(v)$, soit $k = x_{i_0}(n, a_{i(v)}(v) - 1, v)$. D'après le lemme 2.23, il existe k sommets voisins $w_1, \ldots, w_k$ de v tels que pour tout $0 < l \leq k$, $w_l \in V(\mathbf{H}(M, i_0))$ et $n_{i_0}(w_l) = n$ et v a entendu un message

$< (n, n_{old}, M, a_{i(v)}(v) - 1) >$ depuis w_l avant l'étape $i(v)$. Chaque arc correspondant de w_l vers v appartient à $A(\mathbf{H}(M, i_0))$. Dans la suite de ce chapitre, nous prouvons que tant que $\mathbf{H}(M, i_0) \neq Dir(\mathbf{G})$, alors l'exécution de l'algorithme d'énumération $\mathcal{M}$ n'est pas terminée.

Pour tout sommet $v \in V(\mathbf{G})$, puisque le rayon de confiance $a(v)$ et le nombre d'identités attribuées sont tous deux bornés par le nombre de sommets $|V(G)|$, nous savons que toute exécution de l'algorithme $\mathcal{M}_e$ termine. Dans le lemme suivant, nous montrons que le rayon de confiance d'un sommet permet de connaître à quelle distance de v les sommets ont la même boîte-aux-lettres.

Lemme 2.24 *Considérons une étape i_0 et une boîte-aux-lettres M. Pour tous sommets $v, w \in V(\mathbf{H}(M, i_0))$, si $dist_{\mathbf{H}(M,i_0)}(w, v) \leq a_{i(v)}(v)$, alors $a_{i(w)}(w) \geq a_{i(v)}(v) - dist_{\mathbf{H}(M,i_0)}(w, v)$.*

Preuve : Soit $\mathbf{H} = \mathbf{H}(M, i_0)$. Ce lemme peut être prouvé par récurrence sur la distance d entre un sommet w et un sommet v dans $\mathbf{H}$. Supposons que $d = 1$. Par conséquent, $a_{i(v)}(v) \geq dist_{\mathbf{H}}(w, v) \geq 1$ et $w \in N_{\mathbf{H}}(v)$. Puisque $a_{i(v)}(v) \geq 1$, on sait que pour tout $(m, p, a) \in A_{i(v)}(v)$, $a \geq a_{i(v)}(v) - 1$. Ainsi, par définition du graphe $H(M, i)$ et d'après le lemme 2.22, pour tout sommet $w \in N_{H(M,i)}(v)$, w a envoyé un message $< (n(w), n_{old}(w), M, a_{i(v)}(v) - 1) >$. Par conséquent, pour chaque $w \in N_{H(M,i)}$, il existe une étape $j < i(v) \leq i_0$ telle que $M_j(v) = M$ et $a_j(w) \geq a_{i(v)}(v) - 1$, et ainsi $a_{i(w)}(w) \geq a_{i(v)}(v) - 1$.

On fait l'hypothèse que cela est vérifié pour tous sommets v, w tels que $dist_{\mathbf{H}}(w, v) \leq d$. Considérons deux sommets v, w tels que $a_{i(v)}(v) \geq d + 1$ et $dist_{\mathbf{H}}(w, v) = d + 1$. Soit un sommet $u \in \mathbf{H}$ tel que $(w, u) \in A(H)$ et $dist_{\mathbf{H}}(u, v) = d$. Par hypothèse de récurrence, $a_{i(u)}(u) \geq a_{i(v)}(v) - d$ et $a_{i(w)}(w) \geq a_{i(u)}(u) - 1$. Et donc, $a_{i(w)}(w) \geq a_{i(v)}(v) - (d + 1)$. □

Rappelons que $\mathbf{B_G}$ est le graphe dirigé tel que $Dir(\mathbf{G})$ est t-fibré sur $\mathbf{B_G}$ par l'intermédiaire d'une relation de fibration φ et $\mathbf{B_G}$ est la base minimale de $Dir(\mathbf{G})$. Quand on considère une exécution de l'algorithme $\mathcal{M}_e$ dans laquelle l'envoi de certains messages est retardé, tout processus impliqué dans le calcul appartient donc à un sous graphe dirigé $\mathbf{H}$ de $Dir(\mathbf{G})$. Dans le lemme suivant, nous montrons que lorsque $\mathbf{H}$ est fibré sur $\mathbf{B_G}$, la vue de chaque sommet $v \in V(\mathbf{H})$ est isomorphe à la vue de ce même sommet $v \in V(\mathbf{G})$.

Lemme 2.25 *Soit $\mathbf{H}$ un sous graphe dirigé de $Dir(\mathbf{G})$ et le graphe dirigé $\mathbf{B_G}$ tels que $Dir(\mathbf{G})$ (resp. $\mathbf{H}$) est fibré sur $\mathbf{B_G}$ par l'intermédiaire d'une relation de fibration $\varphi_{\mathbf{G}}$ (resp. $\varphi_{\mathbf{H}}$). Si x_0 est un sommet avec une vue maximale dans $\mathbf{B_G}$, alors $\varphi_{\mathbf{H}}(v) = x_0 \implies \varphi_{\mathbf{G}}(v) = x_0$. De plus, pour tout sommet $v \in \mathbf{H}$, $T_{\mathbf{G}}(v) \approx T_{\mathbf{H}}(v)$ et ainsi $\mathbf{H} \approx \mathbf{G}$.*

Preuve : Puisque $\mathbf{H}$ est un sous graphe dirigé de $\mathbf{G}$, d'après la Remarque 2.7, pour chaque sommet v, $T_{\mathbf{H}}(v) \preceq T_{\mathbf{G}}(v)$. Puisque $\mathbf{H}$ est fibré sur $\mathbf{B_G}$ via φ, pour chaque sommet w_0 dans $\mathbf{B_G}$ qui a une vue maximale, pour chaque sommet $v_0 \in \varphi^{-1}(w_0)$, $T_{\mathbf{H}}(v_0)$ est maximale dans $\mathbf{G}$ et ainsi $T_{\mathbf{H}}(v_0) = T_{\mathbf{G}}(v_0)$.

On montre maintenant que pour chaque sommet v dans $V(\mathbf{G})$, $T_{\mathbf{H}}(v) = T_{\mathbf{G}}(v)$. Soit X_0 l'ensemble des sommets de vues maximales. Soit v_0, le sommet le plus proche de v dans $\mathbf{G}$ tel que $T_{\mathbf{G}}(v_0)$ est maximal, et soit $dist_{\mathbf{G}}(v, X_0)$ la distance de v à v_0 dans $\mathbf{G}$. Nous prouvons cette propriété par récurrence sur la distance $dist_{\mathbf{G}}(v, X_0)$. Si $v \in X_0$, alors nous

savons déjà que la propriété est satisfaite. Sinon, il existe un sommet voisin u de v tel que $dist_{\mathbf{G}}(u, X_0) = dist_{\mathbf{G}}(v, X_0) - 1$. Par récurrence, on sait que $T_{\mathbf{G}}(u) \approx T_{\mathbf{H}}(u)$, et ainsi u à le même degré dans $\mathbf{G}$ et dans $\mathbf{H}$. De plus, le multi-ensemble des vues des voisins de u devrait être le même aussi bien dans $\mathbf{H}$ que dans $\mathbf{G}$. Par conséquent, si $T_{\mathbf{H}}(v) \prec T_{\mathbf{G}}(v)$, il existe un autre sommet v' de v tel que $T_{\mathbf{G}}(v) \prec T_{\mathbf{H}}(v)$, ce qui n'est pas possible Ainsi, pour tout sommet $v \in V(H)$, $T_{\mathbf{G}}(v) \approx T_{\mathbf{H}}(v)$ et $N_{\mathbf{G}}(v) = N_{\mathbf{H}}(v)$. Puisqu'on sait que $\mathbf{G}$ est connexe, $V(G) = V(H)$ et $Dir(\mathbf{G}) \approx \mathbf{H}$. □

D'après la proposition 2.16, une fois que l'algorithme d'énumération $\mathcal{M}$ est terminé sur le graphe $\mathbf{H}(M, i_0)$, chaque sommet v possède la même boîte-aux-lettre $M = M(v)$ et est capable de construire un graphe dirigé étiqueté $\mathbf{D}(M(v))$. Nous devons montrer que si $\mathbf{D}(M(v))$ est fibré sur $\mathbf{B}_{\mathbf{G}}$, alors $\mathbf{H}(M, i_0) = Dir(\mathbf{G})$.

On prouve maintenant dans le lemme suivant que dès lors qu'une sommet obtient un rayon de confiance plus grand que la taille du graphe, tous les sommets du graphe ont la même boîte-aux-lettre et ont un rayon de confiance plus grand que 0.

Lemme 2.26 *S'il existe une transition i_0 et un sommet v tels que $a_{i_0}(v) > |V(G)|$, alors il existe un sous-graphe dirigé $\mathbf{H}'$ de $\mathbf{H}(M_{i_0}(v), i_0)$ tel que $\mathbf{H}'$ est fibré sur $\mathbf{D}(M_{i_0}(v))$.*

Preuve : On pose $M = M_{i_0}(v)$ et on considère le graphe $\mathbf{H}(M, i_0)$ défini plus haut. Soit V' l'ensemble des sommets $w \in V(H(M, i_0))$ tels qu'il existe un chemin de w à v dans $\mathbf{H}(M, i_0)$. Soit $\mathbf{H}'$ le sous-graphe dirigé de $\mathbf{H}(M, i_0)$ induit par V'. D'après le lemme 2.24, pour chaque sommet $w \in V(H')$, $M_{i(w)}(w) = M$ et $a_{i(w)}(w) \geq 1$. Puisque $a_{i(w)}(w) \geq 1$, il n'existe donc pas $(n_{i(w)}(w), \ell', N') \in M$ tel que $(\lambda(w), N_{i(w)}(w)) \prec (\ell', N')$. Par conséquent, pour tous sommets $w, w' \in V(H')$, si $n_{i(w)}(w) = n_{i(w')}(w')$, alors $\lambda(w) = \lambda(w')$ et $N_{i(w)}(w) = N_{i(w')}(w')$.

Notons que, puisque $a_{i(w)}(w) \geq 1$, pour tout $(n, p, a) \in A_{i(w)}(w)$, $a \geq 0$. Par conséquent, pour tout $(n, p) \in N_{i(w)}(w)$, $x_{i(w)}(n, a_{i(w)}(w) - 1, w) = p$. Ainsi, dans $\mathbf{D}(M)$, pour tout $(n, p) \in N_{i(w)}(w)$, il y a p arcs du sommet étiqueté n vers le sommet $n_{i(w)}(w)$.

Il ne nous reste plus qu'à définir l'homomorphisme φ du graphe $\mathbf{H}'$ vers le graphe $\mathbf{D}(M)$. Pour chaque sommet $w \in V(H')$, soit $\varphi(w) = n_{i(w)}(w)$. En considérant un sommet $w \in V(H')$, nous définissons l'image par φ de tous ses arcs entrants de la façon suivante. Par construction de $\mathbf{H}(M, i_0)$, pour chaque $(n, p) \in N_{i(w)}(w)$, on sait qu'il existe exactement $a_1, \ldots, a_p \in A(H(M, i_0))$ tel que pour chaque $l \in [1, p]$, $t(a_l) = w$ et $n_{i(s(a_l))}(s(a_l)) = n$. Ainsi, on pose $\varphi(a_l) = a_{n, n_{i(w)}(w), l}$. Par construction, $\mathbf{H}'$ est fibré sur $\mathbf{D}(M)$ par l'intermédiaire de φ. □

Ainsi, s'il existe un sommet v tel que le graphe dirigé $\mathbf{D}(M(v))$ reconstruit depuis sa boîte-aux-lettre $M(v)$ n'est pas fibré sur la base minimale $\mathbf{B}_{\mathbf{G}}$ de $Dir(\mathbf{G})$, l'exécution de l'algorithme d'élection n'est pas terminée.

Dans le lemme suivant, on montre que, à la fin de toute exécution de l'algorithme $\mathcal{M}_e$ sur un graphe nt-minimal, un seul et unique sommet est déclaré comme étant *élu*.

Lemme 2.27 *Pour toute exécution de l'algorithme $\mathcal{M}_e$ sur un graphe $\mathbf{G}$ tel que $Dir(\mathbf{G})$ est nt-minimal, exactement un seul sommet v est déclaré comme étant* élu.

Preuve : Nous savons que toute exécution maximale de $\mathcal{M}_e$ termine. D'abord, supposons qu'après la transition finale i, il existe un sommet v tel que $a_i(v) \leq |V(G)|$. Puisque tous les messages envoyés ont été entendus, pour tout sommet $v \in V(G)$, pour tout $(n,p) \in N(v)$, $p > 0$ et pour tout $(n,p,a) \in A(v)$, $p > 0$. Parmi tous les sommets v tels que $a_i(v)$ est minimal, soit v le dernier sommet ayant entendu un message et soit i_0 l'étape durant laquelle v a entendu ce dernier message. Après que v ait traité ce message $a_{i_0}(v) = 1 + \min\{a \mid \exists (n,p,a) \in A(v)\}$. Ainsi il existe un voisin w de v tel que $a_i(w) = a_{i_0}(v) - 1 = a_i(v) - 1$. Ce qui amène à une contradiction au regard de notre choix de v.

D'après les lemmes 2.25 and 2.26, s'il existe une étape i_0 et un sommet $v \in V(G)$ tels que $a_{i_0}(v) > |V(G)|$, alors les graphes $\mathbf{H}(M(v), i_0)$ et $Dir(\mathbf{G})$ sont isomorphes. De plus, d'après le lemme 2.24, on sait que tout les sommets ont la même boîte-aux-lettre et que pour chaque w, $n(w), N(w)$ et $M(w)$ ne changeront plus par la suite. Par conséquent, après l'étape i_0, pour tout sommet w, le graphe dirigé $\mathbf{D}(M(w))$ est toujours $\mathbf{D}(M_{i_0}(v))$. Ainsi, il existe une étape i telle que pour tous les sommets $w \in V(G)$, $M_i(w) = M_{i_0}(v)$ et $a_i(w) > |V(G)|$. Soit $M = M_{i_0}(v)$. Puisque $Dir(\mathbf{G})$ est t-minimal, l'ensemble des candidats $C_{\mathbf{G},\mathbf{D}(M)}$ n'est pas vide. Donc, il existe un unique sommet $v \in V(G)$ tel que $n_i(v) = \min C_{\mathbf{G},\mathbf{D}(M)}$, et ce sommet est *élu* localement par chaque processus. □

Finalement nous avons prouvé le théorème suivant :

Théorème 2.28 *Pour tout graphe simple étiqueté* $\mathbf{G}$*, les assertions suivantes sont équivalentes :*

1. *il existe un algorithme d'élection polynomial en mémoire, en nombre de messages et pour la taille des messages pour* $\mathbf{G}$ *utilisant des communications par diffusion asynchrone,*
2. *le graphe* $Dir(\mathbf{G})$ *est* nt*-minimal.*

Remarque 2.29 *D'après le lemme 2.30, étant donné un graphe dirigé minimal* $\mathbf{B}$*, nous savons que pour tout graphe simple* $\mathbf{G}$ *fibré sur* $\mathbf{B}$*, l'ensemble des candidats* $C_{\mathbf{G},\mathbf{B}}$ *ne dépend pas de* $\mathbf{G}$*, mais de* $\mathbf{B}$ *uniquement.*

Dans l'algorithme 2, puisque les processus n'utilisent que la base minimale $\mathbf{B_G}$ *de* $Dir(\mathbf{G})$*, il est possible de relaxer la connaissance initiale de chaque processus. Afin de résoudre le problème de l'élection dans notre modèle asynchrone, il suffit que chaque processus connaisse une borne sur la taille du graphe et la base minimale* $\mathbf{B_G}$ *— et non nécessairement le graphe initial* $\mathbf{G}$*.*

2.4.5 Quelques Remarques sur la Connaissance Initiale : le Degré

D'après nos hypothèses précédentes sur la connaissance initiale de chaque processus, une question intéressante pourrait être de savoir ce qu'il se passe lorsque chacun des processus connaît initialement son degré, c.-à-d., le nombre de processus voisins.

Soit $\mathbf{G}$ un graphe dirigé étiqueté tel que $Dir(\mathbf{G})$ est nt-minimal. Si chaque processus connaît son degré et la taille du graphe, il est possible de modifier l'algorithme $\mathcal{M}_e$ (voir l'algorithme 2) afin de prendre en considération cette combinaison de connaissances. Avant

d'augmenter son rayon de confiance dans lequel tous les processus ont la même boîte-aux-lettre, chaque processus v attend jusqu'au moment où il a reçu un message depuis tous ses processus voisins. Une fois la somme des p telle que $(n,p) \in N(v)$ est égale au degré $deg(v)$ de v, nous en déduisons que v a reçu un message de tous ses processus voisins au moins une fois. D'après le lemme 2.24, pour chaque étape i, la boule dans $\mathbf{G}$ centrée en v et de rayon $a_i(v)$ appartient à $H(M,i)$. Par conséquent, si $a_i(v) > |V(G)|$ alors $\mathbf{H}(M,i(v))$ et $Dir(\mathbf{G})$ sont isomorphes. Notons que de connaître le diamètre du graphe peut être suffisante. Le rayon de la boule centrée en v ne peut qu'augmenter quand $a_i(v) \leq a_i(w)$ pour tout sommet $w \in N_{\mathbf{G}}(v)$. Ainsi soit $Diam(G)$ le diamètre du graphe $\mathbf{G}$, si $a_i(v) > Diam(G)$, on peut étendre facilement nos preuves et déduire que $\mathbf{H}(M,i(v))$ et $Dir(\mathbf{G})$ sont isomorphes.

Nous avons montré précédemment (lemmes 2.25 et 2.26) qu'une fois que chacun des processus a un rayon de confiance plus grand que la taille du graphe, alors tous les processus ont la même boîte-aux-lettres et sont capables de reconstruire le même graphe dirigé $Dir(\mathbf{G})$ fibré sur $\mathbf{D}$. Le lemme suivant fait le lien entre la connaissance du degré pour chaque processus et la taille de sa fibre.

Lemme 2.30 ([BCG+96]) *Soit $\mathbf{D}$ un graphe dirigé étiqueté. On note $d_{(v,v')}$ (resp. $d_{(v',v)}$), le nombre d'arcs a tels que $s(a) = v$ et $t(a) = v'$ (resp. $s(a) = v'$ et $t(a) = v$) dans $\mathbf{D}$. Pour toute paire de sommets $v, v' \in V(D)$, il existe deux entiers $d_{(v,v')}$ et $d_{(v',v)}$ tels qu'étant donné un graphe simple $\mathbf{G}$, si $Dir(\mathbf{G})$ est fibré sur $\mathbf{D}$ par l'intermédiaire d'une fibration φ, alors $d_{(v,v')}|\varphi^{-1}(v)| = d_{(v',v)}|\varphi^{-1}(v')|$.*

Avec la connaissance initiale de son degré, un processus peut calculer, d'après le lemme 2.30, la taille de la fibre de chaque processus appartenant au graphe dirigé $\mathbf{D}(M(v))$ reconstruit depuis sa boîte-aux-lettres $M(v)$. Ainsi, chaque processus peut localement identifier les processus qui appartiennent à l'ensemble des candidats (voir la définition 2.8) du graphe $\mathbf{D}$ reconstruit. Donc, le processus élu est le sommet avec la plus petite identité de cet ensemble. Par conséquent, notre algorithme d'élection peut être trivialement adapté dans le modèle où chaque sommet connaît son degré (voir [BCG+96]) tout en gardant une complexité polynomiale et des communications asynchrones.

2.5 Extension à l'Environnement de Diffusion Synchrone

Dans cette section, nous proposons d'étendre nos résultats du modèle de communications asynchrones vers le modèle de communications synchrones.

2.5.1 Modèle de Diffusion Synchrone

Dans le modèle de communication par diffusions synchrones, il n'existe pas nécessairement d'horloge globale. Les processus, initialement dans l'état passif, sont activés en fonction de l'environnement de telle sorte que deux processus séparés par un chemin de longueur au plus deux ne peuvent pas émettre un message en même temps. Nous supposons que chaque processus qui a besoin d'émettre un message est activé. Lorsqu'un message est émis par un processus, il est instantanément entendu par tous les processus

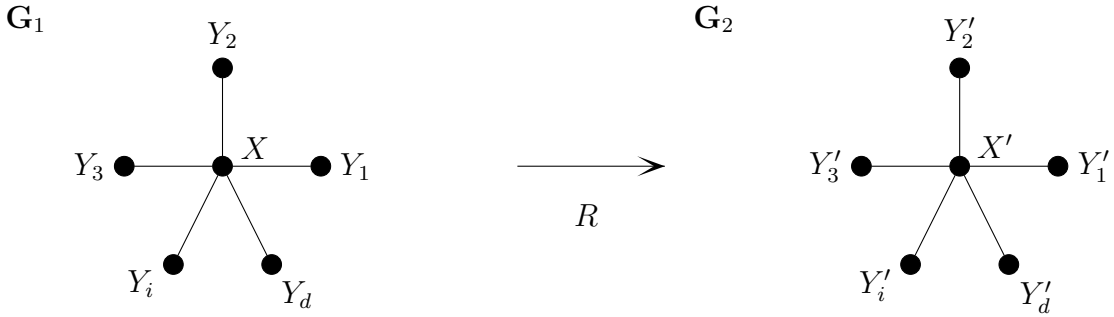

FIGURE 2.3 – Représentation graphique d'une règle dans le modèle synchrone où $Y'_i = f(X, Y_i)$ et $X' = g(X)$ où f, et g sont des fonctions de transition sur les états des sommets. La règle R est aussi notée $R = ((X, (\{Y_i \mid Y_i \in N_{\mathbf{G}_1}(v_0)\})), (g(X), (\{f(X, Y_i) \mid Y_i \in N_{\mathbf{G}_1}(v_0)\})))$.

voisins. Nous rappelons que tous les messages entendus par un processus arrivent sans erreur.

De façon informelle, le modèle de calcul associé à ce type de communication est appliqué à un graphe étiqueté et modifie les étiquettes d'un sous-graphe composé d'un sommet et de ses voisins, suivant des règles dépendantes de ce sous-graphe uniquement. Il est représenté par la forme donnée schématiquement en figure 2.3. Nous utilisons ici les notions de réétiquetage présentées dans le chapitre 1.

Soit $N_G(v)$, l'ensemble des voisins du sommet v dans G. Dans un graphe étiqueté $\mathbf{G}$, s'il est possible d'appliquer une étape de calcul sur une étoile centrée en v_0, qui est dans l'état X, alors cela produit un nouvel étiquetage de $\mathbf{G}$ comme suit : $\forall v_i \in N_G(v_0)$, si $\lambda(v_i) = Y_i$, alors Y_i devient $Y'_i = f(X, Y_i)$ et X devient $X' = g(X)$ où f, et g sont des fonctions de transition sur les états des sommets.

Plus précisément, soit $\mathcal{R}$, l'ensemble des règles comme définies précédemment. Soient $\mathbf{G}_1$ et $\mathbf{G}_2$, deux graphes étiquetés. On dit que $\mathbf{G}_1$ donne $\mathbf{G}_2$ en une étape, notée $\mathbf{G}_1 \mathcal{R} \mathbf{G}_2$, si et seulement s'il existe des sommets $\{v_0\} \cup N_{\mathbf{G}_1}(v_0)$, un règle $R = ((X, (\{Y_i \mid v_i \in N_{\mathbf{G}_1}(v_0) \wedge Y_i = \lambda(v_i)\})), (g(X), (\{f(X, Y_i) \mid v_i \in N_{\mathbf{G}_1}(v_0) \wedge Y_i = \lambda(v_i)\})))$ tels que :

- dans $\mathbf{G}_1$, X est l'étiquette du sommet v_0, Y_i est l'étiquette du sommet voisin v_i de v_0 ;
- $\mathbf{G}_2$ est obtenu depuis $\mathbf{G}_1$ en remplaçant l'étiquette X de v_0 par $X' = g(X)$ et l'étiquette Y_i du sommet voisin v_i de v_0 par $Y'_i = f(X, Y_i)$.

Les étiquettes de tous les autres sommets ne sont pas pertinentes pour une telle étape de calcul et demeurent inchangées. Comme expliqué précédemment, notre modèle assure que tous les messages sont transmis de manière effective.

Les sommets du graphe $\mathbf{G}$ qui modifient leurs étiquettes sont qualifiés comme étant *actifs* (et représentés par des cercles remplis de noir dans les figures), le sommet central des étoiles utilisé pour correspondre à la règle est qualifié de *passif*. Tous les autres sommets de $\mathbf{G}$ qui ne participent pas à une étape de calcul élémentaire sont au *repos* (ou *idle*).

2.5.2 Algorithmes Distribués dans le Modèle de Diffusion Synchrone

Un algorithme dans le modèle de diffusion synchrone est donné par un certain (peut-être infini, mais toujours récursif) ensemble de règles de la forme présentée par la figure 2.3.

Nous rappelons la notion de sérialisation introduite dans le chapitre 1. L'exécution d'un algorithme consiste en l'application de règles jusqu'au moment où aucune règle ne peut s'appliquer, ce qui termine l'exécution. Nous pouvons définir une manière distribuée de calcul comme suit. Deux étapes consécutives sur des supports (étoiles) disjoints peuvent être appliquées dans n'importe quel ordre (et concurremment). En conséquence, suivant les premières règles appliquées, plusieurs résultats sont possibles d'une exécution à l'autre.

Notre modèle est asynchrone, dans le sens où plusieurs étapes peuvent être effectuées au même moment, mais il n'est pas nécessaire que toutes celles-ci soient réalisées.

Nous considérons toujours les problèmes de l'énumération et de l'élection et nous présentons une caractérisation des graphes pour lesquels il existe une solution. Nous présentons les résultats d'impossibilité correspondants et nous montrons la façon de modifier les algorithmes $\mathcal{M}$ et $\mathcal{M}_e$ précédemment pour correspondre au modèle de diffusions synchrones.

Les résultats d'impossibilités sont basés sur le lemme de relèvement associé au modèle synchrone :

Lemme 2.31 (Lemme de Relèvement Synchrone) *Considérons un graphe dirigé $\mathbf{D}_1$ fibré discrètement sur un graphe dirigé $\mathbf{D}_2$ par l'intermédiaire d'une fibration φ. Soit $\mathcal{A}$ un algorithme basé sur le modèle de diffusion synchrone détaillé précédemment. S'il existe une exécution maximale ρ_2 de $\mathcal{A}$ sur $\mathbf{D}_2$ qui donne une configuration $\mathbf{D}'_2$, alors il existe une exécution maximale ρ_1 de $\mathcal{A}$ sur $\mathbf{D}_1$ qui donne une configuration $\mathbf{D}'_1$ de telle sorte que $\mathbf{D}'_1$ est fibré discrètement sur $\mathbf{D}'_2$ via φ.*

Preuve : Soient $\mathbf{D}_1 = (D_1, \lambda_1), \mathbf{D}_2 = (D_2, \lambda_1)$, deux digraphes tels que (D_1, λ_1) est fibré discrètement sur (D_2, λ_2) via φ. Il est suffisant de prouver ce lemme pour une exécution ρ de $\mathcal{A}$. Supposons que ρ est lancée sur le sommet actif $v \in V(\mathbf{D}_2)$ et nous notons λ'_2 le nouvel étiquetage de $\mathbf{D}_2$.

Considérons une exécution arbitraire pour ρ dans le modèle de diffusion synchrone. On considère une étape de l'exécution dans laquelle le sommet v émet un message dans $\mathbf{D}_2$. On peut ainsi relever cette exécution sur $\mathbf{D}_1$ dans laquelle chaque sommet de l'ensemble $\varphi^{-1}(v)$ émet le même message. On note alors λ'_1, le nouvel étiquetage de $\mathbf{D}_1$. Chaque sommet $w \in N_{\mathbf{D}_2}(v)$ entend k messages, avec k dépendant du nombre d'arcs $a \in A(\mathbf{D}_2)$ tels que $s(a) = v$ et $t(a) = w$. Puisque la relation φ est une fibration discrète, il n'y a pas de boucle et cela assure que deux processus voisins ne peuvent pas être dans le même état. Plus particulièrement, deux processus voisins ne peuvent pas émettre un message au même moment. Pour tout sommet $w' \in \varphi^{-1}(w)$, w' a k processus voisins dans $\varphi^{-1}(v)$ et entend k fois le même message. En ce sens, $\lambda'_1(w') = \lambda'_2(w)$ et les étiquettes de tous les autres sommets ne sont pas modifiées. Nous pouvons ainsi déduire qu'à la fin de cette étape, l'étiquetage λ'_2 of $\mathbf{D}_2$ peut être relevé sur l'étiquetage λ'_1 of $\mathbf{D}_1$. Nous pouvons donc conclure que le graphe dirigé (D_1, λ'_1) est fibré discrètement sur le graphe dirigé (D_2, λ'_2) via φ. □

Remarque 2.32 *Dans le modèle de diffusion synchrone, nous pouvons assurer que, à la fin de l'exécution, deux sommets voisins sont dans des états différents. Par conséquent, ils ne peuvent pas avoir la même image via φ. Ainsi, il n'existe pas de boucle et la relation φ est une* fibration discrète*, qui est plus restrictive qu'une fibration classique introduite dans les sections précédentes.*

2.5.3 Résultats d'Impossibilité

Comme précisé dans la Remarque 2.32, le modèle de communication synchrone diffère du modèle asynchrone par le fait que deux sommets voisins ne peuvent pas avoir la même étiquette. Nous en déduisons donc la proposition suivante :

Proposition 2.33 *Soit $\mathbf{G}$ un graphe étiqueté tel que $Dir(\mathbf{G})$ n'est pas minimal pour les fibrations discrètes (resp. discrètes et nt), alors il n'existe pas d'algorithme d'énumération (resp. d'algorithme d'élection) pour $\mathbf{G}$ dans le modèle de diffusion synchrone.*

Preuve : Cette proposition peut être prouvée en suivant le même modèle que pour la preuve de la proposition 2.10 en supposant que φ est une fibration discrète. Considérons que $Dir(\mathbf{G})$ est fibré discrètement sur $\mathbf{D}$ via φ. Etant donné un algorithme $\mathcal{A}$ dans le modèle synchrone, considérons une exécution particulière de $\mathcal{A}$ comme proposée dans le lemme 2.31

Dans ce modèle, deux processus voisins ne peuvent pas avoir la même image par φ puisqu'il n'existe pas de boucle. D'après cette hypothèse, on peut montrer facilement que $\mathcal{A}$ n'est pas un algorithme d'énumération pour $\mathbf{G}$. □

Remarque 2.34 *Par exemple, soit $\mathbf{G}$ le graphe présenté dans la figure 2.1 et $\mathbf{D}$ le graphe dirigé tel que $Dir(\mathbf{G})$ est fibré sur $\mathbf{D}$ par l'intermédiaire d'une fibration discrète φ. Si une étape de calcul dans le modèle synchrone est appliquée à un processus v, le résultat produit deux émissions d'un même message. Ce type de calcul ne peut donc pas être considéré comme une véritable exécution dans le modèle de diffusion synchrone : la processus w entend deux fois le même message. Ici, il faut juste garder à l'esprit que les fibrations ne sont qu'un outil mathématique permettant d'aider à la caractérisation de certaines propriétés dans l'algorithmique distribuée.*

2.5.4 L'Algorithme d'Énumération et d'Élection

Pour l'extension des algorithmes $\mathcal{M}$ et $\mathcal{M}_e$ (voir l'algorithme 3), nous devons assurer que pendant l'exécution, deux processus voisins ne peuvent pas être dans le même état. Pour cela, lorsqu'un processus v entend un message provenant d'un voisin qui a le même numéro, la même étiquette et la même vue locale, celui-ci peut changer son numéro. Cette condition doit donc être prise en considération et ajoutée à la condition effectuée permettant à un processus de changer son identité.

Algorithme 3: Extension des algorithmes d'énumération $\mathcal{M}$ et d'élection $\mathcal{M}_e$ pour le modèle de communication par diffusion synchrone.

R : {Un message $< (n', n'_{old}, M') >$ (resp. $< (n', n'_{old}, M', a') >$) est arrivé à v_0}
début
  [...]
  //ℓ' et N' sont, respectivement, l'étiquette initiale et la vue locale de processus ayant envoyé le message.
  $(\ell', N') := \max_{\prec}\{(\ell, N) \mid \exists (n, \ell, N) \in M'\}$;
  si $n(v_0) = 0$ *ou* $\exists (n(v_0), \ell, \mathcal{N}) \in M(v_0)$ *tel que* $(\lambda(v_0), N(v_0)) \prec (\ell, \mathcal{N})$ *ou* $(n(v_0) = n'$ *et* $(\ell', N') = (\lambda(v_0), N(v_0)))$ **alors**
    $n(v_0) := 1 + \max\{n \mid \exists (n, \ell, \mathcal{N}) \in M(v_0)\}$;
  [...]
fin

2.5.5 Correction de l'Algorithme dans le Modèle Synchrone

En considérant l'algorithme d'énumération $\mathcal{M}$, les lemmes 2.12 et 2.13 démontrés dans le cas des communications asynchrones restent satisfaits. Pour le lemme 2.14, un sommet v peut changer son numéro à l'étape i même s'il n'existe pas $(n_i(v), \ell, N) \in M(v)$ tel que $(\lambda(v), N_i(v)) \prec (\ell, N)$. Cependant, dans ce cas, nous savons qu'il existe $w \in N_G(v)$ tel que $n_{i+1}(v) = n_i(w) = n_i(v)$, $N_i(v) = N_i(w)$ et $\lambda(v) = \lambda(w)$. Par conséquent, l'identité $n_i(v)$ continue d'exister sur un sommet lorsque le processus v change son identité. Ainsi, le lemme 2.14 est toujours satisfait.

Toutefois, comme remarqué précédemment à plusieurs reprises, si deux sommets voisins ont des identités supérieures à zéro, l'exécution de l'algorithme assure qu'à aucun moment ils ont la même identité.

Lemme 2.35 *Pour tous sommets v et $w \in N_G(v)$ et toute transition i, $n_i(v) > 0 \Rightarrow n_i(w) \neq n_i(v)$*

Preuve : Considérons le modèle illustré par la figure 2.3 dans lequel un sommet change son état en fonction de son état précédent. Nous montrons ce lemme par récurrence sur le numéro d'étape i de l'exécution.

Initialement, la propriété est trivialement vraie. Supposons maintenant qu'elle est vraie pour $i \geq 0$. On sait que $(n_i(v), N_i(v)) \in M_i(v)$. De plus, lorsque $M(v)$ change, $M(v) = M(w)$, $\forall w \in N_G(v)$. À l'étape $i+1$, $n_{i+1}(v) = 1 + max\{n | \exists (n, M) \in M_i(v)\}$. Mais, $(n_i(w), N_i(w)) \in M_i(w) = M_i(v)$. Par conséquent, $n_{i+1}(v) > n_i(w) = n_{i+1}(w)$. Dans ce cas, à l'étape $i+1$, la propriété est satisfaite pour tout sommet. □

D'après les affirmations précédentes, nous en déduisons que le lemme 2.15 est lui aussi satisfait. Suivant le lemme 2.35, nous savons que dans la configuration finale, deux sommets adjacents sont dans des états différents et ont donc des identités différentes. Ainsi, le graphe dirigé **D** construit comme décrit dans la proposition 2.16 ne contient pas de boucles.

Concernant les lemmes 2.21, 2.24, 2.26, 2.25 pour la correction de l'algorithme $\mathcal{M}_e$, ils sont eux aussi satisfaits.

Par conséquent, en utilisant les mêmes techniques de caractérisation que dans le cas des communications asynchrones, nous en déduisons le théorème suivant :

Théorème 2.36 *Pour tout graphe simple étiqueté* $\mathbf{G}$*, les assertions suivantes sont équivalentes :*

1. *il existe un algorithme d'élection polynomial en mémoire, en nombre de messages et pour la taille des messages pour* $\mathbf{G}$ *utilisant des communications par diffusions asynchrones,*
2. *le graphe* $Dir(\mathbf{G})$ *est* nt*-minimal.*

Théorème 2.37 *Pour tout graphe simple étiqueté* $\mathbf{G}$*, les assertions suivantes sont équivalentes :*

1. *il existe un algorithme d'énumération (resp. d'élection) polynomial en mémoire, en nombre de messages et pour la taille des messages pour* $\mathbf{G}$ *utilisant des communications par diffusions synchrones,*
2. *le graphe* $Dir(\mathbf{G})$ *est minimal pour les fibrations discrètes (resp. et* nt*-minimal).*

2.5.6 Complexité en Temps des Algorithmes $\mathcal{M}$ et $\mathcal{M}_e$

Puisque dans le cas des communications synchrones, les temps d'émission et d'écoute des messages sont bornés, nous pouvons étendre l'analyse de la complexité proposée par la proposition 2.18 pour y ajouter l'analyse de la complexité en temps des algorithmes. Dans la proposition suivante, nous montrons que toute exécution des algorithmes $\mathcal{M}$ et $\mathcal{M}_e$ nécessite $O(Dn)$ unités de temps.

Proposition 2.38 *Pour tout graphe simple étiqueté* $\mathbf{G}$ *à* n *sommets et de diamètre* D*, toute exécution des algorithmes* $\mathcal{M}$ *ou* $\mathcal{M}_e$ *dans le modèle de diffusions synchrones nécessite* $O(Dn)$ *unités de temps.*

Preuve : On considère un graphe $\mathbf{G}$ à n sommets et de diamètre D. On considère une exécution ρ de l'algorithme $\mathcal{M}$ sur $\mathbf{G}$. D'après le lemme 2.14, on sait que chaque sommet ne modifie pas son numéro plus de n fois.

Si nous considérons la première étape dans laquelle un sommet change son numéro pour k, alors il faut D unités de temps pour que tout les autres sommets de $\mathbf{G}$ soient informés de ce changement. De plus, après $D+1$ unités de temps, chaque sommet v connaît $(n(v), N(v))$ et $(n(w), N(w))$ pour chaque sommet voisin w de v. Par conséquent, si après $D+1$ unités de temps, aucun sommet n'a changé son numéro, alors aucun autre message n'est émis/entendu et l'exécution ρ est terminée. D'après les remarques présentées dans la section 2.3.4 concernant l'étiquetage final, on sait que les identités attribuées sont comprises entre 1 et n. Par conséquent, chaque sommet a récupéré son identité finale après $n(D+1)$. Nous en déduisons alors que l'exécution ρ nécessite au plus $O(Dn)$ unités de temps. □

2.6 Conclusion

Dans ce chapitre, nous avons considéré un modèle de calcul pour les réseaux de diffusion multi-saut partiellement anonymes. Pour ce modèle, nous avons présenté les conditions nécessaires qui doivent être satisfaites par les graphes pour admettre des solutions pour les problèmes de nommage/énumération et d'élection. Nous avons montré que ces conditions sont suffisantes en donnant un algorithme d'énumération d'une part, et un algorithme d'élection d'autre part. Finalement, nous avons obtenu une caractérisation complète des graphes qui admettent des solutions pour ces problèmes.

Par ailleurs, nous avons étudié l'importance de la connaissance initiale et des hypothèses de communications. Dans un premier temps, nos algorithmes s'appuient sur des communications asynchrones. Les hypothèses ne requièrent pas que les communications soient FIFO. Afin de donner un identifiant unique à chacun des processus dans les graphes minimaux pour les fibrations, les processus n'ont pas besoin de connaître initialement leur degré s'ils connaissent la taille du réseau. Au contraire, cette combinaison de connaissances initiales (chaque processus connaît la taille du réseau, mais pas son degré) n'est pas suffisante pour résoudre le problème de l'élection dans les graphes nt-minimaux. Il suffit que les processus connaissent initialement une carte du graphe (ou sa base minimale). Nous avons aussi montré qu'il suffit que chaque processus connaisse son degré et la taille du réseau. Il reste donc ouvert de déterminer exactement quelle connaissance initiale à propos du réseau est nécessaire et/ou suffisante pour résoudre le problème de l'élection dans notre modèle.

Notre contribution se situe également au niveau de la complexité des algorithmes aussi bien au niveau du nombre de messages échangés, qu'au niveau de leur taille, de la mémoire nécessaire à chaque processus ou du temps d'exécution. Ainsi, nos algorithmes ont une complexité polynomiale concernant le nombre de messages et la mémoire de chaque processus. Nous rappelons que les algorithmes basés sur les vues proposés par Yamashita et Kameda [YK99] et Boldi *et al.* [BCG+96] requièrent que les processus échangent des messages de taille polynomiale et aient un nombre exponentiel de bits en mémoire.

Chapitre 3

État Global d'un Système Distribué Anonyme et Calcul de Propriétés Stables

Sommaire

Dans ce chapitre, nous étudions les problèmes du calcul d'état global (ou snapshot) et, plus généralement, de la détection de propriétés stables dans les systèmes totalement distribués et anonymes. Nous considérons le modèle classique à passage de messages dans lequel, dans une étape de calcul, chaque élément du système peut changer son état, envoyer ou recevoir un message à travers des liens de communication. La plupart des algorithmes existants pour résoudre le problème du calcul d'un état global supposent que les éléments du système ont des identifiants uniques ou qu'il existe un unique noeud initiateur. Ce travail concerne le calcul d'un état global dans les systèmes anonymes et plus généralement quelles sont les propriétés stables d'un système distribué qui peuvent être détectées anonymement par l'utilisation de snapshots locaux tout en autorisant des initiateurs multiples et en ne connaissant qu'une borne supérieure sur le diamètre du réseau.

3.1 Introduction

Récemment, Guerraoui et Ruppert [GR05], considérant qu'une grande majorité des travaux menés dans le domaine de l'algorithmique distribuée suppose que les processus ont des identifiants uniques, pose la question suivante : *Qu'est-il possible de calculer dans le cas où les processus n'ont pas d'identifiants uniques ou, pour des raison de vie privée, ne veulent pas divulguer leur identité ?*

3.1.1 Contexte du Problème

Un système distribuée (P, C) est composé d'un ensemble P de processus et d'un sous-système de communication noté C. Il est représenté par un graphe simple connexe et non dirigé $G = (V, E)$ dans lequel les sommets correspondent au processus et les arêtes correspondent aux canaux de communication bidirectionnels.

Un algorithme à base de passage de messages est défini comme suit : à chaque processus est associé un état et un système de transition pouvant modifier l'état du processus et pouvant interagir avec le sous-système de communication. Les événements qui sont associés aux processus sont les événements internes, les événements d'envoi et de réception. Lors d'un événement d'envoi (resp. de réception), un message est produit (resp. consommé).

On note par Q l'ensemble (récursif) des états possibles d'un processus p. Soit $\mathcal{M}$ l'ensemble des messages possibles. L'état d'un canal de communication est le multi-ensemble

des messages envoyés à travers ce canal et pas encore reçus. Soit p un processus, le snapshot local, aussi appelé *local snapshot*, de p est défini par l'état de p et par les états des tous les canaux entrants de p.

L'état d'un système distribué G, aussi appelé sont *état global*, est définis par l'état de chacun des processus et par l'état de chacun des canaux. De façon équivalente, l'état global d'un système est l'ensemble des snapshot locaux des éléments de ce système : c'est précisément un *snapshot global*. Ainsi, un snapshot d'un système distribué est un carte (ou photographie) instantanée de celui-ci, où chaque sommet (resp. chaque arête) est étiqueté par son propre état. Cette carte est représentée par un graphe $\mathbf{G} = (G, \lambda)$ pour lequel λ est la fonction d'étiquetage qui associe à chaque sommet (resp. chaque arête) sont état.

Par définition, dans un système totalement distribué asynchrone il n'y a pas d'horloge globale et, par conséquent, aucun processus n'a la connaissance d'un snapshot ; chaque processus, connaît seulement, a priori, son propre état. Il ne connaît ni les états des autres processus, ni les états des canaux de communication.

Étant donné un système distribué, le but d'un algorithme de snapshot est le calcul d'un tel état global. Comme présenté par Tel [Tel00] (p. 335-336) et en introduction de ce mémoire, la construction d'un snapshot est motivée par :

- la détection de propriétés stables d'un système distribué (propriétés qui, lorsqu'elles deviennent vraies, reste vraies par la suite),
- si le système doit être relancé (p. ex., *crashs* d'un ou plusieurs éléments du système), il peut être relancé depuis le dernier état global calculé (et non depuis le début de l'exécution),
- c'est une notion nécessaire dans le cas du débogage de systèmes distribués.

Un *snapshot consistant* d'un système distribué est un état global du système distribué ou un état global que le système aurait pu atteindre à un certain moment. Depuis le travail précurseur mené par Chandy et Lamport [CL85] qui présente un algorithme qui calcule des snapshot consistants, beaucoup de travaux proposent de tels algorithmes suivant le modèle de système distribué [KS08, Mat87, BT87, MS94, KW07, Ksh10]. Ils supposent que les processus ont des identifiants uniques et/ou qu'il existe exactement un seul initiateur. Beaucoup d'articles présentent des algorithmes spécifiques à la détection de certaines propriétés comme la terminaison ou les interblocages (*deadlocks*).

Dans ce chapitre, nous considérons cette question dans le contexte du calcul de snapshots et en considérant les propriétés stables d'un système distribué qui peuvent être détectées anonymement. De plus, nous voulons une solution totalement distribuée admettant plusieurs initiateurs.

3.1.2 Le Modèle

Comme présenté précédemment et en introduction de ce chapitre, notre modèle est le modèle à base de passage de messages asynchrone habituel tel que celui utilisé dans [Tel00, YK96b].

Un réseau est représenté par un graphe simple connexe $G = (V(G), E(G)) = (V, E)$ où les sommets correspondent aux processus et les arêtes aux liens de communication directs. L'état de chacun des processus v (resp. chacun des liens e) est représenté par une étiquette $\lambda(v)$ (resp. $\lambda(e)$) associée au sommet $v \in V(G)$ (resp. au lien $e \in E$) ; nous notons par

$\mathbf{G} = (G, \lambda)$ un tel graphe étiqueté. Nous supposons que le réseau est anonyme : la même identité peut être attribuée à plusieurs processus (voir le chapitre 2 pour plus de précision sur l'anonymat dans les systèmes distribués).

Nous supposons que chacun des processus peut distinguer les arêtes qui lui sont incidentes, c.-à-d., pour chaque $u \in V(G)$, il existe une bijection δ_u entre les arêtes incidentes à u et $[1, \deg_G(u)]$. Nous noterons par la suite δ l'ensemble des fonctions $\{\delta_u \mid u \in V(G)\}$. Les numéros associés par chacun des sommets à ses sommets voisins est appelé *numéros de port* et la fonction δ est appelée *numérotation des ports* de G. Nous noterons par $(\mathbf{G}, \delta)$ le graphe étiqueté $\mathbf{G}$ ayant la numérotation de ports δ.

Remarque 3.1 *Soit (G, λ) un graphe étiqueté avec la numérotation de ports δ. Les numéros de port des arêtes incidentes à un sommet peuvent être utilisés par un processus représenté par ce sommet pour mémoriser les étiquettes (états) associées aux arêtes incidentes correspondantes. Ainsi, dans la suite de ce chapitre, nous supposons que seulement les sommets du graphe sont étiquetés.*

Chaque processus v du réseau représente une entité capable de réaliser des étapes de calcul, d'envoyer des messages par l'intermédiaire d'un port et de recevoir des messages par l'intermédiaire d'un port qui a été envoyé au préalable par le processus voisin correspondant. Nous considérons des systèmes asynchrones, c.-à-d., aucune horloge globale n'est disponible et chaque calcul peut prendre un temps fini, mais non prévisible. Il est à noter que nous considérons seulement les systèmes fiables : aucune erreur ne peut se produire sur les processus ou les liens de communication. Nous supposons aussi que les canaux sont FIFO, c.-à-d., pour chaque canal, les messages sont transmis dans le même ordre que celui dans lequel ils ont été envoyés. Dans ce modèle, un algorithme distribué est donné par un algorithme local que tous les processus doivent exécuter. Un algorithme local est composé d'une séquence d'étapes de calcul entrelacées par des instructions d'envois et de réceptions de messages.

3.1.3 Résultats

Nous supposons que le réseau est anonyme et que plusieurs processus peuvent initialiser le lancement d'un calcul. Ainsi, il n'est pas toujours possible de calculer un snapshot, c.-à-d., de connaître une carte du réseau composée des sommets et des arêtes étiquetées par les états des processus et des canaux de communication, si le réseau est trop "symétrique" (voir détails dans l'introduction et dans le chapitre précédent). Ce résultat d'impossibilité est une conséquence directe du Théorème 5.5 présentée par Angluin dans [Ang80]. De plus, nous supposons que chacun des processus connaît une borne sur le diamètre du réseau.

D'abord, nous donnons un algorithme simple basé sur la composition d'un algorithme proposé par Szymanski, Shy, et Prywes [SSP85a] et de l'algorithme de Chandy et Lamport qui permet à chaque processus :

- de détecter un instant où tous les processus ont terminé de calculer leur snapshot local,
- d'associer le même numéro d'identification à tous les snapshots locaux.

De cette façon, nous obtenons deux applications : la première permet de mettre en oeuvre un mécanisme de points de reprise, la seconde permet de détecter la terminaison de l'exécution d'un algorithme distribué sous-jacent.

Ensuite, nous prouvons que les propriétés stables peuvent être détectées anonymement en montrant que nous pouvons calculer un snapshot à un revêtement près. Brièvement, soit (G, λ) un graphe étiqueté avec la numération de ports δ. Nous dénotons par $(Dir(\mathbf{G}), \delta)$ le graphe dirigé symétrique $(Dir(G), (\lambda, \delta))$ construit dans la façon suivante. Les sommets de $Dir(G)$ sont les sommets de G et ont la même étiquette dans $\mathbf{G}$ et dans $Dir(\mathbf{G})$. Chaque arête $\{u, v\}$ de G est remplacée dans $(Dir(\mathbf{G}), \delta)$ par deux arcs $a_{(u,v)}, a_{(v,u)} \in A(Dir(G))$ tels que $s(a_{(u,v)}) = t(a_{(v,u)}) = u$, $t(a_{(u,v)}) = s(a_{(v,u)}) = v$, et $\delta(a_{(v,u)}) = (\delta_v(u), \delta_u(v))$.

Un graphe dirigé D est un revêtement d'un autre graphe dirigé D' s'il existe un homomorphisme surjectif φ de D vers D' qui est localement bijectif sur les arcs.

Nous donnons un algorithme totalement distribué avec détection de la terminaison qui, étant donné un graphe étiqueté (G, λ) avec une numérotation de ports δ, permet à chacun des processus de calculer localement un graphe dirigé étiqueté $\mathbf{D}$ tel que $(Dir(\mathbf{G}), \delta)$ est un revêtement symétrique de $\mathbf{D}$. Nous prouvons que les propriétés stables comme la terminaison, les *deadlocks*, le *garbage collection* (ramasse-miette distribué) ou la perte de jetons, détectées en ayant la connaissance de (G, λ) peuvent être détectées en ayant simplement la connaissance de $\mathbf{D}$, appelé dans ce chapitre, un *weak snapshot*.

Remarque 3.2 *En un certain sens, un weak snapshot* $\mathbf{D}$ *est la "connaissance maximale" du système distribué que chacun des processus peut obtenir.*

Ainsi, si nous supposons que le réseau est anonyme et si nous autorisons tout sommet à être initiateur (en particulier plusieurs initiateurs simultanément), le calcul d'un snapshot n'est plus possible ; néanmoins le calcul d'un weak snapshot (un snapshot à un revêtement près) est possible, et tout processus peut détecter les deadlocks, les pertes de jetons ou les éléments non accessibles (garbage collection).

Les résultats présentés dans ce chapitre ont été obtenus en collaboration avec Jérémie Chalopin et Yves Métivier et une partie de ces résultats a été publiée dans [CMM12b].

3.1.4 Travaux Liés

Une vaste étude des notions et des algorithmes en rapport avec les calculs de snapshots, les mécanismes de détection de propriétés stables, de calculs de points de reprises avec restauration de données, est répertoriée dans [KS08].

D'un point de vue théorique, il est simple de savoir si l'état global d'un système distribué satisfait telle ou telle propriété stable. Pour cela, un processus identifié particulier lance une exécution de l'algorithme de Chandy-Lamport, puis il s'occupe de collecter les états des processus pour finalement tester si le réseau ainsi étiqueté satisfait une propriété donnée.

Afin de collecter ou d'analyser les snapshots locaux, plusieurs hypothèses différentes peuvent être faites (voir [KRS95] pour plus de détails sur ces hypothèses) : les processus ont des identifiants uniques, il y a exactement un seul processus initiateur ou un seul processus collecteur. L'analyse peut être faite grâce à l'utilisation d'une *vague*. Comme

expliqué dans [KRS95] : *une vague est un flot de messages de contrôle tel que tout processus dans le système est visité exactement une fois par un message de contrôle, et au moins un processus dans le système peut déterminer quand ce flot de messages termine.* De plus, les séquences de vagues peuvent être implémentées grâce à une structure transversale telle qu'un arbre ou un anneau. De telles possibilités n'existent pas sous nos hypothèses.

Certains travaux présentent des algorithmes spécialisés pour obtenir des solutions efficaces pour détecter les propriétés stables d'un réseau [Mat87, BT87, MS94, KW07, Ksh10]. Dans tous les cas, il est supposé que les processus ont des identifiants uniques et/ou qu'il n'existe qu'un seul initiateur.

Quelques résultats ont cependant été obtenus pour le calcul de snapshots dans le cas des systèmes asynchrones à mémoires partagées et anonymes : [GR05] (Section 5) présente une étude des ces solutions. Cet article y présente aussi des résultats concernant le consensus et l'horodatage (timestamping).

La question "Qu'est-il possible de calculer anonymement ?" a été explorée dans le modèle asynchrone à base de passage de message pour les problèmes de l'élection, de symétries et, plus généralement, pour les fonctions calculables [Ang80, BCG$^+$96, JS85, YK96b, YK96a, YK99, BV99]. Angluin a introduit les techniques de preuves classiques utilisées pour montrer des résultats d'impossibilité basées sur les revêtements de graphes. Puis, les caractérisations des graphes pour lesquels il existe un algorithme d'élection ont été obtenues. Comme précisé dans le chapitre précédent, deux outils principaux sont utilisés dans ces travaux : les revêtements et les vues. Ces outils s'avèrent incontournables dans la définition de propriétés et pour délimiter précisément la frontière entre les résultats positifs et négatifs.

3.1.5 Résumé du Chapitre

D'abord, nous détaillons, dans la section 3.2 , le modèle à base de passage de messages. Puis dans la section 3.3 , nous donnons les définitions essentielles concernant la notion de snapshots. La section 3.4 présente un algorithme de détection de la terminaison de l'algorithme de Chandy-Lamport avec la connaissance d'une borne supérieure sur le diamètre du réseau. Suite à ce premier résultat, la section 3.5 donne deux applications : "calcul de points de reprise et reprises sur incident" et "détection de la terminaison". Nous montrons, dans la section 3.6, le lien qu'il existe entre les revêtements de graphes, les propriétés stables et les snapshots. Nous introduisons la notion de weak snapshots et la section 3.7 prouve que les weak snapshots peuvent être calculés anonymement avec détection de la terminaison en connaissant une borne supérieure sur le diamètre du réseau.

3.2 Précisions sur le Modèle de Passage de Messages

La présentation et les définitions données dans cette section sont tirées de l'ouvrage de Tel [Tel00] (p. 45-47) ou d'Attiya [AW04] (p. 10-12).

Système de Passage de Messages

Les processus communiquent en échangeant des messages de façon asynchrone et chaque processus sait à travers quel canal il reçoit ou il envoie un message : une arête entre deux processus p_1 et p_2 (ou sommets v_1 et v_2) représente un canal connectant un port i de p_1 (ou v_1) à un port j de p_2 (ou v_2) ; ce canal sera noté par p_1p_2 (v_1v_2) ou $c_{p_1,i} = c_{p_2,j}$.

Algorithme de Passage de Messages

Remarque 3.3 *D'une façon générale, les noms des processus ne sont pas accessibles par les processus eux-mêmes. Néanmoins, afin de faciliter les présentations, un message m en transit est noté par (p, m, p') où p est le processus qui envoie le message et p', le processus qui le reçoit.*

Soit $\mathcal{M}$ l'ensemble des messages possibles. Soit p, un processus. L'algorithme local du processus p noté $\mathcal{D}_p$, est défini par :
- l'ensemble (récursif) Q des états possibles de p,
- le sous-ensemble I de Q des états initiaux,
- l'état initial de p est identifié par $\lambda(p)$,
- une relation $\vdash_p$ des événements (événements internes, événement d'envoi, événement de réception).

Soit p un processus. Soit q un processus voisin de p, c.-à-d., pq est un canal de communication. Soit M le multi-ensemble des messages en transit (initialement M est vide). L'état associé à p est noté par $\texttt{state}(p)$. Le multi-ensemble des messages en transit associé au canal pq est noté par M_{pq}.

La transition associée au processus p est notée par :

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

(où c et d sont des états, in et out des entiers, et m et m' des messages). Ce qui signifie que :
- si $in = out = 0$, alors $m = m' = \perp$ (m et m' sont indéfinis) : c'est un événement interne, le nouvel état du processus p est d (préalablement, c'était c) ;
- si $in \neq 0$, alors $out = 0$ et $m' = \perp$ (m' est indéfini) : c'est un événement de réception, l'état du processus p était c, p a reçu le message m par l'intermédiaire du port in et son nouvel état est d ; une occurrence de m (de la forme (p', m, p) où $\delta_p(p') = in$) est retirée de M et donc de $M_{p'p}$.
- si $out \neq 0$, alors $in = 0$ et $m = \perp$ (m n'est pas défini) : c'est un événement d'envoi, initialement, l'état du processus p est égal à c ; après la transition, l'état est égal d et le message m' est envoyé par l'intermédiaire out ; une occurrence de m' (de la forme (p, m', p') où $\delta_p(p') = out$) est ajoutée à M et ainsi $M_{pp'}$.

Un algorithme dans lequel les processus échangent des messages $\mathcal{D}$ pour le système de passage de messages (P, C, λ) est une collection d'algorithmes locaux $\mathcal{D}_p$, un pour chacun des processus $p \in P$. Il est noté par $\mathcal{D} = (\mathcal{D}_p)_{p \in P}$. Un événement d'un tel algorithme est défini par un événement e_p sur un processus p. Le $i$ème événement sur un processus p est noté $e_p^{(i)}$.

Exécution d'un Algorithme de Passage de Messages

Une exécution $\mathcal{E}$ d'un algorithme à base de passage de messages est définie par une séquence $(\texttt{state}_0, M_0), (\texttt{state}_1, M_1), \ldots, (\texttt{state}_i, M_i), \ldots$ telle que :
- pour chaque i, M_i est le multi-ensemble des messages en transit,
- $M_0 = \emptyset$,
- pour tout i et pour tout processus p, $\texttt{state}_i(p)$ représente l'état du processus p,
- pour tout processus p, $\texttt{state}_0(p) = \lambda(p) \in I$. Dans la cas du calcul du snapshot $\lambda(p)$ est l'étiquetage de l'algorithme sous-jacent,
- pour tout i, il existe un unique processus p tel que :
  - si $p' \neq p$, alors $\texttt{state}_{i+1}(p') = \texttt{state}_i(p')$,
  - $\texttt{state}_{i+1}(p)$ et M_{i+1} sont obtenus depuis $\texttt{state}_i(p)$ et M_i par un événement sur le processus p.

Par définition, $(\texttt{state}_i, M_i)$ est une configuration. L'exécution $\mathcal{E}$ de l'algorithme de passage de messages est définie par :

$$\mathcal{E} = (\texttt{state}_i, M_i)_{i \geq 0}.$$

Concernant les événements des processus, l'ordre causal local est défini par :

$$e_p^{(i)} \prec_p e_p^{(j)} \text{ if i} < \text{j}.$$

L'ordre causal $\prec$ est défini comme la plus petite relation qui satisfait :
- si $e_p^{(i)} \prec_p e_p^{(j)}$, alors $e_p^{(i)} \prec e_p^{(j)}$,
- si s est un événement d'envoi et t est l'événement de réception correspondant alors $s \prec t$,
- $\prec$ est transitive.

Deux exécutions sont équivalentes si les ordres causaux associés sont les mêmes. Un calcul $\mathcal{C}$ est une classe d'équivalence de cette relation.

Une configuration finale est une configuration dans laquelle plus aucun événement n'est applicable. Nous pouvons noter que dans une configuration finale, le multi-ensemble M des messages en transit est vide.

3.3 Précisions sur la Notion de Snapshot

Dans cette section, nous présentons les notions relatives à un snapshot suivant les définitions précédentes et le modèle que nous considérons dans ce chapitre. Ces notions sont, elles aussi, adaptées de [Tel00].

3.3.1 Quelques Définitions

Soit $\mathcal{C}$ un calcul d'un algorithme à base de passage de messages $\mathcal{D}$. Nous rappelons qu'un snapshot d'un système distribué est une carte du réseau où les sommets et les arêtes sont étiquetés par les états des processus et des canaux de communication correspondants. Ainsi, un snapshot S^* est défini naturellement par les états locaux des processus, c.-à-d.,

par l'état `state`(p) de chacun des processus p, et par les états des canaux de communication, c.-à-d., par le multi-ensemble M_{qp} de chaque canal reliant le processus q au processus p.

Pour chacun des processus p, l'état `state`(p) et $\{M_{qp} | q \in N(p)\}$ (l'ensemble de canaux de communication incidents à p) définissent le snapshot local du processus p.

Cet ensemble d'états forme la configuration γ^* associée au snapshot S^*.

Un snapshot est réalisable si pour chaque canal pq, depuis le processus p au processus q, l'ensemble des messages reçus par q est un sous-ensemble de l'ensemble des messages envoyés par p.

Soit E_v l'ensemble des événements de $\mathcal{C}$, $E_v = \{e_p^{(i)} \mid \text{p est un processus et } i \geq 0\}$ (l'ensemble E_v est l'union des événements locaux de chaque processus p).

Remarque 3.4 *Nous rappelons un des exemples proposés dans [CL85]. Considérons un système composé de deux processus p et q. Soit A un algorithme dans lequel les processus s'échangent un simple jeton. Supposons que p prenne son snapshot local à l'étape i durant laquelle il possède le jeton et que q prenne son snapshot local à l'étape j, $j < i$ lorsque lui aussi possède le jeton. Le snapshot ainsi calculé consiste alors en la composition des snapshot locaux des deux processus. Cette configuration calculée contenant deux jetons n'est pas une situation possible du système.*

C'est alors qu'interviennent les notions de coupe *et* consistance.

Une coupe de E_v est un ensemble $L \subseteq E_v$ tel que :

$$(e \in L) \text{ and } e' \preccurlyeq_p e \Rightarrow e' \in L.$$

Une coupe consistante de E_v est un sous-ensemble L de E_v tel que :

$$(e \in L) \text{ and } e' \preccurlyeq e \Rightarrow e' \in L.$$

Un snapshot S^* est significatif pour un calcul $\mathcal{C}$ s'il existe une exécution $\mathcal{E} \in \mathcal{C}$ telle que γ^* est une configuration de $\mathcal{E}$, dans laquelle γ^* est la configuration associée à S^*.

Des exemples de coupes d'un système distribué sont illustrés par la figure 3.1.

Finalement, nous rappelons le Théorème 10.5 ([Tel00], p. 339) :

Théorème 3.5 *[Tel00] Soient S^* un snapshot et L la coupe induite par S^*, les assertions suivantes sont équivalentes :*

1. *S^* est réalisable,*
2. *L est une coupe consistante,*
3. *S^* est significatif.*

Remarque 3.6 *Finalement, un snapshot réalisable correspond à un instantané d'un état global du système ou d'un état global que le système pourrait avoir atteint.*

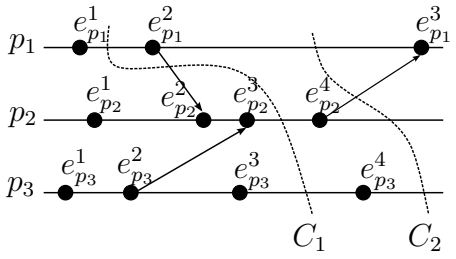

FIGURE 3.1 – Diagramme d'espace-temps [KS08] représentant un système distribué composé de trois processus p_1, p_2 et p_3. Les lignes horizontales représentent la progression d'un processus, un point indique un événement et une flèche correspond au transfert d'un message. L'événement d'envoi $e^2_{p_1}$ du processus p_1 n'appartient pas à la coupe C_1 tandis que l'événement de réception correspondant $e^2_{p_2}$ de p_2 appartient à la coupe : C_1 n'est pas consistante. La coupe C_2 est consistante.

3.3.2 L'Algorithme de Chandy-Lamport

Le but d'un algorithme de snapshot est de construire une configuration du système définie par l'état de chacun des processus et de l'état de chacun des canaux de communication.

Cette section présente l'algorithme de snapshot proposé par Chandy-Lamport [CL85] ; celui-ci est décrit par l'algorithme 4.

Chaque processus p se voit attribué :

- une variable booléenne $taken_p$, initialisée à $faux$, qui indique si le processus p a déjà calculé son état ;
- une variable booléenne $local\text{-}snapshot_p$, initialisée à $faux$, qui indique si le processus p a enregistré sont état et l'état de ses canaux entrants ;
- un multi-ensemble des messages $M_{p,i}$. Initialement, $M_{p,i} = \emptyset$, pour chaque canal entrant i de p.

Nous supposons que l'algorithme 4 est initialisé par au moins un processus qui : sauve son état, envoie un marqueur à travers chacun de ses ports sortants et pour chaque canal entrant, mémorise les messages qui arrivent jusqu'au moment où il reçoit un marqueur par ce même port. Lorsqu'un processus reçoit un marqueur pour la première fois, il effectue les mêmes opérations que dans le cas d'un initiateur ; le canal entrant par lequel il a reçu le marqueur pour la première fois est marqué comme étant vide. Un exemple d'exécution est donné par la figure 3.2.

Si nous considérons une exécution de l'algorithme de Chandy-Lamport, nous obtenons un snapshot consistant en un temps fini après son initialisation par au moins un processus (voir [Tel00] théorème 10.7). En particulier :

Fait 3.7 *En un temps fini après l'initialisation de l'algorithme de Chandy-Lamport, chaque processus p a calculé son snapshot local ($local\text{-}snapshot_p = vrai$).*

Une fois que le calcul des snapshots locaux est effectué, c.-à-d., pour chaque processus p, la variable booléenne $local\text{-}snapshot_p$ est $vraie$, la connaissance du snapshot est tota-

Algorithme 4: L'algorithme de snapshot de Chandy-Lamport.

I*nit-CL*$_p$: {Pour initialiser l'algorithme par au moins un processus p tel que $taken_p = faux$}
début
  enregistrer(`state`(p)) ;
  $taken_p := vrai$;
  envoyer$< mkr >$ à chaque voisin de p ;
  Pour chaque port i, le processus p **mémorise** les messages arrivant via i ;
fin

R-*CL*$_p$: {Un marqueur est arrivé à p via le port j}
début
  recevoir$< mkr >$;
  marquer le port j;
  si *non* $taken_p$ **alors**
    $taken_p := vrai$;
    enregistrer(`state`(p)) ;
    envoyer$< mkr >$ via chaque port ;
    Pour chaque port $i \neq j$, le processus p **mémorise** les messages arrivant via i dans $M_{p,i}$;
  sinon
    Le processus p **cesse de mémoriser** les messages provenant du canal identifié par j de p;
    enregistrer($M_{p,j}$) ;
  si *p a reçu un marqueur via tous ses canaux entrants* **alors**
    local-snapshot$_p := vrai$;
fin

lement distribuée sur le système. Il est alors naturel de se demander "comment exploiter cette connaissance distribuée ?".

Une première solution est obtenue par la construction de l'état global du système par un processus identifié et centralisé. Comme précisé par Raynal [Ray88] : *fournir un algorithme pour le calcul d'un état global est un problème basique dans les systèmes distribués.* Plusieurs hypothèses peuvent être faites pour obtenir un état global : exactement un initiateur pour l'algorithme de Chandy-Lamport, les processus ont des identifiants uniques ou des couleurs globales associées à chaque calcul d'un état global...

Une horloge globale peut être simulée par des horloges locales logiques [Ray88], néanmoins cette solution ne permet pas d'itérer sur les snapshots préalablement calculés.

Un autre moyen d'exploiter cette connaissance locale est basé sur les algorithmes de vague : un message est transmis à chacun des processus par un initiateur unique suivant la topologie du réseau ou une topologie virtuelle (anneau, arbre, graphe complet, ...), voir [MC98].

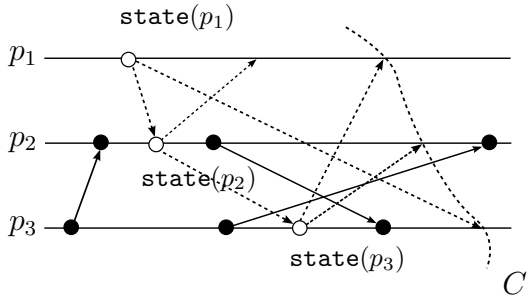

FIGURE 3.2 – Un diagramme d'espace-temps d'une exécution de l'algorithme 4 entre trois processus voisins. Le processus p_1 initialise l'exécution : il sauve son état $\texttt{state}(p_1)$ puis envoie un marqueur à travers chacun de ses ports sortants. Pour chaque canal entrant, il mémorise les messages qui arrivent jusqu'à la réception d'un marqueur sur chacun d'entre eux. Une fois que p_2 reçoit un marqueur, il sauve son état $\texttt{state}(p_2)$ et envoie à son tour un marqueur. Il est à noter que la réception du marqueur par p_2 sauve le canal correspondant comme étant vide. Une fois qu'un processus p_i reçoit un marqueur, il sauve l'état du canal. Dès que tous les marqueurs ont été reçus, le processus p_i a calculé son snapshot local, c.-à-d., $\textit{local-snapshot}_p = \textit{vrai}$. C est la coupe correspondant à l'exécution.

Ces solutions ne sont malheureusement pas disponibles dans le contexte des réseaux anonymes avec aucun processus distingué et aucune topologie particulière.

3.4 Détection de le Terminaison de l'Algorithme de Chandy-Lamport

Le premier problème réside dans la détection de la terminaison du calcul de tous les snapshots locaux. Il est requis que tous les processus certifient, en un temps fini, qu'ils aient terminé le calcul de leur snapshot local.

L'algorithme proposé par Szymanski, Shy, and Prywes [SSP85a] (ou SSP) présenté dans le chapitre 2 réalise cette certification pour une région d'un diamètre pré-spécifié. L'algorithme suppose qu'une borne supérieure sur le diamètre du réseau entier est connue par chacun des processus. Dans la suite, cette borne supérieure est notée β et nous faisons l'hypothèse que chaque processus connaît cette borne. Nous présentons ici une autre variante de l'algorithme SSP.

3.4.1 L'Algorithme SSP

Nous considérons qu'un algorithme distribué est terminé dès lors que tous les processus ont atteint lors condition de terminaison. Dans le cas de l'algorithme de Chandy-Lamport présenté dans la section précédente, cette condition est satisfaite lorsque la variable $\textit{local-snapshot}_p$ est vraie. L'algorithme de SSP va donc détecter ici l'instant pour lequel le calcul du snapshot est entièrement terminé.

Soit G un graphe, nous associons à chaque processus p un prédicat $P(p)$ et un entier $a(p)$ correspondant à son *rayon de confiance*. Initialement, $P(p)$ est *faux* et la valeur de $a(p)$ est -1. Les modifications des valeurs de $a(p)$ sont définies par les règles suivantes.

Chaque calcul local agit sur l'entier $a(p_0)$ associé au processus p_0 ; la nouvelle valeur de $a(p_0)$ dépend des valeurs associées aux voisins de p_0. Plus précisément, soit p_0 un processus et $\{p_1, ..., p_d\}$ l'ensemble des processus voisins de p_0.

- si $P(p_0) = faux$ alors $a(p_0) = -1$,
- si $P(p_0) = vrai$ alors $a(p_0) = 1 + min\{a(p_k) \mid k \in [0; k]\}$.

Nous supposons que pour chaque processus p la valeur de $P(p)$ devient vraie et reste vraie pour toujours. Afin d'appliquer l'algorithme SSP, nous ajoutons deux éléments à l'étiquette de chaque processus :

- $a(p) \in \mathbb{Z}$ est un compteur et, initialement $a(p) = -1$. $a(p)$ représente la distance jusqu'à laquelle tous les processus ont le prédicat satisfait.
- $A(p) \in \mathcal{P}_{\text{fin}}(\mathbb{N} \times \mathbb{Z})$ code les informations que p possède à propos des valeurs de $a(q)$ pour chaque processus voisin q. Initialement, $A(p) = \{(i, -1) \mid i \in [1, \deg_G(p)]\}$.

Un description précise est donnée par l'algorithme 5.

Algorithme 5: L'algorithme de détection de la terminaison SSP.

I*nit-SSP*$_p$: {Pour initialiser la détection de la terminaison au processus p tel que $P(p) = true$ }
début
  $a(p) := 0$;
  $m := Min\{x \mid (i, x) \in A(p)\}$;
  si $m \geq a(p)$ **alors**
    $a(p) := m + 1$;
  envoyer$< a(p) >$ à chaque processus voisin de p ;
fin

R-*SSP*$_p$: {Un entier $< \alpha >$ est arrivé à p via le port j}
début
  recevoir$< \alpha >$;
  $A(p) := (A(p) \setminus \{(j, x)\}) \cup \{(j, \alpha)\}$;
  $m := Min\{x \mid (i, x) \in A(p)\}$;
  si $(m \geq a(p)$ *et* $P(p) = true)$ **alors**
    $a(p) := m + 1$;
  si $a(p) \geq \beta$ **alors**
    p détecte la terminaison complète : la prédicat P est satisfait pour chaque processus ;
  sinon
    envoyer$< a(p) >$ à chaque processus voisin de p ;
fin

Nous considérons une exécution $\mathcal{E}$ de l'algorithme SSP sur le graphe **G** et $(a_i(p))_{i \geq 0}$

la séquence définie par les valeurs successives de $a(p)$ pour l'exécution $\mathcal{E}$. (Soit p un processus, $N(p)$ représente l'ensemble des processus voisins de p). Dans le lemme suivant nous montrons quelques propriétés satisfaites par toute exécution de l'algorithme SSP.

Lemme 3.8 *Nous avons :*

1. $a_i(p) \leq i$,
2. $a_{i-1}(p) \leq a_i(p)$,
3. $q \in N(p) \Rightarrow |a_i(p) - a_i(q)| \leq 1$,
4. *si* $h = a_i(p) \geq 0$ *alors* $\forall q \in V(G)\ d(p,q) \leq h \Rightarrow a_{i-h}(q) \geq 0$.

Preuve : Nous prouvons ce lemme par récurrence sur i. Soit p un processus, $a_0(p) = -1$, ainsi $a_0(p) \leq 0$. Soit k_0 tel que $a(k_0) = Min\{a(p_k) \mid 1 \leq k \leq d\}$. Nous avons $a(k_0) \leq i$, ainsi $a_{i+1}(p) \leq i+1$, et la première propriété du lemme est satisfaite.

Évidemment, $a_0(p) \leq a_1(p)$ et $q \in N(p) \Rightarrow |a_1(p) - a_1(q)| \leq 1$. Pour l'étape de récurrence, soit $i > 1$.. Nous supposons que $\forall p \in V(G)\ a_{i-1}(p) \leq a_i(p)$ et $q \in N(p) \Rightarrow |a_i(p) - a_i(q)| \leq 1$. Soit p un processus. Si l'étape i de l'algorithme SSP n'est pas appliquée sur p, alors $a_{i+1}(p) = a_i(p)$ et l'entier associé au processus p vérifie les propriétés *2* et *3* pour $i+1$.

Supposons maintenant que l'étape i de l'algorithme SSP est appliquée sur p et, plus précisément, il calcule une nouvelle valeur $a_{i+1}(p)$. Si $a_i(p) = Min\{x \mid (j,x) \in A(p)\}$, alors $a_{i+1}(p) = a_i(p) + 1$ et la propriété *2* est vérifiée.

Comme $|a_i(p) - a_i(q)| \leq 1$ et $a_i(p)$ est le minimum, on a $\forall q \in N(p), q \neq p$ et soit $a_{i+1}(q) = a_i(q) = a_i(p) = a_{i+1}(p) - 1$ ou $a_{i+1}(q) = a_i(q) = a_i(p) + 1 = a_{i+1}(p)$; par conséquent la propriété *3* est vérifiée.

Il reste à prouver la propriété *4*. Si $i = 1$ alors la propriété est triviale. Supposons que $h = a_{i+1}(p) = a_i(p)$. Si $d(q,p) \leq h$, alors $a_{(i+1)-h}(q) \geq a_{i-h}(q)$ et par récurrence $a_{i-h}(q) \geq 0$. Le cas restant est lorsque $h = a_{i+1}(p) = a_i(p) + 1$. Si $d(q,p) < h$ et $q \neq p$, alors $a_{(i+1)-h}(q) = a_{i-h}(q)$ et le résultat suit par récurrence. Si $d(q,p) = h$, puisque $a_{i+1}(p) = a_i(p) + 1$, on a $\forall q \in N(p)\ a_i(q) \geq a_i(p)$. De plus, il existe $p' \in N(p)$ tel que $d(p',q) = h - 1 \leq a_i(p')$. En s'aidant une fois de plus de l'hypothèse de récurrence, avec p' et $h-1$, nous terminons la preuve. □

D'après les propriétés *2* et *4* du lemme précédent, nous en déduisons immédiatement que le prédicat P est vrai pour chaque processus de la boule de rayon $a_i(p)$ centrée en p, c.-à-d. :

Proposition 3.9 *Soit p un processus de G, nous supposons que $h = a_i(p) \geq 0$. Alors,*

$$\forall q \in V(G) \quad d(p,q) \leq h \Rightarrow a_i(q) \geq 0.$$

Ainsi le processus p tel que $a(p)$ est supérieur ou égal au diamètre du graphe est capable de savoir que pour chaque processus q du graphe $P(q)$ est vrai, c.-à-d., il détecte la terminaison de l'algorithme.

3.4.2 Un Algorithme pour la Détection de la Terminaison du Calcul des Snapshots Locaux

Maintenant, nous composons l'application de l'algorithme de Chandy-Lamport avec l'algorithme SSP afin de permettre à chaque processus de détecter un instant dans lequel tous les processus ont terminé le calcul de leur snapshot local : une description précise est donnée par l'algorithme 6.

Afin d'appliquer correctement l'algorithme SSP, nous ajoutons trois éléments à l'étiquette de chaque processus :

- $a(p) \in \mathbb{Z}$ est un compteur. Initialement, $a(p) = -1$. Dans un certain sens, $a(p)$ représente la distance jusqu'à laquelle les processus ont terminé le calcul de leur snapshot local ;
- $A(p) \in \mathcal{P}_{\text{fin}}(\mathbb{N} \times \mathbb{Z})$ code l'information que le processus p possède à propos de la valeur $a(q)$ pour chacun de ses voisins q. Initialement, $A(p) = \{(i, -1) \mid i \in [1, \deg_G(p)]\}$;
- $\textit{snapshot-number}_p$ indique le numéro du snapshot. Initialement, $\textit{snapshot-number}_p = 0$; pour cette application cette variable n'est pas nécessaire. Elle sera utilisée dans les sections suivantes.

Si un processus p a terminé le calcul de son snapshot local, alors il change la valeur de $a(p)$ pour 0 et il en informe ses processus voisins. Lorsque p reçoit une valeur $a(q)$ depuis un certain processus voisin q par l'intermédiaire du port i, alors il substitue la nouvelle valeur $(i, a(q))$ par l'ancienne valeur (i, x) dans $A(p)$. Finalement, p calcule la nouvelle valeur $a(p) = 1 + Min\{x \mid (i, x) \in A(p)\}$.

Un processus p sait que chaque processus a terminé le calcul de son snapshot local aussitôt que $a(p) \geq \beta$ (nous rappelons que β est une borne supérieure sur le diamètre du graphe). Ainsi, nous ajoutons une variable booléenne *snasphot* initialisée à *faux* et qui indique si le processus sait si tous les processus ont terminé le calcul de leur snapshot local.

D'après le fait 3.7 et la proposition 3.9, nous en déduisons que :

Proposition 3.10 *Soit (G, λ) un réseau. Soit β une borne supérieure sur le diamètre de G connue par chacun des processus. En un temps fini après l'initialisation de l'algorithme de Chandy-Lamport, l'algorithme 6 permet à chaque processus de détecter le moment où tous les processus ont terminé le calcul de leur snapshot local.*

3.5 Deux Applications : "Calcul de Points de Reprise" et "Détection de la Terminaison"

Cette section présente deux applications simple de l'algorithme de Chandy-Lamport et de l'algorithme 6 dans le contexte des réseaux anonymes sans que l'unicité d'un initiateur soit requise et en supposant que chaque processus connaît une borne supérieure sur le diamètre du réseau :

1. pour calculer des configurations afin de redémarrer le système lorsqu'un processus échoue (voir [KS08] au chapitre 13 pour une présentation complète des différentes

Algorithme 6: Détection de la terminaison de l'algorithme de snapshot de Chandy-Lamport.

I*nit-SSP*$_p$: {Pour initialiser la détection de la terminaison au processus p tel que *local-snapshot*$_p = vrai$, $a(p) = -1$ et *snapshot* $= faux$}
début
  $a(p) := 0$;
  $m := Min\{x \mid (i,x) \in A(p)\}$;
  si $m \geq a(p)$ **alors**
    $a(p) := m + 1$;
  envoyer$< a(p) >$ à tous les voisins de p ;
fin

R-*SSP*$_p$: {Un entier $< \alpha >$ est arrivé à p via le port j}
début
  recevoir$< \alpha >$;
  $A(p) := (A(p) \setminus \{(j,x)\}) \cup \{(j,\alpha)\}$;
  $m := Min\{x \mid (i,x) \in A(p)\}$;
  si $(m \geq a(p)$ *et local-snapshot*$_p = vrai)$ **alors**
    $a(p) := m + 1$;
  si $a(p) \geq \beta$ **alors**
    snapshot-number$_p$:= *snapshot-number*$_p + 1$;
    snapshot$_p := vrai$;
  sinon
    envoyer$< a(p) >$ à tous les voisins de p ;
fin

méthodes de calcul de point de reprise),

2. pour détecter la terminaison de l'exécution d'un algorithme distribué.

3.5.1 Une Application pour le Calcul de Points de Reprise

Un snapshot permet de redémarrer un système lorsqu'il est dans une situation d'échec. Comme expliqué dans [KS08] p. 456 :

Définition 3.11 *L'état sauvegardé lors du calcul d'un snapshot est appelé un* point de reprise (ou checkpoint) *et la méthode qui consiste à redémarrer depuis un point de reprise précédent est appelée* rollback recovery.

Un description précise de notre solution est donnée par l'algorithme 7. Notre solution est obtenue par l'exécution répétée des étapes suivantes :

1. au moins un processus décide d'initier une exécution de l'algorithme de Chandy-Lamport (algorithme 4) ;

2. chaque processus p détecte un instant où le calcul de son snapshot local est terminé, c.-à-d., lorsque $local\text{-}snapshot_p = vrai$;
3. chaque processus p détecte un instant où le calcul de tous les snapshots locaux est terminé : $snapshot_p = vrai$ (algorithme 6) ;
4. un nouveau numéro (obtenu en incrémentant de 1 le compteur $snapshot\text{-}number_p$; initialement $snapshot\text{-}number_p = 0$) est associé à ce snapshot et chaque processus p donne ce numéro à son snapshot local. Chacun des processus p sauvegarde son dernier snapshot local correspondant au numéro $snapshot\text{-}number_p$. Cela permet de redémarrer le système depuis une configuration stable dans le cas d'un échec ;
5. pour finir, les variables de l'algorithme 4 et l'algorithme 6 sont réinitialisées à la fin de l'exécution.

Remarque 3.12 *Nous supposons que le snapshot initial, dont le numéro est* 0*, correspond à l'initialisation du système distribué, c.-à-d., l'état de chacun des processus est son état initial et chaque canal est marqué comme étant vide.*

Remarque 3.13 *Notre solution est totalement distribuée, plusieurs processus peuvent prendre l'initiative de lancer le calcul d'un snapshot au même moment et grâce à la dernière étape, nous simulons une solution centralisée qui est obtenue en faisant l'hypothèse qu'il existe une horloge globale et que tous les processus connaissent, a priori, la date d du calcul des snapshots locaux.*

Remarque 3.14 *Il n'y pas de chevauchement parmi les requêtes répétées de l'algorithme 4 et de l'algorithme 6. Ceci est rendu possible par l'utilisation de la variable* $taken_p$ *qui est réinitialisée seulement à la fin de l'exécution de l'algorithme 7 et par le fait que les canaux de communication sont FIFO.*

3.5.2 Du Calcul de Snapshots Locaux à la Détection de la Terminaison de l'Exécution d'un Algorithme Distribué

Soient $\mathcal{A}$ un algorithme distribué et $\mathcal{E}$ une exécution de l'algorithme $\mathcal{A}$. Notre objectif, ici, est de détecter la terminaison de l'exécution $\mathcal{E}$.

Nous rappelons qu'une exécution $\mathcal{E}$ est terminée si et seulement si tous les processus du système sont dans l'état passif et tous les canaux sont vides. Ainsi, afin de détecter la terminaison de l'exécution $\mathcal{E}$, il suffit que, de temps en temps (à définir), au moins un processus initialise le calcul d'un snapshot et si son état est passif et ses canaux entrants sont vides, il doit détecter si cette même propriété est satisfaite pour tous les processus. Cela est réalisé par l'utilisation d'une occurrence supplémentaire de l'algorithme SSP. Si les variables d'un processus p indiquent que l'exécution n'est pas encore terminée, alors q, processus voisin de p, émet un signal à travers tout le réseau pour en informer chacun des autres processus.

De cette façon, nous obtenons un algorithme pour détecter la terminaison globale de l'exécution d'un algorithme distribué. Ces requêtes de détection de terminaison sont analogues à la solution présentée par Santoro à la section 8.3 de [San06].

Les idées principales sont :

Algorithme 7: Algorithme du calcul d'un point de reprise d'un système distribué.

I*nit-Checkpoint* : {Pour initialiser le calcul d'un point de reprise par un processus p tel que $snapshot = vrai$}
début
  mémoriser les informations concernant le snapshot local identifié par *snapshot-number*;
  $next_p := snapshot\text{-}number + 1$;
  $snapshot_p := vrai$;
  $local\text{-}snapshot_p := faux$;
  $a(p) := -1$;
  $A(p) = \{(i, -1) \mid i \in [1, \deg_G(p)]\}$;
  démarquer chaque port de p;
  envoyer $<$ *new-snapshot*,$next_p >$ via chaque port de p;
  $taken_p := faux$;
fin

R-$NewS_p$: {Un message *new-snapshot* est arrivé à p via le port j}
début
  recevoir$<$ *new-snapshot*, $k >$;
  si $k > snapshot\text{-}number_p$ **alors**
    mémoriser les informations concernant le snapshot local identifié par *snapshot-number*;
    $next_p := snapshot\text{-}number + 1$;
    $snapshot_p := faux$;
    $local\text{-}snapshot_p := faux$;
    $a(p) := -1$;
    $A(p) = \{(i, -1) \mid i \in [1, \deg_G(p)]\}$;
    démarquer chaque port de p;
    envoyer $<$ *new-snapshot*,$next_p >$ via chaque port de p;
    $taken_p := faux$;
fin

1. au moins un processus initialise le calcul d'un snapshot (algorithme 4) ;
2. si le snapshot local du processus p est tel que son état est passif, que tous ses canaux entrants sont vides et que tous les snapshots locaux sont calculés (algorithme 6), alors p initialise une occurrence de l'algorithme SSP (algorithme 5) afin de vérifier si l'exécution de $\mathcal{A}$ est terminée ;
3. si le snapshot local du processus p est tel que son état est actif ou qu'au moins un de ses canaux entrants n'est pas vide, alors, p envoie un signal pour informer tout les processus que l'exécution de $\mathcal{A}$ n'est pas terminée. Par conséquent, au moins un autre snapshot doit être calculé. Les variables de l'algorithme 4 et de l'algorithme 6 sont réinitialisées.

Par conséquent, l'algorithme de détection de la terminaison est une composition de l'algorithme 4, de l'algorithme 6 et de l'algorithme 8.

Toutefois, nous avons besoin d'une autre variable booléenne $Term\text{-}Detection_p$ sur chaque processus p, initialement $Term\text{-}Detection_p = faux$, qui indique si le processus essaie de vérifier si l'exécution est terminée ou non.

Les variables aTD_p et ATD_p sont utilisées par l'algorithme SSP qui s'occupe de vérifier si l'exécution de $\mathcal{A}$ est terminée.

Remarque 3.15 *Comme pour le cas du calcul de points de repris, il n'y a pas de chevauchement parmi les requêtes répétées de l'algorithme 4 et de l'algorithme 6 grâce à l'utilisation de la variable booléenne $taken_p$ qui est réinitialisée seulement à la fin de l'algorithme 8 et grâce au fait que les canaux sont FIFO.*

Remarque 3.16 *Pour tout type de médium de communication (p. ex., filaire), on définit la notion de* Bit Error Rate, *$BER = 10^{-x}$, soit 1 bit faux sur 10^x bits corrects, comme la probabilité qu'un bit soit erroné. La probabilité Pr_{err} qu'un message de taille k bits soit erroné est donnée par : $Pr_{err} = (1 - BER)^k$ avec $(1 - BER) < 1$. Par conséquent, nous en déduisons que plus un message est de taille importante, plus il a de chance d'être erroné pour un BER donné.*

Pour les algorithmes présentés jusqu'ici, la taille d'un message est limitée à 1 bit (simple marqueur) ou $O(logn)$ bits. Selon la remarque précédente, nous déduisons qu'il est possible de réaliser certaines tâches liées au calcul de l'état global en limitant la taille des messages échangés et donc en limitant le taux d'erreur de transmission.

3.6 Revêtements, Propriétés Stables et Weak Snapshot

Depuis le début de ce chapitre nous faisons l'hypothèse que le réseau est anonyme et que plusieurs processus peuvent jouer le rôle de l'initiateur d'un calcul. Chaque processus ne connaît qu'une borne supérieure du diamètre de ce réseau, notée β. Sous ces hypothèses, aucun processus ne peut calculer une carte du réseau. Nous prouvons que chaque processus peut calculer un graphe dont le réseau est revêtement, c.-à-d., un weak snapshot. Nous prouvons aussi que les propriétés classiques, comme les propriétés stables, étudiées à travers les snapshots subsistent toujours dans les weak snapshots. Tout d'abord, nous avons besoin de quelques définitions.

3.6.1 Quelques Définitions

Pour la suite de cette section, nous considérons les graphes dirigés pouvant avoir des arcs multiples ou des boucles. Nous présentons diverses définitions à propos des revêtements qui sont, comme pour les fibrations présentées dans le chapitre précédent, une forme particulière d'homomorphismes de graphes. À la suite de ces définitions, nous montrons le lien qu'il existe entre le calcul d'un snapshot et les revêtements.

Algorithme 8: Algorithme permettant d'envoyer une requête de détection de la terminaison.

I*nit-Term*$_p$: {Pour initialiser la détection de la terminaison par un processus p tel que : $snapshot_p = vrai$, p est passif, les canaux entrants de p sont vides et $Term\text{-}Detection_p = faux$. }
début
  $Term\text{-}Detection_p := vrai$;
  $aTD_p := 0$;
  $ATD_p := \{(i, -1) \mid i \in [1, \deg_G(p)]\}$;
  envoyer$< TD, aTD_p >$ à chaque voisin de p;
fin
R*TD_p* : {Un message $< TD, \alpha >$ est arrivé à p via le port j}
début
  recevoir$< TD, \alpha >$;
  si $Term\text{-}Detection_p = vrai$ **alors**
    $ATD_p := (ATD_p \setminus \{(j, x)\}) \cup \{(j, \alpha)\}$;
    $m := Min\{x \mid (i, x) \in ATD_p\}$;
    si $m \geq aTD_p$ **alors**
      $aTD_p := m + 1$;
    si $aTD_p \geq \beta$ **alors**
      $TERM := vrai$;
    sinon
      envoyer$< TD, aTD_p >$ via chaque port de p;
  sinon
    si $Term\text{-}Detection_p = faux$ *et p est passif, et les canaux entrants de p sont vides et* $local\text{-}snapshot_p = vrai$ **alors**
      $Term\text{-}Detection_p := vrai$;
      $aTD_p := 0$;
      $ATD_p := \{(i, -1) \mid i \in [1, \deg_G(p)]\}$;
      $ATD_p := (ATD_p \setminus \{(j, x)\}) \cup \{(j, \alpha)\}$;
      envoyer$< TD, aTD_p >$ via chaque port de p;
fin

I*nit-Term*$_p$: {Pour envoyer un signal de non-terminaison depuis un processus p tel que : $snapshot_p = vrai$, p n'est pas passif ou il existe au moins un canal entrant qui n'est pas vide.}
début
  $next_p := snapshot\text{-}number_p + 1$;
  envoyer$< not\text{-}terminated, next_p >$ à tous les voisins de p;
  $snapshot_p := faux$;
  $local\text{-}snapshot_p := faux$;
  $a(p) := -1$;
  $A(p) = \{(i, -1) \mid i \in [1, \deg_G(p)]\}$;
  démarquer chaque port de p;
  envoyer $< new\text{-}snapshot, next_p >$ via chaque port de p;
  $taken_p := faux$;
fin
R*T*$_p$: {Un message $< not\text{-}terminated, \alpha >$ est arrivé à p via le port j}
début
  recevoir$< not\text{-}terminated, \alpha >$;
  si $snapshot\text{-}number_p < \alpha$ **alors**
    $next_p := snapshot\text{-}number_p + 1$;
    envoyer$< not\text{-}terminated, next_p >$ à tous les voisins de p;
    $snapshot_p := faux$;
    $local\text{-}snapshot_p := faux$;
    $a(p) := -1$;
    $A(p) = \{(i, -1) \mid i \in [1, \deg_G(p)]\}$;
    démarquer chaque port de p;
    envoyer $< new\text{-}snapshot, next_p >$ via chaque port de p;
    $taken_p := faux$;
fin

Homomorphismes et Revêtements.

Les notions de revêtements et de revêtements symétriques sont fondamentales pour ce travail ; les définitions ainsi que les propriétés principales sont présentées dans [GT87, BV02a]. Ces notions permettent d'exprimer la similarité entre deux graphes dirigés.

Remarque 3.17 *Étant donné un graphe simple connexe étiqueté* $\mathbf{G} = (G, \lambda)$ *avec une numérotation des ports* δ*, soit* $\mathbf{D} = (Dir(\mathbf{G}), \delta)$ *le graphe dirigé étiqueté correspondant* $(Dir(G), (\lambda, \delta))$*. Soit* $\mathcal{A}$ *un algorithme distribué. Nous parlons indifféremment d'une exécution de* $\mathcal{A}$ *sur* $(\mathbf{G}, \delta)$ *d'une exécution de* $\mathcal{A}$ *sur* $\mathbf{D}$*.*

Définition 3.18 *Un graphe dirigé étiqueté* $\mathbf{D}$ *est un* revêtement *d'un graphe dirigé étiqueté* $\mathbf{D}'$ *via* φ *si* φ *est un homomorphisme de* $\mathbf{D}$ *vers* $\mathbf{D}'$ *tel que pour chaque arc* $a' \in A(D')$ *et pour chaque sommet* $v \in \varphi^{-1}(t(a'))$ *(resp.* $v \in \varphi^{-1}(s(a'))$*, il existe un unique arc* $a \in A(D)$ *tel que* $t(a) = v$ *(resp.* $s(a) = v$*) et* $\varphi(a) = a'$*.*

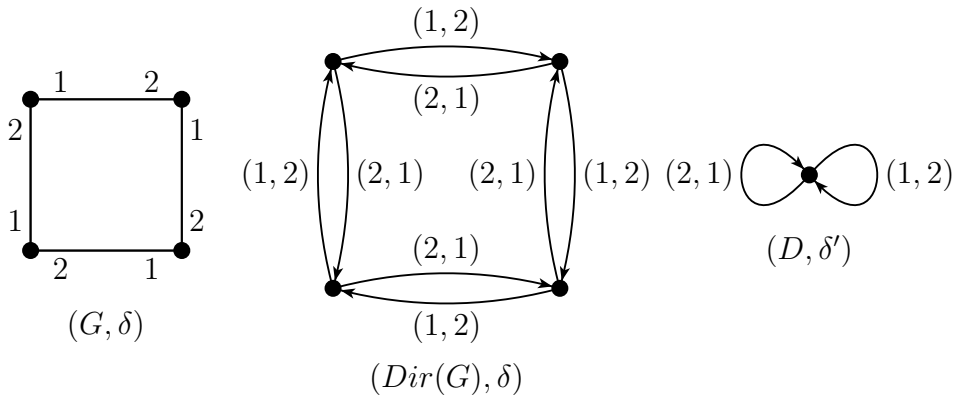

FIGURE 3.3 – Un graphe G avec une numérotation des ports δ et le graphe dirigé étiqueté $(Dir(G), \delta)$ qui est un revêtement de (D, δ').

Définition 3.19 *Un graphe dirigé symétrique étiqueté* $\mathbf{D}$ *est un* revêtement symétrique *d'un graphe dirigé symétrique étiqueté* $\mathbf{D'}$ *via* φ *si* $\mathbf{D}$ *est un revêtement de* $\mathbf{D'}$ *par l'intermédiaire de* φ *et si pour chaque arc* $a \in A(D)$, $\varphi(Sym(a)) = Sym(\varphi(a))$. *L'homomorphisme* φ *est un* revêtement de projection symétrique *de* $\mathbf{D}$ *vers* $\mathbf{D'}$.

Dans la définition suivante, nous présentons le lien qu'il existe entre les revêtements et la notion de weak snapshots.

Définition 3.20 *Étant donné un graphe simple connexe étiqueté* $\mathbf{G} = (G, \lambda)$ *avec une numérotation des ports* δ *qui définit un snapshot du réseau* G. *Soit* $\mathbf{D} = (Dir(\mathbf{G}), \delta)$ *le graphe dirigé étiqueté correspondant* $(Dir(G), (\lambda, \delta))$. *Soit* $\mathbf{D'}$ *le graphe dirigé étiqueté tel que* $\mathbf{D} = (Dir(\mathbf{G}), \delta)$ *est un revêtement de* $\mathbf{D'}$. *Le graphe dirigé étiqueté* $\mathbf{D'}$ *est appelé* weak snapshot *de* G.

3.6.2 Snapshots et Propriétés Stables dans les Systèmes Distribués

Soit $\mathbf{G}$ un graphe. Soit $\mathbf{D}$ un algorithme à base de passage des messages. On considère une exécution $\mathbf{E}$ de $\mathbf{D}$ sur $\mathbf{G}$. Soit S^* un snapshot consistant correspondant à l'exécution de $\mathbf{E}$.

Définition 3.21 *Une propriété* P *de configurations de* $\mathcal{E}$ *est qualifiée de* stable *lorsque : si* P *est vraie pour une configuration* $(\texttt{state}, M)$, *alors* P *est aussi vraie pour toute configuration obtenue depuis* $(\texttt{state}, M)$.

Parmi les propriété stables des systèmes distribués détectées à l'aide d'un snapshot, nous considérons la terminaison, les interblocages ou *deadlocks*, la perte de jetons et le ramasse-miette ou *garbage collection* (voir [Tel00, San06, KS08] pour plus de détails sur ces propriétés).

Terminaison.

Le rapport entre cette propriété et le calcul d'un snapshot a déjà été traité dans la section 3.5.2. Nous rappelons la définition :

Définition 3.22 *Une exécution $\mathcal{E}$ est terminée si et seulement si tous les processus du système sont dans l'état passif et tous les canaux sont vides.*

Interblocage.

Un interblocage ou *deadlock* survient dans un système distribué s'il existe un cycle composé de processus dans lequel chacun est en attente d'un autre et dans lequel il n'y a aucun message en transit.

Il peut être détecté par la construction d'un *Wait-For-Graph* (ou WFG) : les sommets de ce graphe correspondent au processus et il existe un arc entre le sommet (processus) p et le sommet q si p est bloqué et est en attente de q. Il y a un deadlock si et seulement si il existe un cycle dans le WFG (voir [San06, KS08]).

Perte de Jetons.

Certains systèmes distribués requièrent la circulation d'un jeton parmi les processus. Certains jetons peuvent disparaitre ou, plus simplement, être consommés par les processus eux-mêmes. En conséquence, il peut être intéressant de vérifier des propriétés telles que "il y a exactement k tokens" ou "il y a au plus k tokens" ; ces propriétés sont stables.

Ramasse-miettes Distribué ou Distributed Garbage Collection.

L'objectif, ici, est de décider si un objet du système est nécessaire ou non dans l'exécution d'une tâche. Nous nous inspirons de la présentation de Schiper et Sandoz [SS94] : considérons un système composé d'un ensemble d'objets O, ainsi que d'un ensemble statique $Root \subseteq 0$, correspondant à des objets racines. Les objets racines sont invoqués ou utilisés par certains processus. Une invocation sur un objet o_i implique l'exécution des actions fournies par o_i. Par conséquent, l'ensemble des objets peut être représenté comme un ensemble de processus échangeant des messages lors d'une invocation, et les messages peuvent comporter des références.

Sur l'ensemble O des objets, nous définissons une relation de descendance comme suit : descendant (o_i, o_j) si l'objet o_i possède une référence vers l'objet o_j ou une référence vers l'objet o_j est effective pour l'objet o_i. Un objet o_i est accessible soit si $o_i \in Root$ ou si o_i est un descendant d'un objet accessible. Par définition, un objet est nécessaire s'il est accessible. Seul les objets accessibles peuvent référencer d'autres objets. Un objet qui n'est plus accessible dans le système est appelé *garbage* et doit être détruit.

Ainsi, les objets accessibles sont détectés comme des sommets o pour lesquels il existe un chemin depuis une racine vers o.

Propriétés Stables et Snapshots.

Habituellement, après un calcul de snapshot, les propriétés stables sont vérifiées à l'aide d'un arbre recouvrant ou d'un anneau ; elles peuvent aussi être vérifiées de façon centralisée par un processus calculant une carte du réseau dans laquelle chaque sommet et chaque canal est étiqueté par son état.

Ces solutions ne sont plus possibles dans les réseaux anonymes avec aucun sommet distingué. Dans ce contexte, nous utilisons la notion de revêtements de graphes.

3.6.3 Propriétés Stables dans les Systèmes Distribués et Weak Snapshots

Soient $\mathbf{D}_1$ et $\mathbf{D}_2$ deux graphes dirigés étiquetés tels que $\mathbf{D}_1$ est un revêtement de $\mathbf{D}_2$ via l'homomorphisme φ. Un relèvement du chemin W_2 dans $\mathbf{D}_2$ est un chemin W_1 dans $\mathbf{D}_1$ tel que φ associe les arcs de W_1 sur les arcs de W_2 dans l'ordre de traversée.

Soit $(\mathbf{G}, \delta)$ un réseau avec une numérotation de ports δ. Soit $\mathbf{D}'$ un graphe étiqueté tel que $(Dir(\mathbf{G}), \delta)$ est un revêtement de $\mathbf{D}'$.

Avec nos notations, le théorème 2.4.1 et le théorème 2.4.3 de [GT87] peuvent être traduits comme suit :

1. Soit W un chemin dans $\mathbf{D}'$ tel que le sommet initial est u. Alors pour chaque sommet u_i in $\varphi^{-1}(u)$, il existe un unique relèvement de W qui débute depuis u_i.
2. Soit C un cycle dans $\mathbf{D}'$. Alors $\varphi^{-1}(C)$ est une union disjointe de cycles.

D'après ces deux résultats, nous en déduisons que si $\mathbf{D}_1$ est un revêtement de $\mathbf{D}_2$, alors les deadlocks et les garbages détectés dans $\mathbf{D}_1$ peuvent aussi être détectés dans $\mathbf{D}_2$.

Dans la proposition suivante, nous donnons la relation qu'il existe entre les images et les antécédents d'un revêtement.

Proposition 3.23 *Soient $\mathbf{D}_1$ et $\mathbf{D}_2$ deux graphes dirigés étiquetés. Si $\mathbf{D}_1$ est un revêtement de $\mathbf{D}_2$ via l'homomorphisme φ, alors il existe un entier α tel que pour tout sommet u de $\mathbf{D}_2$, la cardinalité de $\varphi^{-1}(u)$ est égale à α.*

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

Preuve : On considère deux graphes dirigés étiquetés $\mathbf{D}_1$ et $\mathbf{D}_2$ tels que $\mathbf{D}_1$ est un revêtement de $\mathbf{D}_2$ via l'homomorphisme φ. On considère un sommet $v \in V(D_2)$ et un sommet $v' \in N_{D_2}(v)$.

Puisque φ est un homomorphisme localement bijectif, pour tout sommet $u \in \varphi^{-1}(v)$, il existe un unique sommet $u' \in N_{D_1}(u)$ tel que $\varphi(u') = v'$. De plus, puisque φ est localement bijectif en u', pour tout sommet $u'' \in N_{D_1}(u')$ différent de u, $\varphi(u'') \neq \varphi(u)$. Par conséquent, $|\varphi^{-1}(v)| \leq |\varphi^{-1}(v')|$ et par symétrie, $|\varphi^{-1}(v)| = |\varphi^{-1}(v)|$. Ainsi, puisque le graphe $\mathbf{D}_2$ est connexe, pour tout $v, v' \in V(D_2)$, $|\varphi^{-1}(v)| = |\varphi^{-1}(v')|$. □

Ainsi, si c_1 jetons apparaissent dans $\mathbf{D}_1$ (pour un entier c_1 positif), alors $c_2 = c_1/\alpha$ jetons apparaissent dans $\mathbf{D}_2$.. De là nous déduisons que si la taille (le nombre de sommets) de $\mathbf{D}_1$ est connue, alors la connaissance de $\mathbf{D}_2$ permet de détecter la perte de jetons $\mathbf{D}_2$.

Ces résultats sont résumés dans la proposition suivante :

Proposition 3.24 *Soit G un système distribué. Tout weak snapshot $\mathbf{D}$ de G permet de détecter les deadlocks et la terminaison et d'effectuer la collecte des objets inaccessibles du système (ramasse-miette distribué). De plus, si les processus connaissent la taille de G, alors la perte de jetons peut aussi être détectée depuis un weak snapshot.*

Dans la suite de ce chapitre nous montrons que, connaissant une borne sur le diamètre d'un réseau anonyme $\mathbf{D}_1$, il existe un algorithme totalement distribué, pouvant admettre plusieurs initiateurs, avec détection de la terminaison, qui peut calculer $\mathbf{D}_2$ tel que $\mathbf{D}_1$ est un revêtement de $\mathbf{D}_2$.

Sous ces hypothèses, nous ne pouvons pas calculer un snapshot. Toutefois, d'après la proposition 3.24, nous pouvons résoudre le problème de la détection de propriétés stables. Il suffit que :

1. au moins un processus initialise une exécution de l'algorithme de Chandy-Lamport (algorithme 4) ;
2. chaque processus détecte un instant dans lequel le calcul de tous les snapshots locaux est terminé (algorithme 6) ;
3. au moins un processus initialise le calcul d'un weak snapshot ;
4. chaque processus détecte un instant dans lequel le calcul du weak snapshot est réalisé et s'informe sur l'existence de propriétés stables.

Maintenant, nous présentons un algorithme totalement distribué qui s'occupe de calculer anonymement un weak snapshot avec détection de la terminaison.

3.7 Calculer Anonymement des Weak Snasphots

Soit $(\mathbf{G}, \delta)$ un graphe étiqueté avec une numérotation de ports δ. Nous supposons que $\mathbf{G}$ est anonyme et que chaque sommet de G connaît une borne supérieure, notée β, sur le diamètre du graphe G. Cette section présente un algorithme, que l'on nomme $\mathcal{M}_{W-S}$, qui s'occupe de calculer un weak snapshot, c.-à-d., un graphe dirigé étiqueté $(\mathbf{D}, \delta')$ tel que $(Dir(\mathbf{G}), \delta)$ est un revêtement de $(\mathbf{D}, \delta')$.

Dans un certain sens, le weak snapshot $(\mathbf{D}, \delta')$ représente la "vue globale" ou la "connaissance maximale" du système distribué que chacun des sommets peut obtenir (corollaire du travail présenté par Angluin [Ang80], théorème 5.5).

Nous prouvons que si les processus connaissent une borne supérieure sur le diamètre du réseau, alors ils peuvent détecter la terminaison de $\mathcal{M}_{W-S}$.

Cet algorithme a été présenté initialement dans [Cha06, CM07b] comme un algorithme d'élection lorsqu'il est exécuté sur des graphes minimaux pour la relation de revêtements. Le graphe $(\mathbf{G}, \delta)$ est minimal si quand $(Dir(\mathbf{G}), \delta)$ est un revêtement de $(\mathbf{D}, \delta')$, alors $(Dir(\mathbf{G}), \delta)$ est isomorphe à $(\mathbf{D}, \delta')$. Cet algorithme est basé sur un autre algorithme d'élection proposé par Mazurkiewicz dans [Maz97b] et sur l'algorithme SSP présenté précédemment.

3.7.1 Description Informelle

Tout d'abord, nous donnons une description générale de l'algorithme $\mathcal{M}_{W-D}$ lorsque celui-ci est exécuté sur un graphe étiqueté simple et connexe $\mathbf{G}$ avec une numérotation de port δ.

On présente maintenant un algorithme d'énumération inspiré de l'algorithme de Mazurkiewicz et adapté au modèle étudié dans ce chapitre.

Durant l'exécution de l'algorithme, chaque sommet u essaie d'obtenir une identité qui est un numéro entre 1 et $|V(G)|$. Chaque sommet u va ensuite envoyer son numéro à chacun de ses voisins v, en leur indiquant le numéro du port $\delta_u(v)$. Lorsqu'un sommet u reçoit un message depuis un de ses voisins v, il mémorise son numéro $n(v)$ ainsi que les numéros de ports $\delta_u(v)$ et $\delta_v(u)$. D'après toutes les informations qu'il a rassemblé depuis ses voisins, chaque sommet est capable de construire sa *vue locale*, c.-à-d., l'ensemble des numéros de ses voisins associés aux numéros de ports correspondants. Ensuite, chaque sommet va diffuser dans tout le graphe son numéro, son étiquette et sa boîte-aux-lettres (qui contient un ensemble de *vues locales*). 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 initiale $\lambda(u)$ et sa vue locale avec l'étiquette initiale $\lambda(v)$ 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 en informe ses voisins et diffuse à nouveau son numéro et sa vue locale. À la fin de l'exécution, chaque sommet a calculé un graphe $(\mathbf{D}, \delta')$ tel que $(Dir(\mathbf{G}), \delta)$ est un revêtement symétrique de $(\mathbf{D}, \delta')$.

3.7.2 Étiquettes

Nous considérons un réseau $(\mathbf{G}, \delta)$ où $\mathbf{G} = (G, \lambda)$ est un graphe simple étiqueté et où δ est une numérotation des ports de $\mathbf{G}$.

Remarque 3.25 *La fonction $\lambda : V(G) \to L$ est un étiquetage initial des sommets qui ne sera pas modifié par l'algorithme ; cela correspond aux états calculés par l'algorithme précédent (l'algorithme de Chandy-Lamport dans notre cas).*

Nous supposons qu'il existe un ordre total $<_L$ sur L. Nous étendons l'ordre $<_L$ à $L \cup \{\perp\}$ (en supposant que $\perp \notin L$) comme suit : pour tout $\ell \in L$, $\perp < \ell$.

Lors de l'exécution, l'état de chaque sommet va être codé par une étiquette de la forme $(\lambda(v), n(v), N(v), M(v))$ qui représente les informations suivantes :

- la première composante $\lambda(v)$ est l'étiquette initiale et ne sera pas modifiée lors de l'exécution et a été calculée par l'algorithme précédent.
- $n(v) \in \mathbb{N}$ est le *numéro* courant du sommet v qui est modifié lors de l'exécution de l'algorithme ; initialement $n(v) = 0$.
- $N(v) \in \mathcal{P}_{\text{fin}}(\mathbb{N}^3)$ est la *vue locale* du sommet v. À la fin de l'exécution, si $(m, \ell, i, j) \in N(v)$, alors le sommet v a un voisin u dont le numéro est m, l'étiquette est ℓ et l'arc de u à v est étiqueté (i, j). Initialement, $N(v) = \{(0, \perp, 0, i) \mid i \in [1, \deg_G(v)]\}$.

– $M(v) \subseteq \mathbb{N} \times L \times \mathcal{P}_{\text{fin}}(\mathbb{N}^3)$ est la *boîte-aux-lettres* du sommet v ; initialement $M(v) = \emptyset$. Un élément de $M(v)$ a la forme suivante : (m, ℓ, N) où $m \in \mathbb{N}$, $\ell \in L$ et N est la vue locale. Elle va contenir toute l'information reçue par v lors de l'exécution de l'algorithme. Si $(m, \ell, N) \in M(v)$, cela signifie qu'à une certaine étape antérieure de l'exécution, il y avait un sommet u tel que $n(u) = m$, $\lambda(u) = \ell$ et $N(u) = N$.

Un sommet va détecter si son état est final en utilisant un compteur. Ainsi, nous ajoutons à l'étiquette de chaque sommet les éléments suivants :

– $a(v) \in \mathbb{Z}$ est un compteur et initialement, $a(v) = -1$. Dans un certains sens, $a(v)$ représente la distance jusqu'à laquelle tous les sommets ont la même boîte-aux-lettre que v.
– $A(v) \in \mathcal{P}_{\text{fin}}(\mathbb{N} \times \mathbb{N})$ encode l'information que v possède à propos des valeurs de $a(u)$ pour chacun des ses voisins u. Initialement, $A(v) = \{(i, -1) \mid i \in [1, \deg_G(v)]\}$.

3.7.3 Messages

Dans notre algorithme, les processus communiquent en échangeant des messages de la forme $< (n, \ell, M, a), i >$. Si un processus u envoie un message à un de ses voisins v via le port i, alors $< (n, \ell, M, a), i >$ contient les informations suivantes :

– n est le numéro courant $n(u)$ de u,
– ℓ est l'étiquette $\lambda(u)$ de u,
– M est la boîte-aux-lettres de u,
– a est la valeur du compteur $a(u)$,
– i est le port par lequel le message a été envoyé, c.-à-d., $i = \delta_u(v)$.

3.7.4 Un Ordre sur les Vues Locales

Comme pour l'algorithme 1 et l'algorithme 2 présentés dans le chapitre précédent, les bonnes propriétés de l'algorithme reposent sur un ordre sur les vues locales, c.-à-d., sur les ensembles finis de triplets de $\mathbb{N}^3$. Pour cela, on considère que $\mathbb{N}^3$ est muni de l'ordre lexicographique usuel : $(n, p, q) < (n', p', q')$ si $n < n'$, ou si $n = n'$ et $p < p'$, ou si $n = n'$, $p = p'$ et $q < q'$.

Étant donnés deux ensembles $N_1, N_2 \in \mathcal{P}_{\text{fin}}(\mathbb{N}^3)$ distincts, on dit que $N_1 \prec N_2$ si le maximum pour l'ordre lexicographique sur $\mathbb{N}^3$ de la différence symétrique $N_1 \triangle N_2 = (N_1 \setminus N_2) \cup (N_2 \setminus N_1)$ appartient à N_2.

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

3.7.5 Éléments Maximaux d'une Boîte-Aux-Lettres

Considérons la boîte-aux-lettres $M = M(v)$ d'un sommet v durant l'exécution de l'algorithme $\mathcal{M}_{W-S}$ sur un graphe $(\mathbf{G}, \delta)$. On dit qu'un élément $(n, \ell, N) \in M$ est *maximal* dans M si il n'existe pas $(n, \ell', N') \in M$ tel que $(\ell, N) \prec (\ell', N')$. On note par $S(M)$

l'ensemble des éléments maximaux de M. D'après la proposition 3.26, après chaque étape de l'algorithme $\mathcal{M}_{W-S}$, $(n(v), \lambda(v), N(v))$ est maximal dans $M(v)$.

3.7.6 L'Algorithme $\mathcal{M}_{W-S}$

L'algorithme $\mathcal{M}_{W-S}$ est décrit pour un sommet v_0. Il est exprimé suivant le modèle *event-driven*. Un sommet qui exécute une des actions suivantes est qualifié comme actif.

La première règle **I** ne peut être appliquée que par un processus v_0 qui lorsqu'il se réveille n'a encore reçu aucun message. Dans ce cas là, le sommet choisit le numéro 1, met à jour sa boîte-aux-lettres et informe ses voisins de son nouveau numéro et de sa nouvelle boîte-aux-lettres.

La seconde règle **R** explique les opérations effectuées par un processus v_0 lorsqu'il reçoit un message m d'un de ses voisins par le port j. Il met d'abord à jour sa boîte-aux-lettres en y ajoutant les éléments de M' puis met à jour sa *vue locale* suivant les informations contenues dans le message m. Ensuite, s'il découvre l'existence d'un autre sommet ayant le même numéro et une vue locale plus forte, il prend un nouveau numéro. Si la boîte-aux-lettres n'a pas été modifiée, il met à jour $A(v_0)$ et incrémente la valeur de $a(v_0)$ si cela est possible. Finalement, si sa boîte-aux-lettres $M(v_0)$ ou son compteur $a(v_0)$ ont été modifiés lors de l'exécution de cette suite d'instruction, il en informe tous ses voisins.

3.7.7 Propriétés de L'Algorithme $\mathcal{M}_{W-S}$.

Nous considérons une exécution ρ de l'algorithme $\mathcal{M}_{W-S}$ sur un graphe $(\mathbf{G}, \delta)$ et pour chacun des sommets $v \in V(G)$, nous notons par $(\lambda(v), n_k(v), N_k(v), M_k(v), a(v), A(v))$ l'état de v après la $k$ième étape de calcul de ρ sur v. Si le sommet v exécute une action pour aller de l'étape k à l'étape $k+1$, il est dit actif à l'étape $k+1$.

La proposition suivante résume certaines propriétés satisfaites durant l'exécution ρ sur $(\mathbf{G}, \delta)$.

Proposition 3.26 ([Cha06, CM07b]) *Considérons un sommet v et une étape i.*

Alors, $n_k(v) \leq n_{k+1}(v)$, $N_k(v) \preceq N_{k+1}(v)$, $M_k(v) \subseteq M_{k+1}(v)$.

Pour chaque $(m, \ell, N) \in M_k(v)$ et chaque $m' \in [1, m]$, $\exists (m', \ell', N') \in M_k(v), \exists v' \in V(G)$ tel que $n_k(v') = m'$.

Nous pouvons montrer que l'algorithme $\mathcal{M}_{W-S}$ termine et que l'étiquetage final vérifie les propriétés suivantes :

Proposition 3.27 ([Cha06, CM07b]) *Étant donné un graphe $\mathbf{G}$ avec une numérotation des ports δ, nous pouvons associer avec l'étiquetage final de toute exécution ρ de $\mathcal{M}_{W-S}$ sur $(\mathbf{G}, \delta)$, un graphe dirigé $(\mathbf{D}, \delta')$ tel que $(Dir(\mathbf{G}), \delta)$ est un revêtement symétrique de $(\mathbf{D}, \delta')$.*

Un corollaire intéressant de la proposition 3.26 est :

Corollaire 3.28 *Nous considérons une exécution ρ de $\mathcal{M}_{W-S}$ sur $(\mathbf{G}, \delta)$. Il existe une étape k_0 telle qu'après celle-ci, pour tout sommet v, les valeurs de $n(v), N(v)$ et $M(v)$ ne sont plus modifiées.*

Algorithme 9: L'algorithme $\mathcal{M}_{W-S}$.

I : $\{n(v_0) = 0$ et aucune message n'a été reçu par $v_0\}$
début
  $n(v_0) := 1$;
  $M(v_0) := \{(n(v_0), \lambda(v_0), \emptyset)\}$;
  pour $i := 1$ ***to*** $\deg(v_0)$ **faire**
    envoyer $< (n(v_0), \lambda(v_0), M(v_0), a(v_0)), i >$ par le port i ;
fin

R : {Un message $< (n_1, \ell_1, M_1, a_1), i_1 >$ est arrivé à v_0 par le port $j_1\}$
début
  $M_{old} := M(v_0)$;
  $a_{old} := a(v_0)$;
  $M(v_0) := M(v_0) \cup M_1$;
  si $n(v_0) = 0$ *or* $\exists (n(v_0), \ell', N') \in M(v_0)$ *tel que* $(\lambda(v_0), N(v_0)) \prec (\ell', N')$ **alors**
    $n(v_0) := 1 + \max\{n' \mid \exists (n', \ell', N') \in M(v_0)\}$;
  $N(v_0) := N(v_0) \setminus \{(n', \ell', i_1, j_1) \mid \exists (n', \ell', i_1, j_1) \in N(v_0)\} \cup \{(n_1, \ell_1, i_1, j_1)\}$;
  $M(v_0) := M(v_0) \cup \{(n(v_0), \lambda(v_0), N(v_0))\}$;
  si $M(v_0) \neq M_{old}$ **alors**
    $a(v_0) := -1$;
    $A(v_0) := \{(i', -1) \mid 1 \leq i' \leq deg(v_0)\}$;
  si $M(v_0) = M_1$ **alors**
    $A(v_0) := A(v_0) \setminus \{(j_1, a') \mid \exists (j_1, a') \in A(v_0)\} \cup \{(j_1, a_1)\}$;
  si $\forall (i', a') \in A(v_0), a(v_0) \leq a'$ *et* $a(v_0) \leq \beta$ **alors** $a(v_0) := a(v_0) + 1$;
  si $M(v_0) \neq M_{old}$ *ou* $a(v_0) \neq a_{old}$ **alors**
    pour $i := k$ ***to*** $\deg(v_0)$ **faire**
      envoyer $< (n(v_0), \lambda(v_0), M(v_0), a(v_0)), k >$ par le port k ;
fin

3.7.8 Détection de la Terminaison de $\mathcal{M}_{W-S}$

Si l'algorithme $\mathcal{M}_{W-S}$ est exécuté sur un graphe minimal G, alors celui-ci termine et attribue un identifiant unique à chaque sommet compris entre 1 et la taille (l'ordre) de G. Nous prouvons que la connaissance de la taille de G permet à chaque sommet de détecter la terminaison (voir [Cha06, CM07b]).

Dans ce chapitre, nous ne faisons pas l'hypothèse que le réseau est minimal pour les revêtements. En conséquence, il n'est pas possible d'appliquer les résultats de [Cha06, CM07b].

Soit β une borne supérieure sur le diamètre $Diam$ du graphe G. Dans cette section, nous prouvons que chaque sommet, connaissant β, est capable de détecter un instant dans lequel les valeurs de $n(v)$, $N(v)$ et $M(v)$ ne sont plus modifiées, c.-à-d., la terminaison de

$\mathcal{M}_{W-S}$.

Proposition 3.29 *Considérons un sommet v et une étape i.*

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

Si $a_{i+1}(v) \geq 1$, alors pour chaque $w \in N(v)$, il existe une étape $j \leq i$ telle que $a_j(w) \geq a_{i+1}(v) - 1$ et $M_j(w) = M_{i+1}(v)$.

Preuve : Nous prouvons cette assertion par récurrence sur l'étape i. Considérons un sommet v qui modifie sont état à l'étape $i+1$.

Si v a appliqué la règle **I**, alors il est trivial de voir que la propriété est satisfaite.

Supposons maintenant que v a appliqué la règle **R** après avoir reçu le message $m_1 =< (n_1, l_1, M_1, a_1), p_1 >$ par l'intermédiaire du port q_1. De par l'algorithme, nous avons : $M_i(v) \subseteq M_{i+1}(v)$.

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

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

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

Supposons maintenant que $M_1 = M_{i+1}(v)$ et $min_A = a_1$. Si $a_1 = 0$, alors v peut incrémenter la valeur de $a(v)$ si elle est égale à -1. Ainsi, $a_{i+1}(v) \geq 0$ et par conséquent, $a_{i+1}(v) \geq min_A$.

Sinon, nous pouvons supposer que $min_A = a_1 > 0$, $M_{i+1}(v) = M_i(v) = M_1$ et que $(q_1, a_1) \notin A_i(v)$. Soit $m_2 =< (n_2, \ell_2, M_2, a_2), p_1 >$ étant le message précédent reçu par l'intermédiaire du port q_1. Puisque les canaux de communication sont FIFO, m_2 a été envoyé avant m_1. Comme $a_1 > 0$, $M_2 = M_1$ et ainsi, par récurrence, $a_2 = a_1 - 1$. Soit $j \leq i$ l'étape où v reçoit m_2. Puisque $M_2 \subseteq M_j(v) \subseteq M_i(v) = M_1 = M_2$, $M_j(v) = M_2$. Par conséquent, $(q, a_2) \in A_i(v)$. Puisque $a_1 = min_A$, $a_2 = a_1 - 1 = min\{a | \exists(q,a) \in A_i(v)\}$. Si $a_i(v) \geq a_1$, alors $a_{i+1}(v) \geq a_i(v) \geq min_A$. Sinon, par récurrence, $a_i(v) = a_2 = a_1 - 1$, et, puisque $a_i(v) < a_1$ et $a_i(v) \leq min_A$, v incrémente $a_i(v)$. Ainsi, $a_{i+1}(v) = 1 + a_i(v) = a_1 \geq min_A$.

Supposons que $a_{i+1}(v) \geq 1$ et considérons un sommet $w \in N(v)$. Il existe $(\delta_v(w), a) \in A_{i+1}(v)$ avec $a \geq a_{i+1}(v) - 1 \geq 0$ et v reçoit un message $m =< (n, \ell, M, a), \delta_w(v) >$ provenant de w durant une étape $i' \leq i+1$. Soit $j < i+1$ l'étape durant laquelle w a envoyé ce message. Puisque $a \geq 0$, $M_{i'}(v) = M_{i+1}(v) = M_j(v)$ et $a_j(w) = a \geq a_{i+1}(v) - 1$. □

D'après cette proposition, nous en déduisons ces deux lemmes suivants et la terminaison de l'algorithme $\mathcal{M}_{W-S}$.

Lemme 3.30 *Pour chaque sommet $v \in V(G)$ et pour chaque étape k, pour chaque sommet $w \in V(G)$ tels que $\mathrm{dist}(v,w) \leq a_k(v)$ il existe une étape $k' \leq k$ telle que $a_{k'}(w) \geq a_k(v) - \mathrm{dist}(v,w)$ et $M_k = M'_k$.*

Preuve : On fait une démonstration par récurrence sur la distance d entre v et w dans G. Si $d = 0$, la propriété est trivialement vraie. On suppose maintenant que la propriété est vraie pour tous sommets v, w tels que $\text{dist}_G(v, w) \leq d$.

On considère deux sommets v, w et une étape k tels que $a_k(v) \geq d + 1 \geq 1$ et $\text{dist}_G(v, w) = d + 1$. Il existe un sommet $u \in N_G(v)$ tel que $\text{dist}_G(u, w) = d$. On sait qu'il existe $(\delta_v(u), a) \in A_k(v)$ tel que $a \geq a_k(v) - 1 \geq 0$. On considère l'étape $k' < k$ lors de laquelle le dernier message reçu par v par le port $\delta_v(u)$ a été envoyé par u. On sait que $M_{k'}(u) = M_k(v)$ et $a_{k'}(u) = a \geq a_k(v) - 1$. Par hypothèse de récurrence, on sait qu'il existe une étape $j < k'$ telle que $a_j(w) \geq a_{k'}(u) - d \geq a_k(v) - (d + 1)$ et $M_j(w) = M_{k'}(u) = M_k(v)$. Ainsi, la propriété est vérifiée pour tous sommets v, w à distance $d + 1$ dans G. □

Lemme 3.31 *S'il existe un sommet $v \in V(G)$ et une étape k telle que $a_k(v) \geq \beta + 1$ alors pour chaque sommet w $M_k(w) = M_k(v)$ et $a_k(w) \geq 0$.*

Preuve : On montre ce lemme par récurrence sur la distance d entre v et w dans G. Si $d = 0$, la propriété est trivialement vraie.

D'après la proposition 3.29 et le lemme 3.30, la propriété est vérifiée pour tous sommets v, w à distance $d + 1$ dans G. □

3.8 Conclusion

Dans ce chapitre, nous avons abordé le problème des propriétés stables qui peuvent être détectées dans un système totalement distribué (dans lequel aucun processus n'est distingué et dans lequel il peut y avoir plusieurs initiateurs) en supposant que les processus sont anonymes et connaissent une borne supérieure sur le diamètre du réseau.

Nous avons prouvé que la détection de la terminaison de l'algorithme de Chandy-Lamport peut être réalisée avec l'algorithme SSP. Ensuite, nous avons appliqué cette méthode pour fournir un algorithme pour la détection de la terminaison d'un algorithme distribué quelconque par l'utilisation de requête de terminaison. Nous avons aussi utilisé cette technique pour le calcul de points de reprises.

Nous avons introduit le concept de *weak snapshot* comme étant la connaissance maximale qu'un processus peut calculer anonymement depuis un réseau en ne connaissant qu'une borne supérieure sur son diamètre. Nous avons prouvé que les propriétés stables usuelles telles que la terminaison, les interblocages, la perte d'un jeton ou encore la notion de *garbage collection* peuvent être détectées à l'aide d'un *weak snapshot*. En conséquence, nous avons proposé un algorithme qui permet à chacun des processus de calculer de façon totalement distribuée et anonyme un *weak snapshot* en connaissant une borne supérieure sur le diamètre du réseau.

Dans une première perspective, il pourrait être intéressant de présenter l'efficacité (complexités) des algorithmes étudiés permettant la détection de telles propriétés.

Chapitre 4

Débogage et Visualisation de l'Exécution d'un Système Distribué Anonyme

Sommaire

Comme présenté dans le chapitre précédent, calculer l'état global d'un système distribué asynchrone est un problème très largement étudié pour lequel une multitude de solutions existe sous différents modèles et hypothèses. La plupart de ces modèles ne correspondent pas aux contraintes de la vie réelle. En ce sens, nous présentons une adaptation des solutions du chapitre 3 pour le débogage totalement distribué et anonyme de systèmes distribués asynchrones. Nous montrons que notre contribution peut être implémentée et ajoutée comme une nouvelle fonctionnalité dans des logiciels de simulation en décrivant les spécifications du modèle utilisé et les détails de développement. Comme illustration, nous ajoutons cette fonctionnalité au sein du logiciel ViSiDiA.

4.1 Introduction

Beaucoup de systèmes d'informations complexes nécessitent un nombre important d'entités et de serveurs interconnectés à travers des réseaux locaux ou distants. De telles applications distribuées ont pour objectif commun de mettre en collaboration l'exécution d'une même tâche. Des problèmes sont soulevés quant à l'accès simultané à des ressources partagées, la détection de faille critique, ou même la stratégie de communication entre processus. La généralisation de réseaux hétérogènes de plus en plus grands implique alors l'étude intensive des algorithmes distribués.

La conception et la validation d'applications distribuées dépendent essentiellement de l'analyse et de la compréhension de l'algorithme sous-jacent qui doit être prouvé, implémenté, débogué et testé. L'émergence de logiciels de simulation est une étape importante vers cet objectif. La visualisation de l'exécution d'un algorithme distribué composée d'observations ordonnées des informations transférées et des changements de propriétés des entités du réseau (processus et liens de communications) est d'une aide importante. La quantification de tous les événements survenant au sein d'un réseau conduit à l'évaluation d'algorithmes distribués en terme de complexité ou de performances globales.

Parmi les services nécessaires pour un système distribué, surveiller l'exécution de chacun des processus de façon distribuée peut être un problème critique, mais s'avère indispensable pour l'évaluation de certaines propriétés d'un réseau ou pour le débogage du système dans sa globalité. Par définition, dans un système totalement distribué asynchrone, il n'y a pas d'horloge globale et il est ainsi impossible de prendre un instantané du réseau entier de manière distribuée. Chaque processus ne connaît ni l'état des autres processus, ni l'état des canaux de communication correspondants. Cette impossibilité est renforcée par l'analyse proposée par Guerraoui et Ruppert [GR05] dans laquelle ils considèrent qu'une vaste majorité des travaux dans le domaine de l'algorithmique distribuée fait l'hypothèse que les processus n'ont pas d'identifiant unique ou ne veulent pas partager leur identité pour des raisons de vie privée.

Par conséquent, il existe trois défis importants dans le développement d'une interface visuelle pour le débogage d'algorithmes distribués. D'abord, un débogueur a besoin de snapshots du système, composés de l'état de chacun des processus et de l'état de chaque lien de communication. Comment pouvons-nous calculer de tels snapshots sur un réseau en utilisant uniquement les informations locales à chacun des processus ? Ensuite, comment satisfaire cet objectif si les processus n'ont pas d'identifiants uniques ou ne veulent pas partager leurs données ? Enfin, ces informations de débogage doivent être disponibles au fur et à mesure de l'exécution de l'algorithme. Comment pouvons-nous obtenir un calcul efficace et la visualisation d'informations de débogage associées à la simulation et l'affichage des événements à l'écran ?

4.1.1 Une Utilité du Débogage : Détection d'Inondations

La motivation principale pour le problème d'évaluation de prédicats est induite par le besoin de réagir relativement à certaines situations qui peuvent survenir dans un système distribué. Par exemple, considérons les zones de catastrophes naturelles placées sous haute surveillance par la mise en place d'un réseau de capteurs (voir l'introduction du chapitre 2).

Pour le cas des inondations, il pourrait être utile de vérifier si le niveau de l'eau ne dépasse pas un certain seuil déterminé sur chacun des capteurs. Lorsque le niveau de l'eau est correct pour un capteur, celui-ci vérifie si cette condition est vérifiée pour ses voisins, les voisins de ses voisins, etc. Si un capteur détecte un problème, un effet "boule de neige" peut survenir, c.-à-d., l'inondation se propage et les autres capteurs n'ayant, jusqu'alors, pas détecté d'anomalies, s'enclenchent. La surveillance doit être recommencée depuis le début.

4.1.2 Résultats

Pour relever les défis mentionnés ci-dessus, ce chapitre présente un cadre complet et stable, nommé *ViSiDiA*, pour la simulation, la visualisation et le débogage d'algorithmes distribués dans les réseaux anonymes. Notre approche repose à la fois sur le développement d'une technique entièrement distribuée avec de solides fondements théoriques, et la conception d'un moteur interactif de simulation de haut niveau offrant une interface simple pour le test et la validation de nouveaux algorithmes. Ce chapitre présente les contributions suivantes :

- nous proposons une méthode originale dans le contexte des réseaux anonymes pour déboguer l'exécution d'algorithmes distribués pour laquelle les informations globales peuvent être calculées par l'intermédiaire des données locales collectées ;
- les informations de débogage sont visualisées en parallèle de l'exécution de l'algorithme distribué (modèle asynchrone). L'utilisateur peut interagir et observer avec le réseau ;
- nous présentons une plate-forme pour la création, la visualisation et la simulation d'algorithmes distribués. C'est un intérêt particulier pour le test et le débogage de nouveaux algorithmes ;
- nous ajoutons au sein du logiciel un débogueur visuel puis nous étendons son API (Application Programming Interface), c.-à-d., les possibilités permettant aux développeurs d'utiliser le débogueur.

Les résultats présentés dans ce chapitre ont été obtenus en collaboration avec Cédric Aguerre et Mohamed Mosbah dont une partie a été publiée dans [AMM12a] et [AMM12b].

4.1.3 État de l'Art

ViSiDiA (Visualization and Simulation of Distributed Algorithms) figure parmi les premières plateformes offrant une API complète pour la visualisation et l'expérimentation d'algorithmes distribués dans les modèles les plus utilisés actuellement : passage de messages, agents mobiles, réécriture de graphes. Toutefois, dès lors que l'on s'intéresse aux champs des logiciels de simulation et de visualisation de réseaux, *ViSiDiA* n'est pas le seul outil développé. De même, lorsque l'on considère le test et le débogage de systèmes distribués, plusieurs solutions existent parmi lesquelles certaines sont commerciales. La particularité de *ViSiDiA* est de reposer sur des fondations théoriques solides.

Étude des Outils de Visualisation et de Simulation Existants

VADE [MPTU98] (Visualisation of Algorithms in Distributed Environnements) permet de visualiser les algorithmes distribués dans un modèle de communication asynchrone. Il permet, entre autres, à l'utilisateur de visualiser l'exécution d'un algorithme depuis une page Web tandis que l'exécution réelle se déroule sur un serveur distant. Ce qui permet une utilisation à distance sur des systèmes hétérogènes. Les hypothèses du modèle sont similaires à celles de *ViSiDiA* dans le cas des algorithmes à base de passage de messages asynchrone : les communications sont fiables et les canaux de communication sont FIFO. L'utilisation d'un système événementiel permet de maintenir la cohérence entre l'animation et l'exécution. Il n'y a pas eu de développement récent pour ce logiciel.

Principalement développé à des fins pédagogiques, LYDIAN [KPT03] (Library of Distributed Algorithms and Animations) propose un environnement pour le développement, la visualisation et l'expérimentation d'algorithmes distribués. LYDIAN s'appuie sur le modèle à base de passage de messages et sur un gestionnaire d'événements. Le développement de nouveaux algorithmes est réalisé par l'intermédiaire de l'API proposée. L'outil propose différentes configurations de simulations (p. ex., gestion de pannes) et de visualisations : visualisation du réseau, des événements par processus, des statistiques ou encore des liens de causalités entre les processus. En raison de ses fonctionnalités, LYDIAN est l'outil se rapprochant le plus de *ViSiDiA*.

Plus récemment, DAJ [BA01] (Distributed Algorithms in Java) est aussi un outil pédagogique et de recherche permettant de concevoir des algorithmes distribués en Java. Concernant la visualisation, l'exécution d'algorithme distribué est accessible depuis un navigateur Web à travers de *applets* Java. Un système est développé en deux étapes. La première étape consiste à développer l'algorithme qui sera exécuté par les processus du système. Dans un second temps, il est nécessaire de développer le moteur du système qui se chargera de créer le réseau et de lancer l'exécution des algorithmes sur chacun des processus. L'outil offre ainsi une granularité plus fine pour la création de schémas d'exécution.

D'un point de vue moins spécifique à l'algorithmique distribuée, PARADE [SK93] (PARallel program Animation Development Environment) est un outil permettant la visualisation de différents types de programmes, de différentes architectures et implémentés dans des langages différents. L'outil se compose de trois modules différents. Le premier s'occupe de rassembler les traces et les actions des programmes. La visualisation est rendue possible par l'intermédiaire d'un second module en liaison avec le premier module par un troisième qui se charge de traduire les traces récupérées en une source de données permettant leur visualisation. POLKA est une extension de PARADE utilisée pour visualiser l'exécution de programmes distribués (et séquentiels) depuis des langages différents.

Concernant les simulateurs réseau les plus connus, OMNet++ [Pon93] et son extension pour les réseaux sans fil [KSW+08], les produits OPNET [Cha99] ou encore NS, figurent parmi les plus utilisés par la communauté scientifique et industrielle. Ils font partie de la catégorie des simulateurs réseau par événements discrets. Ils ne permettent pas explicitement de concevoir et visualiser des algorithmes distribués, mais ils permettent de simuler différentes configurations réseau existantes à travers différents protocoles des couches du modèle OSI. Cependant, OMNeT++ diffère d'autres outils dans le sens où il offre une

plateforme complète allant de l'environnement de développement à la visualisation en passant par la gestion des paramètres réseau.

Étude des Outils de Débogage

Dans la littérature, on distingue deux phases pour le déploiement d'applications distribuées. La première consiste à simuler l'application à l'aide d'un logiciel de simulation qui diffère selon le type de réseaux [Pon93, KSW+08]. Une fois validées à l'aide des simulations, les applications sont ensuite déployées puis testées sur des plateformes de tests. Malgré les nombreux travaux réalisés autour du problème du calcul de l'état global et de ses applications (voir le chapitre précédent), peu d'entre eux sont utilisés par des logiciels de simulation ou de test pour le débogage de systèmes et d'applications distribués.

Parmi les plateformes de test les plus utilisées pour le débogage d'applications distribuées, la majorité est spécifique aux réseaux de capteurs sans fil. Sympathy [RCK+05, RKE05] propose d'analyser le trafic des communications d'un réseau ou de certaines métadonnées, spécialement générées pour le débogage et envoyées à l'aide de messages particuliers, afin de localiser l'endroit d'une défaillance. Malgré la possibilité d'obtenir une vision partielle, mais limitée de la globalité du système, cette technique de débogage est de haut niveau et ne permet donc pas une analyse fine de l'état des processus. Spyglass [BPF+04] trace, affiche et analyse des réseaux de capteurs. Cet outil nécessite qu'un logiciel particulier pour l'agrégation des informations de débogage fonctionne sur chacun des capteurs. Les données ainsi collectées sont ensuite diffusées à la passerelle. Deux problèmes se posent alors, il existe une entité centralisée et la diffusion des informations peut s'avérer très élevée dans le cas de systèmes complexes.

Marionette [WTT+06] et Clairvoyant [YSSW07] offrent tous les deux la possibilité au développeur de déboguer chacun des capteurs directement au niveau de leur code source. Cependant, ces deux solutions font un usage conséquent des canaux de communications et par conséquent soulignent d'autant plus l'impact du débogage dans un système. S^{2}DB [WWG06] offre une plateforme de simulation permettant d'abstraire les éléments d'un système. Cette abstraction s'effectue à différents niveaux comme au niveau réseau ou au niveau processus. La plateforme autorise un développeur à placer des points d'arrêt (*breakpoint*) afin de connaître l'état de certains éléments. En conséquence, ces outils n'offrent aucun moyen de visualisation. De plus, puisque le développeur ne se concentre que sur quelques éléments du système, il est difficile d'en évaluer le comportement global et d'étudier des systèmes complexes. Toutefois, ce type d'approche permet une évaluation en temps réel des états des éléments.

D'autres projets se concentrent sur l'aspect simulation et visualisation de l'exécution tels que VIsage [LBM09]. Ce dernier présente une plateforme qui permet la visualisation et le débogage de systèmes plus complexes en faisant usage de la réalité augmentée, mais se basant, toutefois, sur une entité centralisée.

Nous en déduisons alors que la recherche d'un mécanisme de débogage en temps réel limite l'usage des algorithmes de calcul de snapshot et, a fortiori, limite la vision globale du système. Toutefois, couplé avec une interface de visualisation, un débogueur basé sur les résultats du chapitre 3 fournit de meilleures évaluations du comportement global d'un système distribué. L'évaluation en temps réel peut être émulée par une interface de

visualisation comme proposée par *ViSiDiA*. De plus, il est à noter qu'il n'existe pas de solution qui soit à la fois totalement distribuée et qui considère les réseaux anonymes.

4.1.4 Résumé du Chapitre

Pour ce chapitre, nous allons tout d'abord, dans la section 4.2, présenter la plateforme de simulation et de visualisation d'algorithmes distribués *ViSiDiA*. Nous y détaillons les différents modèles supportés par le logiciel. Nous décrivons brièvement la structure et l'architecture de *ViSiDiA* ainsi que son API. Afin d'illustrer cette présentation, nous utilisons le même algorithme d'exemple durant la totalité du chapitre.

Puis dans la section 4.3, nous montrons les choix et le processus de développement de l'intégration d'un débogueur au sein de *ViSiDiA* dont les fondements théoriques sont inspirés du chapitre 3. Enfin, nous terminons par la présentation de l'extension de l'API permettant d'utiliser le débogueur.

Il est à noter que le code source correspondant aux travaux présentés dans ce chapitre sera prochainement disponible en ligne.

4.2 Introduction à ViSiDiA

La généralisation des réseaux hétérogènes et de grande taille dans de multiples domaines implique des études conséquentes en algorithmique distribuée. Le développement d'applications distribuées est rendu difficile par des problèmes d'accès concurrents aux ressources et de communication entre processus. Une solution a été apportée avec l'émergence de logiciels de simulation. De tels outils permettent de suivre graphiquement et en temps réel l'évolution d'algorithmes distribués, dans le but de trouver des propriétés et des invariants nécessaires aux preuves formelles.

ViSiDiA permet de concevoir, expérimenter et visualiser des algorithmes distribués grâce à l'utilisation d'une interface de programmation spécifique et d'une sémantique visuelle appropriée. Un système distribué est défini par l'intermédiaire d'un éditeur graphique. La simulation d'un algorithme distribué peut être visualisée en temps réel. Le terme "temps réel" signifie ici qu'on observe les changements sur le réseau avec la garantie que l'ordonnancement des différents événements est respecté. C'est un point fondamental grâce auquel il est possible d'étudier les algorithmes, de détecter des erreurs ou encore de calculer leur complexité. Nous présentons dans cette section l'ajout d'une nouvelle fonctionnalité au sein de *ViSiDiA*.

ViSiDiA s'adresse majoritairement à un public scientifique, enseignants chercheurs et étudiants, avec l'objectif de les aider dans la compréhension et l'étude d'algorithmes distribués. Pour être accessible à tous les utilisateurs, cet outil est mis à disposition librement, et est multi-plateforme (Windows, Linux, Mac). À l'instar de VADE et de DAJ, il est également utilisable via une interface Web.

4.2.1 Fondements Théoriques et Modèles

La force de *ViSiDiA*, en plus d'offrir une plateforme complète, réside dans l'utilisation de bases théoriques solides [BGM+01, DM03, BM03]. Ainsi, *ViSiDiA* propose différents modèles de l'algorithmique distribuée tels que le modèle à base de passages de messages, les agents mobiles, les réseaux dynamiques ou encore les réécritures de graphes (présentées dans le chapitre 1 et utilisées dans le chapitre 2).

Passage de Messages

Il existe plusieurs modèles pour gérer les algorithmes distribués. Le plus utilisé d'entre eux est le modèle de communication par passage de messages [Tel00, YK96b]. Chaque sommet du réseau est actif, car il correspond à un processus qui exécute un algorithme de manière indépendante, et qui communique avec ses voisins par envoi de messages (figure 4.1).

Ce modèle présente cependant un inconvénient dans le cas d'un réseau de grande taille. En effet, il conduit à une surconsommation de ressources machine, puisqu'un processus est associé à chaque sommet du graphe réseau.

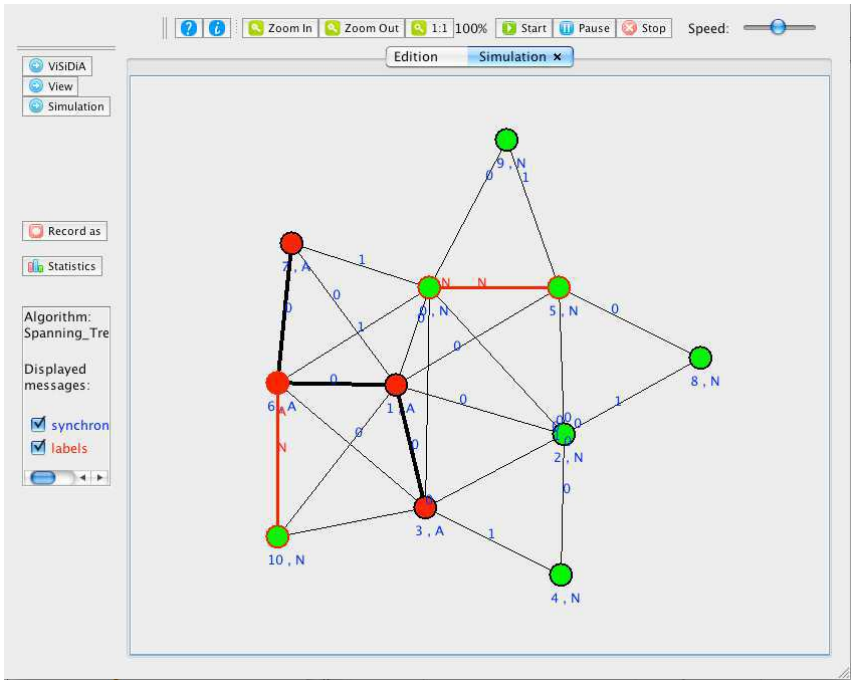

FIGURE 4.1 – Simulation par passage de messages.

Agents Mobiles

Les agents mobiles représentent une alternative à la communication par passage de messages. Les sommets sont passifs et les processus sont des entités (appelées agents) se

déplaçant sur le réseau. Lorsqu'un agent arrive sur un sommet du réseau, il y effectue les tâches nécessaires puis se déplace vers un autre sommet (figure 4.2).

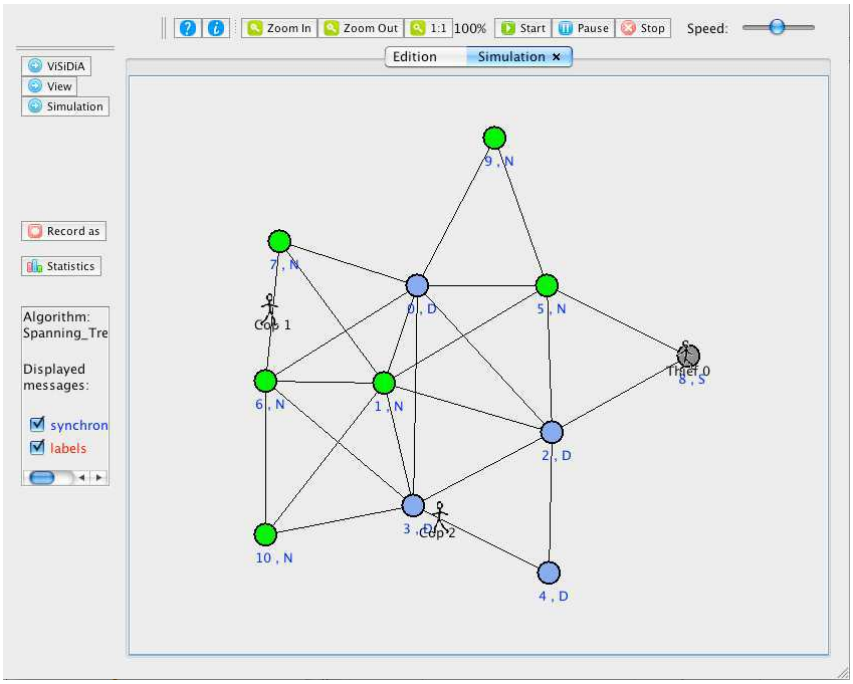

FIGURE 4.2 – Simulation avec agents mobiles.

Processus Itinérants

La simulation par passage de messages ou par agents mobiles telle que présentée précédemment supposait que le réseau était fixe (géométriquement et topologiquement). Dans *ViSiDiA*, il est possible de faire une simulation par messages ou agents sur un réseau dynamique, c'est-à-dire une structure de graphe qui va changer au cours de la simulation.

Les sommets du graphe sont des processus itinérants (appelés aussi capteurs mobiles) qui vont se déplacer au cours de la simulation. Si deux sommets sont suffisamment proches, ils se connectent l'un à l'autre par une arête ; s'ils deviennent trop éloignés, deux sommets se déconnectent l'un de l'autre. Deux processus ne peuvent communiquer que s'ils sont connectés l'un à l'autre, ce qui revient à dire que la simulation par messages ou agents s'effectue sur un ensemble de sous-graphes connexes.

La liberté de déplacement des capteurs mobiles est régie par un deuxième graphe, appelé graphe support. Celui-ci est fixe, et peut être le plus simplement une grille cartésienne ou bien un graphe plus complexe préalablement défini par l'utilisateur. Les capteurs mobiles se déplacent le long des arêtes du graphe support. Les arêtes du graphe dynamique peuvent ne pas correspondre à des arêtes du graphe support (figure 4.3).

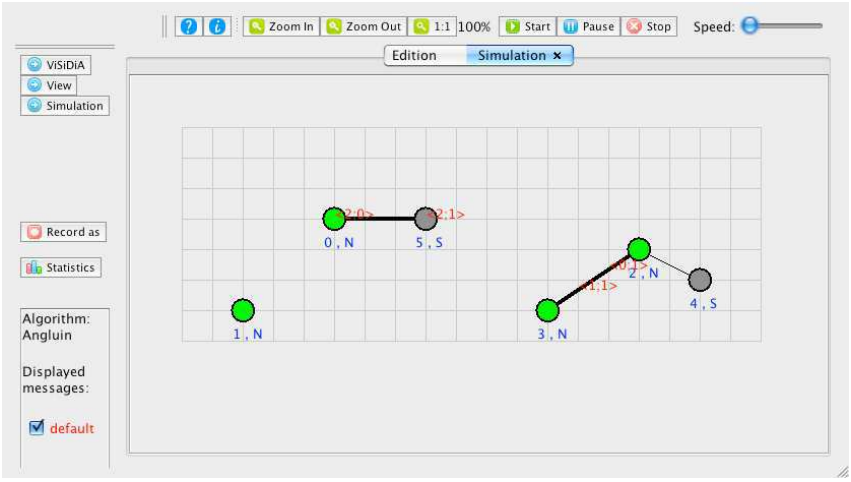

FIGURE 4.3 – Simulation par envoi de messages avec des processus itinérants.

Réécriture de Graphes

ViSiDiA peut effectuer la simulation d'un algorithme prédéfini, par exemple contenu dans un fichier. Il existe cependant une alternative : l'utilisateur peut "dessiner" un algorithme avec la souris. Il s'agit de mettre en place un système de réécriture de graphe qui va se substituer à l'usage classique d'un algorithme. Un tel système se compose d'un ensemble de règles qui vont être spécifiées par l'utilisateur grâce à une interface graphique dédiée. Ces règles vont s'appliquer comme des calculs locaux sur le réseau, en fonction d'un contexte également défini par l'utilisateur (figure 4.4).

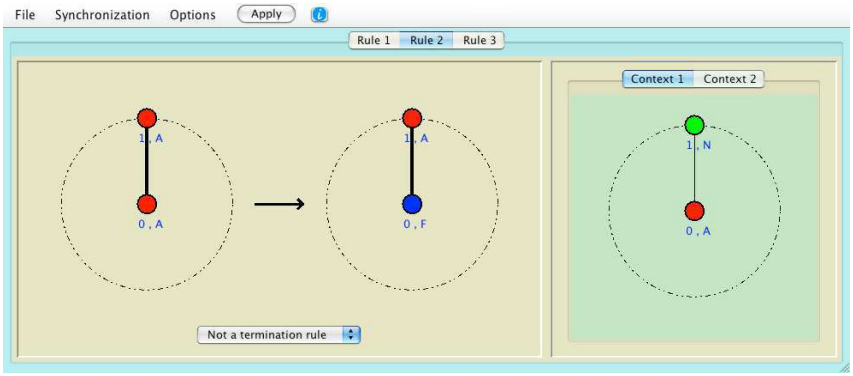

FIGURE 4.4 – Exemple de règles de réécriture.

4.2.2 Illustration par un Exemple : Algorithme de Diffusion

Dans un souci de compréhension, nous allons illustrer notre contribution à l'aide d'un algorithme de diffusion (ou *broadcast*) simple appliqué à un système distribué dans le modèle à base de passage de messages. Le réseau est modélisé par un graphe dont les

sommets et les arêtes représentent respectivement les processus autonomes et les liens de communication.

Le graphe est tel qu'initialement un processus est étiqueté A et tous les autres sont étiquetés N. L'algorithme est le suivant. À tout moment, chaque processus étiqueté A envoie un message à tous ses voisins, et tous les processus N recevant un message changent leur étiquette pour A et propagent le message à tous leurs voisins. Lorsqu'un message est reçu, l'arête entre le processus émetteur et le processus récepteur est marquée. L'algorithme termine dès qu'il n'y a plus aucun processus étiqueté N. Toutes les arêtes marquées et les sommets forment alors un arbre couvrant du graphe initial. L'algorithme est détaillé par l'algorithme 10.

Algorithme 10: Exemple de l'algorithme de diffusion.

début
  si $\lambda(v) = A$ **alors**
    envoyer$< Wave >$ à tous les voisins de v ;
  sinon
    recevoir$< Wave >$ par le port p ;
    envoyer$< Ack >$ via le port p ;
    $\lambda(v) := A$;
    marquer l'arête e identifiée par p ;
    envoyer$< Wave >$ via tous les port $q \neq p$;
  ignorer et **accuser** les autres messages $< Wave >$;
  attendre l'arrivée des messages $< Ack >$ par tous les port $q \neq p$;
fin

Dans la suite de ce chapitre, nous donnerons l'implémentation détaillée de l'algorithme à l'aide de l'API de *ViSiDiA*. Nous utiliserons ce même algorithme pour illustrer l'extension de l'API pour la mise en place d'un mécanisme de débogage.

4.2.3 Structure et Architecture

L'outil *ViSiDiA* est conçu selon un schéma multi-couches qui s'articule autour du simulateur. Au-dessus de celui-ci se trouve l'interface graphique (Graphical User Interface, GUI) qui permet de visualiser la simulation en cours, et avec laquelle l'utilisateur peut interagir. Enfin, on trouve une interface de programmation qui permet aux utilisateurs de développer leurs propres algorithmes et de les utiliser dans *ViSiDiA*. Dans ce système, chaque couche communique uniquement avec ses voisins immédiats. Le schéma multi-couches représente la composante structurelle de *ViSiDiA*.

Architecture Système

La conception de *ViSiDiA* adopte le schéma classique modèle-vue-contrôleur (Model-View-Controller, MVC). Le modèle correspond au graphe représentant le réseau distribué,

la vue est l'interface graphique, et le contrôleur est la console de simulation. La console de simulation reçoit les événements relatifs aux actions utilisateurs depuis l'interface graphique, et s'occupe des mises à jour et des événements de synchronisations entre le modèle et la vue. Ce schéma d'implémentation est illustré par la figure 4.5.

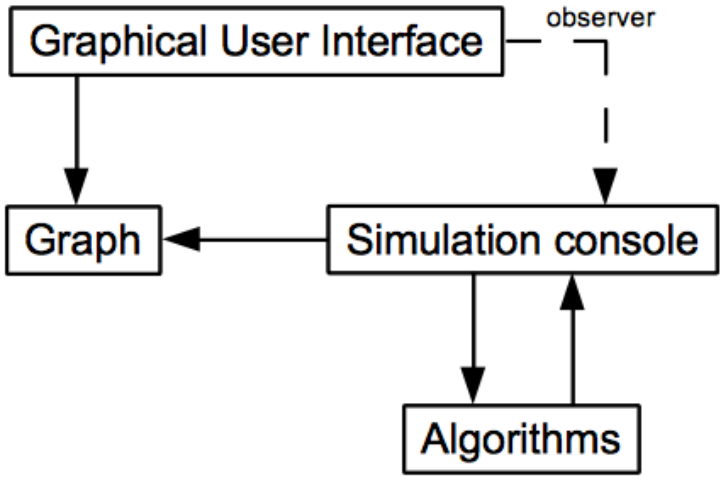

FIGURE 4.5 – Organisation structurelle et logique de *ViSiDiA*. Les flèches pleines représentent les associations directes (l'interface graphique et la console ont toutes deux accès au graphe ; la console et les algorithmes peuvent communiquer entre eux). La flèche pointillée est une association indirecte ; l'interface graphique observe la console (p. ex., pour mettre à jour l'affichage) mais la console ne peut pas directement transmettre des informations à l'interface graphique.

L'API de *ViSiDiA* permet d'implémenter des algorithmes distribués grâce à un ensemble de primitives permettant, en particulier, l'accès aux états des processus ou des canaux de communication ou la communication entre les processus. Cette API est liée au coeur du simulateur, incluant le graphe du réseau et la console, au-dessus duquel l'interface graphique est construite. Ainsi, cela forme une architecture en trois couches qui définit la structure de l'application. Il est à noter que la console est indépendante de l'interface graphique autorisant ainsi la simulation sans la visualisation. *ViSiDiA* suit une approche de développement orientée objet, et l'utilisation du langage Java permet d'avoir une application qui peut être utilisée soit localement sur un ordinateur soit par un navigateur web (p. ex., applet).

Sémantiques Visuelles

L'interface graphique de *ViSiDiA* est un environnement graphique qui permet à l'utilisateur de dessiner facilement un réseau et de visualiser l'exécution d'un algorithme distribué. À travers cette interface, l'utilisateur peut interagir avec les processus et les liens

de communication pour ajouter, supprimer, déplacer ou connecter. Des opérations d'importation et d'exportation sont aussi disponibles pour faciliter la gestion de différents scénarios de tests. Les sommets et les arêtes ont des propriétés spécifiques telles que l'étiquette ou le poids. L'utilisateur peut donc ajuster ces propriétés prédéfinies, en ajouter de nouvelles, décider de leur affichage par l'intermédiaire d'une fenêtre de dialogue (voir la figure 4.6).

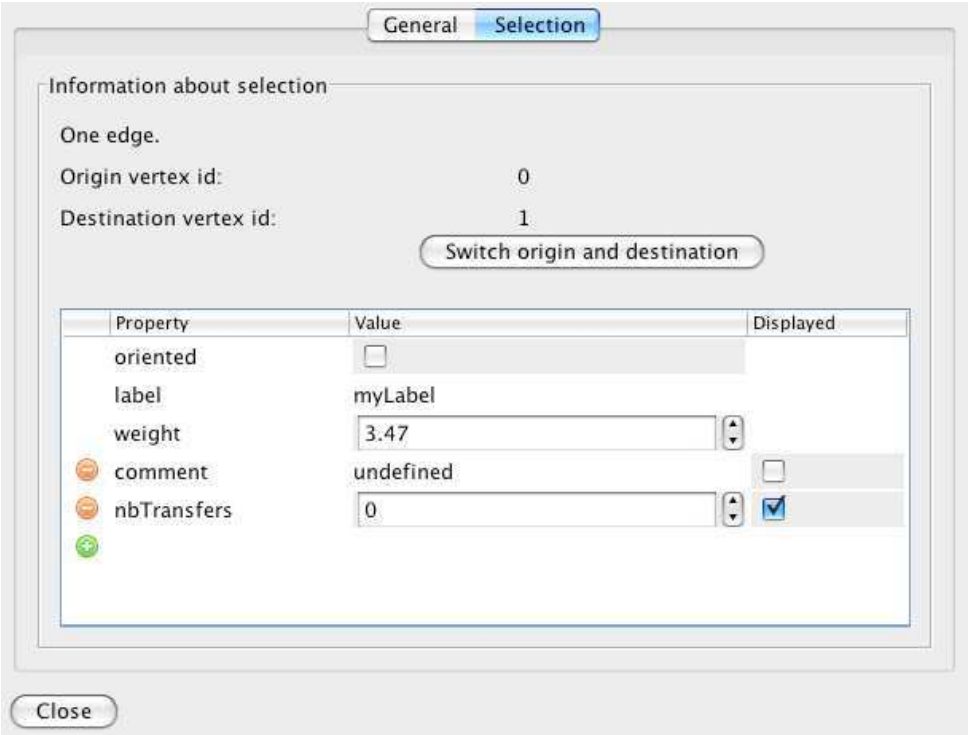

FIGURE 4.6 – Exemple de configuration des propriétés d'une arête. Trois propriétés sont liées à une arête et ne peuvent être supprimées (orientation, étiquette et poids). Dans cet exemple, l'utilisateur a ajouté deux autres propriétés (commentaires et nombre de transferts) dont seule la seconde sera affichée durant la simulation.

Depuis un graphe donné, il est possible de lancer l'exécution d'une ou plusieurs simulations pour notamment étudier les comportements d'un algorithme sur un même réseau pour des exécutions différentes. Il est possible, durant une exécution, d'interagir avec le réseau (p. ex., changer certaines propriétés).

Chaque sommet du graphe correspond au processus, et les arêtes représentent les canaux de communication entre les processus sur lesquels les messages circulent. Les sommets sont des cercles dont l'étiquette est représentée sur la partie interne et la couleur est modifiable via une palette de couleurs. Dans la figure 4.7(a), les parties internes des sommets A et des sommets N sont respectivement rouges et vertes. La partie externe d'un sommet indique si celui-ci est sélectionné (rouge) ou non (noir). En dessous d'un sommet apparaissent ses propriétés : son identifiant, son étiquette ou toute autre valeur préalablement définie. L'utilisateur contrôle l'information à afficher. Les arêtes sont des

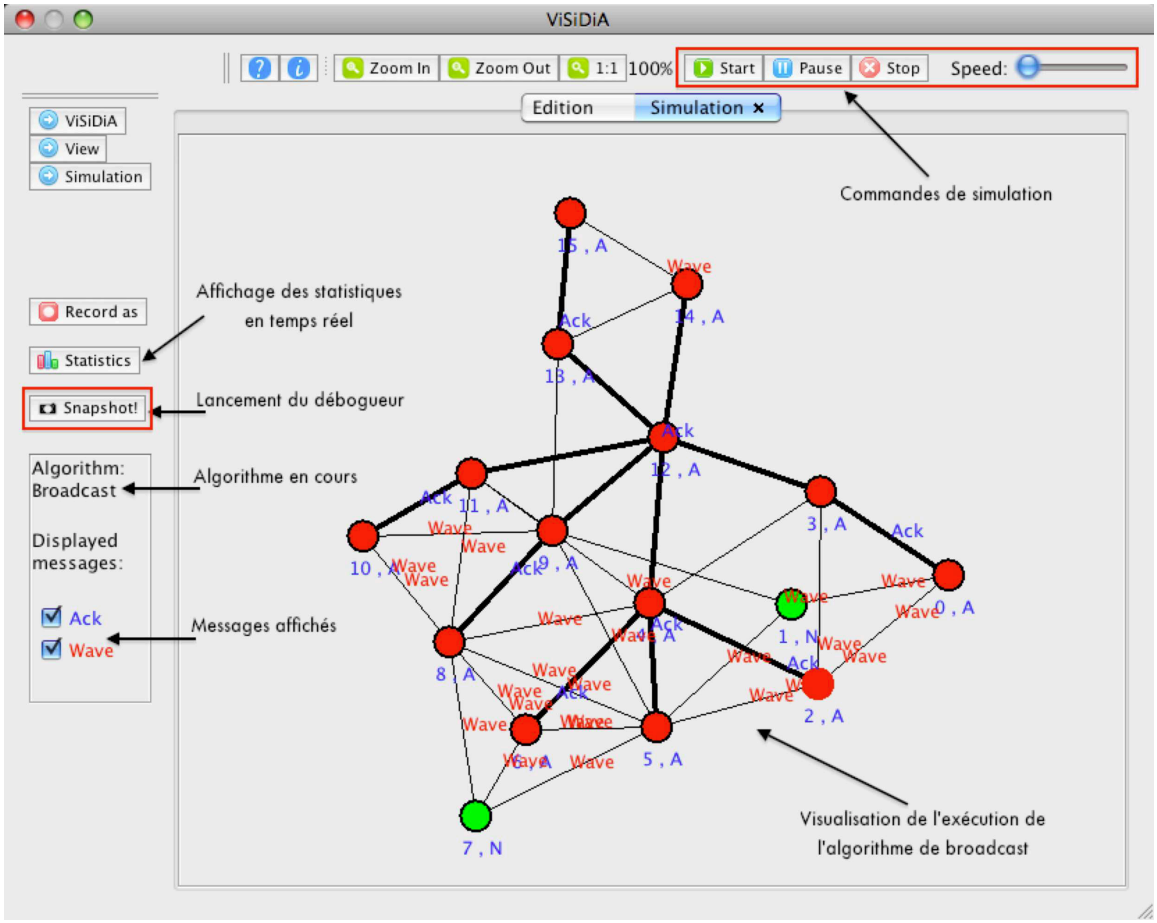

FIGURE 4.7 – Aperçu de l'interface graphique. Les paramètres sont accessibles en haut et à gauche du panneau. Le système distribué est affiché tandis que l'algorithme s'exécute. L'utilisateur peut observer les changements sur les états des processus et des canaux aussi bien que la circulation des messages. Un bouton permettant de lancer le débogage (*Snapshot !*) est disponible sur la gauche de l'interface.

segments (ou des flèches dans le cas orienté). L'épaisseur d'une arête indique si celle-ci est sélectionnée ou non. À l'instar des sommets, il est possible de changer la couleur des arêtes.

Un processus est implémenté par un *thread* autonome associé à une copie locale de l'algorithme simulé. Pour notre approche, nous considérons les réseaux anonymes et il est indispensable de faire des copies de l'algorithme sur chacun des processus afin de ne pas distinguer certaines parties du réseau avec des algorithmes différents. Puisque les processus n'ont qu'une connaissance limitée à leur voisinage immédiat et quelques propriétés globales sur le réseau (taille, degré), les processus ne peuvent que modifier leur état et envoyer ou recevoir des messages par les canaux de communications à travers les ports correspondants.

Les messages sont des informations textuelles qui transitent d'un processus émetteur vers un processus récepteur. Des couleurs différentes sont utilisées selon le type de mes-

sage défini par l'algorithme. Dans le cas de l'algorithme de diffusion, il y a des messages de *vagues* (en rouge) et des messages d'accusé de réception (en bleu). Les changements dynamiques de couleurs ou de propriétés des éléments du réseau donnent une perception instantanée et globale de l'algorithme en cours d'exécution. Suivant la complexité des algorithmes, il est possible d'ajuster la vitesse d'exécution ou de la suspendre afin d'accéder aux informations détaillées plus aisément ou encore de visualiser les statistiques de l'algorithme (figure 4.8).

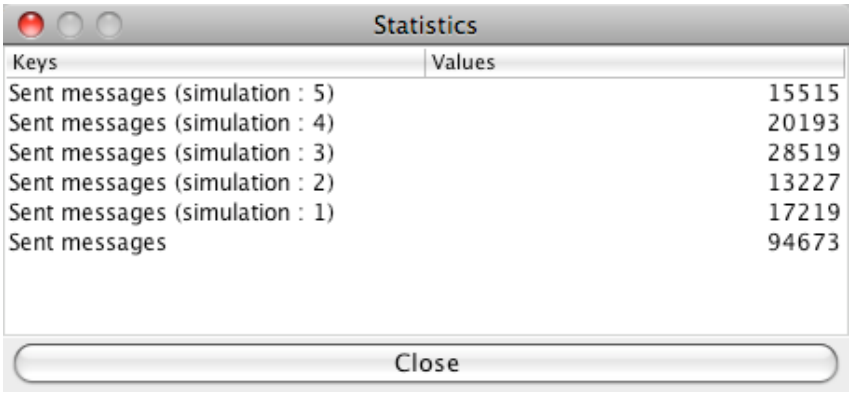

FIGURE 4.8 – Quelques statistiques sur le nombre de messages envoyés par cinq simulations de d'exécution d'un algorithme à la suite.

Gestion des Messages

Un processus peut envoyer (resp. recevoir) des messages par le biais de portes sans, pour autant, connaître l'identité du processus récepteur (resp. émetteur). En conséquence, un processus détermine les messages à envoyer, puis en informe la console de simulation afin de les envoyer aux destinataires appropriés. Un message est livré lorsque la console a placé celui-ci dans la queue des messages du récepteur. Selon les hypothèses de notre modèle, voir chapitre 3, section 3.1.2, les canaux de communications sont FIFO. Il est à noter que l'utilisation d'une console de simulation est uniquement nécessaire à la visualisation. L'API de *ViSiDiA* ne permet pas à un processus d'accéder à l'ensemble du réseau par l'intermédiaire de la console et ainsi de mettre à mal le fonctionnement légitime d'un système distribué.

Console de Simulation et Ordonnancement des Événements

La console est responsable de l'ordonnancement de toutes les opérations sur le réseau, incluant la remise des messages. Ces opérations sont des commandes auto-exécutables gérées par la console par l'intermédiaire d'un mécanisme d'événements et d'accusés de réception. Lorsqu'un processus demande une nouvelle commande à traiter, la console génère tout d'abord un événement et le verrouille jusqu'au traitement complet de la commande. Ensuite, la console demande l'exécution de la commande puis attend l'accusé de réception qui indique la terminaison de la commande. Le verrou sur l'événement est finalement

levé laissant place au traitement de la commande suivante. Cette gestion d'événements est utilisée dans le but de synchroniser les événements de simulations avec l'affichage sur l'interface graphique.

Serveur Local et Serveur Distant

La manière dont les commandes sont échangées entre la console de simulation et les processus dépend de la nature du réseau : local (les processus sont sur une même machine) ou distant (les processus sont répartis sur plusieurs machines). Cet échange est géré par un ou plusieurs serveurs, locaux ou distants, qui sont les seuls à savoir la manière dont il faut échanger les commandes, sans pour autant en connaître leur contenu exact.

En réseau local, une seule machine abrite la console, les processus et un unique serveur local. En réseau distant, chaque machine abrite un serveur ainsi que plusieurs processus gérés par le serveur auquel ils sont rattachés. La console gère plusieurs serveurs, et peut se situer soit sur une machine abritant un serveur, soit sur une machine tierce.

La communication est toujours locale entre un serveur et ses processus. En réseau distant, seule la communication entre la console et les serveurs est distante. Peu importe le type du réseau, deux processus (qu'ils soient sur le même serveur ou non) ne communiquent jamais directement entre eux. Un processus "émetteur" doit demander à son serveur de contacter un processus "récepteur". Si le serveur n'a pas connaissance de ce récepteur lui-même, il délègue cette tâche à la console.

4.2.4 Aperçu de l'API ViSiDiA

Un système distribué est composé de processus exécutant le même algorithme et communiquant entre eux. Classiquement, les processus n'ont que très peu d'informations sur le réseau et leurs connaissances se limitent principalement au voisinage immédiat. Un ensemble de fonctionnalités est ainsi disponible dans l'API de *ViSiDiA* :

- accès/modification des propriétés des sommets et des arêtes du réseau ;
- communication par envoi/réception de messages ;
- connaissance du nombre de voisins et, éventuellement, de la taille du réseau.

Les processus, par l'intermédiaire de leur algorithme, ont également accès à des primitives de synchronisation plus élaborées pour la mise en oeuvre des calculs locaux (p. ex., par rendez-vous, par élection locale de rayon 1 ou 2).

Le modèle de l'API de *ViSiDiA* impose que la classe d'un algorithme hérite de la classe *Algorithm* afin d'avoir accès à toutes les primitives de l'API. Comme expliqué précédemment, chaque processus du *ViSiDiA* est un *thread*. Le comportement de l'algorithme est donc dépendant du contenu de l'implémentation de la méthode *init()* (voir l'implémentation de l'algorithme de diffusion dans la figure 4.9).

Envoi et Réception de Messages

L'API propose différentes méthodes pour l'envoi et la réception de messages. Il est possible d'envoyer un message à travers un port particulier avec la méthode *sendTo()* ou d'envoyer un message à tous ses voisins avec la méthode *sendAll()*. De même, il existe la

```
[...]

public class BroadcastAlgorithm extends Algorithm{

  static Message wave = new Message("Wave", true);
  static Message ack = new Message("Acknowledgment", true);

  public void init(){

    int arity = getArity();
    String label = getProperty("label");

    if(label.compareTo("A") == 0)
      for{int neighbor = 0 ; neighbor <= arity-1 ; neihgbor++}
        sendTo(neighbor, wave);
    else{
      Door door = receiveMessage();
      int doorNum = door.getNum();
      sendTo(doorNum, ack)

      setProperty("label", "A")
      setDoorState(new MarkedState(true), doorNum);

      for{int neighbor = 0 ; neighbor <= arity-1 ; neighbor++}
        if(neighbor != doorNum)
          sendTo(neighbor, wave);
    }

    /* Accuser la réception d'autres messages Wave.
     * Attendre les accusés de réception de voisins
     * autres que ceux identifiés par doorNum. */
    waitAcks();
  }
}
```

FIGURE 4.9 – Code source de l'algorithme de diffusion (algorithme 10).

méthode *receiveFrom()* qui permet la réception d'un message depuis un port particulier. Il est à noter que la fonction *receiveFrom()* est surchargée :

- attente de la réception d'un message depuis un port, peu importe lequel ;
- attente de la réception d'un message depuis un port spécifique

Toutefois, l'appel à cette fonction est bloquant, c.-à-d., le processus attend uniquement la réception d'un message. Il est donc nécessaire de prendre des précautions quant à son utilisation. *ViSiDiA* fournit des primitives de tests pour évaluer le contenu de la queue des messages afin de limiter les appels bloquants. Cet aspect prend tout son sens lorsqu'il est nécessaire de répartir des messages suivant leur type ou d'effectuer certaines actions durant l'attente de l'arrivée d'un message.

Messages et Structure de Données

Par l'intermédiaire de la classe abstraite *Message*, *ViSiDiA* offre la possibilité au développeur de définir ses propres messages dans la conception de ses algorithmes. Il existe des implémentations déjà existantes de la classe *Message* telles que *StringMessage* ou *IntegerMessage*. Un message possède quelques propriétés comme sa couleur ou son type. Le fait d'utiliser un langage-objet comme Java permet à un processus de déterminer son type lors de sa réception. Il est alors possible d'exécuter certaines tâches spécifiques selon les

messages reçus. Durant le processus de développement d'un algorithme complexe, il peut être est intéressant de modifier la couleur d'un message afin d'offrir une meilleure lisibilité lors de la visualisation de l'exécution. Dans l'exemple d'une exécution de l'algorithme de diffusion (figure 4.7), deux types de messages sont distingués : *Ack* en bleu et *Wave* en rouge (par défaut).

Afin d'aider la visualisation et la compréhension de l'exécution d'un algorithme, *ViSiDiA* propose de lier des propriétés particulières à chacun des processus. L'étiquette d'un processus permet de le distinguer des autres processus, cette propriété ne peut donc pas être retirée. D'un point de vue de l'implémentation, les propriétés d'un processus sont regroupées dans un ensemble $\{(clé, valeur)\}$. Deux méthodes sont donc accessibles depuis l'API : *getProperty()* pour récupérer la valeur identifiée par le nom d'une propriété, et *setProperty()* pour sa mise à jour. Dans la figure 4.9, initialement, c.-à-d., avant le lancement de l'algorithme, tous les processus ont l'étiquette N sauf un seul. Une fois qu'un processus a été visité, l'étiquette prend la valeur A.

Autres Propriétés

L'API de *ViSiDiA* ne se limite pas aux seules méthodes citées précédemment. Un processus a la possibilité de connaître certaines propriétés globales du réseau comme le nombre de processus (méthode *getNetSize()*) ou encore le nombre de voisins (méthode *getArity()*).

Exemple 4.1 *Dans l'exemple de code source (voir la figure 4.9), il est à noter que le développeur n'intervient aucunement dans la gestion de la visualisation de l'exécution. Il ne peut simplement interagir avec les propriétés d'un processus et marquer ses arêtes incidentes. L'API offre une souplesse de développement permettant à un développeur de se focaliser sur l'essentiel.*

La simplicité de développement d'algorithmes distribués ainsi que la clarté et la fluidité de la visualisation permettent d'affirmer que *ViSiDiA* est une solution complète.

C'est donc tout naturellement que nous nous sommes orientés dans le choix de ce logiciel par rapport à des solutions existantes. Dans la suite, nous montrons l'intégration d'une solution de débogage dans *ViSiDiA* en améliorant l'expérience utilisateur aussi bien au niveau de la visualisation qu'au niveau des possibilités offertes par l'API.

4.3 Visualisation des Informations de Débogage

Cette section présente l'implémentation des solutions présentées dans le chapitre 3. Nous développons une couche de débogage au sein d'un logiciel de visualisation dans laquelle les problèmes de calculs d'états globaux et d'évaluation de prédicats sont pris en considération.

4.3.1 Idée Générale

La conception et le développement d'algorithmes distribués dans *ViSiDiA* est facilitée par l'accès à des primitives de débogage étroitement liées à de nouveaux éléments de

visualisation.

D'après l'architecture proposée et présentée précédemment, chaque processus, correspondant à un *thread* Java, fonctionne de manière autonome. Tous les événements de simulation tels que l'envoi ou la réception et les événements de visualisation tels que l'affichage des messages ou des propriétés des éléments sont obligatoirement répertoriés et gérés par la console. Dans ce contexte, il est donc trivial d'utiliser cette console afin de collecter les informations nécessaires à la réalisation d'un snapshot du système à un instant donné. Toutefois, nous souhaitons fournir une solution totalement distribuée.

Pour cela, nous utilisons les algorithmes présentés dans le chapitre précédent. L'idée générale consiste en l'adjonction d'un algorithme de snapshot au-dessus d'un algorithme développé par un utilisateur. Ainsi, le débogage et l'exécution d'un algorithme sont entrelacés et ne nécessitent aucune intervention de la console, si ce n'est pour la gestion des événements. Nous insistons sur l'aspect totalement distribué de cette solution : les informations de débogage sont transmises unilatéralement des processus vers la console, et non l'inverse.

4.3.2 Évaluation d'un Prédicat Global

Afin de pouvoir généraliser la détection de propriétés stables présentée dans le chapitre 3, nous étendons l'idée de l'algorithme 8 pour considérer des prédicats autres que la terminaison d'un algorithme.

Ainsi, depuis les snapshots locaux, on peut détecter un instant de l'exécution durant lequel un prédicat global est satisfait. Cette procédure permet de répondre au problème de la détection d'inondations présenté dans l'introduction. Les idées principales de cette procédure sont les suivantes :

1. au moins un processus initialise le calcul d'un snapshot (algorithme 4) ;
2. si le snapshot local du processus p est tel que le prédicat est satisfait et que tous les snapshots locaux sont calculés (algorithme 6), alors p initialise une occurrence de l'algorithme SSP (algorithme 5) afin de vérifier si ce prédicat est vérifié dans tout le système ;
3. si le snapshot local du processus p est tel le prédicat n'est pas satisfait, alors, p envoie un signal pour informer tous les processus de l'existence d'un problème. Par conséquent, au moins un autre snapshot doit être calculé. Les variables de l'algorithme 4 et de l'algorithme 6 sont réinitialisées.

4.3.3 Architecture et Processus de Développement

La mise en place d'une couche de débogage et du mécanisme de détection de prédicats précédent au sein d'un logiciel existant tel que *ViSiDiA* soulève des problématiques inhérentes à l'algorithmique distribuée. Comme précisé précédemment, nous voulons une solution totalement distribuée et anonyme, c.-à-d., n'autorisant pas la communication entre la console et les processus, et une solution fiable, c.-à-d., permettant de diagnostiquer un problème même lors d'un *crash* d'application surveillée.

Couche de Débogage

Considérons un système distribué dans lequel un processus échoue (*crash*) : le thread correspondant provoque ainsi une situation d'interblocage puisque le comportement des voisins du processus est dépendant de son état. Nous souhaitons que le débogueur propage une information selon laquelle "un voisin a échoué". Puisque le débogueur ne peut pas être le thread en situation d'interblocage, nous associons un second thread à chaque processus uniquement utilisé pour le débogage. Un processus est composé alors de deux threads : un pour l'exécution de l'algorithme, l'autre pour le débogage. Dans la console, transitent maintenant les messages de l'algorithme et les messages de débogage. Un processus distribue chaque message selon son type au thread approprié. Le thread de débogage surveille le thread de l'algorithme et ses messages entrants. Si l'algorithme échoue, tous les messages sont traités par le thread de débogage (voir la figure 4.10).

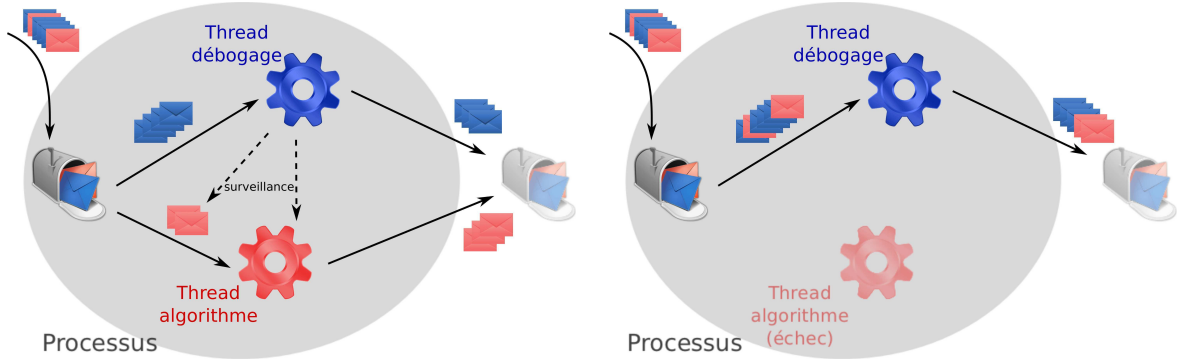

FIGURE 4.10 – L'échange de messages à l'échelle d'un processus. Un processus est composé d'une boîte-aux-lettre et de deux threads : un pour l'exécution normale de l'algorithme, l'autre pour le débogage. À gauche : une situation normale, les messages sont routés selon leur type. À droite : si l'exécution de l'algorithme échoue, le thread de débogage prend le relais.

Composants Visuels

L'interface graphique de *ViSiDiA* contient un bouton permettant de lancer le débogueur, uniquement pendant l'exécution d'un algorithme. Les informations de débogage apparaissent sous la forme d'une arborescence composée des processus et de chacun de leurs canaux entrants (figure 4.11(b)), et certains prédicats globaux sont évalués et listés dans un autre onglet (figure 4.11(c)). Ces informations peuvent aussi être visualisées au passage de la souris sur un sommet du graphe. Nous développons actuellement une version améliorée de la visualisation basée sur la notion de *weak snapshots*. Un *weak snapshot* calculé par chacun des processus apparaît à côté de chacun des sommets du graphe au passage de la souris.

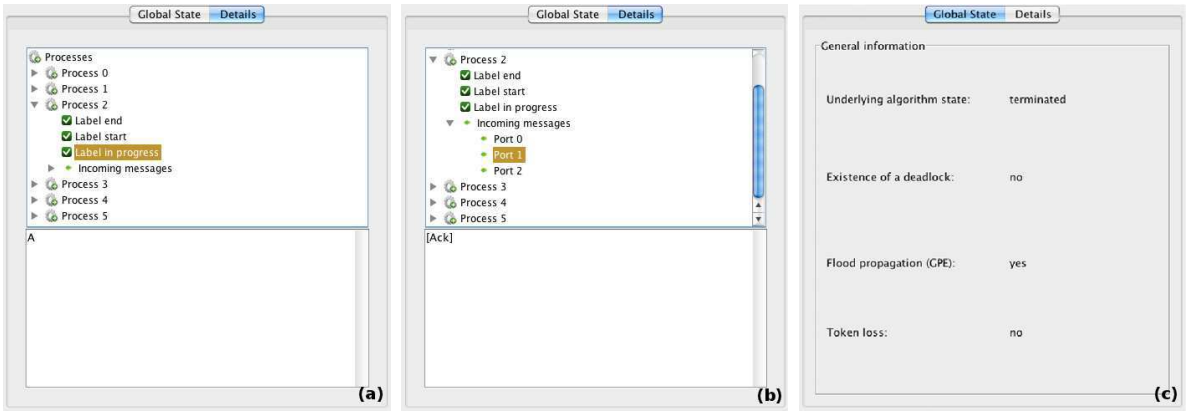

FIGURE 4.11 – Aperçu des résultats du calcul d'un snapshot réalisé durant l'exécution de l'algorithme de diffusion (algorithme 10). (a) et (b) : détails du processus 2 comprenant la valeur de son étiquette et les contenus de ses canaux entrants. (c) : liste des prédicats globaux évalués durant l'exécution telss que la terminaison de l'algorithme, la perte d'un jeton ou encore la propagation d'une inondation.

4.3.4 Extension de l'API

Afin de surveiller l'évolution de la valeur d'une variable spécifique d'un processus ou de vérifier l'existence d'un prédicat dans le système au cours de l'exécution, il est indispensable d'étendre l'API initiale proposée par *ViSiDiA*. Nous présentons, à l'aide du même exemple, une façon d'utiliser le mécanisme de débogage.

Une erreur fréquente lors de l'implémentation de l'algorithme 10 est l'oubli de l'envoi des accusés de réception ou l'envoi d'un accusé à tous les voisins. Ainsi, le comportement de l'algorithme n'est pas celui escompté par le développeur. Pour corriger ce problème, un développeur pourrait aimer connaître l'évolution de l'état d'un processus (p. ex., l'étiquette du processus) et les messages échangés (p. ex., le contenu des messages). Sans mécanisme de débogage, une approche classique consisterait en l'écriture de certaines valeurs sur la sortie standard (ou dans un fichier, un par processus) telles que le snapshot local de chacun des processus ou en l'envoi des snapshots locaux des processus directement à la console afin de les afficher sur l'interface graphique. Mais, comme expliqué précédemment, notre modèle suppose des communications asynchrones et il n'est pas possible pour un processus d'accéder à une horloge globale. Par conséquent, il n'est pas possible de s'appuyer sur ces méthodes de débogage puisque les données récupérées pourraient être non cohérentes.

Afin de surveiller une variable particulière, il suffit d'utiliser la primitive *registerVariable()*. Cette méthode est suffisante pour détecter les deux erreurs citées pour la conception de l'algorithme 10. Une fois que le calcul du snapshot est réalisé, même si l'exécution du *broadcast* échoue pour certains processus, le panneau de *ViSiDiA* (voir les figures 4.11(b) et (c)) montre l'état de tous les processus et des canaux de communication correspondants. Ainsi des erreurs de conception peuvent être détectées et corrigées.

De même, nous proposons un ensemble de méthodes permettant d'évaluer l'existence

d'un prédicat dans le système. L'une de ces méthodes permet de spécifier si l'exécution de l'algorithme sous-jacent est terminée (méthode *setTerminated()*). À la fin du calcul du snapshot, si l'algorithme est terminé, cette information sera affichée dans le panneau de *ViSiDiA* (figure 4.11(c)). De plus, nous permettons aux développeurs de définir leurs propres prédicats (p. ex., la détection d'inondation). Pour cela, il suffit de créer un nouveau prédicat et d'utiliser la primitive *addGlobalPredicate()* permettant de préciser au débogueur le besoin d'évaluer un prédicat sur chacun des processus. Il est à noter qu'il est possible de vérifier l'existence de plusieurs prédicats simultanément durant l'exécution d'un algorithme (un *ET* logique sur les prédicats est effectué).

Dans le cas de l'algorithme de diffusion (figure 4.12), nous demandons au débogueur de suivre l'évolution de l'étiquette du processus à l'aide de la méthode *registerVariable()*. Nous avons ensuite créé un prédicat *sp* puis nous avons informé le débogueur de l'évaluer (*addGlobalPredicate()*). Enfin, nous voulons être informés de la terminaison de l'algorithme.

```
[...]

int arity = getArity}();
String label = getProperty}("label");

registerVariable("label␣start", label);
addGlobalPredicate}(sp);

if(label.compareTo("A") == 0)
  for{int neighbor = 0 ; neighbor <= arity-1 ; neihgbor++}
    sendTo(neighbor, wave);
else{
  Door door = receiveMessage();
  int doorNum = door.getNum();
  sendTo(doorNum, ack)

  putProperty("label", new String("A"));
  registerVariable("label␣in␣progress", label);

  setDoorState(new MarkedState(true), doorNum);

  for{int neighbor = 0 ; neighbor <= arity-1 ; neighbor++}
    if(neighbor != doorNum)
      sendTo(neighbor, wave);

  registerVariable}("label␣end", label);
}
  waitAcks();
  setTerminated(true);

[...]
```

FIGURE 4.12 – Code source de l'algorithme de diffusion avec usage des primitives de débogage.

4.4 Conclusion

Dans ce chapitre, nous présentons une solution complète pour le débogage s'appuyant sur une architecture totalement distribuée et ne supposant aucune hypothèse sur le ré-

seau. Plusieurs processus peuvent initialiser une requête de débogage simultanément et n'ont connaissance que de la taille du réseau. Le modèle considéré est le modèle distribué anonyme à base de passage de messages présenté dans le chapitre précédent. En raison de sa nature distribuée, nous montrons que notre solution peut être aisément ajoutée au sein d'un logiciel de simulation et visualisation tel que *ViSiDiA*. Ainsi, nous permettons aux concepteurs d'algorithmes distribués de connaître le comportement d'un système et de déterminer l'endroit où quelque chose tourne mal. La fonctionnalité que nous rajoutons permet aussi, de manière distribuée, de savoir ce qu'il se passe globalement dans le réseau durant une exécution particulière d'un algorithme à l'aide d'un mécanisme de détection de propriétés stables telles que les deadlocks ou la terminaison.

Comme première évolution, un système de *checkpoint* et de *rollback recovery* (voir le chapitre 3 à la section 3.5 pour les fondements théoriques) est en cours d'intégration au sein de *ViSiDiA*. C'est une extension des commandes de simulation du logiciel dans laquelle des boutons *précédent* et *suivant* sont ajoutés. À l'aide de ces boutons, l'utilisateur a la possibilité de parcourir les différents *checkpoints* réalisés au cours de l'exécution et ainsi d'avoir une visualisation précise des états de chacun des éléments du système. Parallèlement, nous mettons en place le calcul et la visualisation de *weak snapshots*. Nous avons l'intention de nous concentrer sur cette technique afin d'améliorer l'expérience utilisateur lors de l'utilisation de *ViSiDiA*. Chaque weak snapshot calculé est accessible depuis les propriétés d'un processus et permet à ce processus de détecter d'autres propriétés stables. Enfin, nous allons étendre cette solution aux autres modèles intégrés dans *ViSiDiA* : réécriture de graphes et agents mobiles.

Chapitre 5

Conclusion et Perspectives

Dans ce mémoire, nous avons étudié deux grands problèmes de l'algorithmique distribuée ainsi que certaines variantes. Pour ces problèmes, nous considérons les réseaux anonymes. Au fil des chapitres, nous montrons les limites et la puissance de différents modèles de systèmes distribués. Pour ce faire, nous présentons des résultats d'impossibilité nécessaires à l'évaluation de ces limites.

D'abord, nous avons introduit le modèle de diffusion (asynchrone et synchrone) pour ensuite proposer une caractérisation complète des graphes, ou réseaux, pour lesquels il existe une solution pour les problèmes de nommage/énumération et de l'élection. Pour cela, nous avons donné des résultats d'impossibilité basés sur des homomorphismes de graphes. Nous avons montré que ces résultats sont nécessaires et suffisants en développant des algorithmes qui permettent de résoudre ces problèmes. Nos résultats viennent alors compléter ceux initiés par Y. Métivier *et al.*.

Cette caractérisation peut être comparée à des travaux existants comme ceux de Yamashita et Kameda [YK99], Boldi *et al.* [BCG$^+$96] ou encore plus récemment les travaux de Cidon et Mokryn [CM98]. Toutefois, le modèle que nous proposons n'impose pas que les processus connaissent le voisinage qui les entoure. De plus, les algorithmes que nous proposons offrent une meilleure complexité en regard de ces travaux la réduisant ainsi d'exponentielle à polynomiale en la taille du réseau. Pour un modèle qui peut s'adapter naturellement aux réseaux ad hoc sans fil tels que les réseaux de capteurs dans lesquels les entités sont de très faible capacité, il est indispensable de réduire la complexité en temps, en mémoire, en nombre de messages et en taille de messages.

Dans cette contribution, nous avons aussi souligné l'importance de la connaissance initiale. Comme précisé précédemment, le modèle impose la non connaissance du degré par chaque sommet. Cette hypothèse, bien que contraignante, correspond d'autant plus à la réalité : la spontanéité d'un réseau de capteurs ne peut qu'affirmer ce fait. En ce sens, nous montrons que cette absence de connaissance du degré permet de résoudre le problème du nommage/énumération à partir du moment où chaque processus connaît la taille du graphe. Comme précisé dans l'introduction de ce manuscrit, il semble intuitif que dès lors que l'algorithme de nommage/énumération terminé, il devient trivial de résoudre le problème de l'élection. On montre alors qu'il n'est pas possible de résoudre l'élection avec comme seule connaissance la taille du réseau. Nous autorisons ainsi chaque processus à connaître une carte initiale du réseau sans pour autant connaître sa position dans

celui-ci. Cette combinaison de connaissance, c.-à-d., ne pas connaître le degré et connaître une carte du réseau reste justifiable encore une fois de par la spontanéité d'un réseau de capteurs. Pour l'administrateur de ce réseau, il devient alors plus aisé de fournir la même carte à chacun des sommets plutôt que de devoir déterminer le nombre de voisins de chacun d'entre eux. Nous remarquons cependant que la connaissance de la base minimale de ce réseau suffit, ce qui autorise plusieurs topologies de réseau sans devoir mettre à jour la connaissance des processus. De plus, nous démontrons que si le degré et la taille du graphe sont connus par les processus, alors la caractérisation est aussi satisfaite. Dans une dernière partie, nous étendons les résultats précédents à un autre modèle de communication. Nous considérons le modèle de diffusion synchrone dans lequel, à l'instar du modèle asynchrone, il n'existe pas nécessairement d'horloge globale. Les processus sont initialement passifs et sont activés selon l'environnement de telle façon que deux processus voisins ne peuvent pas émettre un message en même temps. En un pas de calcul, lorsqu'un processus émet un message celui-ci est instantanément entendu par tous les voisins. La caractérisation des graphes pour lesquels il existe une solution pour les problèmes de nommage/énumération est basée sur la notion de réécriture de graphes introduite dans le chapitre 1 et d'homomorphismes de graphes. Nous rappelons que ce modèle de calcul modifie les étiquettes d'un sous-graphe composé d'un sommet et de ses voisins, suivant des règles dépendantes de ce sous-graphe uniquement. L'exécution d'un algorithme distribué correspond donc à un ensemble de sous-graphes disjoints sur lesquels s'appliquent des règles prédéfinies. Globalement, un algorithme est considéré comme asynchrone dans le sens où chacune des règles s'applique indépendamment les unes des autres. Toutefois, à l'échelle d'un sous-graphe, une exécution est dite localement synchrone, c.-à-d., une règle ne s'applique que si le sommet et tous ses voisins ne participent pas déjà à l'application d'une autre règle d'un autre sous-graphe. Il est donc nécessaire d'assurer une synchronisation entre le sommet central et ses voisins. En ce sens, nous notons que deux sommets voisins ne peuvent pas être dans le même état à la fin d'une étape de calcul. Ainsi, nous montrons comment modifier les algorithmes présentés et utilisés par le modèle asynchrone afin de les adapter au modèle synchrone. Dans ce modèle, il est possible d'étudier la complexité en temps de l'algorithme et nous montrons que les algorithmes offrent une complexité en temps polynomiale.

Dans un tout autre cadre, nous avons également étudié la très vaste notion du calcul de l'état global, ou snapshot, d'un système. La littérature offre un large éventail de solutions pour des problèmes spécifiques et des modèles très différents [KS08]. Il est donc normal d'en déduire que c'est un problème actuel qui trouve ses origines par les travaux de Chandy et Lamport [CL85]. Toutefois, la contribution que nous présentons dans ce mémoire est différente de celles présentées jusqu'alors. Comme précisé au début de cette conclusion, nous considérons uniquement les réseaux anonymes, et, sous cette hypothèse, il n'existe aucune contribution équivalente. Nous considérons le modèle classique à base de passage de messages pour lequel nous apportons de nouvelles idées pour la résolution du problème du calcul de l'état global et de ses applications telles que le calcul de point reprise ou la détection de propriétés stables. Depuis Angluin [Ang80], il est connu que pour une configuration dite trop "symétrique", il est impossible pour un processus de connaître une carte du réseau composée des sommets et des arêtes étiquetées par les états des processus et des canaux de communication. Toutefois, bien que considérant les réseaux anonymes,

nous montrons tout d'abord que les algorithmes de Chandy-Lamport et Shi, Szymanski et Prywes peuvent être adaptés pour fournir une solution aux applications du calcul de points de reprise et de la détection de la terminaison d'un algorithme distribué. Dans un second temps, nous nous sommes interrogés sur l'utilisation des connaissances locales de chaque processus calculées par une exécution de l'algorithme de Chandy-Lamport. Nous introduisons alors la notion de *weak snapshot*. Un *weak snapshot* est un snapshot à un revêtement près d'un système distribué. Un revêtement est un homomorphisme de graphe localement bijectif initialement utilisé par les résultats d'impossibilités des problèmes du nommage/énumération et de l'élection [Ang80, God02, Cha06]. Nous montrons que la notion de *weak snapshot* conserve certaines propriétés du réseau et notamment la grande majorité des propriétés stables telles que la terminaison, la perte de jetons, les deadlocks ou encore le garbage collection. Pour l'application de la détection de propriétés stables, nous offrons de plus la possibilité d'avoir plusieurs processus initiateurs. Par conséquent, là où les travaux existants se restreignent à des réseaux où les processus sont identifiés de façon unique et à un unique initiateur, nous relaxons ces hypothèses et nous proposons une nouvelle contribution originale pour ce domaine.

Comme précisé précédemment, le domaine d'étude du problème du calcul de l'état global est vaste. Parmi les applications proposées par le chapitre 3, il y a le calcul de point de reprise (et la reprise lors d'un échec) et la détection de propriétés stables. Dans le chapitre 4, nous montrons un autre aspect de ce domaine en introduisant la notion de débogage distribué. Lors de la conception et de la mise en place d'applications distribuées, la notion de débogage est intéressante à plus d'un titre. Tout d'abord, elle permet, en amont, de faciliter la mise au point d'un algorithme en améliorant la compréhension d'un système distribué par l'utilisation de snapshot. En aval, elle permet ensuite de mettre en place une surveillance du système. La surveillance d'un système distribué intervient, entre autres, dans les réseaux de capteurs et permet la détection de problèmes particuliers ou d'évaluer certains paramètres du système évoluant au fur et à mesure de l'exécution. Dans ce dernier chapitre, nous présentons l'intégration d'un mécanisme de débogage distribué au sein d'un logiciel de simulation et de visualisation nommé *ViSiDiA*. La plateforme *ViSiDiA* offre une solution complète et intuitive pour la conception d'algorithmes distribués. Nous avons ainsi implémenté et ajouté en tant que fonctionnalité inhérente du logiciel, l'algorithme de snapshot en respectant les hypothèses initiales sur l'anonymat du réseau et sur la possibilité d'avoir plusieurs initiateurs. L'API du logiciel a été étoffée afin de permettre le débogage et la surveillance d'un système distribué. Ainsi, nous confirmons de façon pratique les résultats du chapitre 3.

Dans le cadre présenté dans le chapitre 2, les perspectives de recherche restent encore nombreuses. La réduction de la connaissance initiale est une première étape pour la conception d'algorithmes fiables. Lorsque l'on considère le problème de l'élection, nous avons mis en avant que seule la connaissance de la taille combinée à l'absence de connaissance du degré ne suffisait pas. Nous avons donc supposé que chaque processus du réseau connaît une carte du réseau ou, plus généralement, une carte de sa base minimale. Nous étudions actuellement la possibilité de limiter la connaissance de chaque processus à la taille du graphe et au nombre de liens de communication. L'idée consiste à mettre en relation l'étiquetage final produit par une exécution de l'algorithme avec ces hypothèses afin de permettre à tous les processus de détecter la terminaison de l'algorithme.

Dans un second temps, nous nous sommes intéressés aux graphes aléatoires géométriques qui traduisent au mieux le modèle de diffusion. Les processus sont placés aléatoirement sur un plan et deux sommets communiquent entre eux dès lors qu'ils se trouvent tout deux dans un rayon déterminé r. Nous essayons alors de répondre à la question de savoir avec quelle probabilité un graphe aléatoire géométrique respecte les conditions nécessaires données par la caractérisation, c.-à-d., est minimal pour les fibrations. Nous utilisons pour cela la notion de coloration semi-régulière utilisée par Chalopin et Paulusma [CP11] permettant d'exprimer les fibrations en terme de colorations de graphes. Pour un graphe étiqueté aléatoire $\mathbf{G}$, si le nombre de couleurs nécessaires à la réalisation d'une coloration semi-régulière est inférieur à $|V(G)|$ alors le graphe n'est pas minimal. Ainsi, une première étape consiste à évaluer la probabilité qu'un graphe aléatoire possède au moins deux sommets *jumeaux*. Deux sommets u, v sont *jumeaux* si $N(u) \cup \{u, v\} = N(v) \cup \{u, v\}$. Nous concluons, expérimentalement, qu'il y a environ une chance sur deux d'avoir des sommets jumeaux dans un graphe aléatoire géométrique. Toutefois, l'existence de sommets jumeaux ne permet que d'obtenir une borne supérieure sur la probabilité. Il est donc nécessaire d'affiner les résultats en utilisant, par exemple, l'algorithme polynomial proposé par [Lei82].

Concernant le domaine du calcul de l'état global, une première direction possible est l'étude de la complexité des algorithmes proposés. Il semble intéressant d'étudier les complexités en temps des différentes compositions d'algorithmes. La composition d'algorithmes séquentiels est un aspect largement éprouvé mais il n'en est pas de même pour le domaine de l'algorithmique distribuée. Il peut être judicieux de s'inspirer des résultats présentées dans [CF11] afin de les adapter à notre modèle et d'utiliser des outils d'aide à la preuve comme *Coq* afin de prouver les algorithmes composés du chapitre 3.

Une autre piste intéressante consiste à étendre les solutions du chapitre 3 dans le modèle de diffusion asynchrone du chapitre 2. Une fois de plus, la connaissance initiale est problématique : il est contradictoire de proposer une carte du réseau comme connaissance dans le cadre d'un algorithme de calcul d'état global. Dans ce cadre, de par l'asynchronisme du modèle et l'absence de connaissance sur le voisinage, il est alors difficile pour un processus de détecter la terminaison d'un algorithme et de déterminer le contenu d'un canal de communication. D'une façon plus générale, il pourrait être envisagé d'étudier les conditions nécessaires et suffisantes de l'algorithme de détection de la terminaison *SSP* afin de connaitre les limites exactes de cet algorithme.

Dans le contexte expérimental proposé par le dernier chapitre de ce mémoire, nous intégrons un mécanisme de *rollback* au sein de *ViSiDiA*. Par l'intermédiaire de ce mécanisme, les développeurs peuvent interagir avec le déroulement d'un algorithme distribué et navigant à travers les différents états du système en cours d'exécution. La grande difficulté d'intégration de ce système réside dans la complexité à placer les processus et les liens de communication dans des états antérieurs (p. ex., inconsistance de certaines variables, de certains messages). De façon analogue, l'utilisation de la notion de *weak snapshot* est en cours de développement afin de permettre une meilleure visualisation et interaction dans la conception, le test et la supervision d'applications distribuées. Enfin, il existe un projet commun entre l'outil d'aide à la preuve *B* et *ViSiDiA* nommé *B2ViSiDiA*. Il paraît aussi judicieux d'intégrer le débogueur au sein de ce projet afin de faciliter le débogage d'algorithmes provenant de l'outil *B*.

Bibliographie

[AAER07] D. Angluin, J. Aspnes, D. Eisenstat, and E. Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4) :279–304, November 2007.

[Ahu92] M. Ahuja. Global snapshots for asynchronous distributed systems with non-fifo channels. Technical Report CS92-268, University of California, 1992.

[AMM12a] C. Aguerre, T. Morsellino, and M. Mosbah. Debugging the Execution of Distributed Algorithms over Anonymous Networks. In *Proceedings of the 16th International Conference on Information Visualisation (IV)*. IEEE Computer Society, 2012.

[AMM12b] C. Aguerre, T. Morsellino, and M. Mosbah. Fully-Distributed Debugging and Visualization of Distributed Systems in Anonymous Networks. In *Proceedings of the International Conference on Information Visualization Theory and Applications*, pages 764–767. INSTICC, 2012.

[Ang80] D. Angluin. Local and global properties in networks of processors. In *Proc. of the 12th Symposium on Theory of Computing (STOC 1980)*, pages 82–93, 1980.

[AW04] H. Attiya and J. Welch. *Distributed computing : fundamentals, simulations, and advanced topics*. John Wiley and Sons, 2004.

[AWW05] I. F. Akyildiz, X. Wang, and W. Wang. Wireless mesh networks : a survey. *Computer Networks*, 47(4) :445 – 487, 2005.

[BA01] M. Ben-Ari. Interactive execution of distributed algorithms. *J. Educ. Resour. Comput.*, 1, August 2001.

[BCG+96] P. Boldi, B. Codenotti, P. Gemmell, S. Shammah, J. Simon, and S. Vigna. Symmetry breaking in anonymous networks : characterizations. In *Proc. of the 4th Israeli Symposium on Theory of Computing and Systems (ISTCS 1996)*, pages 16–26. IEEE Press, 1996.

[BGM+01] M. Bauderon, S. Gruner, Y. Métivier, M. Mosbah, and A. Sellami. visualization of distributed algorithms based on labeled rewriting systems. In *Proc. of the 2nd International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2001)*, volume 50 of *ENTCS*, pages 229–239, 2001.

[BM03] M. Bauderon and M. Mosbah. A unified framework for designing, implementing and visualizing distributed algorithms. *ENTCS*, 72(3) :13 – 24, 2003.

[Bod89] H.L. Bodlaender. The classification of coverings of processor networks. *Journal of parallel and distributed computing*, 6(1) :166–182, 1989.

[BPF+04] C. Buschmann, D. Pfisterer, S. Fischer, S. P. Fekete, and A. Kröller. Spyglass : taking a closer look at sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, pages 301–302, New York, NY, USA, 2004. ACM.

[BT87] G. Bracha and S. Toueg. Distributed deadlock detection. *Distributed Computing*, 2(3) :127–138, 1987.

[BV99] P. Boldi and S. Vigna. Computing anonymously with arbitrary knowledge. In *Proc. of the 18th ACM Symposium on principles of distributed computing (PODC 1999)*, pages 181–188. ACM Press, 1999.

[BV01] P. Boldi and S. Vigna. An effective characterization of computability in anonymous networks. In *Proc. of Distributed Computing, 15th International Conference (DISC 2001)*, volume 2180 of *Lecture Notes in Computer Science*, pages 33–47. Springer-Verlag, 2001.

[BV02a] P. Boldi and S. Vigna. Fibrations of graphs. *Discrete Mathematics*, 243(1-3) :21–66, 2002.

[BV02b] P. Boldi and S. Vigna. Universal dynamic synchronous self-stabilization. *Distributed Computing*, 15(3) :137–153, 2002.

[BYGI91] R. Bar-Yehuda, O. Goldreich, and A. Itai. Efficient emulation of single-hop radio network with collision detection on multi-hop radio network with no collision detection. *Distributed Computing*, 5 :67–71, 1991.

[BYGI92] R. Bar-Yehuda, O. Goldreich, and A. Itai. On the time-complexity of broadcast in multi-hop radio networks : An exponential gap between determinism and randomization. *Journal of Computer and System Sciences*, 45(1) :104 – 126, 1992.

[CF11] P. Castéran and V. Filou. Tasks, types and tactics for local computation systems. *Studia Informatica Universalis*, 9(1) :39–86, 2011.

[Cha99] X. Chang. *Network simulations with OPNET*, pages 307–314. ACM, 1999.

[Cha06] J. Chalopin. *Algorithmique distribuée, calculs locaux et homorphismes de graphes*. PhD thesis, université Bordeaux 1, 2006.

[Chl01] B.S. Chlebus. Randomized communication in radio networks. In P.M. Pardalos, S. Rajasekaran, J.H. Reif, and J.D.P. Rolim, editors, *Handbook of Randomized Computing*, volume I, pages 401–456,. Kluwer Academic Publishers, 2001.

[CL85] K. M. Chandy and L. Lamport. Distributed snapshots : Determining global states of distributed systems. *ACM Trans. Comput. Syst.*, 3(1) :63–75, 1985.

[CM94] B. Courcelle and Y. Métivier. Coverings and minors : applications to local computations in graphs. *Europ. Journal of Combinatorics*, 15(2) :127–138, 1994.

[CM98] I. Cidon and O. Mokryn. Propagation and leader election in a multihop broadcast environment. In *DISC*, pages 104–118, 1998.

[CM07a] J. Chalopin and Y. Métivier. An efficient message passing election algorithm based on mazurkiewicz's algorithm. *Fundamenta Informaticae*, 80(1-3) :221–246, 2007.

[CM07b] J. Chalopin and Y. Métivier. An efficient message passing election algorithm based on mazurkiewicz's algorithm. *Fundam. Inform.*, 80(1-3) :221–246, 2007.

[CM10a] J. Chalopin and Y. Métivier. On the power of synchronization between two adjacent processes. *Distributed Computing*, 23(3) :177–196, 2010.

[CM10b] J. Chalopin and Y. Métivier. On the power of synchronization between two adjacent processes. *Distributed Computing*, 23 :177–196, 2010.

[CMM12a] J. Chalopin, Y. Métivier, and T. Morsellino. Enumeration and leader election in partially anonymous and multi-hop broadcast networks. *Fundamenta Informaticae*, 119(1) :1–27, 2012.

[CMM12b] J. Chalopin, Y. Métivier, and T. Morsellino. On Snapshots and Stable Properties Detection in Anonymous Fully Distributed Systems (Extended Abstract). In *Proceedings of the 19th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, volume 7355, pages 207–218. LNCS, 2012.

[CMM12c] J. Chalopin, Y. Métivier, and T. Morsellino. Snapshots et Détection de Propriétés Stables dans les Systèmes Distribués Anonymes. In *14èmes Rencontres Francophones sur les Aspects Algorithmiques de Télécommunications (AlgoTel)*, 2012.

[CMZ06] J. Chalopin, Y. Métivier, and W. Zielonka. Local computations in graphs : the case of cellular edge local computations. *Fundamenta Informaticae*, 74(1) :85–114, 2006.

[CP11] J. Chalopin and D. Paulusma. Graph labelings derived from models in distributed computing : A complete complexity classification. *Networks*, 58(3) :207–231, 2011.

[Der06] B. Derbel. *Local aspects in distributed algorithms.* PhD thesis, Université Bordeaux 1, 2006.

[DM03] B. Derbel and M. Mosbah. Distributing the execution of a distributed algorithm over a network. In *INFOVIS'03*, pages 485 – 490, july 2003.

[Gal85] R. Gallager. A perspective on multiaccess channels. *Information Theory, IEEE Transactions on*, 31(2) :124 – 142, mar 1985.

[GM02] E. Godard and Y. Métivier. A characterization of families of graphs in which election is possible (*ext. abstract*). In *Proc. of Foundations of Software Science and Computation Structures, 5th International Conference (FOSSACS 2002)*, volume 2303 of *Lecture Notes in Computer Science*, pages 159–172. Springer-Verlag, 2002.

[GM03] E. Godard and Y. Métivier. Deducible and equivalent structural knowledges in distributed algorithms. *Theory of Computing Systems*, 36(2) :631–654, 2003.

[GMM04] E. Godard, Y. Métivier, and A. Muscholl. Characterization of classes of graphs recognizable by local computations. *Theory of Computing Systems*, 37(2) :249–293, 2004.

[God02] E. Godard. *Réécritures de Graphes et Algorithmique Distribuée*. PhD thesis, Université Bordeaux 1, 2002.

[GR05] R. Guerraoui and E. Ruppert. What can be implemented anonymously ? In *DISC*, pages 244–259, 2005.

[GT87] J. L. Gross and T. W. Tucker. *Topological graph theory*. Wiley Interscience, 1987.

[Hel89] J-M. Helary. Observing global states of asynchronous distributed applications. In J-C. Bermond and M. Raynal, editors, *Distributed Algorithms*, volume 392, pages 124–135. 1989.

[HR88] J.-M. Helary and M. Raynal. Assigning distinct identities to sites on an anonymous distributed system. In *Distributed Computing Systems in the 1990s, 1988. Proceedings., Workshop on the Future Trends of*, pages 82 –86, sep 1988.

[JS85] R.E. Johnson and F.B. Schneider. Symmetry and similarities in distributed systems. In *Proc. of the 4th Annual ACM Symposium on Principles of Distributed Computing (PODC 1985)*, pages 13–22, 1985.

[KPT03] B. Koldehofe, M. Papatriantafilou, and P. Tsigas. Integrating a simulation-visualisation environment in a basic distributed systems course : a case study using lydian. In *ITiCSE'03*, pages 35–39. ACM, 2003.

[KRS95] A. D. Kshemkalyani, M. Raynal, and M. Singhal. An introduction to snapshot algorithms in distributed computing. *Distributed Systems Engineering*, 2(4) :224–233, 1995.

[KS08] A. D. Kshemkalyani and M. Singhal. *Distributed computing*. Cambridge, 2008.

[Ksh10] A. D. Kshemkalyani. Fast and message-efficient global snapshot algorithms for large-scale distributed systems. *IEEE Trans. Parallel Distrib. Syst.*, 21(9) :1281–1289, 2010.

[KSW+08] A. Köpke, M. Swigulski, K. Wessel, D. Willkomm, P. T. Klein Haneveld, T. E. V. Parker, O. W. Visser, Hermann S. Lichte, and Stefan Valentin. Simulating wireless and mobile networks in omnet++ the mixim vision. In *SimuTools*, page 71, 2008.

[KW07] A. D. Kshemkalyani and B. Wu. Detecting arbitrary stable properties using efficient snapshots. *IEEE Trans. Software Eng.*, 33(5) :330–346, 2007.

[Lam78] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7) :558–565, 1978.

[LBM09] C. Lipski, K. Berger, and M. Magnor. visage - A visualization and debugging framework for distributed system applications. In Vaclav Skala, editor, *Proc. WSCG 2009*, volume 2009, pages 1–7, Plzen, Czech Republic, February 2009. UNION Agency – Science Press.

[Lei82] F.T. Leighton. Finite common coverings of graphs. *J. Combin. Theory, Ser. B*, 33(3) :231–238, 1982.

[LeL77] G. LeLann. Distributed systems, towards a formal approach. In *Information processing 1977*, pages 155–160. North-Holland, 1977.

[LY87] T.H. Lai and T.H. Yang. On distributed snapshots. *Inf. Process. Lett.*, 25(3) :153–158, 1987.

[Lyn96] N. Lynch. *Distributed algorithms*. Morgan Kaufman, 1996.

[Mas91] W.S. Massey. *A basic course in algebraic topology*. Springer-Verlag, 1991. Graduate texts in mathematics.

[Mat87] F. Mattern. Algorithms for distributed termination detection. *Distributed computing*, 2(3) :161–175, 1987.

[Mat89] F. Mattern. Virtual time and global states of distributed systems. In *Parallel and Distributed Algorithms*, pages 215–226. North-Holland, 1989.

[Maz87] A. Mazurkiewicz. Trace theory. In *Advances in Petri Nets*, volume 255 of *Lecture notes in computer science*, pages 279–324. Spinger, 1987.

[Maz97a] A. Mazurkiewicz. Distributed enumeration. *Information Processing Letters*, 61(5) :233–239, 1997.

[Maz97b] A. Mazurkiewicz. Distributed enumeration. *Inf. Processing Letters*, 61 :233–239, 1997.

[MC98] J. Matocha and T. Camp. A taxonomy of distributed termination detection algorithms. *Journal of Systems and Software*, 43(3) :207–221, 1998.

[MMW97] Y. Métivier, A. Muscholl, and P.-A. Wacrenier. About the local detection of termination of local computations in graphs. In *Proc. of the 4th International Colloquium on Structural Information and Communication Complexity (SIROCCO 1997)*, Proceedings in Informatics, pages 188–200. Carleton Scientific, 1997.

[MPTU98] Y. Moses, Z. Polunsky, A. Tal, and L. Ulitsky. Algorithm visualization for distributed environments. In *INFOVIS'98*, pages 71–78, 1998.

[MR83] M.A. Marsan and D. Roffinella. Multichannel local area network protocols. *Selected Areas in Communications, IEEE Journal on*, 1(5) :885 – 897, nov 1983.

[MS94] K. Marzullo and L. S. Sabel. Efficient detection of a class of stable properties. *Distributed Computing*, 8(2) :81–91, 1994.

[MSZ02] Y. Métivier, N. Saheb, and A. Zemmari. Randomized local elections. *Information Processing Letters*, 82(6) :313–320, 2002.

[MSZ03] Y. Métivier, N. Saheb, and A. Zemmari. Analysis of a randomized rendezvous algorithm. *Inf. Comput.*, 184(1) :109–128, 2003.

[MT00] Y. Métivier and G. Tel. Termination detection and universal graph reconstruction. In *Proc. of 7th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2000)*, Proceedings in Informatics, pages 237–251. Carleton Scientific, 2000.

[Nor95] N. Norris. Universal covers of graphs : isomorphism to depth $n-1$ implies isomorphism to all depths. *Discrete Applied Math.*, 56(1) :61–74, 1995.

[Pon93] G. Pongor. Omnet : Objective modular network testbed. In *MASCOTS '93*, pages 323–326, 1993.

[Ray88] M. Raynal. *Networks and distributed computation.* MIT Press, 1988.

[RCK+05] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin. Sympathy for the sensor network debugger. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, SenSys '05, pages 255–267, New York, NY, USA, 2005. ACM.

[RKE05] N. Ramanathan, E. Kohler, and D. Estrin. Towards a debugging system for sensor networks. *Int. J. Netw. Manag.*, 15(4) :223–234, July 2005.

[Sak99] N. Sakamoto. Comparison of initial conditions for distributed algorithms on anonymous networks. In *Proc. of the 18th Annual ACM Symposium on Principles of Distributed Computing (PODC 1999)*, pages 173–179. ACM Press, 1999.

[San06] N. Santoro. *Design and Analysis of Distributed Algorithms (Wiley Series on Parallel and Distributed Computing).* Wiley-Interscience, 2006.

[Sel04] A. Sellami. *Des calculs locaux aux algorithmes distribués.* PhD thesis, Université Bordeaux 1, 2004.

[SK93] J. T. Stasko and E. Kraemer. A methodology for building application-specific visualizations of parallel programs. *J. Parallel Distrib. Comput.*, 18 :258–264, June 1993.

[SS94] A. Schiper and A. Sandoz. Strong stable properties in distributed systems. *Distributed Computing*, 8(2) :93–103, 1994.

[SSP85a] B. Szymanski, Y. Shy, and N. Prywes. Synchronized distributed termination. *IEEE Transactions on software engineering*, 11(10) :1136–1140, 1985.

[SSP85b] B. Szymanski, Y. Shy, and N. Prywes. Terminating iterative solutions of simultaneous equations in distributed message passing systems. In *Proc. of the 4th Annual ACM Symposium on Principles of Distributed Computing (PODC 1985)*, pages 287–292. ACM Press, 1985.

[Sto05] I. Stojmenovic. *Handbook of Sensor Networks : Algorithms and Architectures.* Wiley, 2005.

[Tan02] A. Tanenbaum. *Computer Networks.* Prentice Hall Professional Technical Reference, 2002.

[Tel00] G. Tel. *Introduction to distributed algorithms.* Cambridge University Press, 2000.

[TvS02] A. Tanenbaum and M. van Steen. *Distributed Systems - Principles and Paradigms.* Prentice Hall, 2002.

[WTT+06] K. Whitehouse, G. Tolle, J. Taneja, C. Sharp, S. Kim, J. Jeong, J. Hui, P. Dutta, and D. Culler. Marionette : using rpc for interactive development

and debugging of wireless embedded networks. In *Proceedings of the Fifth International Conference on Information Processing in Sensor Networks*, IPSN '06, 2006.

[WWG06] Ye Wen, Rich Wolski, and Selim Gurun. S2db : a novel simulation-based debugger for sensor network applications. In *Proceedings of the 6th ACM & IEEE International conference on Embedded software*, EMSOFT '06, pages 102–111, New York, NY, USA, 2006. ACM.

[YK96a] M. Yamashita and T. Kameda. Computing functions on asynchronous anonymous networks. *Math. Systems Theory*, 29(4) :331–356, 1996.

[YK96b] M. Yamashita and T. Kameda. Computing on anonymous networks : Part i - characterizing the solvable cases. *IEEE Transactions on parallel and distributed systems*, 7(1) :69–89, 1996.

[YK96c] M. Yamashita and T. Kameda. Computing on anonymous networks : Part ii - decision and membership problems. *IEEE Transactions on parallel and distributed systems*, 7(1) :90–96, 1996.

[YK98] M. Yamashita and T. Kameda. Erratum to computing functions on anonymous asynchronous networks. *Theory of Computing Systems*, 31(1) :109, 1998.

[YK99] M. Yamashita and T. Kameda. Leader election problem on networks in which processor identity numbers are not distinct. *IEEE Transactions on parallel and distributed systems*, 10(9) :878–887, 1999.

[YSSW07] J. Yang, M. L. Soffa, L. Selavo, and K. Whitehouse. Clairvoyant : a comprehensive source-level debugger for wireless sensor networks. In *Proceedings of the 5th international conference on Embedded networked sensor systems*, SenSys '07, pages 189–203, New York, NY, USA, 2007. ACM.