# Cohérence de calculs répartis face aux défaillances, à l'anonymat et au facteur d'échelle

François Bonnet

N° d'ordre : 3968 ANNÉE 2010

**THÈSE / UNIVERSITÉ DE RENNES 1**
*sous le sceau de l'Université Européenne de Bretagne*

pour le grade de

**DOCTEUR DE L'UNIVERSITÉ DE RENNES 1**

*Mention : Informatique*

**Ecole doctorale Matisse**

présentée par


# François Bonnet

préparée à l'unité de recherche ASAP - IRISA
Institut de Recherche en Informatique et Systèmes aléatoires
IFSIC


---

# Cohérence de calculs répartis face aux défaillances, à l'anonymat et au facteur d'échelle

**Thèse soutenue à Rennes le 22 juillet 2010**

devant le jury composé de :

**Claude Jard**
Prof. Ens Cachan, Antenne de Bretagne / président

**Olivier Beaumont**
DR INRIA-Bordeaux / rapporteur

**Pierre Sens**
Prof. Paris 6 / rapporteur

**Rachid Guerraoui**
Prof. EPFL / examinateur

**Anne-Marie Kermarrec**
DR INRIA-Rennes / examinateur

**Michel Raynal**
Prof. Rennes 1 / directeur de thèse

# Table des matières

# Chapitre 1

# Introduction

Aujourd'hui les systèmes répartis, ou systèmes distribués si l'on s'autorise un anglicisme habituel, existent sous de multiples formes. En premier lieu, tout le monde pense évidemment, et à juste titre, à Internet ; ce gigantesque réseau reliant plusieurs centaines de millions d'utilisateurs. Internet constitue actuellement le plus gros réseau d'ordinateurs au monde, et un tel réseau implique nécessairement une forte collaboration entre toutes les entités participantes pour fonctionner correctement. Son fonctionnement est basé sur un modèle hiérarchique reposant sur un squelette de routeurs interconnectés par des liaisons à très haut débit. De manière invisible pour l'utilisateur final, de nombreux algorithmes sont utilisés pour garantir la bonne marche du réseau, et ce à différents niveaux ; des algorithmes de résolutions de noms Internet (Domain Name System en anglais) permettant d'associer un nom de domaine à une adresse Internet, des protocoles de communications garantissant la non-corruption des paquets transitant dans le réseau, des algorithmes de routage de paquets permettant d'acheminer les données à la bonne destination, ...

Mais Internet n'est pas le seul réseau au monde, il existe d'autres systèmes répartis qui nécessitent eux aussi une forte collaboration entre les entités participantes. Nous considérons l'exemple des réseaux téléphoniques cellulaires (téléphones portables). Ici aussi nous retrouvons de multiples algorithmes répartis. En premier lieu, au niveau d'une antenne relai : à tout instant, plusieurs téléphones mobiles sont connectés à une même antenne. Cela nécessite un protocole d'accord entre les téléphones et l'antenne sur la fréquences de communication que chaque téléphone doit utiliser afin d'éviter les pertubations avec les autres mobiles. En second lieu, au niveau supérieur, un algorithme doit pouvoir gérer la mobilité des téléphones et s'assurer du transfert de service d'une antenne à une autre. En effet chaque antenne ne dispose que d'une portée d'émission limitée (moins d'un kilomètre) et ne couvre par conséquent qu'une petite zone géographique. Or un téléphone mobile doit, de par sa définition, être capable de fonctionner lorsque l'utilisateur se déplace, d'où la nécessité d'algorithmes gérant le changement d'antenne. Enfin, à un dernier niveau, tout réseau cellulaire doit permettre les communications vers d'autes réseaux téléphoniques (filaires ou cellulaires) ; comme pour Internet cela nécessite des algorithmes de routage des communications.

Afin de compléter cette rapide présentation de systèmes répartis, nous citons un dernier exemple plus récent ; les réseaux de capteurs. Un capteur désigne une petite machine (quelques

centimètres) de faible puissance, munie d'une batterie, communiquant de manière sans fil (radio ou wifi) et disposant d'une sonde. Un réseau de capteurs correspond à une multitude de ces capteurs dispersés sur une zone géographique où aucune infrastructure n'est disponible (pas de prise de courant, pas de cables réseaux,...). Les applications les plus fréquemment évoquées sont la surveillance/détection des feux de forêts (capteurs de températures), l'étude d'animaux sauvages (capteurs fixés sur les animaux), la surveillance d'un champ de bataille (capteurs de détections de mouvements). Ces entités créent un réseau ad-hoc et doivent coopérer pour effectuer la tâche qui leur est confiée. Au niveau algorithmique, il faut réussir à concevoir des algorithmes permettant de faire remonter les informations de manière efficace, tout en limitant au maximum la consommation énergétique.

Dans les paragraphes précédents, nous avons brièvement présenté trois exemples de systèmes répartis. Les trois cas étudiés entrent dans le cadre des systèmes répartis car ils mettent tous en œuvre plusieurs entités indépendantes ayant besoin de communiquer et coopérer. Néanmoins nous pouvons constater que ces trois systèmes sont très différents sur plusieurs critères :

- Les entités étudiées. La puissance d'un ordinateur connecté à Internet, celle d'un téléphone mobile, ou celle d'un capteur, sont difficilement comparables. Le nombre de participants varie lui aussi de plusieurs ordres de grandeur d'un système à un autre.
- Le système lui-même. Dans le cas d'Internet, les communications peuvent être considérées comme fiables, car les messages transitent par des cables. Dans les deux autres exemples les communications sont sans-fil et sont donc soumises à d'éventuelles pertubations.
- Les impondérables. Un monde parfait n'existant pas, des pannes peuvent apparaitre dans les systèmes. Celles-ci varient aussi suivant le système concerné ; d'une panne franche d'un routeur pour le réseau Internet ou d'une panne de batterie sur un téléphone ou un capteur, de nombreuses pannes peuvent se produire. Chaque système doit être capable de les gérer au mieux, impliquant des mécanismes différents pour chacun d'entre eux.

Les trois critères précédents montrent que bien que regroupés sous une même dénomination de "systèmes répartis", les trois systèmes présentés sont en réalité très différents. La suite de cette introduction présente une caractérisation des systèmes répartis suivant une décomposition en adversaires.

## 1.1 Décomposition en adversaires

Dans le domaine qui nous concerne, l'étude théorique des systèmes répartis, les axes de recherche peuvent être résumés d'une manière simple. Étant donné un problème que l'on souhaite résoudre, on ajoute un certain nombre d'adversaires au système (ou au chercheur, suivant le point de vue) qui complexifient la recherche de la solution. Ces adversaires choisis, le but est de prouver l'existence ou l'absence de solution au problème considéré. Ensuite dans le cas où le problème est solvable sous des contraintes données, il s'agit de trouver les meilleures solutions possibles ; algorithmes utilisant le minimum de messages, algorithmes

offrant des performances optimales,... Dans le cas contraire, lorsque le problème est prouvé insolvable, il devient intéressant de trouver quel ajout (minimal) au système permet de lever cette absence de solution.

Dans cette vision basée sur les adversaires, il est possible de distinguer trois types principaux d'adversaires. Tout d'abord les adversaires directement liés aux caractéristiques intrinsèques des entités participantes ; cet ensemble inclut par exemple le nombre d'entités, ou la puissance de calcul de chacune d'entre elle. Ensuite il est possible de séparer les adversaires liés au système étudié ; cet ensemble inclut en particulier les primitives de communications accessibles aux entités du système. Enfin le dernier grand ensemble d'adversaires correpond aux contraintes non contrôlables du système ; la présence de pannes étant l'exemple le plus significatif.

Dans la suite, nous présentons plusieurs adversaires pouvant apparaître dans des systèmes répartis. À chaque fois, nous montrons en quoi ces adversaires soulèvent une difficulté supplémentaire dans la conception d'algorithmes répartis.

Dans ce travail de thèse, nous avons étudié plusieurs systèmes dans lesquels une partie de ces adversaires intervient. Il est souvent intéressant de se concentrer sur un adversaire en particulier ; ainsi certains travaux considèrent des systèmes où le simple fait d'ajouter une contrainte supplémentaire (l'anonymat) impose de repenser entièrement les solutions aux problèmes abordés.

## 1.2 Adversaires liés au type d'entités étudiées

Tout travail de recherche sur les systèmes répartis commence par une description précise du modèle étudié. La première étape correspond à la description des entités concernées. En effet, contrairement à un système simple composé d'un seul processeur, la présence de plusieurs entités rend la conception d'algorithmes plus ardue. Dans cette section nous décrivons les adversaires liés aux entités du système. Remarquons toutefois qu'une partie des adversaires suivants n'est pas limitée aux seuls systèmes répartis ; en effet certains peuvent aussi se retrouver dans la conception d'algorithmes non-répartis.

### 1.2.1 Faiblesse des entités

La faiblesse des entités, par opposition à la puissance de celles-ci, rassemble les faiblesses intrinsèques des participants du système. D'une part, cela concerne la potentielle faible puissance de calcul ; un capteur par exemple ne dispose ni d'un processeur aussi puissant qu'un ordinateur de bureau, ni d'une quantité de mémoire (vive ou de stockage) aussi importante. D'autre part, cela concerne l'éventuelle faiblesse d'énergie ; comme signalé précédemment, certaines entités fonctionnent sur batterie, sans apport extérieur d'énergie. Ces deux contraintes de faiblesse constituent un premier adversaire dont il faut tenir compte lorsque des algorithmes sont conçus pour de telles entités.

Nous devons prendre en considération cet adversaire afin de garantir le fonctionnement global du système. Ainsi il est parfois crucial de dégrader l'efficacité d'un système pour pro-

longer l'autonomie, ou bien d'utiliser des algorithmes moins précis pour abréger les temps de calcul. Il peut aussi être nécessaire de distribuer le calcul ou la tâche à accomplir sur plusieurs entités afin d'économiser certains participants clés du système (les capteurs centraux, par exemple dans un réseau de capteurs).

### 1.2.2 L'anonymat des entités

Dans un système traditionnel (réparti ou non), chaque entité possède une identité propre. Sur Internet, chaque machine accessible est ainsi désignée par une adresse IP unique, alors qu'à tout téléphone portable (toute carte SIM pour être précis) est associé un unique numéro de téléphone. Cependant il existe des systèmes répartis anonymes dans lesquels les participants sont indiscernables les uns des autres. L'anonymat constitue un second adversaire directement lié à la nature des entités. En effet il est plus compliqué de concevoir des algorithmes dans un système anonyme. Dans un tel système, il est par exemple impossible, a priori, de comptabiliser le nombre de participants puisque ceux-ci sont parfaitement identiques.

### 1.2.3 Le nombre d'entités

L'exemple précédent du comptage du nombre de participants d'un système, permet d'introduire un nouvel adversaire ; le nombre d'entités, ou plus précisément le passage à l'échelle du nombre d'entités ("scalability problem" en anglais). Autant il est relativement simple de réfléchir à des algorithmes pour de petits systèmes répartis, de l'ordre de quelques dizaines de participants ; autant il est difficile de concevoir des algorithmes efficaces pour des milliers/millions d'entités. Cependant il est nécessaire de concevoir de tels algorithmes ; Internet étant un cas concret d'un gigantesque réseau. Le passage à l'échelle est un point crucial dans la conception de systèmes répartis, car rien ne garantit qu'un algorithme efficace pour 10 entités reste correct et efficace dans un système avec 1 000 000 participants[1].

### 1.2.4 La dynamique des entités

Quel que soit le nombre de participants, rares sont les systèmes répartis qui restent statiques, c'est-à-dire sans variation du nombre ou de l'identité des participants. Nous pouvons distinguer deux types de dynamisme, mais d'un point de vue algorithmique, ils seront souvent traités de la même manière, c'est pourquoi nous les regroupons sous le même adversaire. D'une part, le "churn"[2] désigne le taux de connexions et de déconnexions des entités d'un système. Ce terme est fréquemment utilisé dans les réseaux pair-à-pair pour décrire

[1] La même problématique apparaît lorsque l'on s'intéresse aux hommes ; il est relativement facile de se mettre d'accord dans un groupe de 3/4 personnes, mais il est quasiment impossible d'arriver à un consensus dans un groupe de 20/30 personnes. Pour cela la société a inventé des "algorithmes" sociaux, plus couramment appelés votes ou élections.

[2] Nous avons choisi de conserver ce terme anglais dans cette thèse en français car, à notre connaissancec, il n'existe pas d'équivalent français simple.

l'évolution de la population ; on parlera de churn élévé si les participants se renouvellent rapidement, au contraire on parlera de churn faible si globablement le réseau reste statique. D'autre part, la mobilité active désigne la mobilité physique que peut avoir une entité ; un robot peut choisir par exemple de se déplacer pour accomplir la tâche qui lui est demandée. Cependant un tel mouvement peut le mettre hors de portée d'un émetteur, ou d'un autre robot ; d'une certaine manière, comme pour le churn, il apparaît alors comme déconnecté du système. Le churn et la mobilité active correspondent tous les deux à une arrivée ou un départ volontaire du système de la part de l'entité. Dans les paragraphes suivants, nous aborderons d'autres types de déconnexion, non-volontaire cette fois, que sont la panne franche et la mobilité passive.

## 1.3 Adversaires inhérents au système

Jusqu'à présent, nous avons décrit les adversaires relatifs aux entités présentes dans un système réparti. Ces adversaires ont montré la grande diversité d'entités auxquelles il est possible d'être confronté. Néanmoins décrire les participants ne suffit pas à caractériser un système réparti. Pour que l'on puisse réellement parler de système réparti, il est certes nécessaire d'avoir plusieurs entités, mais il faut aussi qu'elles collaborent entre elles, et donc qu'elles communiquent. La notion d'adversaire est peut être ici plus difficile à comprendre, cependant comme nous allons le voir par la suite, le type de communication accessible peut soulever des problèmes. Cette section est décomposée en deux parties ; dans un premier temps elle présente trois modes de communication pouvant être disponibles puis trois caractéristiques pouvant modifier les communications.

### 1.3.1 Modes de communication

Nous présentons ici les trois principaux modes de communication qui existent dans les systèmes répartis.

**Communication par diffusion** La communication par diffusion ("broadcast" en anglais) signifie qu'un message envoyé par un participant est reçu par tous les participants du système. Cela correspond à la méthode utilisée actuellement pour difuser la télévision hertzienne, ou la radio ; il y a un émetteur qui émet, une seule fois, à destination de tous les récepteurs. Dans le cadre des systèmes répartis, chaque entité joue à la fois le rôle d'émetteur et de récepteur et chacune peut envoyer et recevoir des messages. Le problème de ce mode de communication est l'impossibilité d'échanges privés entre deux participants ; lorsqu'un message est émis, tout le monde peut écouter et donc recevoir ce message. C'est pour cela que nous considérons la communication par diffusion comme un adversaire ; en effet, dans certains cas, cela implique d'ajouter des mécanismes cryptographiques au système pour permettre des communications sécurisées.

La communication par diffusion existe bien evidemment dans les systèmes sans-fil, mais on peut aussi les retrouver dans les systèmes répartis filaires : Le Protocole Internet (IP)

autorise la diffusion de messages à un ensemble de machines d'un même sous-réseau en utilisant une adresse IP spécifique (255.255.255.255 par exemple).

**Communication point-à-point** À l'opposé de la communication par diffusion, il existe le paradigme de communication point-à-point dans lequel les participants s'échangent des messages de manière privé ; c'est-à-dire que lorsque un processus $A$ envoie un message au processus $B$, seul $B$ reçoit le message. De manière duale au précédent, ce mode de communication apparait comme un adversaire lorsqu'un participant souhaite envoyer le même message à plusieurs destinataires. L'impossibilité d'envoyer simultanément ce message à tous les destinataires peut impliquer des délais de réception différents, ou même des absences de réception, si l'emetteur tombe en panne (section 1.4).

La communication point-à-point existe principalement dans les réseaux filaires, où la présence de cables physiques impliquent justement une telle communication : deux entités ne peuvent communiquer que s'il existe un cable entre les deux.

**Communication indirecte/cachée** Le dernier mode de communication que nous avons choisi de présenter est celui de la communication indirecte, ou cachée. Le terme "caché" provient de la théorie de l'information, qui introduit la notion de "canal caché" désignant un canal de communication utilisé de manière détournée par l'utilisateur[3]. Dans le cadre des systèmes répartis, nous parlerons de communication cachée lorsque les entités du systèmes ne disposent pas d'un réel moyen de communication permettant de s'envoyer des messages, mais arrivent tout de même à cummuniquer via des moyens détournés. L'observation du mouvement des autres entités constitue un exemple de communication cachée ; c'est celui qui sera utilisé et détaillé dans le chapitre 3. Un robot (entité participante du système) peut transmettre des informations aux autres robots du système en se déplaçant suivant des motifs particuliers. Nous pouvons faire le rapprochement avec la danse des abeilles qui est un moyen de communication utilisé chez les abeilles pour indiquer la localisation d'une source de nourriture.

Ce type de communication joue le rôle d'adversaire, car il implique des algorithmes complexes pour échanger de l'information. De plus, il implique aussi parfois un accord préalable des participants sur la méthode d'utilisation du canal caché (motifs de mouvement dans le cas précédent).

**Remarques** De manière amusante, nous pouvons retrouver ces trois mêmes modes de communications au niveau de la communication entre les êtres humains. Dans une salle de classe, (1) le professeur s'adresse aux élèves suivant une communication par diffusion, (2) les élèves communiquent entre eux par une communication point-à-point en se faisant passer "discrètement" des petits papiers (ou par sms pour faire plus moderne), (3) les élèves

[3]Un exemple simple, mais très inefficace, consiste à transmettre gratuitement des informations via le réseau téléphonique. Il suffit de laisser sonner une ou deux fois le téléphone du destinataire pour transmettre un 0 ou un 1. Puisque l'appelant ne paye la communication que lorsque le destintaire décroche, on obtient bien une communication gratuite.

communiquent entre eux de manière caché, vis-à-vis du professeur, en mimant ou en utilisant des comportements communautaires que le professeur ne connait/comprend pas.

### 1.3.2 Caractéristiques des communications

Nous présentons ici trois caractéristiques des communications qu'il est possible de trouver dans des systèmes répartis. Ces trois aspects peuvent modifier les modes de communications décrits précédemment.

**Communication synchrone/asynchrone** La première caractéristique est temporelle. Nous distinguons les communications synchrones des communications asynchrones. Dans le premier cas, le modèle considère que, par définition, les temps de transmission des messages sont bornés. Cela signifie qu'il existe une borne, connue ou non par les entités du système, telle que le temps entre le début de transmission d'un message par un émetteur et la fin de la réception de ce message par un destinataire ne peut dépasser cette borne. Au contraire, dans un système où les communications sont asynchrones, il n'existe aucune borne limitant la durée de transmission d'un message. Un message peut prendre un temps arbitrairement long pour être reçu, tout en restant néanmoins fini. L'asynchronisme constitue un puissant adversaire principalement car il empêche la détection des pannes des entités participantes : dans un système asynchrone il est impossible de différencier une entité en panne (qui donc ne répond plus aux messages) d'une entité dont les messages sont très lents à arriver.

Cette notion d'asynchronisme permet de modéliser de manière simple des réseaux complexes. En effet le mode de communication point-à-point fait abstraction de la réalité physique. Dans le cas d'un réseau simple de deux entités connectées directement par un cable, on peut raisonablement considérer le réseau comme sychrone puisque les messages transitent tous physiquement par le même medium. Au contraire dans un grand réseau (tel Internet), les messages échangés par deux entités ne suivent pas tout le temps le même chemin physique, et ne requièrent donc pas forcément le même temps de transmission ; on modélisera alors le système comme asynchrone.

Dans ce manuscrit, c'est la première fois qu'apparait une notion temporelle. Lors de la description des entités (section 1.2), nous n'avons pas directement fait référence à l'éventuel synchronisme ou asychronisme des entités. Ce n'était nullement un oubli puisque, dans les modèles théoriques de systèmes répartis, le contraintes temporelles sont limitées aux communications. Il est en effet équivalent d'inclure les temps de calculs aux temps de transmission ; ainsi nous considérerons les temps de calculs comme nuls.

**Communication partielle** La deuxième caractéristique concerne la portée des communications. Chacun des trois modes de communications présentés peuvent être soit total soit partiel. Nous considérons les communications comme totales, si il n'y a aucune restriction de portée, c'est à dire qu'elles sont telles que définies dans les paragraphes précédents. Nous les considérons comme partielle s'il existe une telle restriction de portée :

– Une diffusion est partielle si chaque entité ne peut émettre que pour un sous-ensemble des entités participantes.
– Une communication point-à-point est partielle si seules certaines paires d'entités peuvent communiquer entre elles.[4]
– Une communication indirecte est partielle si seul un sous-ensemble des entités participantes peuvent capter/comprendre chaque communication.

Une communication partielle constitue un adversaire dont il faut tenir compte lors de la conception d'algorithmes répartis. En effet, il est alors souvent nécessaire de mettre en place des mécanismes de relais pour permettre la transmission d'informations.

**Communication parfaite/imparfaite** La troisème caractéristique reflète la qualité des communications. Une communication parfaite est une communication dans laquelle il n'y a aucun risque de perte ou d'altération du message : lorsqu'une entité envoie un message, le destinataire (ou les destinataires suivant le mode de communication) reçoit le message intact. Une communication imparfaite ne garantit pas que le message émis arrive intact au destinataire ; il peut être perdu, altéré, ou dupliqué durant le transfert.

Le caractère imparfait des communications joue un rôle d'adversaire puisqu'il implique d'ajouter des mécanismes de contrôles supplémentaires pour pallier les défaillances des communications. Encore une fois cette notion de qualité de communication dépend du niveau d'abstraction auquel on se place : sur Internet, les échanges ne sont pas fiables, néanmoins il existe déjà des mécanismes de contrôle (code correcteur+réémissions) au niveau du protocole TCP-IP qui garantissent que les messages arrivent correctement au destinaire.

**Remarque** Exceptée la notion de communication synchrone ou asynchrone, les deux autres caractéristiques sont aussi présentes au niveau de la communication entre humains. La notion de communication partielle ou totale se retrouve lorsqu'on différencie une personne parlant faiblement d'une personne parlant à voix haute. Par ailleurs, des troubles dysphasiques ou des bruits extérieurs peuvent pertuber le message transmis, ce qui d'une certaine façon correspond à une communication imparfaite.

## 1.4 Adversaires imprévisibles du système

Dans les deux sections précédentes nous avons abordé les adversaires liés aux entités et les adversaires liés aux communications. Ces deux catégories d'adversaires correspondent à des notions prévisibles du système[5]. Nous abordons ici les adversaires non prévisibles du système, les impondérables qui font que (si l'on ne fait rien) rien ne marche jamais comme prévu.

[4]Suivant le niveau d'abstraction que l'on considère, Internet peut être vu comme un système réparti avec une communication point-à-point totale (on fait abstraction du routage), ou comme une communication point-à-point partielle (on tient compte des liens physiques reliant chaque machine).

[5]La notion de qualité de communication aurait pu apparaitre dans cette section mais dans une optique de simplicité, nous avons préféré la regrouper avec les autres caractéristiques des communications.

### 1.4.1 Panne franche

L'adversaire imprévisible le plus fréquent est la panne franche d'une entité. À un instant donné une entité du système stoppe prématurément sa participation ; elle n'envoie ni ne reçoit plus aucun message, elle arrête tout mouvement et toute communication indirecte. Cette défaillance est permanente. D'un point de vue théorique, la panne franche permet de modéliser de nombreuses réalités ; panne effective d'un composant, coupure de courant, arrachage de cable (involontaire par le personnel de ménage), ...

Lors de la conception d'algorithmes répartis, nous devons tenir compte d'éventuelles pannes. Cependant il est rarement possible de continuer à fonctionner lors de panne totale. Ainsi nous concevons des algorithmes tolérant uniquement un certain nombre de pannes ; typiquement nous pouvons garantir la validité de certains algorithmes uniquement si moins de la moitié des entités sont défaillantes.

### 1.4.2 Mobilité passive

Précedemment, nous avons introduit la dynamique des entités (le churn) comme étant la capacité des entités à entrer et sortir d'un système réparti. Ce changement est choisi par les entités qui décident de démarrer ou de stopper leur participation au système. Il est cependant possible que ces entrées/sorties du système ne soient pas volontaires, mais au contraire contraintes par des mécanismes non contrôlables par les entités. Nous parlerons dans ce cas de mobilité passive, par opposition à la mobilité active préalablement introduite.

Cette notion est utile pour les réseaux mobiles, dans lesquels les entités ne contrôlent pas leurs mouvements ; c'est le cas par exemple des capteurs fixés sur des animaux. Il est cependant aussi possible de retrouver cette "mobilité passive" dans le cas d'un réseau fixe : Dans un réseau de capteurs fonctionnant à l'énergie solaire ; il est possible que certains capteurs manquent temporairement d'énergie et donc qu'ils soient temporairement déconnectés du réseau. Dans ce cas particulier la terminologie panne/rétablissement (crash/recovery en anglais) est aussi utilisée ; c'est une généralisation du cas particulier de la panne dans lequel une entité en panne peut récupérer de sa défaillance.

### 1.4.3 Comportement malicieux

Le dernier comportement imprévisible des systèmes répartis présenté ici constitue l'adversaire le plus contraignant dans l'optique de concevoir des algorithmes. Il s'agit de la possibilité pour les entités de se comporter de manière malicieuse. Contrairement à la panne franche où l'entité concernée stoppe sa participation, dans le cas d'une défaillance par comportement malicieux, l'entité continue à participer au système mais de manière erronée : elle ne suit plus l'algorithme défini initialement. Le principal risque réside dans la possibilité pour cette entité d'induire en erreur les autres participants et de compromettre ainsi le bon fonctionnement global du système.

Les comportements malicieux permettent de modéliser plusieurs réalités. Il arrive parfois que des données en mémoire ou sur le disque dur soient corrompues, sans que l'entité ne le

détecte ou ne s'arrête ; cela se modélise par un comportement malicieux car après la corruption des données, l'entité peut se comporter de manière arbitraire. Néanmoins l'exemple le plus fréquent consiste en la prise de contrôle d'une entité par un ennemi du système (un pirate sur Internet par exemple). Dans ce cas l'entité ne se comportera non pas de manière arbitraire, mais de manière malsaine vis-à-vis du système si le but de l'ennemi est de compromettre le système. On parlera de comportement byzantin (en référence aux généraux byzantins qui ont trahi leur armée).

## 1.5 Quelques exemples

Au début de cette introduction, nous avons briévement présenté trois exemples de systèmes répartis. Maintenant que nous avons défini plus précisément les différents adversaires existants dans les systèmes répartis, nous pouvons relever les adversaires apparaissant dans chacun des cas. Ceux-ci sont résumés dans le tableau suivant :

| | Adversaires liés aux entités | Adversaires liés au système | Adversaires imprévisibles |
|---|---|---|---|
| Internet (haut niveau) | Grand nombre de participants, grande dynamique | Communication point-à-point asynchrone, totale, parfaite | Comportement malicieux |
| Internet (bas niveau) | Grand nombre de participants, faible dynamique | Communication point-à-point synchrone, partielle, imparfaite | Panne franche |
| Réseau céllullaire | Nombre moyen de participants, moyenne dynamique | Communication par diffusion synchrone, totale, imparfaite | Mobilité passive |
| Réseau de capteurs | Nombre moyen de participants, faible puissance, anonymat | Communication par diffusion synchrone, partielle, imparfaite | Panne franche |

Tab. 1.1 – Les adversaires de quelques systèmes répartis.

Pour l'exemple d'Internet, nous avons choisi de distinguer deux modèles pour montrer qu'un même réseau peut comporter différents adversaires suivant le niveau d'abstraction que l'on considère. À haut niveau, c'est-à-dire au niveau des utilisateurs finaux, le réseau comporte un très grand nombre de machines qui peuvent toutes communiquer entre elles (grâce aux mécanismes de routage) de manière parfaite (grâce aux protocoles de communications). Au contraire, à bas niveau, en considérant les routeurs qui constituent le cœur du réseau, les communications sont partielles (limitées aux cables physiques existants) et potentiellement imparfaite (pertubations magnétiques par exemple). Par ailleurs à haut niveau, nous considérons les comportements malicieux comme principal adversaire, tandis qu'à bas niveau la panne franche prédomine selon nous.

Le réseau céllulaire comporte des adversaires différents, principalement une communication par diffusion imparfaite et une mobilité passive due aux mouvements des porteurs des téléphones. Le réseau de capteurs quant à lui permet d'introduire un système réparti anonyme dont les entités, faiblement puissantes, ne peuvent communiquer qu'avec leurs voisins proches.

## 1.6 Plan de travail

Dans cette introduction, nous avons présenté une liste, non-exhaustive, d'adversaires qu'il est possible de retrouver dans les systèmes répartis ainsi que quelques exemples particuliers de systèmes. Cependant les exemples brièvement expliqués jusqu'à présent ne sont pas ceux qui sont étudiés dans ce manuscrit. Parmi les différents sujets traités durant cette thèse, nous avons choisi d'en présenter trois principaux en détail.

**Chapitre** 2 Dans une première partie, nous nous intéressons à la modélisation de réseaux petits-mondes. Nous proposons une analyse théorique permettant d'évaluer la qualité d'un algorithme de routage glouton dans les réseaux petits-mondes. Cette analyse permet d'obtenir, sans utiliser de longues simulations, une estimation du nombre moyen d'étapes de routage nécessaires pour atteindre une entité du système. Nous proposons ensuite des algorithmes de gossip (bavardage) permettant la construction simple de réseaux petits-mondes, qui malgré leur simplicité demeurent efficaces en terme de routage. Ces travaux ont été effectués en collaboration avec Anne-Marie Kermarrec, Vincent Leroy, et Michel Raynal.

**Chapitre** 3 Dans une seconde partie, nous établissons de nouveaux résultats dans le domaine de l'exploration de graphes par des robots mobiles anonymes. En premier lieu, nous démontrons une borne topologique sur le nombre de robots pouvant être utilisés pour effectuer une telle exploration. En second lieu, nous proposons plusieurs algorithmes effectuant cette exploration suivant que la communication (cachée) entre robots est totale ou partielle. Ces travaux ont été effectués en collaboration avec Roberto Baldoni, Alessia Milani, et Michel Raynal.

**Chapitre** 4 Dans une troisième partie, nous étudions les problèmes d'accord, le consensus en particulier, dans les systèmes anonymes. Nous introduisons de nouveaux détecteurs de fautes "anonymes" permettant la conception d'algorithmes résolvant ces problèmes de manière anonyme. D'un part, nous montrons une nouvelle hierarchie permettant de comparer ces détecteurs. D'autre part, nous proposons deux algorithmes résolvant le consensus dans ce contexte puis abordons la notion de plus faible détecteur de fautes dans les systèmes anonymes. Ces travaux ont été effectués en collaboration avec Michel Raynal.

**Adversaires concernés** Sans rentrer dans les détails décrits ultérieurement dans chaque partie, nous représentons ci-dessous un résumé des adversaires qui entrent en jeu dans les trois systèmes que nous avons choisi d'étudier. Les termes entre parenthèses font référence à des adversaires qui ne seront pas considérés ici, mais qui le sont dans d'autres travaux.

**Remarque** Les deux dernières parties exposent des travaux purement théoriques qui correspondent exactement au schéma décrit au début de la Section 1.1 ; (1) description d'un modèle, (2) description d'un problème, (3) recherche de l'existence de solution, (4) quête

| | Adversaires liés aux entités | Adversaires liés au système | Adversaires imprévisibles |
|---|---|---|---|
| Réseaux petits-mondes | Grand nombre de participants, (dynamique) | Communication point-à-point synchrone, partielle, parfaite | (Panne franche) |
| Robots mobiles | Anonymat, mobilité | Communication indirecte synchrone, partielle ou totale, parfaite | Aucun |
| Système anonyme | Anonymat | Communication point-à-point asynchrone, totale, parfaite | Panne franche |

TAB. 1.2 – Les adversaires des systèmes étudiés.

d'optimalité. Ces deux chapitres, en particulier le dernier, entrent parfaitement dans le cadre de travail habituel de Michel Raynal, le directeur de cette thèse.

Au contraire, les premiers travaux présentés correspondent à une analyse un peu plus pratique d'un système réparti ; cela comprend, par exemple, du codage réel (et non pas juste du pseudo-code) d'algorithmes afin de réaliser des simulations. Ce travail ne rentre ainsi pas exactement dans le cadre théorique donné précédemment, mais il nous a néanmoins semblé important de le présenter afin de donner une ouverture plus large à ce manuscrit.

# Chapitre 2

# Routage dans les réseaux petits-mondes ; théorie et pratique

Les résultats présentés dans ce chapitre correspondent globalement à des traductions de la publication [14] et du rapport technique [15]. Des résultats complémentaires, non présentés ici, apparaissent dans [11].

## 2.1 Introduction

Durant la dernière décennie, les systèmes répartis ont connu une croissance très rapide. Cette évolution n'est pas exclusivement limitée à la recherche académique ; le grand public a lui aussi bénéficié de l'essor des systèmes répartis, principalement par l'apparition des réseaux Pair-à-Pair (PàP) permettant le partage de fichiers (souvent illégaux). Ce nouveau besoin explique en partie pourquoi les réseaux PàP ont pris une place importante dans la communauté scientifique, ou pour être plus précis, dans les communautés scientifiques. En effet deux groupes de personnes, étrangement distinctes (en majorité), étudient ces systèmes pair-à-pair. D'un côté les "praticiens" cherchent à concevoir des algorithmes globalement performants, sans se soucier des pires exécutions possibles. D'un autre côté, les "théoriciens" essayent de trouver des bornes asymptotiques valables pour des systèmes extrêmement grands, sans réellement chercher des algorithmes atteignant ces bornes en pratique. Dans les deux cas, la question du routage est une des problématiques majeures abordées ; cela consiste à être capable de se repérer et se déplacer dans ces grands réseaux. La qualité du routage est une caractéristique cruciale pour l'efficacité globale d'un système PàP. Dans ce chapitre, nous tenterons (1) de considérer à la fois les aspects théoriques et pratiques des réseaux petits-mondes dans lesquels chaque pair[1] est connecté à ses plus proches voisins et à quelques pairs éloignés, et (2) de "réconcilier" les deux domaines pour en tirer bénéfice et créer de nouveaux systèmes plus performants.

[1]Cela correspond à l'entité, comme décrite dans le chapitre 1. Cependant nous conservons le terme pair qui est habituellement utilisé dans ce cadre.

**Contributions** La première contribution de nos travaux consiste en une nouvelle méthode d'analyse des performances de routage dans les réseaux petits-mondes. Cette analyse basée sur des equations récursives se positionne à un niveau intermédiaire entre les analyses théoriques pures (formules closes, bornes asymptotiques) et les études pratiques (simulations, expérimentations). Notre objectif est d'arriver à obtenir des résultats aussi proches que possible des simulations sans avoir à développer un simulateur et exécuter un grand nombre de simulations. Comme nous le verrons ultérieurement, le calcul des performance de routage obtenu par nos formules et celui obtenu par des simulations sont sensiblement équivalents. L'avantage des formules (par rapport aux simulations) réside dans leur généralité et dans le fait qu'il est nettement plus rapide de les utiliser que d'effectuer des simulations.

La deuxième contribution de ce chapitre tient en la description et l'évaluation d'un nouveau protocole épidémique construisant des réseaux petits-mondes. Nous adaptons un protocole existant en utilisant les travaux théoriques de Kleinberg : nous biaisons un algorithme fournissant des liens aléatoires de manière à obtenir des voisins choisis selon une distribution harmonique, comme Kleinberg le préconise [54, 55].

**Remarque** Le Tableau 2.1 résume les différentes méthodes d'analyse d'algorithmes. Dans ce chapitre, nous nous intéressons principalement aux deux lignes intermédiaires ; les formules récursives et les simulations.

| | | |
|---|---|---|
| Théorie | Notation $O(\ )$ | Décrit les comportements asymptotiques pour tous les cas |
| | Formule récurrente | Donne le comportement précis pour un cas particulier |
| Expérimentations | Simulations | |
| | Expérimentations réelles | |

TAB. 2.1 – Les méthodes d'évaluation de performance (de routage)

**Plan du chapitre** Le chapitre comporte 8 sections. La Section 2.2 présente un bref état de l'art sur les réseaux petits-mondes et sur les protocoles épidémiques puis donne quelques motivations ayant entraîné nos recherches . La Section 2.3 décrit les différentes variantes des systèmes étudiés. Les Sections 2.4, 2.5, et 2.6 explicitent le calcul précis du coût du routage pour trois configurations différentes ; respectivement, lorsque les contacts distants sont choisis aléatoirement de manière uniforme sur une topologie en grille, lorsque les contacts distants sont choisis aléatoirement de manière uniforme sur une topologie uniforme, et lorsque les contacts distants sont choisis *à la Kleinberg* sur une topologie uniforme. La Section 2.7 propose un nouvel algorithme épidémique permettant de construire un réseau petit-monde *à la Kleinberg*. Enfin la Section 2.8 conclut le chapitre.

## 2.2 État de l'art et motivation

**Réseaux petits-mondes et résultats de Kleinberg** Les réseaux petits-mondes ont été introduits afin de proposer un cadre théorique permettant de comprendre et d'exploiter la notion des *six degrés de séparation* qui suggère que deux personnes, choisies au hasard, peuvent être reliées par une courte chaîne de connaissances [57]. Cette notion fait référence aux réseaux sociaux entre les individus ; il est néanmoins possible d'obtenir des résultats similaires pour les réseaux informatiques. Chaque pair du réseau doit connaître ses voisins les plus proches (dans une certaine topologie) et quelques pairs distants dans le graphe. Nous appellerons ces liens supplémentaires des raccourcis car ils permettent de se déplacer rapidement dans le graphe sans utiliser une longue liste de voisins successifs. Alors que Watts et Strogatz [68] choisissent aléatoirement ces raccourcis de manière uniforme dans le réseau, Kleinberg a montré qu'il était possible d'obtenir un routage glouton polylogarithmique si les raccourcis sont choisis non pas de manière uniforme, mais suivant une distribution spécifique ($D$-harmonique, $D$ étant la dimension de l'espace) [54, 55]. L'un des principaux résultats de Kleinberg tient dans la détermination de l'ordre de grandeur de la complexité du routage glouton, suivant la méthode de sélection des raccourcis. Ce résultat a ouvert la voie à de nombreux travaux améliorant la complexité du routage, en augmentant la connaissance du réseau ou en changeant légèrement l'algorithme glouton (connaissance/routage à deux sauts par exemple) [9, 39, 38, 61].

**Réseaux épidémiques** Les protocoles épidémiques (ou de gossip en anglais) ont d'abord été introduits pour permettre la diffusion fiable et rapide d'informations dans des réseaux de grande taille [13, 31]. Leurs propriétés de convergence, de fiabilité, et de simplicité les rendent néanmoins intéressants pour beaucoup d'autres tâches que de la diffusion [34, 53]. Ces protocoles peuvent aussi servir à la construction de réseaux PàP ; ils permettent de construire et de maintenir de manière fiable et simple une grande variété de structures [51, 52]. Plus précisément, en jouant sur les choix d'interaction et les données échangées, les protocoles épidémiques permettent de réaliser aussi bien des réseaux aléatoires non-structurés que des réseaux fortement structurés, tels des tables de hachages distribuées (DHT en anglais) [37]. Nous montrons dans ce chapitre que les protocoles de gossip peuvent en outre créer des réseaux petits-mondes *à la Kleinberg*.

Les protocoles épidémiques peuvent servir à construire des réseaux dont les caractéristiques sont très proches des graphes aléatoires [33, 52, 65]. Typiquement, un algorithme épidémique d'échantillonnage aléatoire de pairs (random peer sampling en anglais) fournit à chaque pair une liste de contacts aléatoires du système [52]. Par ailleurs, il existe aussi des protocoles épidémiques de regroupement (clustering en anglais) permettant aux pairs de découvrir leurs voisins proches suivant une certaine métrique [66, 67].

**Motivation** Ce chapitre s'intéresse aux systèmes dans lesquels deux protocoles sont utilisés : d'un côté un algorithme d'échantillonnage fournissant des contacts distants à chaque pair,

de l'autre côté un algorithme de clustering fournissant une liste de voisins proches. La combinaison des deux construit un réseau petit-monde correspondant au modèle de Watts et Strogatz [68].

Kleinberg a montré [55] que dans une telle configuration, le coût du routage est polynomial en $\Omega(n^{\frac{1}{3}})$ nombre de sauts ($n$, étant le nombre de pairs). Toujours d'après les travaux de Kleinberg, le routage devrait être en $\Theta(\log^2 n)$ si l'échantillonnage n'est plus uniforme mais biaisé suivant une distribution harmonique.

Cette disparité théorique n'est pas pertinente pour des "petites" tailles de réseaux, y compris jusqu'à plusieurs centaines de milliers de pairs. Nous souhaitons savoir précisément comment se comportent ces réseaux petits-mondes pour des tailles raisonnables où les complexités asymptotiques ne sont pas suffisament pertinentes. Ainsi, nous avons simulé des systèmes de 5 000 à 500 000 pairs dans lesquels chaque pair est relié à ses 6 voisins les plus proches et à 1 ou 10 contacts lointains. Ces simulations sont effectuées sans l'utilisation d'algorithmes ; il existe un observateur global qui "choisit" les contacts de chaque pair. Cela permet de garantir que la sélection est parfaitement uniforme ou suivant une distribution harmonique. Les résultats apparaissent sur la Figure 2.1.

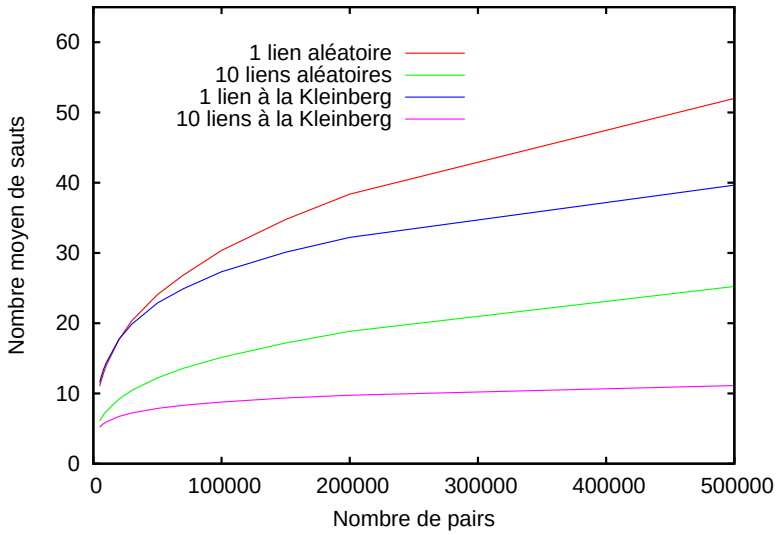



FIG. 2.1 – Liens aléatoire *vs* liens *à la Kleinberg*

Comme attendu, lorsque la sélection des contacts distants est biaisée, le routage est plus efficace. Néanmoins la différence entre les deux situations ne reflète pas entièrement la différence d'ordre de grandeur, même pour des réseaux relativement grands de l'ordre du demi-million de pairs. Cela nous conforte dans l'idée qu'il serait intéressant d'avoir une connaissance plus fine du coût du routage.

## 2.3 Description des systèmes

Cette section décrit les systèmes étudiés dans ce chapitre. Nous considérons un système statique où les pairs ne peuvent défaillir. Nous analysons deux types de topologie ; la topologie en grille et la topologie uniforme. Comparée à la grille, la topologie uniforme est plus pertinente car elle est plus facilement réalisable en pratique. Nous considérons aussi deux types de sélections de raccourcis ; les liens longs choisis aléatoirement de manière uniforme et ceux choisis aléatoirement suivant une distribution harmonique (*à la Kleinberg*).

### 2.3.1 Topologies

Les deux topologies considérées ici sont basées sur un tore de dimension 2. La différence réside dans la répartition des pairs sur le tore comme cela apparaît sur la Figure 2.2. Dans la suite, nous identifierons un pair et une position sur le tore, en interdisant donc d'avoir deux pairs situés exactement au même endroit.

**Topologie en grille** Cette topologie (représentée sur la Figure 2.2(a)) consiste en un réseau de $n = \ell \times \ell$ pairs situés sur une grille carrée de taille $\ell \times \ell$. La position de chaque pair est définie par ses coordonnées $(i, j)$ avec $i, j \in \{0, 1, \ldots, \ell - 1\}$. Comme nous travaillons sur un tore, il faut comprendre que le pair situé en $(0, 0)$ est entre les pairs situés en $(0, 1)$ et en $(0, \ell - 1)$. Dans cette topologie, la distance de Manhattan est utilisée : la distance entre deux pairs $A$ et $B$ situés en $(i_A, j_A)$ et $(i_B, j_B)$ vaut :

$$d_m = \min(|i_B - i_A|, \ell - |i_B - i_A|) + \min(|j_B - j_A|, \ell - |j_B - j_A|).$$

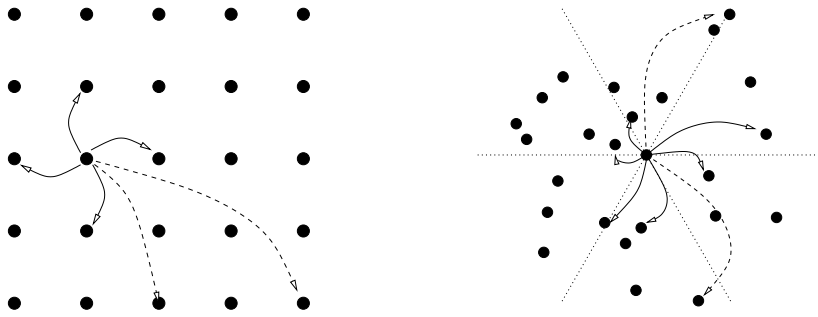
**Topologie uniforme** Dans cette topologie, les positions des $n$ pairs sont choisies aléatoirement de manière uniforme sur le tore $[0 : 1] \times [0 : 1]$. Plus précisément, les coordonnées $(i_A, j_A)$ d'un pair $A$ sont tirées entre 0 et 1 suivant une distribution uniforme. Dans cette topologie, c'est la distance Euclidienne qui est utilisée : la distance entre deux pairs $A$ et $B$ vaut :

$$d_e = \sqrt{\min(|i_B - i_A|, 1 - |i_B - i_A|)^2 + \min(|j_B - j_A|, 1 - |j_B - j_A|)^2}.$$

### 2.3.2 Sélection des contacts

Dans un réseau petit-monde, chaque pair connaît un certain nombre de pairs du système. Ces contacts sont répartis en deux ensembles (appelées vues) ; (1) les voisins ou contacts locaux et (2) les raccourcis ou contacts distants. Les premiers sont des pairs situés à proximité du pair concerné, suivant la distance considérée. Les seconds peuvent être situés n'importe où sur le tore et sont choisis selon une méthode spécifique.

Nous considérons un routage glouton qui permet de naviguer dans le réseau petit-monde. À chaque étape de routage, le message est dirigé vers le pair dont la position sur le tore est la plus proche de la destination. Ainsi à chaque saut, la distance entre le pair actuel et la destination décroît. Le pair actuel sélectionne le prochain pair en cherchant dans ses deux vues, celle des voisins et celle des raccourcis.



(a) Topologie en grille (b) Topologie uniforme

FIG. 2.2 – Les deux topologies avec 25 pairs

**Sélection des voisins**

Afin de garantir la validité du routage glouton (le message atteint la destination en réduisant la distance vers cette destination à chaque étape), chaque pair doit connaître ses voisins les plus proches. Il n'est pas nécessaire d'avoir des contacts distants pour garantir le routage, les contacts locaux suffisent. Les raccourcis sont utilisés uniquement pour améliorer (grandement) les performances.

La sélection des voisins varie légèrement suivant la topologie choisie. Il faut en effet que les vues vérifient quelques propriétés pour garantir qu'à chaque étape de routage, la distance vers la destination diminue.

- **Topologie en grille** Chaque pair doit connaître au moins ses quatre plus proches voisins. Plus précisément le pair en position $(i, j)$ possède les quatre pairs suivants dans sa vue : $(i, j+1 \mod \ell)$, $(i, j-1 \mod \ell)$, $(i+1 \mod \ell, j)$, et $(i-1 \mod \ell, j)$.
- **Topologie uniforme** Chaque pair doit connaître au moins six voisins ; le plus proche dans chacune des six sections de l'espace représentées sur la Figure 2.2(b). Tout découpage régulier de l'espace en un nombre inférieur de sections ne permet pas de garantir la validité du routage glouton ; cela repose en partie sur les résultats de [71] et nous proposons un contre-exemple dans le paragraphe suivant où l'espace est divisé en cinq quartiers. Il est facile de montrer qu'à partir de six quartiers ; ces mauvaises situations ne peuvent plus se produire.

**Contre-exemple avec cinq secteurs** Nous considérons la situation présentée sur la Figure 2.3. Nous étudions une étape de routage à partir de la source $S$ vers la destination $D$ séparée par une distance $d$. Le pair $V_1$ (resp. $V_2$) se situe à une distance $d - \epsilon'$ de $S$ avec un angle $\widehat{DSV_1}$ (resp. $\widehat{DSV_2}$) lègerement inférieur à $\frac{2\pi}{5}$. Dans cette configuration, $S$ choisit $V_1$ (resp. $V_2$) comme voisin de la section correspondante. La distance séparant la destination $D$ du pair $V_1$ (resp. $V_2$) est égale à $\left((d-\epsilon')^2 + d^2 - 2d(d-\epsilon')\cos(\frac{2\pi}{5} - \epsilon)\right)^{\frac{1}{2}}$. Pour $\epsilon$ et $\epsilon'$ suff-

isament petits cette distance est strictement plus grande[2] que $d$. $S$ ne possède alors aucun voisin lui permettant de se rapprocher de la destination ; le routage glouton se termine donc sans atteindre la destination. (L'utilisation d'une autre metrique pour évaluer la distance entre les pairs pourrait permettre de réussir un routage glouton dans cette situation.)

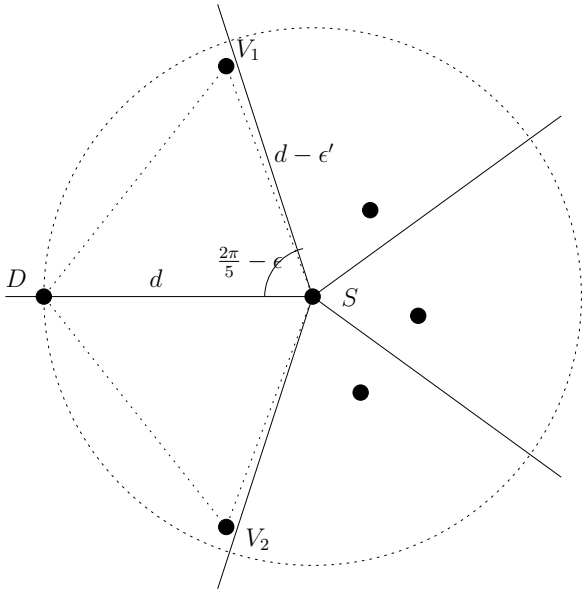



FIG. 2.3 – Contre-exemple avec cinq sections

### Sélection des raccourcis

Les raccourcis constituent la deuxième vue de contacts dont dispose chaque pair. Comme déjà mentionné, elle n'est pas nécessaire pour la validité du routage, mais elle en améliore grandement les performances. Intuitivement, utiliser ces raccourcis permet de diminuer le nombre d'étapes de routage en effectuant des sauts plus importants que ceux disponibles grâce aux voisins de la première vue. Dans ce travail, nous considérons deux méthodes différentes de sélection aléatoire de voisins : une méthode uniforme, que, par abus, nous appellerons aléatoire, et une méthode *à la Kleinberg* issue des travaux de Kleinberg.

- **Raccourcis aléatoires** Comme introduit par le modèle de Watts et Strogatz [68], cette sélection consiste en un choix aléatoire uniforme des raccourcis parmi tous les pairs du réseaux.
- **Raccourcis *à la Kleinberg*** Comme proposé dans [54], les raccourcis peuvent être sélectionnés de manière non uniforme, suivant des distributions spécifique de probabilité. Biaiser le choix des raccourcis permet d'améliorer davantage les performances du routage. Dans le modèle de Kleinberg, un pair $B$ est sélectionné par un pair $A$ pour

[2]Pour $\epsilon = \epsilon' = 0$, la distance $DV_1$ vaut approximativement $1,17d$. Lorsque l'espace est divisé en six quartiers, cette même distance (en remplaçant $\frac{2\pi}{5}$ par $\frac{2\pi}{6}$) est toujours plus petite que $d$.

contact distant suivant une probabilité proportionnelle à $\frac{1}{d(A,B)^2}$ ($d()$ correspondant à la distance de Manhattan ou la distance Euclidienne suivant la topologie).

## 2.4 Réseau petit-monde avec liens aléatoires dans la topologie en grille

### 2.4.1 Remarques et calculs préliminaires

Kleinberg a prouvé que l'utilisation d'un routage glouton dans un réseau petit-monde avec une sélection de raccourcis aléatoires implique une complexité en nombre de sauts d'au moins $\alpha n^{\frac{1}{3}}$, où $n$ correspond au nombre de pairs et $\alpha$ est un coefficient -non déterminé- indépendant de $n$ [54, 55].

Les bonnes performances obtenues dans les réseaux petit-monde créés à partir d'algorithmes de gossip, donc utilisant des raccourcis aléatoires, nous incitent à rechercher la valeur précise de $\alpha$. L'idée sous-jacente est que connaître cette valeur de $\alpha$, et ainsi connaître précisément une borne sur les performances du routage devrait nous permettre d'analyser et de comprendre pourquoi les algorithmes de gossip sont performants.

Bien que les études de Kleinberg ne portent que sur une topologie en grille, nous pensons que ses résultats peuvent aussi s'adapter sur une topologie uniforme. Sans montrer formellement cette adaptation, nous vérifions néanmoins expérimentalement le coût du routage glouton dans les deux topologies.

Les simulations sont effectuées en utilisant PeerSim [64]. Ce simulateur que nous avons complété nous permet de choisir la topologie utilisée, le nombre de pairs, le nombre de voisins, et le nombre de raccourcis. Pour chaque réseau généré, nous testons l'efficacité du routage glouton en sélectionnant aléatoirement un grand nombre (typiquement 500 000) de pairs $(A, B)$, puis en comptant le nombre d'étapes nécessaires pour "router" de $A$ vers $B$.

Les résultats sont présentés sur la Figure 2.4. Nous observons que dans les deux cas, les résultats des simulations confirment l'analyse de Kleinberg : l'exposant $\frac{1}{3}$ apparaît clairement pour les deux topologies.

Afin de comprendre les surprenantes (car inattendues) bonnes performances des mises en œuvres pratiques, nous évaluons ici le coût précis d'un algorithme de routage dans une topologie en grille. Pour obtenir le coût exact, nous ne pouvons pas nous intéresser uniquement aux ordres de grandeurs, mais nous calculons en détail le nombre de sauts nécessaire pour atteindre un pair depuis n'importe quel autre pair. Nous n'obtenons malheureusement pas une formule close, qui reste un problème ouvert, mais une formule récursive qui nous permet de connaître rapidement le nombre moyen d'étapes nécessaires pour atteindre une destination située à une certaine distance d'un point de départ.

Obtenir ces valeurs exactes est très utile car, contrairement aux simulations qui donnent des informations sur un système particulier (nombre de pairs, nombre de voisins, ...), les résultats obtenus à partir de nos formules sont vrais pour tous les réseaux.

Comme décrit précédemment, dans la topologie en grille, il y a $n = \ell^2$ pairs placés sur une grille $\ell \times \ell$, elle même située sur un tore. Soit $r$ le rayon des contacts proches : chaque

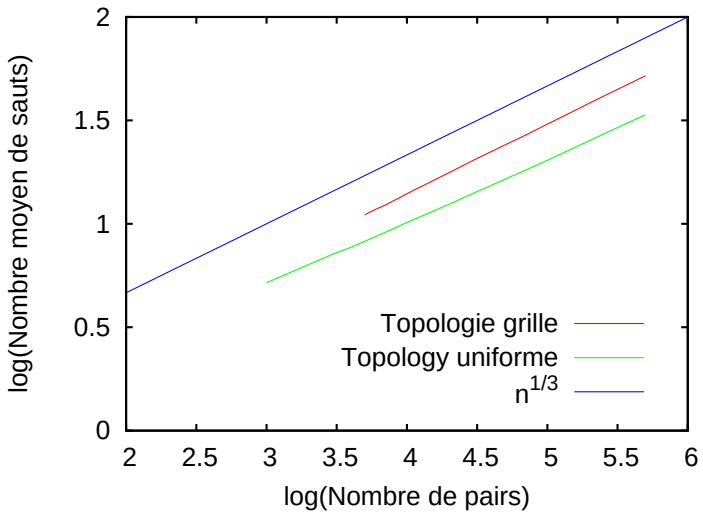



FIG. 2.4 – Coût du routage avec des raccourcis aléatoires dans les deux topologies

pair connaît comme voisin tous les autres pairs qui sont à une distance d'au plus $r$ de sa propre position. Soit $q$ le nombre de contacts distants que chaque pair possède : ici, ces $q$ raccourcis sont choisis de manière aléatoire uniforme.

**Taille du $k$-voisinage** Nous introduisons le $k$-voisinage comme étant l'ensemble des pairs à distance $k$ d'un pair donné. La connaissance de la taille des $k$-voisinages est indispensable pour calculer les probabilités relatives aux positions des raccourcis. De part la régularité de la topologie en grille, nous observons facilement qu'il y a 1 pair à distance 0, 4 pairs à distance 1, 8 pairs à distance 2, et plus généralement $4d$ pairs à distance $d$ lorsque $d < \frac{\ell}{2}$. Pour $d \geq \frac{\ell}{2}$, le nombre de pairs à distance $d$ varie suivant la parité de $\ell$. Si $\ell$ est impair, il y a $4(\ell - d)$ pairs à distance $d$ pour $d \geq \frac{\ell}{2}$. Si $\ell$ est pair, il y a $2\ell - 2$ pairs à distance $\frac{\ell}{2}$, $4(\ell - d)$ à distance $d$ pour $\frac{\ell}{2} < d < \ell$, et 1 pair à distance $\ell$. Ces valeurs sont résumées dans le tableau ci-dessous. Pour des raisons de simplicité, nous supposerons que $\ell$ est impair ; ainsi, chaque pair possède $4\min(\ell - d, d)$ pairs à distance $d$ $0 < d < \ell$.

### 2.4.2 Nombre moyen de sauts pour $r = q = 1$

Le nombre moyen de sauts nécessaires pour atteindre un pair à partir d'un autre pair ne dépend que de la distance entre ces deux pairs. Intuitivement, si la destination est proche de la source, le routage sera plus rapide que si les pairs sont éloignés l'un de l'autre. Par conséquent, nous pouvons définir la variable aléatoire entière $X_d$ qui, étant données une source et une destination séparées d'une distance $d$, retourne le nombre de sauts nécessaire

| | impair $\ell$ | pair $\ell$ |
|---|---|---|
| $d = 0$ | 1 | 1 |
| $0 < d < \frac{\ell}{2}$ | $4d$ | $4d$ |
| $d = \frac{\ell}{2}$ | | $2\ell - 2$ |
| $\frac{\ell}{2} < d < \ell$ | $4(\ell - d)$ | $4(\ell - d)$ |
| $d = \ell$ | 0 | 1 |

TAB. 2.2 – Taille du $k$-voisinage

pour rejoindre la destination à partir de la source. Nous définissons alors la fonction $f$ qui, pour chaque valeur de $d$, associe l'espérance de la variable aléatoire $X_d$ :

$$\forall d \geq 0 \qquad f(d) = \mathbb{E}(X_d)$$

Lorsque $r = q = 1$, nous pouvons faire les observations suivantes :

- $d = 0$ : La destination est la source. Aucune étape de routage n'est requise ; $\mathbb{P}(X_0 = 0) = 1$ et donc $f(0) = 0$.
- $d = 1$ : La destination est à distance 1 de la source. La destination fait partie des contacts locaux de la source. En une étape, il est possible d'atteindre la destination ; $\mathbb{P}(X_1 = 1) = 1$ et donc $f(1) = 1$.
- $d > 1$ : la destination ne fait pas partie des contacts locaux de la source car $d > r = 1$. Le routage glouton doit donc choisir une prochaine étape intermédiaire entre la source et la destination. Cette prochaine étape peut être le raccourci ou un pair choisi parmi les contacts locaux ; les deux cas doivent être distingués. Si un contact local est choisi, la distance vers la destination est réduite d'une unité. Si le raccourci est choisi, la distance vers la destination est aussi réduite, mais potentiellement de plusieurs unités, tout dépend de la position relative du raccourci vis-à-vis de la destination. Il est possible de quantifier le bénéfice apporté par le raccourci, c'est d'ailleurs un des points clés sur lequel repose l'analyse présentée ici. Soit $d(i)$ la probabilité (calculée ultérieurement) que le raccourci du pair courant soit situé à une distance $i$ de la destination. Nous obtenons alors l'expression suivante :

$$\forall d > 1 \quad \forall k \geq 0 \quad \mathbb{P}(X_d = k) = \left( \sum_{i=0}^{d-2} d(i) \mathbb{P}(X_i = k - 1) \right) + \left( 1 - \sum_{i=0}^{d-2} d(i) \right) \mathbb{P}(X_{d-1} = k - 1). \tag{2.1}$$

Le premier terme $(\sum d(i)\mathbb{P}(X_i = k - 1))$ correspond aux cas où les raccourcis sont intéressants, c'est-à-dire situés à une distance au plus $d - 2$ de la destination. Les évenements "le raccourci est à distance $i$" et "il faut $k - 1$ étapes pour atteindre la destination à partir d'une source éloignée à distance $i$" sont indépendants d'où la multiplication des probabilités. Le second terme correspond aux cas restants, lorsque les raccourcis ne permettent pas de progresser vers la destination. Pour obtenir la valeur $f(d)$, nous calculons l'espérance de $X_d$, à partir de l'Équation 2.1 :

$$
\begin{aligned}
\forall d > 1 \quad f(d) &= \sum_{k=0}^{\infty} k\mathbb{P}(X_d = k) \\
&= \sum_{k=0}^{\infty} \left[ k \left( \sum_{i=0}^{d-2} d(i)\mathbb{P}(X_i = k-1) \right) \right] + \sum_{k=0}^{\infty} \left[ k \left( 1 - \sum_{i=0}^{d-2} d(i) \right) \mathbb{P}(X_{d-1} = k-1) \right] \\
&= \sum_{k=0}^{\infty} \left( \sum_{i=0}^{d-2} d(i)(k-1)\mathbb{P}(X_i = k-1) \right) + \sum_{k=0}^{\infty} \left[ \left( 1 - \sum_{i=0}^{d-2} d(i) \right) (k-1)\mathbb{P}(X_{d-1} = k-1) \right] \\
&+ \sum_{k=0}^{\infty} \left( \sum_{i=0}^{d-2} d(i)\mathbb{P}(X_i = k-1) \right) + \sum_{k=0}^{\infty} \left[ \left( 1 - \sum_{i=0}^{d-2} d(i) \right) \mathbb{P}(X_{d-1} = k-1) \right] \\
&= \sum_{i=0}^{d-2} d(i) \left( \sum_{k=0}^{\infty} (k-1)\mathbb{P}(X_i = k-1) \right) + \left( 1 - \sum_{i=0}^{d-2} d(i) \right) \sum_{k=0}^{\infty} (k-1)\mathbb{P}(X_{d-1} = k-1) \\
&+ \sum_{i=0}^{d-2} d(i) \left( \sum_{k=0}^{\infty} \mathbb{P}(X_i = k-1) \right) + \left( 1 - \sum_{i=0}^{d-2} d(i) \right) \sum_{k=0}^{\infty} \mathbb{P}(X_{d-1} = k-1) \\
&= \sum_{i=0}^{d-2} d(i)f(i) + \left( 1 - \sum_{i=0}^{d-2} d(i) \right) f(d-1) + \sum_{i=0}^{d-2} d(i) \times 1 + \left( 1 - \sum_{i=0}^{d-2} d(i) \right) \times 1.
\end{aligned}
$$

Ce qui donne finalement :

$$
\forall d > 1 \quad f(d) = 1 + \sum_{i=0}^{d-2} d(i)f(i) + \left( 1 - \sum_{i=0}^{d-2} d(i) \right) f(d-1). \tag{2.2}
$$

À partir de $f(0)$, $f(1)$, et des définitions récursives de $f(d)$, il est possible de calculer $f(d)$ pour toutes les valeurs de $d$. Il reste néanmoins à déterminer $d(i)$. En fait, cette probabilité $d(i)$ est déjà connue : le nombre de pairs à distance $i$ d'un pair donné a auparavant été calculé ; c'est le $i$-voisinage de ce pair. $d(i)$ est directement relié à ce nombre : puisque les raccourcis sont choisis aléatoirement de manière uniforme parmi tous les pairs du système, la probabilité que le raccourci soit à distance $i$ de la destination est égal au nombre de pairs à distance $i$ divisé par le nombre total de pairs. Ainsi $d(i) = \frac{4\min(i,\ell-i)}{n}$ pour $i > 0$ et $d(0) = \frac{1}{n}$[3]. Bien que cette analyse ne nous fournit pas une formule close pour la fonction $f$ cherchée, cette fonction peut maintenant être calculée facilement et efficacement par un logiciel de calcul numérique. Nous utilisons Maple qui, bien que plutôt conçu pour du calcul symbolique, convient aussi dans notre situation.

Soit $X$ la variable aléatoire, qui étant données une source et une destination (dont on ne connait pas leur distance relative), retourne le nombre de sauts nécessaires pour rejoindre la destination à partir de la source. L'objectif ultime de notre analyse consiste à déterminer

[3]Le dénominateur devrait être $n-5$ car un raccourci ne peut ni être choisi parmi les contacts locaux, ni être le pair considéré. Cependant dans les conditions qui nous concernent, $n-5$ peut raisonnablement s'approximer en $n$.

l'espérance de cette variable aléatoire. Or en conditionnant les probabilités suivant la distance séparant la source de la destination et en tenant compte de l'indépendance des probabilités, nous obtenons une formule simple à partir des probabilités $d(i)$ et des espérances $f(i)$ :

$$\begin{aligned}
\mathbb{E}(X) &= \sum_{k=0}^{\infty} k\mathbb{P}(X=k) \\
&= \sum_{k=0}^{\infty} k \left( \sum_{i=0}^{\ell-1} \mathbb{P}(\text{la source et la destination sont éloignées de } i)\mathbb{P}(X_i=k) \right) \\
&= \sum_{k=0}^{\infty} k \left( \sum_{i=0}^{\ell-1} d(i)\mathbb{P}(X_i=k) \right) \\
&= \sum_{i=0}^{\ell-1} d(i) \left( \sum_{k=0}^{\infty} k\mathbb{P}(X_i=k) \right) \\
&= \sum_{i=0}^{\ell-1} d(i)f(i)
\end{aligned}$$

La Figure 2.5 représente cette espérance $\mathbb{E}(X)$ ainsi que celle obtenue par simulations pour des réseaux allant de 4 000 à 150 000 pairs. Les résultats concordent parfaitement.

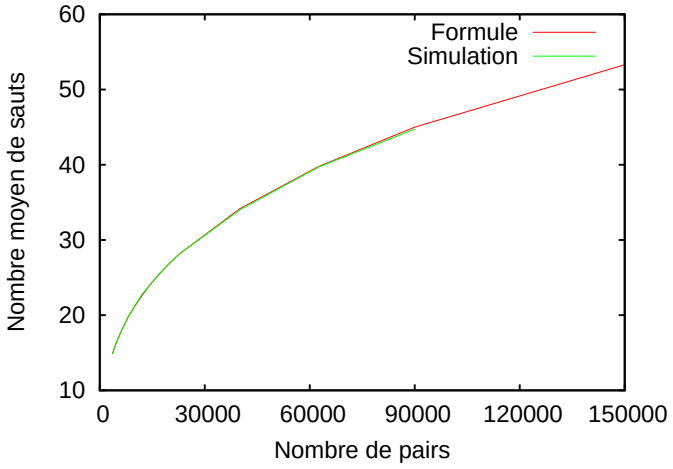



FIG. 2.5 – Comparaison des performances de routage obtenues par simulations et par nos formules, dans une topologie en grille avec sélection aléatoire des raccourcis ($r = q = 1$)

### 2.4.3 Nombre moyen de sauts pour $r$ et $q$ quelconques

L'étude précédente ne considérait que 4 contacts locaux (ceux situés à distance $r = 1$) et $q = 1$ raccourci. Nous pouvons généraliser notre analyse lorsque ces valeurs $r$ et $q$ sont différentes. Les deux cas nécessitent de simples adaptations de la formule récurrente 2.2 :

- $r > 1$ : Tous les pairs à distance au plus $r$ sont des contacts locaux. Lorsque la destination est à une distance inférieure ou égale à $r$, le routage ne requiert qu'un seul saut ; ainsi $f(d) = 1$ pour $1 < d \leq r$. Lorsque la destination est située à une distance supérieure et lorsque les raccourcis ne sont pas utiles, l'utilisation d'un contact local permet de se rapprocher de la destination de $r$ unités en une seule étape.
- $q > 1$ : Chaque pair possède $q$ raccourcis aléatoirement choisis de manière uniforme. Puisque le nombre de raccourcis augmente (par rapport au cas initial), la probabilité de trouver un raccourci utile augmente elle aussi. Notons $d_q(i)$ la probabilité que le meilleur raccourci (le plus proche de la destination) soit à distance $i$ de la destination. Nous avons $d_1(i) = d(i)$ et les autres fonctions pour $q > 1$ se calculent de manière récursive :

$$d_q(i) = d_{q-1}(i) \left(1 - \sum_{k=0}^{i-1} d_1(k)\right) + \left(1 - \sum_{k=0}^{i} d_{q-1}(k)\right) d_1(i). \tag{2.3}$$

  Cette relation récursive s'obtient en décomposant les $q$ raccourcis d'un pair en un ensemble de $q-1$ raccourcis et 1 raccourci supplémentaire. Le premier terme de l'Équation 2.3 correspond au cas où le meilleur des $q-1$ raccourcis est situé à distance $i$ de la destination et le raccourci supplémentaire est au moins à distance $i$ de la destination. Le second terme est le cas inverse ; les $q-1$ raccourcis sont à une distance supérieure à $i$ de la destination et le raccourci supplémentaire est exactement à distance $i$. Ces deux termes décrivent les deux seuls cas (disjoints) qui impliquent que le meilleur des $q$ raccourcis est à distance exactement $i$ de la destination.

En tenant compte des deux améliorations ci-dessus et en reprenant l'Équation 2.2, nous arrivons à la formule générale permettant de calculer l'espérance d'un routage glouton dans une topologie en grille avec des raccourcis choisis aléatoirement de manière uniforme lorsque la source et la destination sont éloignées d'une distance $d$ :

$$\forall d > r \qquad f(d) = 1 + \left(\sum_{i=0}^{d-r-1} d_q(i) f(i)\right) + \left(1 - \sum_{i=0}^{d-r-1} d_q(i)\right) f(d-r). \tag{2.4}$$

L'espérance de la variable alétoire $X$, permettant d'obtenir le coût moyen du routage glouton dans cette configuration, se calcule de la même manière :

$$\mathbb{E}(X) = \sum_{i=0}^{\ell-1} d(i) f(i)$$

Nous comparons les résultats des simulations avec nos résultats analytiques. Comme nous pouvons le constater sur la Figure 2.6, les résultats sont quasiment identiques pour les deux méthodes pour des systèmes avec $r = 2$ et $q = 2$.

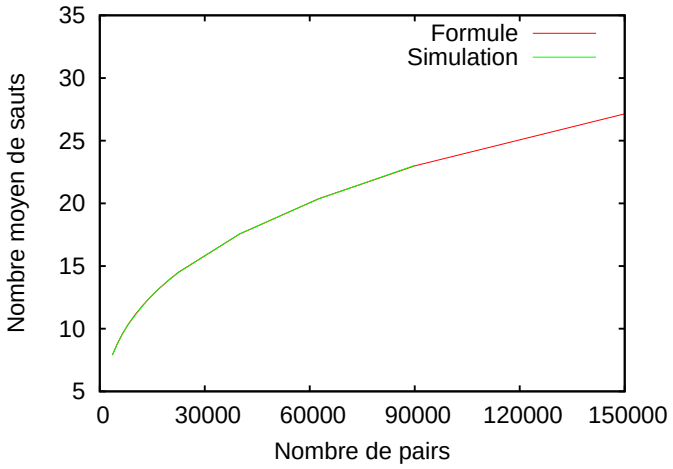



FIG. 2.6 – Comparaison des performances de routage obtenues par simulations et par nos formules, dans une topologie en grille avec sélection aléatoire des raccourcis ($r = q = 2$)

## 2.5 Réseau petit-monde avec liens aléatoires dans la topologie uniforme

### 2.5.1 Remarques préliminaires

À partir de cette section, nous entrons dans le domaine du continu grâce à la topologie uniforme, par opposition à la grille qui est discrète. Les sommes se transforment en intégrales. Puisque le symbole $d$ utilisé jusqu'à présent désigne une distance, nous choisissons d'omettre les variables d'intégration $dx$ pour éviter les ambiguïtés. Par ailleurs nous décidons que $p$ désigne le nombre de contacts locaux et $q$ le nombre de contacts distants.

L'analyse pour la topologie uniforme requiert des calculs nettement plus approfondis. La principale différence réside dans la distance séparant les pairs. Au lieu d'être des entiers, les distances sont maintenant des réels de l'ensemble $[0, \frac{\sqrt{2}}{2}]$ (rappelons que les pairs sont sur un tore $[0:1] \times [0:1]$). Le domaine de définition de la fonction $f$ change aussi en conséquence en passant lui aussi des naturels aux réels : pour tout réel $d$, $f(d)$ correspond à l'espérance de la variable aléatoire entière $X_d$ qui, étant données une source et une destination séparées d'une distance $d$, retourne le nombre de sauts nécessaire pour rejoindre la destination à partir de la source.

### 2.5.2 Gestion des contacts locaux

Dans notre analyse de la topologie en grille, le nombre de contacts locaux n'est pas directement fixé ; c'est le rayon de voisinage $r$ qui détermine le nombre de voisins. Puisqu'ici il

n'y a plus de positions régulières pour les pairs, nous devons estimer un rayon $r_p$ correspondant à une connaissance de $p$ voisins. Comme le tore $[0:1] \times [0:1]$ s'étend sur une surface de superficie 1, la surface moyenne "couverte" par $p$ pairs vaut $\frac{p}{n}$. En utilisant un disque pour approximer cette surface, on obtient un disque de rayon $r_p = \sqrt{\frac{p}{n\pi}}$.

Cette approximation est relativement grossière, elle ne prend pas en compte le fait qu'au moins un voisin doit être choisi dans chacune des 6 "directions" pour garantir la validité du routage glouton (voir Section 2.3.2). Ainsi cette analyse ne peut être considérée comme pertinente que pour des valeurs de $p$ suffisament grandes pour lesquelles la probabilité de connaitre effectivement ces 6 voisins est grande. Dans nos évaluations, nous choisissons la valeur $p = 20$ qui nous semble pertinente.

### 2.5.3 Gestion des contacts distants

Dans l'analyse précédente, nous évaluons $d_q(i)$, la probabilité que le meilleur des $q$ raccourcis se situe à une distance $i$ de la destination. Tout comme pour la fonction $f$, nous passons maintenant au monde continu et $d_q(i)$ devient alors la densité de probabilité que le meilleur des $q$ raccourcis se trouve à une distance $i$ de la destination.

**Détermination de $d_1$** Puisque la sélection du raccourci est réalisée aléatoirement de manière uniforme, nous devons évaluer la proportion de pairs situés à une distance inférieure à $i$ d'un pair donné. Or puisque les pairs sont eux-mêmes répartis de manière aléatoire uniforme sur le tore, il suffit d'évaluer la surface $S(i)$ définie par l'ensemble des points à distance inférieure ou égale à $i$ d'un point donné. $d_1$ s'obtient alors en dérivant la fonction $S$ ; ce calcul correspond à la densité de probabilité que le raccourci se situe à une distance inférieure à $i$.

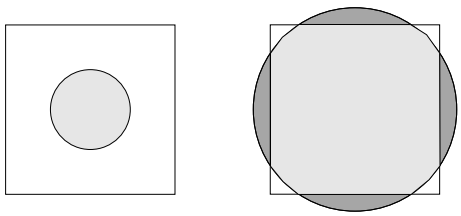


FIG. 2.7 – Détermination de la densité de probabilité $d_1$

Pour $0 \leq i \leq \frac{1}{2}$, le calcul est simple car la surface $S(i)$ est un disque de rayon $i$ : c'est la zone en gris clair représentée sur la partie gauche de la Figure 2.7 dont la surface est donc $S(i) = \pi i^2$. Par conséquent pour $0 \leq i \leq \frac{1}{2}$, nous avons $d_1(i) = 2\pi i$.

Pour $\frac{1}{2} < i \leq \frac{\sqrt{2}}{2}$, les calculs sont légèrement plus complexes. En effet la surface $S(i)$ correspondant à la zone en gris clair sur la partie droite de la Figure 2.7 n'est plus une forme géométrique simple : le disque de rayon $i$ "déborde" en dehors du tore[4]. Nous calculons

[4]Il serait intéressant d'adapter cette étude sur une sphère. Ce problème n'apparaîtrait plus, mais d'autres problèmes relatifs aux distances surgiraient.

alors la zone grise claire en soustrayant les parties grises foncées à la surface du disque. Plus précisément nous obtenons :

$$\begin{aligned}
S(i) &= \pi i^2 - 4\int_{x=\frac{1}{2}}^{i}\int_{y=-\sqrt{i^2-x^2}}^{\sqrt{i^2-x^2}} 1, \\
&= \pi i^2 - 8\int_{x=\frac{1}{2}}^{i}\int_{y=0}^{\sqrt{i^2-x^2}} 1, \\
&= \pi i^2 - 8\int_{x=\frac{1}{2}}^{i}\sqrt{i^2-x^2}, \\
&= \pi i^2 - 2\pi i^2 + \sqrt{4i^2-1} + 4i^2 \arcsin\frac{1}{2i}, \\
&= -\pi i^2 + \sqrt{4i^2-1} + 4i^2 \arcsin\frac{1}{2i}.
\end{aligned}$$

Après dérivation, pour $\frac{1}{2} < i \leq \frac{\sqrt{2}}{2}$, nous avons $d_1(i) = -2\pi i + 8i \arcsin\left(\frac{1}{2i}\right)$.

**Détermination de $d_q$** Les fonctions $d_q$ pour $q > 1$ peuvent être déterminées récursivement à partir de $d_1$ tout comme pour le cas de la topologie en grille. La méthode est similaire à celle décrite à l'équation 2.3 en changeant les sommes en intégrales :

$$d_q(i) = d_{q-1}(i)\ \left(1 - \int_{k=0}^{i} d_1(k)\right) + \left(1 - \int_{k=0}^{i} d_{q-1}(k)\right) d_1(i). \tag{2.5}$$

### 2.5.4 Nombre moyen de sauts

En tenant compte des remarques précédentes sur les contacts locaux et les contacts distants, la formule récursive 2.4 s'adapte à la topologie uniforme :

$$\begin{aligned}
0 < d \leq r_p : &\qquad f(d) = 1, \\
\forall d > r_p \qquad f(d) &= 1 + \left(\int_{i=0}^{d-r_p} d_q(i) f(i)\right) + \left(1 - \int_{i=0}^{d-r_p} d_q(i)\right) f(d - r_p).
\end{aligned} \tag{2.6}$$

Comme pour la topologie en grille, nous comparons les performances du routage obtenues par des simulations et celles obtenues à partir de notre formule. La Figure 2.8 propose cette comparaison pour des systèmes allant de 4 000 à 150 000 pairs où chaque pair possède $p = 20$ voisins et $q = 2$ raccourcis. Contrairement aux résultats de la topologie en grille, nous observons ici une légère différence entre les deux courbes. Cela provient de notre gestion imparfaite des contacts locaux. En effet nous considérons dans notre étude que l'usage d'un voisin permet toujours de réduire la distance de $r_p$ vers la destination ; en pratique ce voisin est situé sur le disque de rayon $r_p$ donc le bénéfice de son utilisation peut être inférieur. Il serait intéressant d'essayer de corriger ce biais en tenant compte de cette remarque et en tentant de moyenner le bénéfice apporté par l'usage d'un raccourci. Cependant cette recherche est actuellement suspendue...

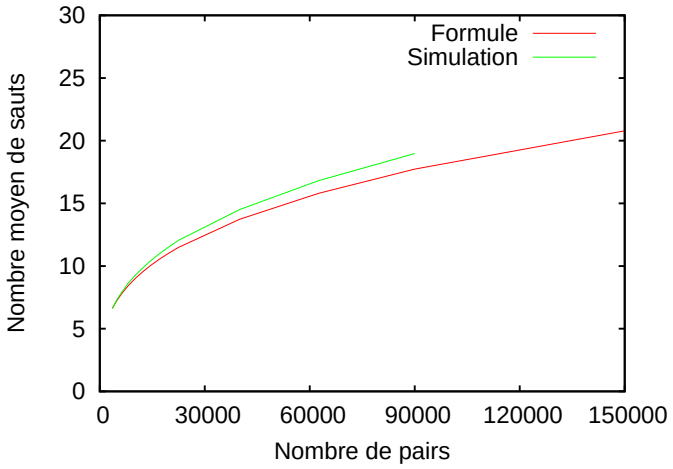



FIG. 2.8 – Comparaison des performances de routage obtenues par simulations et par nos formules, dans une topologie uniforme ($p = 20$ et $q = 2$) avec sélection aléatoire des raccourcis

## 2.6 Réseau petit-monde avec liens Kleinberg dans la topologie uniforme

Les Sections 2.4 et 2.5 ont étudié le coût d'un routage glouton dans un réseau petit-monde où les contacts distants sont sélectionnés aléatoirement de manière uniforme, respectivement dans une topologie en grille et dans une topologie uniforme. Nous montrons maintenant qu'il est possible de faire une étude similaire lorsque la sélection des contacts est biaisée selon une distribution harmonique *à la Kleinberg*. Paradoxalement, la régularité de la topologie en grille complexifie beaucoup (trop) l'étude des raccourcis *à la Kleinberg* : pour être plus précis, c'est la dissymétrie engendré par l'utilisation combinée d'un tore et d'un grille qui rend l'étude impossible, du moins trop complexe selon nous : lorsque deux pairs $A$ et $B$ sont distants de $k$ unités, le nombre de pairs à distance $k-1$ de $A$ et à distance 1 de $B$ dépend de la position relative de $A$ et $B$. Or, pour l'étude des liens longs *à la Kleinberg*, il nous est nécessaire de connaître ce nombre de pairs. Ainsi pour briser cette dissymétrie[5], nous avons choisi de nous limiter à l'étude sur la topologie uniforme où ce problème est moindre.

### 2.6.1 Gestion des contacts locaux

Seule la méthode de sélection des raccourcis change, les $p$ voisins sont traités comme dans la Section 2.5 : nous considérons ici que les voisins couvrent un disque de rayon $r_p = \sqrt{\frac{p}{n\pi}}$

[5]C'est suffisamment inhabituel pour être remarqué ; en temps normal on cherche plutôt à briser la symétrie.

centré sur le pair considéré. Lorsque la destination d'un routage appartient à ce disque, nous estimons que le routage s'effectue en une seule étape.

### 2.6.2 Gestion des contacts distants

Dans cette section, les raccourcis sont choisis suivant une distribution non-uniforme parmi les pairs du réseau. Comme décrit dans la Section 2.3.2, un pair $B$ est choisi comme raccourci par un pair $A$ suivant une probabilité proportionnelle à l'inverse du carré de la distance séparant $A$ et $B$. Nous connaissons déjà la distribution des distances entre deux pairs grâce à la densité de probabilité exprimée par la fonction $d_1$. Nous pouvons alors en déduire la densité de probabilité *dist* qu'un raccourci $B$ d'un pair $A$ soit situé à distance donnée de celui-ci :

$$\forall r_p < i \leq \frac{\sqrt{2}}{2} : \qquad dist(i) = \frac{\frac{d_1(i)}{i^2}}{\int_{k=r_p}^{\frac{\sqrt{2}}{2}} \frac{d_1(k)}{k^2}}. \tag{2.7}$$

Cette densité de probabilité n'est définie qu'à partir de $i = r_p$ car les raccourcis ne peuvent pas être sélectionnés parmi les contacts locaux. Cette remarque permet de préciser qu'en réalité notre modèle est plus proche d'un système où chaque pair connaît tous les pairs situés dans un disque de proximité (quel que soit le nombre de ces pairs) et quelques pairs supplémentaires répartis dans le réseau. Le système de départ que nous souhaitons modéliser (celui décrit dans la Section 2.3) considère lui un nombre prédéfini de voisins.

Nous connaissons la densité de probabilité *dist* qu'un raccourci $R$ d'une source $S$ soit situé à une distance donnée de $S$. Ce n'est cependant pas l'information dont nous avons besoin pour calculer le coût du routage. Il est nécessaire de connaître la densité de probabilité qu'un raccourci $R$ du pair $S$ soit situé à une distance donnée de la destination $D$. Dans le cas où les raccourcis sont choisis de manière uniforme parmi des pairs uniformément répartis, ces deux probabilités sont les mêmes. Dans la situation actuelle, puisque les raccourcis ne sont plus choisis de manière uniforme les deux densités de probabilités diffèrent.

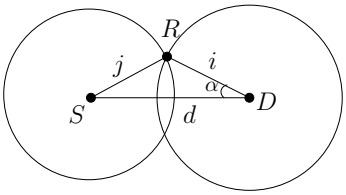



FIG. 2.9 – Position relative d'une Source, d'une Destination, et d'un Raccourci

Considérons tout d'abord que chaque pair ne possède qu'un seul raccourci ($q = 1$). Soit $d'_1$ la densité de probabilité que le raccourci soit à une distance donnée de la destination. La Figure 2.9 présente la situation : $S$, $D$, et $R$ représentent respectivement la source, la destination, et le raccourci ; $d$, $i$, et $j$ les distances $SD$, $DR$, et $SR$. Une simple analyse

géométrique nous permet d'obtenir que $j = \sqrt{d^2 + i^2 - 2di\cos(\alpha)}$. En sommant sur toutes les positions possibles du raccourci $R$ sur le cercle de droite, nous obtenons la formule suivante pour la densité de probabilité :

$$d'_1(d,i) = i \int_{\alpha=0}^{2\pi} \frac{dist\left(\sqrt{d^2 + i^2 - 2di\cos(\alpha)}\right)}{2\pi\sqrt{d^2 + i^2 - 2di\cos(\alpha)}}.$$

Remarquons que contrairement aux deux études précédentes (Sections 2.4 et 2.5), la densité de probabilité prend désormais un paramètre supplémentaire $d$, la distance entre la source et la destination. Ainsi, de manière plus correcte, nous avons déterminé une famille de densités de probabilité, une pour chaque valeur de $d$. À partir de cette famille de fonctions $d'_1$, nous pouvons calculer récursivement la familles de fonctions $d'_q$ qui correspondent aux densités de probabilité lorsqu'il y a $q$ raccourcis. Plus précisément $d'_q(d,-)$ est la densité de probabilité que le meilleur des $q$ raccourcis se situe à une distance donné de la destination, lorsque cette destination est elle-même distante de $d$ par rapport à la source.

$$d'_q(d,i) = d'_{q-1}(d,i)\left(1 - \int_{k=0}^{i} d'_1(d,k)\right) + \left(1 - \int_{k=0}^{i} d'_{q-1}(d,k)\right) d_1(d,i). \tag{2.8}$$

### 2.6.3 Nombre moyen de sauts

De manière similaire à l'Equation 2.6, nous obtenons la formule récursive donnant le coût moyen du routage entre une source et une destination éloignée d'une distance $d$ :

$$\forall d > r_p \qquad f(d) = 1 + \left(\int_{i=0}^{d-r_p} d'_q(d,i) f(i)\right) + \left(1 - \int_{i=0}^{d-r_p} d'_q(d,i)\right) f(d - r_p). \tag{2.9}$$

La Figure 2.10 compare les résultats des simulations avec les valeurs calculées par notre formule, pour des réseaux allant de 4 000 à 150 000 pairs. Pour cette comparaison, nous supposons que chaque pair connaît $p = 20$ contacts locaux et $q = 2$ contacts distants. Comme dans l'étude précédente (Section 2.5), il existe une légère différence entre les deux courbes. Selon nous, cet écart s'explique de nouveau par la gestion des voisins. Néanmoins la formule permet une bonne estimation des performances de routage dans de tels systèmes.

## 2.7 Algorithmes épidémiques fabriquant des réseaux petits-mondes

Les trois sections précédentes ont permis d'analyser précisément le coût des algorithmes de routage glouton dans différents types de réseaux petits-mondes. Cependant, dans tous les cas (simulation et formule), nous avons considéré des systèmes parfaits construits par un observateur global. Cet observateur global connaît les positions de tous les pairs et peut donc tirer aléatoirement les liens distants de manière exacte, par rapport à la distribution

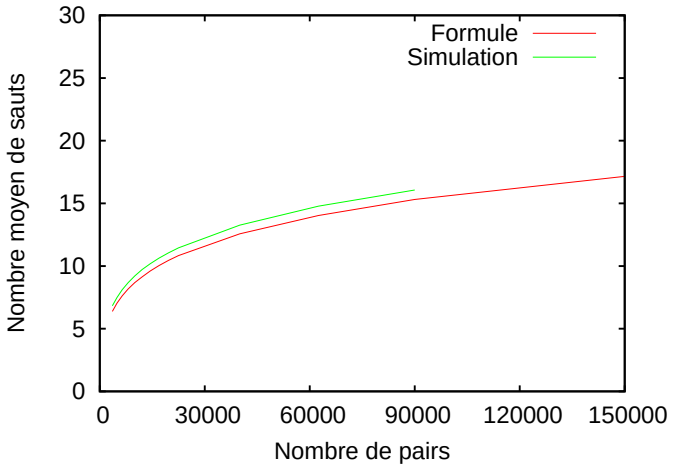



FIG. 2.10 – Comparaison des performances de routage obtenues par simulations et par nos formules, dans une topologie uniforme ($p = 20$ et $q = 2$) avec sélection *à la Kleinberg* des raccourcis

choisie. Dans cette section, nous montrons comment il est possible en pratique de générer des réseaux petits-mondes.

Lors de la présentation des réseaux épidémiques en Section 2.2, nous avons rappelé qu'il existe déjà des algorithmes permettant de construire les deux vues d'un réseau petit-monde. La vue des contacts distants pouvant être conçue de manière à contenir une liste de pairs tirés aléatoirement de manière uniforme dans le réseau. Or comme, nous l'avons compris précédemment dans ce chapitre, l'utilisation de contacts distants sélectionnés suivant une distribution harmonique améliore les performances du routage. Nous proposons alors ici une adaptation d'un algorithme existant afin de permettre un tel tirage aléatoire de contacts distants.

Pour appuyer l'intérêt pratique du biaisement de la sélection des raccourcis, nous comparons les performances d'un système avec les deux méthodes de sélections. Les résultats apparaissant sur la Figure 2.11 sont basés sur un système utilisant une topologie uniforme de 200 000 pairs, où chaque pair connaît 10 contacts proches et où nous avons fait varier le nombre de contacts distants entre 1 et 10. Comme attendu, pour les deux situations, les performances de routage s'améliorent lorsque le nombre de raccourcis augmente. De manière plus inattendue, nous remarquons aussi que le gain apporté par le biaisement de la sélection aléatoire augmente avec le nombre de raccourcis : pour 1 raccourci, le gain est d'environ 16%, alors que pour 10 raccourcis, il atteint presque 50%. Notons tout de même que bien que ce gain soit important en pourcentage, les performances obtenues par une sélection uniforme restent raisonnables et utilisables en pratique (20 sauts en moyenne, pour 10 raccourcis).

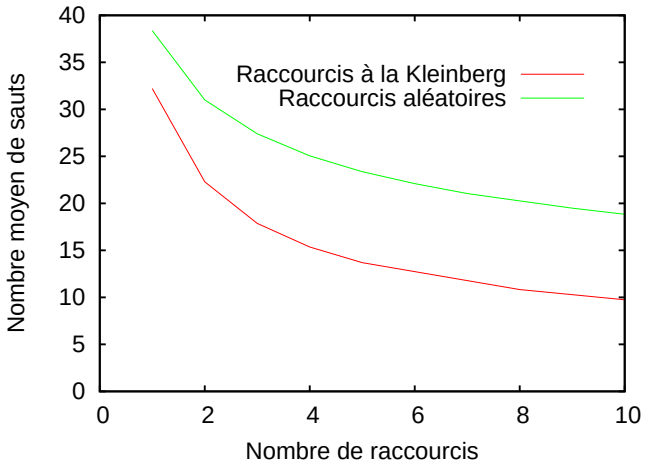



FIG. 2.11 – Comparaison entre les deux méthodes de sélection des raccourcis (cas exacts)

**Remarque** Dans cette section, à l'image du reste du chapitre, nous ne considérons que des systèmes statiques sans panne. Pour avoir une idée de ce que valent les protocoles présentés dans un système dynamique (churn) avec pannes, nous orientons le lecteur vers nos travaux [11] que nous avons décidé de ne pas présenter ici car ils proviennent en majorité des réflexions d'un autre doctorant, Vincent Leroy.

### 2.7.1 Algorithmes existants

Nous considérons ici une topologie uniforme. En effet d'un point de vue algorithmique il est (quasiment) impossible de réaliser une topologie en grille. Initialement chaque pair participant au système détermine ses coordonnées par l'utilisation de deux fonctions de hachages appliquées sur une caractéristique unique de chaque pair (adresse IP, par exemple). Cela garantit, avec une grande probabilité, que chaque pair dispose d'une position unique dans l'espace.

Tous les protocoles épidémiques suivent les mêmes mécanismes d'échange de données. Nous décrivons ces mécanismes puis précisons comment il est possible de les spécialiser pour obtenir ce que l'on souhaite : une liste de contacts locaux d'une part et une liste de contacts distants d'autre part.

**Un protocole épidémique générique** Dans un protocole épidémique, chaque pair conserve une vue partielle du réseau, c'est-à-dire une petite liste de pairs ; typiquement 10 à 20 contacts pour des réseaux de plusieurs centaines de milliers pairs. Cette vue crée un graphe de connexion où un pair $A$ possède une arête vers un pair $B$ lorsque ce dernier appartient à

la vue de $A$. Chaque pair exécute deux tâches simultanément, une active et une passive.

- *Tâche active :* Périodiquement tout pair $A$ (i) choisit un pair $B$ dans sa vue, (ii) envoie un message à $B$ contenant une partie de sa vue, (iii) attend une réponse de $B$ et calcule sa nouvelle vue à partir de son ancienne et des informations transmises par $B$.
- *Tâche passive :* En réponse à une sollicitation d'un pair $B$, le pair $A$ (i) répond à $B$ en envoyant un message contenant une partie de sa vue, (ii) calcule sa nouvelle vue à partir de son ancienne et des informations transmises par $B$.

Dans cette brève description des algorithmes épidémiques, il faut relever trois étapes importantes qui vont influencer la structure vers laquelle le graphe va converger. Ces trois mécanismes permettent de définir une grande variétés de protocoles épidémiques en modifiant la manière de (1) sélectionner le pair avec lequel l'échange va avoir lieu, (2) choisir les informations à partager, (3) construire les nouvelles vues après un échange.

**Sélection des contacts locaux** À l'aide d'un protocole épidémique, il est possible de converger vers un graphe où chaque pair connaît ses plus proches voisins. Il faut essayer néanmoins de maintenir la propriété énoncée en Section 2.3.2 garantissant le routage : tout pair connaît au moins un voisin dans chacun des 6 secteurs de l'espace. Un algorithme basique de "clustering" fonctionne de la façon suivante :

1. *Choix du pair :* De manière cyclique, l'algorithme sélectionne le pair le plus proche dans chacun des quartiers de l'espace.
2. *Choix des données :* Toute la vue est échangée (ainsi qu'éventuellement la vue contenant les contacts lointains afin d'améliorer la vitesse de convergence).
3. *Choix de la nouvelle vue :* Il faut (si possible) conserver le pair le plus proche dans chacun des quartiers de l'espace puis compléter ensuite la vue avec les autres pairs les plus proches.

**Sélection des contacts distants aléatoires** À partir du schéma de base, nous pouvons aussi obtenir un algorithme épidémique construisant une vue de contacts distants aléatoires (uniformément répartis). Cyclon [65] est un protocole épidémique qui construit un graphe dont les propriétés sont très proches de celles d'un graphe aléatoire. Son fonctionnement est brièvement décrit ci-dessous et consiste globalement à un mélange des vues des deux pairs ayant une interaction.

1. *Choix du pair :* Le pair, avec lequel l'échange a lieu, est choisi uniformément au hasard dans la vue.
2. *Choix des données :* Un nombre fixé de pairs de la vue est transmis. Typiquement cela peut être la moitié de la vue.
3. *Choix de la nouvelle vue :* Les pairs transmis sont remplacés par ceux reçus, pour chacun des deux participants de l'échange.

Afin de permettre au lecteur de comprendre précisément le comportement de cet algorithme et les différences avec l'algorithme présenté ensuite, nous donnons l'exemple d'un

échange possible. Les vues sont de taille 6 et l'échange porte sur la moitié de la vue, c'est-à-dire sur 3 pairs.

**Exemple** Considérons deux pairs $A$ et $B$ dont les vues initiales sont respectivement $v_A = \{B, C, D, E, F, G\}$ et $v_B = \{U, V, W, X, Y, Z\}$. Un échange effectué par $A$ pourrait être :

1 $A$ choisit aléatoirement, de manière uniforme, un pair dans sa vue, disons $B$.

2 $A$ choisit aléatoirement, de manière uniforme, 2 autres pairs dans sa vue, disons $C$ et $E$ et envoie alors un message au pair $B$ contenant l'ensemble $v_{A\to B} = \{A, C, E\}$ ($A$ remplace l'identité de $B$ qu'il a choisie, par sa propre identité).

2' $B$ choisit aléatoirement, de manière uniforme, 3 pairs dans sa vue, disons $U$, $W$, et $Z$ et envoie alors un message au pair $A$ contenant la liste $v_{B\to A} = \{U, W, Z\}$.

3 $A$ reçoit le message de $B$ et met à jour sa vue en remplaçant les pairs qu'il a envoyé à $B$ par ceux reçus de $B$ : $v_A = (v_A \setminus v_{A\to B}) \cup v_{B\to A}$. La vue de $A$ devient donc $v_A = \{D, F, G, U, W, Z\}$.

3' De manière similaire $B$ met à jour sa vue : $v_B = (v_B \setminus v_{B\to A}) \cup v_{A\to B}$, ce qui donne $v_B = \{A, C, E, V, X, Y\}$.

### 2.7.2 Nouvel algorithme de sélection *à la Kleinberg*

Afin d'améliorer le routage dans les réseaux petits-mondes obtenus par l'utilisation simultanée des deux algorithmes précédents, nous biaisons l'algorithme fournissant les contacts distants. L'objectif est d'atteindre une sélection des contacts qui soit aussi proche que possible de la sélection optimale démontrée par Kleinberg [54]. Le problème réside dans la connaissance partielle du réseau par chaque pair ; nous montrons cependant qu'une sélection *à la Kleinberg* est possible.

**Sélection des contacts distants** *à la Kleinberg* Le biaisement s'effectue uniquement au niveau du choix des pairs transmis lors d'un échange. Ces pairs ne sont plus choisis aléatoirement de manière uniforme. Plus précisément, chacun des deux pairs participants à l'échange va choisir les contacts qu'il conserve dans sa vue suivant des probabilités proportionnelles à l'inverse du carré de la distance le séparant des contacts.

**Exemple** En considérant le même exemple que précédemment, un échange initié par $A$ pourrait être le suivant :

1 $A$ choisit aléatoirement, de manière uniforme, un pair dans sa vue, disons $B$.

2 $A$ calcule pour tous les autres contacts $x \in v_A \setminus \{B\} = \{C, D, E, F, G\}$ la valeur $p_x = \frac{1}{d(A,x)^2}$. $A$ choisit alors de conserver 3 pairs suivant des probabilités proportionnelles aux $p_x$ calculées, disons $D$, $E$, et $F$. Il envoie donc un message à $B$ contenant l'ensemble $v_{A\to B} = \{A, C, G\}$.

2' De manière similaire $B$ calcule les valeurs $p_x = \frac{1}{d(B,x)^2}$ pour tous les pairs $x$ de sa vue et $B$ choisit de conserver 3 pairs suivant des probabilités proportionnelles aux

$p_x$ calculées, disons $U$, $V$, et $Z$. Il envoie donc un message à $A$ contenant l'ensemble $v_{B\to A} = \{W, X, Y\}$.

3 $A$ reçoit le message de $B$ et met à jour sa vue en remplaçant les pairs qu'il a envoyé à $B$ par ceux reçus de $B$ : $v_A = (v_A \setminus v_{A\to B}) \cup v_{B\to A}$. La vue de $A$ devient donc $v_A = \{D, E, F, W, X, Y\}$.

3' De manière similaire $B$ met à jour sa vue : $v_B = (v_B \setminus v_{B\to A}) \cup v_{A\to B}$, ce qui donne $v_B = \{A, C, G, U, V, Z\}$.

**Évaluation de cet algorithme** L'évaluation de cet algorithme de sélection *à la Kleinberg* porte sur deux points principaux. Il s'agit de vérifier tout d'abord que le routage est plus performant que celui obtenu en utilisant un algorithme choisissant les contacts distants de manière aléatoire. Il s'agit ensuite d'évaluer la qualité de la sélection effectuée localement. Nous avons étudié ces deux aspects et présentons les résultats sur la Figure 2.12. Nous faisons varier la taille du réseau, mais dans tous les cas, chaque pair possède une vue de 10 contacts locaux et une vue de 10 contacts distants.

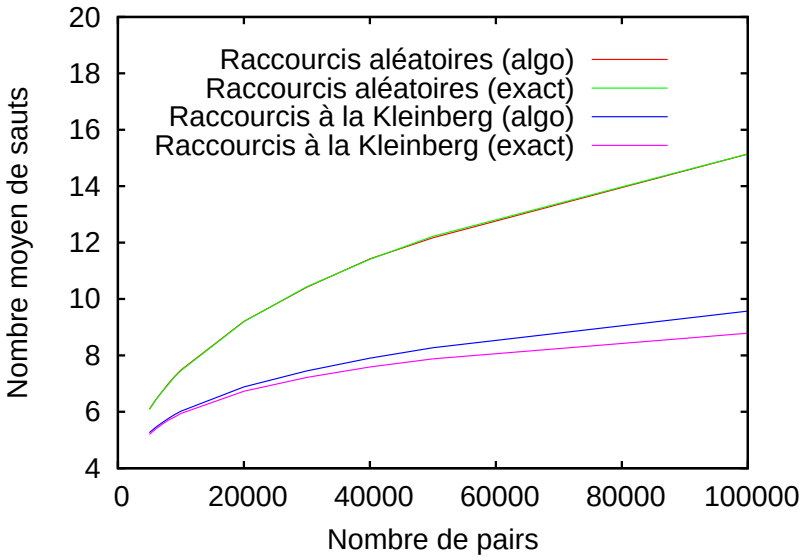



FIG. 2.12 – Comparaison entre les deux méthodes de sélection des raccourcis

Les courbes verte et violette (celles intitulées "exact") correspondent à des simulations où aucun algorithme n'est utilisé. Comme dans les premières sections de ce chapitre, il existe un superviseur global qui détermine les voisins de chaque pair de manière aléatoire, parfaitement uniforme ou suivant une distribution exactement *à la Kleinberg*. Les courbes rouge et bleue (celles intitulées "algo") correspondent à des simulations utilisant des algorithmes

épidémiques pour déterminer les vues de chaque pair. Les vues sont initialisées de manière aléatoire uniforme et les mesures de routage sont effectuées après 20 cycles des algorithmes.

Nous remarquons d'une part que l'utilisation de notre algorithme biaisé (courbe bleue) améliore nettement les performances par rapport à l'utilisation de l'algorithme non-biaisé (courbe rouge). Par ailleurs, nous observons que dans les deux cas, uniforme et biaisé, les performances de routage obtenues par l'utilisation des algorithmes sont très proches de celles obtenues par une simulation parfaite. Cela n'est guère surprenant pour le cas aléatoire, mais est extrêmement intéressant pour le cas biaisé.

Suites à ces constatations, il est raisonnable de penser que notre algorithme biaisé impose une sélection des contacts distants similaire à une sélection exacte *à la Kleinberg*. Pour vérifier ce point, nous comparons les graphes obtenus par l'algorithme et par le calcul exact. Parmi les différents critères de comparaison testés (coefficient de clustering, répartition des degrés entrants, ...), un nous a semblé particulièrement pertinent : la répartition des raccourcis dans l'espace. La Figure 2.13 montre le résultat de nos analyses. Nous considérons un système composé de 20 000 pairs et des vues de taille 10. Les simulations ont été effectuées pendant 150 cycles et durant chaque cycle $e$ pairs sont échangés.

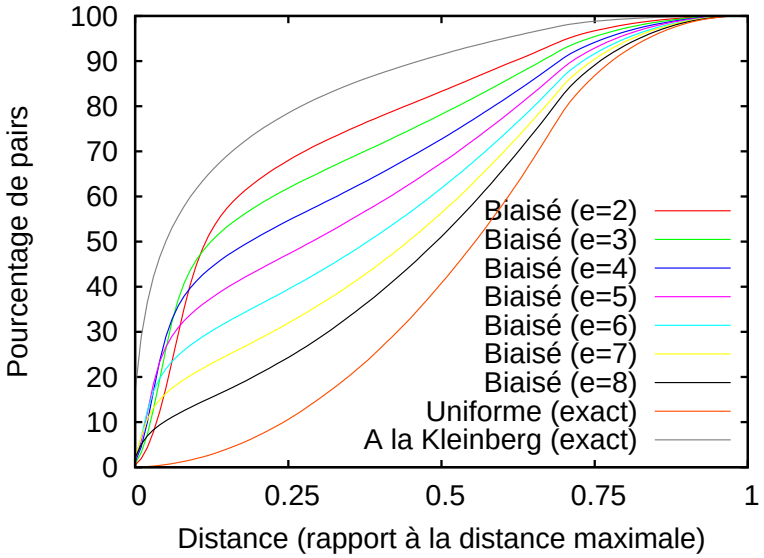



FIG. 2.13 – Répartitions des raccourcis dans l'espace

Les deux courbes extrêmes représentent les répartitions des contacts distants pour des sélections exactes des raccourcis suivant une méthode uniforme (courbe orange, à droite) et *à la Kleinberg* (courbe grise, à gauche). Les autres courbes montrent la répartitions des contacts distants après l'exécution de 150 cycles de notre algorithme biaisé. Suivant le nombre de pairs

échangés lors de chaque étape, la répartition change. Cela peut s'expliquer intuitivement d'après le mécanisme de l'algorithme : lorsqu'un pair $A$ transmet $e$ contacts à un pair $B$, cela signifie que $A$ a choisi de conserver $(10-e)$ contacts suivant la distribution harmonique ; les $e$ contacts restants correspondent à ceux reçus de $B$ sur lesquels $A$ n'a aucun contrôle. Ainsi plus $e$ est faible, plus $A$ choisit ses contacts distants et donc leur répartition est plus proche de celle d'une sélection parfaitement *à la Kleinberg*.

## 2.8 Conclusion

Ce chapitre nous a permis de présenter plusieurs résultats concernant le routage dans les réseaux petits-mondes. D'un coté, nous proposons une approche analytique permettant d'estimer précisément le coût du routage dans de tels réseaux. D'un autre coté, nous décrivons un algorithme épidémique réalisant un réseau petit-monde dont les caractéristiques sont très proches de celles d'un réseau théoriquement optimal. Les points importants à retenir sont les suivants :

- L'utilisation d'équations récursives constitue une méthode intermédiaire pour l'évaluation du routage d'un réseau petit-monde. Par rapport à des calculs de bornes asymptotiques, notre méthode est plus simple (pas de formules trop compliquées) et plus précise. Par rapport à des simulations, notre méthode est plus simple (peu de code à écrire) et plus rapide.
- Les équations récursives permettent d'obtenir des résultats quasiment identiques à ceux obtenus par simulation.
- La combinaison de deux algorithmes épidémiques autorise la création simple et efficace de réseaux petits-mondes de manière décentralisée.
- Il est possible de biaiser un protocole épidémique fournissant une liste de contacts aléatoires de manière à obtenir une liste de contacts répartis suivant une distribution proche de la distribution harmonique.

Ces travaux ne sont néanmoins pas une fin en soi et ouvrent la voie à de potentielles futures recherches :

- Il serait intéressant de chercher d'autres situations où la notion d'équations récursives peut être utilisée.
- Le protocole épidémique que nous introduisons mériterait d'être étudié par de plus amples simulations et par des expérimentations réelles. Une partie a déjà été effectuée et est présentée dans [11].

# Chapitre 3

# Exploration de graphes par des robots anonymes

Les résultats présentés dans ce chapitre correspondent principalement à des traductions des publications [5, 6] et des rapports techniques [7, 8].

## 3.1 Introduction

**Exploration de graphes par des robots** Le problème de l'exploration de graphes consiste à déplacer une ou plusieurs entités sur un graphe afin de visiter tous les sommets de ce graphe. Dans ce contexte particulier, les entités participantes sont plus fréquemment appelées des robots ; nous utilisons donc ce terme dans la suite de ce chapitre. L'exploration peut être perpétuelle si les robots doivent visiter chaque sommet infiniment souvent. Une telle exploration perpétuelle est requise lorsque les robots se déplacent dans le but de récupérer des données variables (dans le temps), ou dans l'optique de collecter des ressources dynamiques (dans l'espace). Si les sommets et les arêtes du graphe exploré sont distinctement identifiés, l'exploration est relativement facile à effectuer. Le problème de l'exploration devient plus intéressant lorsque le graphe est entièrement anonyme ; les sommets et les arêtes n'ont aucun identifiant. Dans ce cadre, plusieurs bornes ont déjà été démontrées. Elle concernent le temps minimal nécessaire pour visiter tous les sommets [32, 49], ou la taille mémoire minimale nécessaire pour réaliser l'exploration ; il est, par exemple, montré dans [40] qu'un robot a besoin d'au moins $O(D \log d)$ bits de mémoire locale pour explorer tout graphe de diamètre $D$ dont le degré est borné par $d$. Des résultats d'impossibilité d'exploration de graphes sont établis dans [63] lorsque les robots ne dispose que d'une mémoire bornée. Cependant une grande partie des résultats sur l'exploration ne considère qu'un seul robot. Ce n'est que récemment que, d'un point de vue pratique, l'exploration par plusieurs robots a été abordée [42]. Les motivations principales sont l'efficacité, la tolérance aux fautes, ou le dépassement d'impossibilités liées aux capacités d'un seul robot.

**Problème de l'exploration contrainte** Sous l'hypothèse de graphes anonymes sur lesquels les robots peuvent se déplacer de manière synchrone mais ne peuvent communiquer, nous considérons ici le problème de l'Exploration Perpétuelle Contrainte de Graphes ($EPCG$). Ce problème consiste à déplacer des robots sur un graphe afin que tous les robots visitent infiniment souvent chaque sommet du graphe en respectant deux contraintes d'exclusions mutuelles : au plus un robot sur chaque sommet, au plus un robot sur chaque arête. Ces contraintes permettent de modéliser un environnement réaliste, dans lequel il est souhaitable d'éviter les collisions. Des contraintes similaires sont étudiées dans [45] où les robots se déplacent sur une grille.

**Contributions** La première contribution de ce chapitre tient en une borne supérieure du nombre de robots pour lequel le problème $EPCG$ reste solvable. Cette borne est obtenue en considérant une puissance de calcul infinie et une synchronisation parfaite des robots ; un superviseur omniscient contrôle les robots. Considérant cela, la borne ne repose que sur la structure du graphe ; c'est une borne topologique. Par exemple, lorsque le graphe considéré est une chaîne, il est impossible pour deux robots de visiter tous les sommets en tenant compte des contraintes d'exclusions mutuelles : dans ce cas, la borne est de un robot. Notons que sous certaines hypothèses de synchronie et/ou de puissance de calcul, il se peut que cette borne ne soit pas atteignable par des algorithmes répartis.

La deuxième contribution de notre travail consiste en des algorithmes garantissant dans tous les cas (quels que soient le graphe et la situation initiale) les deux types d'exclusions mutuelles et résolvant l'exploration lorsque le nombre de robots le permet. Il se peut que l'exploration ne soit pas atteinte si le nombre de robots est trop important. Les résultats présentés dans ce cadre relèvent de trois paramètres ; $p$, $q$, et $\rho$. Le premier paramètre, $p$, correspond au nombre de sommets du graphe considéré. Le second, $q$, est relié à une caractéristique topologique du graphe que nous avons introduite. Le dernier, $\rho$, représente le rayon du champ de vision de chaque robot : un robot peut voir tous les sommets (et les éventuels robots situés sur ces sommets) situés à une distance inférieure à $\rho$ de sa position actuelle.

Les deux cas extrêmes ($\rho = 0$ et $\rho = \infty$) sont relativement faciles à étudier :

- Cas $\rho = 0$ : Le problème $EPCG$ ne peut pas être résolu si le graphe comporte plus d'un sommet, *i.e.* $p > 1$. L'hypothèse $\rho = 0$ implique que chaque robot n'a aucune vision du graphe et donc, intuitivement, aucun robot ne peut se déplacer sans risquer d'enfreindre une contrainte d'exclusion mutuelle.
- Cas $\rho = \infty$ : Le problème $EPCG$ peut être résolu si et seulement si le nombre de robots est inférieur à $p - q$, ce qui correspond précisément à la borne topologique démontrée.

Le cas particulier $\rho = 1$ est nettement plus coriace et en conséquence nous l'étudions plus en détail. Lorsque $q = 0$ le problème $EPCG$ est solvable avec au plus $p - 1$ robots (un seul sommet vide), et lorsque $q > 0$ avec au plus $p - q$ robots. C'est intéressant puisque cela prouve qu'avec une vision très limitée, les résultats obtenus sont très proches de ceux atteints avec une vision infinie. Cette dernière remarque n'est pertinente que du point de vue de la calculabilité ; du point de vue de la complexité les deux situations diffèrent.

**Plan du chapitre** Le chapitre comporte 7 sections. La Section 3.2 présente brièvement un état de l'art de l'exploration de graphes. La Section 3.3 décrit les deux modèles étudiés et formalise le problème de l'Exploration Perpétuelle Contrainte de Graphes ($EPCG$). La borne topologique sur le nombre de robots est énoncée et démontrée dans la Section 3.4. La Section 3.5 introduit une nouvelle notion, la $f$-solvabilité, et étudie la $f$-solvabilité du problème $EPCG$ lorsque les robots dispose d'une vision nulle ou inifinie. La Section 3.6, quant à elle analyse le cas particulier d'une vision limitée. Enfin la Section 3.7 conclut le chapitre en proposant quelques perspectives de recherche.

## 3.2 État de l'art

**Sur les hypothèses initiales** L'exploration de graphe consiste à faire visiter chaque sommet d'un graphe par des entités. Suivant le type de robots, le type de graphe, les primitives de communications, ..., le problème peut être solvable ou non. Ainsi, comme souvent dans les systèmes répartis (voir section 1.1), de gros efforts de recherche ont été faits pour trouver les hypothèses minimales permettant une telle exploration (par exemple dans [41]). Certains travaux considèrent des robots avec une mémoire bornée et une communication directe entre les robots [4]. D'autres recherches partent du principe que les robots possèdent la capacité de visualiser la position des autres robots [36]. Enfin d'autres études analysent si la connaissance initiale du graphe (par les robots) permet de diminuer la complexité de l'exploration [62].

**Sur le type d'exploration de graphe** Les problèmes d'exploration sont séparés en deux grandes familles ; d'un côté l'exploration perpétuelle [32], où les robots doivent parcourir le graphe infiniment souvent ; d'un autre coté l'exploration avec arrêt où les robots doivent s'arrêter après avoir parcouru une seule fois le graphe [41]. Certains travaux considèrent l'exploration de graphes par un seul robot [1, 4, 40], tandis que d'autres étudient l'exploration coopérative par un ensemble de robots [10, 36, 45]. Il est, par exemple, démontré dans [36] que le nombre minimum requis de robots pour parvenir à l'exploration d'un anneau de taille $n$ est en $O(\log n)$ lorsque cette exploration souhaite être obtenue par des robots asynchrones sans-mémoire.

Des bornes inférieures ont aussi été données pour le problème de l'exploration perpétuelle dans [32, 43]. Ces bornes concernent la durée nécessaire pour un robot pour (re)visiter entièrement le graphe, en considérant certaines hypothèses sur les robots ou sur le graphe. De manière différente, nous proposons ici une borne supérieure sur le nombre de robots permettant l'exploration tout en évitant les collisions.

**Sur l'exploration contrainte de graphe** Le problème de l'exploration perpétuelle contrainte (formellement défini section 3.3.3) considère deux contraintes d'exclusions mutuelles ; une sur les sommets, une sur les arêtes. Ces deux contraintes ont déjà été définies et utilisées dans [45] où les graphes considérés sont des grilles. Cependant le problème étudié dans [45] est différent : chaque robot possède une destination précise sur la grille qu'il doit atteindre (chaque destination est différente). Le papier établit alors une borne inférieure sur le nombre

de rondes nécessaires pour résoudre ce problème, ainsi qu'un algorithme atteignant cette borne.

Le problème de collisions de robots apparaît également dans [72], où les auteurs proposent un algorithme permettant à des robots mobiles d'éviter les collisions. Néanmoins le contexte est légèrement différent puisque, dans ces travaux, les robots se déplacent sur un plan (espace continu) au lieu d'un graphe (espace discret).

## 3.3 Modèle(s) et problème

Dans ce chapitre, nous définissons et utilisons deux modèles légèrement différents. Dans les deux cas le système est synchrone, le graphe est anonyme. Le premier modèle est plus générique car il n'impose aucune restriction sur le type de graphe, alors que le second ne considère que les graphes qui sont des grilles partielles.

Le premier modèle nous permet d'obtenir une borne topologique la plus forte possible, tandis que le second, grâce aux restrictions, autorise la conception d'algorithmes. La borne topologique obtenue dans le premier modèle restant bien évidemment valide dans le second.

### 3.3.1 Modèle générique

Le système est constitué d'un graphe fini, connecté, non-orienté $G = (S, A)$ et d'un ensemble fini $R$ de robots tel que $|R| \leq |S|$. Le graphe $G$ est anonyme ; les sommets et les arêtes n'ont pas d'identité.

Le système est synchrone ; les robots partagent une horloge commune et le temps est divisé en rondes [48]. Durant chaque ronde, un robot peut (i) se déplacer vers un sommet adjacent, (ii) rester sur le même sommet. Le mouvement vers un autre sommet occupe toute la ronde (un seul déplacement par ronde). Une configuration du système est une répartition des robots sur le graphe telle qu'il y ait au plus un robot par sommet.

Ce modèle sert à obtenir la borne topologique (donc ne dépendant que du graphe), il n'est pas nécessaire de préciser les caractéristiques des robots (puissance de calcul, quantité de mémoire, vision, ...). Nous supposons au contraire qu'un démon omniscient contrôle les déplacements des robots. La borne obtenue ne dépend alors que des contraintes d'exclusions mutuelles que ne doivent pas enfreindre les robots.

### 3.3.2 Modèle restreint

**Le graphe** Toutes les caractéristiques du modèle précédent demeurent dans le modèle restreint. Cependant sans ajout d'hypothèses supplémentaires, il est impossible de concevoir des algorithmes à cause des contraintes d'anonymat et d'exclusions mutuelles. Nous limitons alors le type de graphes sur lesquels évoluent les robots : dans cette variante du modèle nous n'autorisons que les graphes qui sont des grilles partielles (connectées). Une grille partielle étant définie à partir d'une grille planaire sur laquelle des sommets et/ou des arêtes ont été supprimés ; la Figure 3.1 présente un exemple de grille partielle.

Par ailleurs afin de briser la symétrie nous supposons aussi que ce le système est maintenant orienté par une boussole ; chaque robot peut distinguer les quatre directions Nord, Est, Sud, et Ouest. Ces directions seront utilisées par les algorithmes proposés en sections 3.5 et 3.6.

**Les robots** Contrairement au modèle générique, le but de ce modèle est de permettre la conception d'algorithmes. Il est donc nécessaire de préciser les caractéristiques des robots. Tout comme le graphe, les robots sont eux aussi anonymes ; les identifiants utilisés par la suite ne sont là que pour faciliter les explications, mais ne sont pas accessibles par les algorithmes. Les robots disposent d'une mémoire et d'une capacité de calcul infini.

**Définition 3.1** *Étant donné un robot $a$ et une ronde $r$, $S(a, r)$ correspond au sommet occupé par le robot $a$ au début de la ronde $r$.*

Chaque robot dispose d'une vision partielle du graphe caractérisée par son rayon de vision $\rho \in \mathbb{N}$. Dans le cas où $\rho > 0$, cela signifie qu'au début d'une ronde $r$, un robot $a$ situé sur le sommet $S(a, r)$ voit tous les sommets et arêtes à distance au plus $\rho$ du sommet $S(a, r)$[1]. Il voit aussi si ces sommets sont occupés ou non par des robots, néanmoins sans pouvoir distinguer les différents robots les occupant. Un exemple de vision avec un rayon $\rho = 2$ est représenté sur la Figure 3.1 pour le robot $b$.

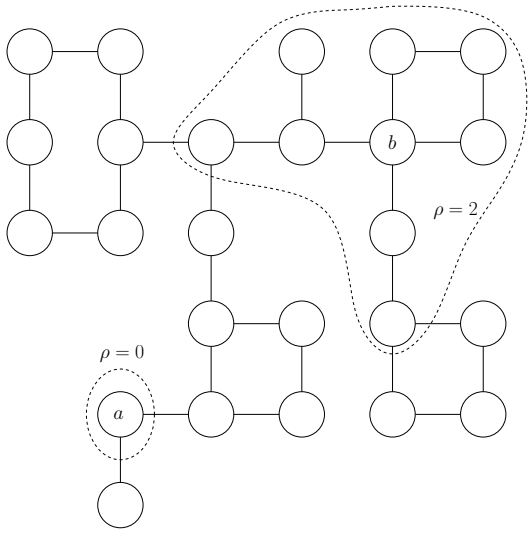



FIG. 3.1 – Une grille partielle et deux rayons de vision.

Avec un léger abus de notation, nous considérons que le cas particulier d'un rayon de vision $\rho = 0$ permet à un robot de voir les arêtes incidentes à sa position actuelle. Sur la

[1]Le robot $a$ considéré ne voit pas les éventuelles arêtes des sommets situés exactement à distance $\rho$ de $S(a, r)$.

Figure 3.1, le robot $a$ sait que le sommet sur lequel il se situe possède une arête vers l'Est et une vers le Sud. La différence fondamentale avec un rayon de vision $\rho = 1$ réside dans la non-connaissance de l'occupation des sommets adjacents.

### 3.3.3 Problème

Le problème de l'Exploration Perpétuelle Contrainte de Graphes ($EPCG$) est formellement défini par les trois propriétés suivantes :

- Exploration Perpétuelle. Tout robot $a$ visite infiniment souvent tout sommet $s$ :

$$\forall s \in S \quad \forall a \in R \quad \{r \mid S(a,r) = s\} \text{ est infini.}$$

- Exclusion Mutuelle sur les Sommets. Au début de chaque ronde, deux robots n'occupent pas le même sommet :

$$\forall r \geq 0 \quad \forall (a,b) \in R \times R \quad (a \neq b) \Rightarrow \big(S(a,r) \neq S(b,r)\big).$$

- Exclusion Mutuelle sur les Arêtes. Lors d'une ronde, deux robots ne peuvent se déplacer en utilisant la même arête ; ils ne peuvent échanger leurs positions :

$$\forall r \geq 0 \quad \forall (a,b) \in R \times R \quad (a \neq b) \Rightarrow \big[\big(S(a,r+1) = S(b,r)\big) \Rightarrow \big(S(b,r+1) \neq S(a,r)\big)\big].$$

## 3.4 Borne topologique

Dans cette section nous montrons qu'il existe une borne topologique (pour chaque graphe) sur le nombre de robots au delà de laquelle le problème $EPCG$ ne peut être résolu. Il existe une borne triviale correspondant au nombre de sommets du graphe considéré ; cependant cette borne n'est pas toujours atteignable même en supposant qu'un démon omniscient contrôle tous les robots. Considérons par exemple un graphe composé de quatre sommets formant une ligne ; il est évident qu'il sera impossible d'explorer entièrement ce graphe avec plus d'un robot, car deux robots n'ont aucune possibilité de se croiser (à cause des exclusions mutuelles).

Nous calculons ici une borne stricte pour chaque graphe. Étant donné un graphe $G$, $R$ robots et la borne $B$ correspondante au graphe :

- Si $R > B$, aucun démon omniscient ne peut résoudre le problème $EPCG$,
- Si $R \leq B$, il existe un démon omniscient résolvant le problème $EPCG$.

Dans le cas où $R \leq B$, l'existence d'un algorithme n'est pas garanti. En effet contrairement à un démon omniscient, un algorithme est exécuté localement par chaque robot. Le manque d'information globale (dû à un faible rayon de vision par exemple) peut empêcher d'atteindre la borne $B$.

### 3.4.1 Définitions préliminaires

Nous raisonnons ici dans le modèle générique ; nous considérons donc un graphe fini, connecté, non-orienté $G = (S, A)$ où toute paire de sommets est connectée par au plus une arête (pas de multiplicité). Un sommet $s$ est appelé *feuille* s'il n'existe qu'un seul autre sommet $s'$ tel que $(s, s') \in A$. Le *degré* $d$ d'un sommet $s$ correspond au nombre d'arêtes incidentes à $s$, c'est-à-dire au cardinal de l'ensemble $\{s', (s, s') \in A\}$. Un *isthme* est une arête dont la suppression entraîne une déconnexion du graphe ; nous utiliserons la terminologie *graphe sans isthme* pour désigner un graphe dont aucune arête n'est un isthme. Un chemin entre un sommet $s$ et un sommet $s'$ est appelé *élémentaire* s'il ne traverse pas deux fois un même sommet.

Un graphe $G' = (S', A')$ est appelé un *sous-graphe* de $G = (S, A)$ si $S' \subseteq S$ et $A' \subseteq A$. Reciproquement, nous disons que $G = (S, A)$ est un *sur-graphe* de $G' = (S', A')$. Un sous-graphe *non-trivial* désigne un sous-graphe qui possède au moins deux sommets. Le sous graphe $G' = (S', A')$ de $G = (S, A)$ est dit *induit* par son ensemble de sommets $S'$ si $A'$ contient toutes les arêtes de $G$ dont les deux extrémités sont dans $S'$ (formellement si $A' = \{(s, s') \in A, s \in S' \text{ et } s' \in S'\}$. Tous les sous-graphes considérés par la suite sont induits, le terme induit est donc omis pour alléger l'écriture.

Un sous-graphe $G'$ d'un graphe $G$ est dit *maximal* pour une propriété $P$ si $G'$ satisfait $P$ et qu'aucun graphe $G''$, sur-graphe de $G'$ et sous-graphe de $G$ (intuitivement $G' < G'' < G$), ne vérifie $P$.

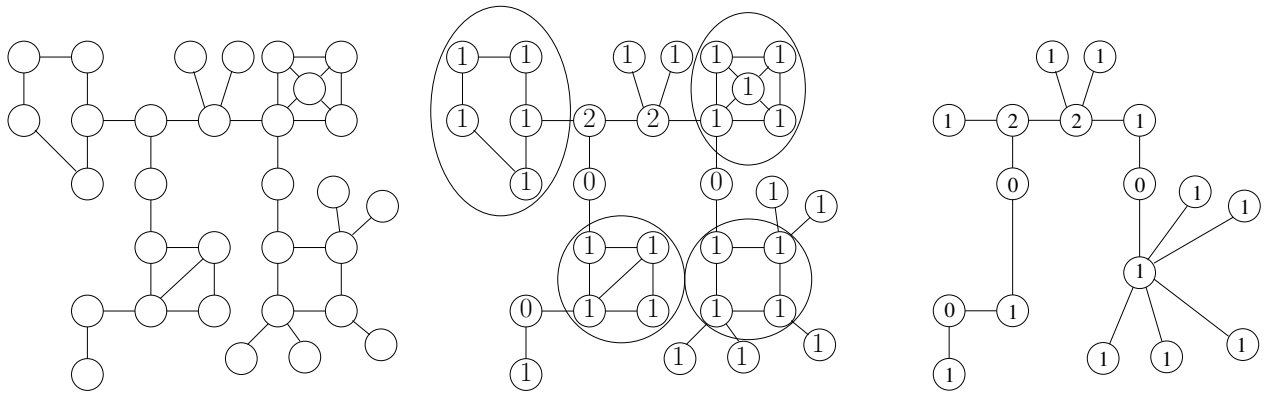
### 3.4.2 Arbre de mobilité

Afin de calculer la borne topologique propre à chaque graphe, nous avons besoin d'introduire une nouvelle notion, celle d'*arbre de mobilité*. L'arbre de mobilité est une réduction du graphe initial qui (informellement) conserve les goulots d'étranglement du graphe. À partir de l'arbre de mobilité, nous pouvons facilement déduire la borne topologique du graphe. L'arbre de mobilité est défini par sa méthode de construction :

**Définition 3.2** *(Arbre de mobilité) L'arbre de mobilité associé à un graphe $G = (S, A)$ est l'unique arbre étiqueté $G' = (S', E')$ dérivé de $G$ par la réduction suivante :*

1. *Étiquetage initial. Chaque sommet $s \in S$ est étiqueté de la manière suivante :*
   - 0 *si $s$ est de degré deux et si les deux arêtes partant de $s$ sont des isthmes dans $G$.*
   - 1 *si $s$ est une feuille ou si $s$ appartient à un cycle élementaire de $G$.*
   -
   - 2 *dans les cas restants (degré supérieur à deux et toutes les arêtes partant de $s$ sont des isthmes).*
2. *Compression. Tout sous-graphe maximal non trivial et sans isthme de $G$ est compressé en un seul sommet étiqueté* 1. *(Tous les sommets fusionnés sont nécessairement étiquetés* 1.*)*

La Figure 3.2 propose un exemple de réduction en arbre de mobilité. Tout d'abord le graphe $G$ est dessiné sur la Figure 3.2(a). Puis le résultat de l'étiquetage initial apparaît sur la Figure 3.2(b) ; les sous-graphes maximaux non triviaux et sans isthme sont entourés sur ce même dessin. Enfin l'arbre de mobilité, après compression, est représenté sur la Figure 3.2(c).



(a) Graphe $G$ (b) Étiquetage initial de $G$ (c) Arbre de mobilité étiqueté $G'$

FIG. 3.2 – Un graphe $G$ et son arbre de mobilité associé $G'$

L'arbre de mobilité associé à un graphe $G$ permet de faire ressortir les caractéristiques cruciales de sa topologie, dans l'optique de résoudre le problème $EPCG$. En premier lieu, l'arbre de mobilité fait disparaître (en les fusionnant en un seul sommet) les sous-graphes de sommets étiquetés 1 car ceux-ci ne posent aucun soucis pour résoudre le problème $EPCG$. En effet, même entièrement remplis de robots, ces sous-graphes peuvent encore être explorés grâce à des cycles élémentaires (voir section 3.5) en déplaçant tous les robots d'un cycle simultanément. En second lieu, l'arbre de mobilité fait apparaître les portions de graphes que chaque robot devra traverser d'un seul coup ; ce sont les arêtes reliant les différents sous-graphes étiquetés 1. Plus formellement ces chemins sont définis comme étant des chemins d'exclusion mutuelle ci-dessous.

**Définition 3.3** *(Chemin d'exclusion mutuelle) Soit $\mathcal{C}$ un chemin $(s, s^1, s^2 \dots s^m, s')$ d'un arbre de mobilité. $\mathcal{C}$ est un chemin d'exclusion mutuelle si et seulement si :*
- *les étiquettes de $s$ et $s'$ sont différentes de 0,*
- *les sommets intermédiaires $s^h$, $1 \leq h \leq m$ (s'ils existent) sont tous étiquetés par 0.*

À titre d'exemple, la Figure 3.3 montre trois chemins d'exclusion mutuelle $\mathcal{C}_1$, $\mathcal{C}_2$, et $\mathcal{C}_3$.

**Définition 3.4** *(Longueur d'un chemin d'exclusion mutuelle) Dans un arbre de mobilité, nous définissons la longueur d'un chemin d'exclusion mutuelle entre deux sommets $v$ et $v'$ comme étant le nombre d'arêtes entre $v$ et $v'$ sur ce chemin (i.e. la longueur usuelle) auquel on ajoute le nombre d'extrémités du chemin étiquetées par 2.*

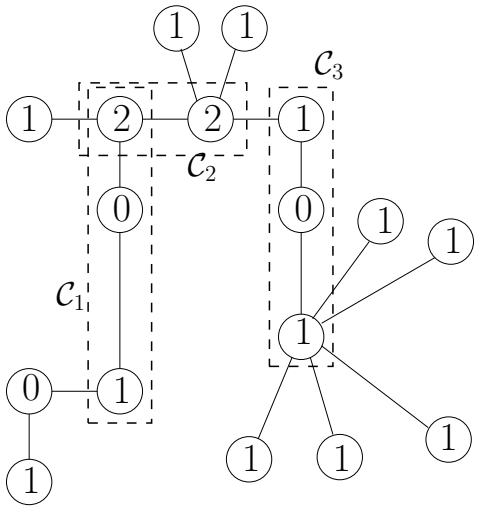



FIG. 3.3 – Chemin d'exclusion mutuelle dans un arbre de mobilité

Utilisant la définition précédente, le chemin d'exclusion mutuelle $\mathcal{C}_1$ de la Figure 3.3 est de longueur $2 + 1 = 3$. Les deux autres chemins $\mathcal{C}_2$ et $\mathcal{C}_3$ sont quant à eux, respectivement, de longueur $1 + 2 = 3$ et $2 + 0 = 2$.

Nous allons voir que la longueur d'un chemin d'exclusion mutuelle correspond au nombre de sommets qui doivent être initialement libres (*i.e.* sans robot) pour permettre la résolution du problème $EPCG$. Cette remarque nous permet de catégoriser les graphes en différentes familles $\mathcal{G}(p, q)$.

**Définition 3.5** *Pour tout $p > 0$ et $q \geq 0$, nous définissons la famille de graphes $\mathcal{G}(p, q)$ comme étant l'ensemble des graphes $G$ tels que :*

- *$G$ possède $p$ sommets,*
- *$q$ est la longueur du plus long chemin d'exclusion mutuelle de l'arbre de mobilité associé à $G$.*

### 3.4.3 Borne supérieure

Grâce aux définitions précédentes, il est maintenant possible de définir formellement la borne topologique du nombre de robots au delà de laquelle le problème $EPCG$ n'est pas solvable. Le théorème correspondant et sa preuve suivent ci-dessous. Pour être cohérent avec ce qui est écrit dans l'introduction, à savoir que cette borne est stricte, la section 3.5 propose un algorithme qui atteint cette borne lorsque le rayon de vision est infini. Bien que cet algorithme soit défini pour le modèle restreint, un démon omniscient peut exécuter le même algorithme dans le premier modèle.

**Théorème 3.6** *Soient $p > 0$ et $q \geq 0$, soit $G$ un graphe de la famille $\mathcal{G}(p, q)$. Il est impossible de résoudre le problème $EPCG$ sur le graphe $G$ pour n'importe quelle configuration initiale possédant strictement plus de $p - q$ robots.*

**Preuve** La preuve est par contradiction. Supposons qu'il existe une stratégie $S$ d'un démon omniscient qui permet de résoudre le problème $EPCG$ pour un graphe $G$ appartenant à la famille $\mathcal{G}(p,q)$ avec une configuration initiale comprenant $p-q+1$ robots. Nous ne supposons aucune restriction sur $S$ ; il se pourrait par exemple que $S$ ne soit pas localement calculable par chaque robot. La preuve étudie uniquement les situations avec $p-q+1$ robots ; en effet s'il n'existe pas de stratégie pour $p-q+1$ robots il n'en existe trivialement pas non plus pour un nombre supérieur.

Dans le cas où $q = 0$ la contradiction est évidente ; il est impossible de placer $p+1$ robots sur un graphe à $p$ sommets sans enfreindre la propriété d'exclusion mutuelle sur les sommets. La suite de la preuve suppose donc $q > 0$. Toute configuration initiale comporte $q-1$ sommet(s) vide(s), puisque $p-q+1$ robots sont présents.

D'après la définition de $\mathcal{G}(p,q)$, $q$ désigne la longueur du plus long chemin d'exclusion mutuelle dans l'arbre de mobilité associé à $G$. Considérons alors un tel chemin $\mathcal{C}$ de longueur $q$ dans $G$ et appelons $x$ et $y$ ses deux extrémités. Suivant les étiquettes de $x$ et $y$, nous distinguons les trois cas suivants :

– $x$ et $y$ sont étiquetés 1. De part la définition 3.4, le chemin $\mathcal{C}$ est composé de $q+1$ sommets. La situation est représentée sur la Figure 3.4. Soient $G_x$ et $G_y$ les deux sous-graphes de $G$ incluant respectivement $x$ et $y$, maximaux vis-à-vis de la propriété suivante : tout chemin élémentaire de $G$ allant d'un sommet $s_x \in G_x$ vers un sommet $s_y \in G_y$ contient les deux sommets $x$ et $y$.

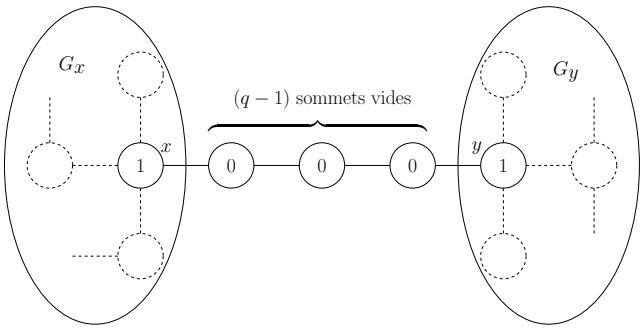



FIG. 3.4 – Chemin d'exclusion mutuelle étiqueté 1 vers 1, de longueur $q = 4$

Considérons une configuration initiale où les $q-1$ sommets vides sont les sommets du chemin $\mathcal{C}$ privé de $x$ et $y$. Ainsi les $p-q+1$ robots occupent entièrement les deux sous-graphes $G_x$ et $G_y$. Puisqu'il n'y a que $q-1$ sommets sans robot (ceux de $\mathcal{C} \setminus \{x,y\}$) et que tout chemin de $G_x$ vers $G_y$ inclut $\mathcal{C}$, il s'ensuit qu'aucun robot d'un sommet $s_x \in G_x$ ne peut atteindre un sommet $v_y \in G_y$ sans violer l'une ou l'autre des propriété d'exclusion mutuelle. Cela prouve le théorème pour ce premier cas.

– $x$ et $y$ sont étiquetés 2 et 1 respectivement. D'après la définition 3.4, le chemin $\mathcal{C}$ est composé de $q$ sommets. La situation est représentée sur la Figure 3.5. Soit $G_y$ le sous-graphe défini comme précédemment et soient $G_x^1$, $G_x^2$ (et éventuellement $G_x^3$, $G_x^4$, ...) les sous-graphes connexes de $G$ ne contenant pas $x$ et maximaux vis-à-vis de la propriété suivante : tout chemin élémentaire de $G$ allant d'un sommet $s_x \in G_x^1 \cup G_x^1 \cup \ldots$ vers

un sommet $s_y \in G_y$ contient les deux sommets $x$ et $y$.

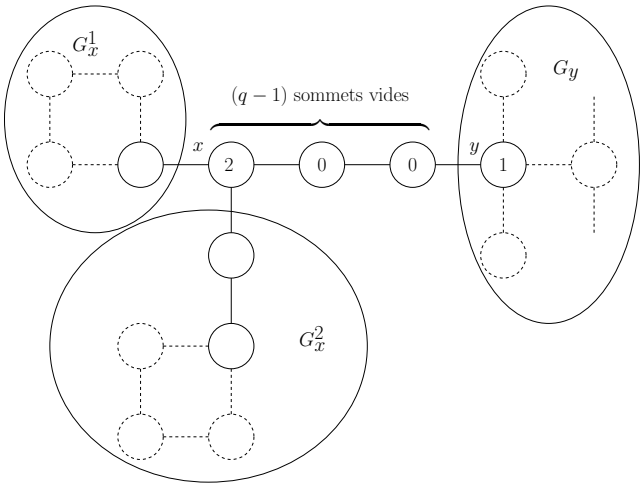



FIG. 3.5 – Chemin d'exclusion mutuelle étiqueté 1 vers 2, de longueur $q = 4$

Comme précédemment, considérons une configuration initiale où les $q-1$ sommets vides sont les sommets du chemin $\mathcal{C}$ privé de $y$. Ainsi les $p-q+1$ robots occupent entièrement les sous-graphes $G_x^1$, $G_x^2$, $G_x^3$, ..., et $G_y$. De même que pour le premier cas, il n'y a pas assez de sommets libres pour permettre à un robot situé sur un sommet $s_x \in G_x^1 \cup G_x^2 \cup \ldots$ d'atteindre un sommet $s_y \in G_y$. Cela prouve le théorème pour ce second cas.

– $x$ et $y$ sont étiquetés 2. De part la définition 3.4, le chemin $\mathcal{C}$ est composé de $q-1$ sommets. La situation est représentée sur la Figure 3.6. Soient $G_x^1$, $G_x^2$ (et éventuellement $G_x^3$, $G_x^4$, ...) les sous-graphes de $G$ définis comme dans le cas précédent et soient $G_y^1$, $G_y^2$ (et éventuellement $G_y^3$, $G_y^4$, ...) leur équivalents du coté du sommet $y$.

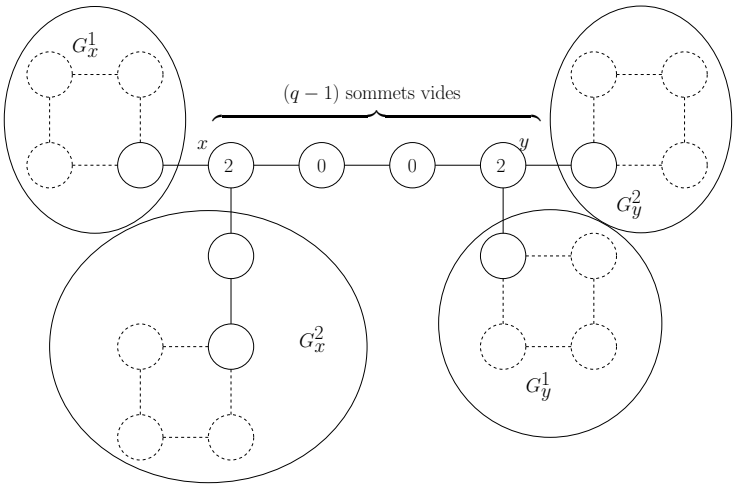



FIG. 3.6 – Chemin d'exclusion mutuelle étiqueté 2 vers 2, de longueur $q = 5$

Considérons de nouveau une configuration initiale où les $q-1$ sommets vides sont les sommets du chemin $\mathcal{C}$ (incluant cette fois $x$ et $y$). Ainsi les $p-q+1$ robots occupent entièrement les sous-graphes $G_x^1$, $G_x^2$, $G_x^3$, ..., et $G_y^1$, $G_y^2$, $G_y^3$, .... L'absence de robots sur les sommets de $\mathcal{C}$ n'est néanmoins pas suffisant pour permettre à un robot situé sur un sommet $s_x \in G_x^1 \cup G_x^2 \cup \ldots$ d'atteindre un sommet $s_y \in G_y^1 \cup G_y^2 \cup \ldots$. Cela prouve le théorème pour ce dernier cas.

□

## 3.5 Algorithmes d'exploration perpétuelle

Dans la section précédente, nous avons montré que pour tout graphe, il existait une borne sur le nombre de robots pouvant simultanément explorer ce graphe. Comme mentionné auparavant, cette borne n'est néanmoins que topologique ; rien ne garantit qu'il existe effectivement des algorithmes permettant à autant de robots d'explorer un graphe. Dans cette section, nous proposons des algorithmes résolvant le problème de l'Exploration Perpétuelle Contrainte de Graphes ($EPCG$) défini section 3.3.3.

**Rappels** Dans toute la suite de ce chapitre, nous considérons le modèle restreint défini à la section 3.3.2 dans lequel les graphes sont des grilles partielles. Par ailleurs nous rappelons que dans ce modèle, les robots sont munis d'une boussole leur indiquant les points cardinaux[2]. Par ailleurs un paramètre $\rho$ compris entre 0 et $+\infty$ décrit le rayon de vision de chaque robot du système.

### 3.5.1 $f$-solvabilité

Dans ce cadre là, nous nous sommes intéressés à une certaine catégorie d'algorithmes, que nous décrivons comme génériques. Ils sont génériques dans le sens où les algorithmes que nous concevons ne sont pas destinés à un graphe en particulier ; ils doivent "fonctionner" quel que soit le graphe à explorer. De manière équivalente, cela signifie que les robots ne connaissent pas (initialement) le graphe sur lequel ils se déplacent.

La connaissance initiale du graphe simplifie le problème de l'exploration : considérons, par exemple, un graphe "carré" composé de 4 sommets formant un carré. Considérons un rayon de vision $\rho = 0$, *i.e.* lorsqu'un robot ne sait même pas si un sommet adjacent est occupé ou non. Si les robots possèdent la connaissance initiale du graphe, ils peuvent de manière déterministe s'accorder sur un sens de rotation ; chaque robot se déplace successivement sur le sommet suivant. Un tel algorithme permet à tous les robots (quelque soit leur nombre) d'explorer perpétuellement le graphe. Si les robots n'ont pas la connaissance du graphe, ils ne

[2]La connaissance d'une seule direction (typiquement le nord) ne suffit pas à distinguer les voisins d'un sommet, il est nécessaire de connaître au moins deux directions (non symétriques) pour posséder la notion de chiralité.

peuvent pas prendre le risque de se déplacer sans violer les contraintes d'exclusions mutuelles comme nous le montrons ultérieurement à la Section 3.5.2.

À partir de la classification des graphes en classes $\mathcal{G}(-,-)$, nous définissons formellement la notion de $f$-solvabilité ci-dessous qui caractérise précisément la généricité des algorithmes étudiés.

**Définition 3.7** *($f$-solvabilité) Soit $f$ une fonction de l'ensemble des classes $\mathcal{G}(-,-)$ vers les entiers naturels $\mathbb{N}$, i.e. $f(p,q) = \{0,1,\ldots\}$[3]. Soit $\rho$ un rayon de vision et $A$ un algorithme d'exploration de graphes. Nous dirons que $A$ $f$-résout le problème $EPCG$ pour une vision $\rho$ si (1) $A$ garantit toujours les deux propriétés d'exclusions mutuelles (sur les sommets et sur les arêtes), quelque soit le graphe et le nombre de robots, et (2) $A$ résout le problème $EPCG$ lorsqu'il est utilisé sur un graphe de la classe $\mathcal{G}(p,q)$ avec au plus $f(p,q)$ robots.*

**Remarques** L'introduction de cette notion de $f$-solvabilité force en effet la généricité (vis-à-vis des graphes) puisqu'un algorithme doit garantir certaines propriétés quelque soit le graphe parcouru. Nous avons délibérément choisi d'assurer les contraintes d'exclusions mutuelles dans tous les cas ; c'est un choix qui nous semblait important puisque, dans le cas d'un système réel, il est crucial de garantir au moins la sûreté, même si le problème ne peut être résolu.

Le Théorème 3.6 implique que, pour toute valeur de $\rho$, il n'existe aucun algorithme $A$ qui $f$-résout le problème $EPCG$ pour $\rho$ si il existe une classe $\mathcal{G}(p,q)$ telle que $f(p,q) > p - q$.

### 3.5.2 Rayon de vision $\rho = 0$

Cette section montre que lorsque $\rho = 0$, le problème $EPCG$ ne peut être $f$-résolu que pour un très petit nombre de robots ; 0 ou 1 selon les classes de graphes. Plus formellement, le théorème s'énonce ainsi :

**Théorème 3.8** *Il existe un algorithme $A$ qui $f$-résout le problème $EPCG$ pour une vision $\rho = 0$ si et seulement si (1) $f(1,0) \leq 1$ et (2) $f(p,q) = 0$ lorsque $(p,q) \neq (1,0)$.*

**Preuve** L'implication $\Leftarrow$ est triviale. Il suffit de prendre l'algorithme qui maintient immobile chaque robot. Cet algorithme n'enfreint pas les contraintes d'exclusions mutuelles et résout l'exploration perpétuelle lorsqu'il n'y a qu'un seul robot sur un graphe réduit à un sommet (le seul graphe de la classe $\mathcal{G}(1,0)$).

L'implication $\Rightarrow$ demande plus de travail et est prouvée par contradiction. Supposons qu'il existe un algorithme $A$ qui $f$-résout le problème $EPCG$ pour $\rho = 0$ telle que $f$ ne vérifie pas les hypothèses du théorème. Nous distinguons trois cas :

- Cas $f(1,0) > 1$. Il est impossible de placer plus d'un robot sur un graphe ne possédant qu'un sommet sans violer la contrainte d'exclusion mutuelle sur les sommets. $f(1,0) > 1$ est donc impossible, ce qui prouve l'implication pour ce cas.

[3]Nous considérons que la fonction $f$ n'est définie que pour les paires $(p,q)$ pour lesquelles la classe $\mathcal{G}(p,q)$ n'est pas vide. En effet certaines classes sont vides, comme la classe $\mathcal{G}(2,4)$ par exemple.

– Cas $f(p,q) > 0$ avec $p > 0$ et $q > 0$. Soit $G$ un graphe de $\mathcal{G}(p,q)$. Par la définition 3.7 de la $f$-solvabilité, $A$ permet en particulier à un robot $\alpha$ d'explorer perpétuellement le graphe $G$ lorsqu'il est seul sur le graphe. Puisque $\rho = 0$, le robot $\alpha$ se comporte de la même manière lorsque le graphe est initialisé avec $p$ robots ou bien lorsque $\alpha$ est le seul robot du graphe.
L'hypothèse $q > 0$ implique l'existence d'un isthme dans $G$. La traversée de cet isthme par $\alpha$ est nécessaire pour résoudre l'exploration du graphe lorsque $\alpha$ est seul. Par contre cette même traversée de l'isthme entraîne obligatoirement une violation de l'exclusion mutuelle sur les sommets lorsque le graphe est entièrement rempli de robots. $f(p,q) > 0$ avec $p > 0$ et $q > 0$ est donc impossible, ce qui prouve l'implication pour ce cas.
– Cas $f(p,q) > 0$ avec $p > 0$ et $q = 0$. Soit $G$ un graphe de $\mathcal{G}(p,q)$. Le raisonnement précédent n'est plus valable car $G$ ne possède pas d'isthme ; la contradiction nécessite cette fois l'introduction d'un autre graphe $G'$. Comme pour le cas précédent, $A$ permet à un robot $\alpha$ d'explorer perpétuellement le graphe $G$ lorsqu'il est seul sur le graphe et les déplacements de $\alpha$ induits par $A$ sont les mêmes si le graphe est initialement rempli de $p$ robots.
Considérons la situation où le graphe est entièrement rempli par $p$ robots, et étudions la première ronde $r$ où un robot se déplace ; une telle ronde existe puisque le robot $\alpha$ se déplace. Comme $G$ est entièrement rempli de robots, il y a plusieurs robots qui se déplacent lors de la ronde $r$. Plus précisément, il existe au moins un cycle $C$ de $G$ sur lequel les robots se déplacent simultanément.
Soit $s$ un sommet appartenant au cycle $C$. Nous définissons $G'$ comme étant le sous-graphe de $G$ dans lequel le sommet $s$ a été supprimé. Soit $s_1$ et $s_2$ les deux sommets adjacents à $s$ dans le cycle $C$. Puisque $\rho = 0$, les robots situés sur $C \setminus \{s, s_1, s_2\}$[4] ne différencient pas $G$ de $G'$ et donc se déplacent lors de la ronde $r$. Comme dans le paragraphe précédent, afin de garantir l'exclusion mutuelle sur les sommets, cela implique qu'il existe un cycle $C'$ sur lequel tous les robots se déplacent simultanément. Soit $G''$ le sous-graphe de $G'$ dans lequel un sommet $s'$ de $C'$ a été supprimé. Le même raisonnement peut ainsi être appliqué récursivement. Le graphe initial $G$ étant fini, on aboutit à un graphe $G^\omega$ qui ne possède pas de cycle, ce qui entraîne une contradiction. $f(p,q) > 0$ avec $p > 0$ et $q = 0$ est donc impossible, ce qui prouve l'implication pour ce cas et conclut la preuve du théorème.

□

### 3.5.3 Rayon de vision $\rho = \infty$

Cette section montre que lorsque le rayon de vision est infini, *i.e.* les robots ont une vision globale du graphe à tout instant, la borne topologique du nombre maximal de robots est atteignable. Formellement :

[4]Puisque les graphes sont restreints à des grilles partielles, les cycles sont au moins de longueur quatre ; cet ensemble de sommets n'est donc pas vide.

**Théorème 3.9** *Il existe un algorithme $A$ qui $f$-résout le problème $EPCG$ pour une vision $\rho = \infty$ si et seulement si $f(p,q) \leq p - q$ pour toute paire $(p,q)$.*

**Preuve** L'implication $\Rightarrow$ est immédiate. Elle découle du théorème 3.6. Il reste à montrer l'implication opposée $\Leftarrow$ ; pour ce faire nous proposons ci-dessous un algorithme qui $f$-résout le problème $EPCG$ pour une vision $\rho = \infty$ lorsque $f(p,q) \leq p - q$ pour toute paire $(p,q)$. Grâce à leur vision globale, les robots connaissent directement le nombre de robots présents sur le graphe. Si ce nombre est supérieur à $p - q$ les robots demeurent immobiles, car il est impossible d'explorer le graphe (borne topologique). Si il y a au plus $p-q$ robots, l'exploration est possible et les mouvements à effectuer sont décrits ci-dessous.

Pour des raisons de simplicité, l'algorithme proposé à la Figure 3.7 est écrit du point de vue d'un observateur extérieur ; il ne correspond pas directement à l'algorithme qu'exécutent les robots. Cependant cela n'ajoute aucune restriction puisque nous considérons ici le cas d'un rayon de vision $\rho$ infini ; tous les robots ont une connaissance globale du système et, par conséquent, chacun peut localement agir comme un observateur extérieur et se déplacer lorsque l'algorithme l'exige. Cela correspond d'une certaine manière au démon omniscient décrit dans la Section 3.4 prouvant la borne topologique.

```
Algorithme explore_∞ :
(01) Obtenir une vue globale de la grille partielle ;
(02) Associer déterministiquement un numéro à chaque robot (de 1 à R le nombre de robots) ;
(03) Calculer le cycle canonique orienté C incluant tous les sommets ;
(04) tant que vrai faire
(05)     pour x de 1 à R faire % déplacer le robot x sur le cycle C %
(06)         Soit s le sommet actuellement occupé par le robot x ;
(07)         Trouver le rang i de la première occurrence du sommet s dans le cycle C ;
(08)         tant que (x n'a pas atteint le i − 1 sommet du cycle C) faire
(09)             Trouver la prochaine arête a de C que le robot x doit traverser ;
(10)             si (il existe un cycle contenant a)
(11)             alors Calculer le cycle canonique orienté élémentaire C_a comprenant l'arête a ;
(12)                  Tous les robots sur le cycle C_a se déplacent d'un sommet le long de C_a ;
(13)             sinon Déplacer les robots pour créer un sommet vide à l'extrémité de a ;
(14)                  Déplacer le robot x sur l'arête a
(15)             fin si
(16)         fin tant
(17)     fin pour
(18) fin tant
```

FIG. 3.7 – Algorithme résolvant le problème $EPCG$ pour $\rho = \infty$

**Description globale** L'algorithme comporte deux phases distinctes ; (a) une phase d'initialisation de la Ligne 1 à la Ligne 3 qui permet aux robots de s'accorder sur la situation globale du système, et (b) une phase de mouvement durant laquelle, chaque robot va successivement parcourir tout le graphe.

**Phase d'initialisation** Les Lignes 1 et 2 permettent aux robots de se mettre d'accord pour (a) identifier de manière unique chaque sommet du graphe par un entier de 1 à $p$, et (b) nommer de manière unique chaque robot par un entier de 1 à $R$, $R$ étant le nombre de robots présents. Puisque chaque robot dispose d'une boussole, il est possible d'effectuer ces deux tâches en parcourant le graphe de manière déterministe.

Le *cycle canonique orienté* $C$ introduit à la Ligne 3 est un cycle particulier du graphe qui traverse tous les sommets. Ce cycle n'est pas obligatoirement élémentaire ; il peut traverser plusieurs fois certains sommets. La contrainte principale est que tous les robots calculent le même cycle canonique ; nous proposons une manière d'en obtenir un. Pour cela, nous définissons tout d'abord le *chemin canonique orienté* d'un sommet $s_1$ vers un sommet $s_2$ comme étant le plus court chemin orienté reliant $s_1$ à $s_2$ sur la graphe ; en cas de non-unicité, le chemin est choisi de manière déterministe. Le *cycle canonique orienté* $C$ est composé de la concaténation des chemins canoniques élémentaires suivants : tout d'abord celui reliant le sommet 1 au sommet 2, puis celui reliant le sommet 2 au sommet 3, ..., puis celui reliant le sommet $p-1$ au sommet $p$, et enfin celui reliant le sommet $p$ au sommet 1.

**Phase de mouvements** Cette phase consiste en une boucle infinie (Lignes 4 à 18). Chaque itération de la boucle permet à tous les robots de parcourir (au moins) une fois le graphe tout entier. Ces parcours sont effectués successivement par chaque robot, du robot 1 au robot $R$ (Ligne 5). Pour un robot $x$ fixé, l'algorithme détermine tout d'abord la position actuelle de ce robot dans le cycle canonique orienté $C$ (Lignes 6 et 7). La boucle interne (Lignes 8 à 16) contrôle alors le déplacement du robot $x$ le long du cycle canonique orienté $C$ et se termine lorsque le robot $x$ a effectué un parcours complet du cycle $C$. Les Lignes 9 à 15 décrivent les comportements des robots pour permettre au robot $x$ d'avancer. Différents cas sont étudiés, suivant la situation de l'arête $a$ que doit franchir le robot $x$. Soient $s_1$ et $s_2$ les extrémités de l'arête $a$, le robot $x$ étant positionné sur le sommet $s_1$.

- Il existe un cycle du graphe contenant l'arête $a$ (Lignes 11 et 12). Dans ce cas, l'algorithme calcule déterministiquement un cycle canonique orienté $C_a$ incluant $a$, puis tous les robots situés sur $C_a$ se déplacent simultanément d'un sommet. Cela permet au robot $a$ de franchir l'arête $a$.
- Il n'existe aucun cycle contenant l'arête $a$ (Lignes 13 et 14), l'arête $a$ est donc un isthme. Nous supposons que cet isthme est situé sur un chemin d'exclusion mutuelle de type $1-1$ comme décrit dans la Section 3.4[5]. Considérons alors les trois sous-graphes $G_1$, $G_2$, et $G_3$ représentés sur la Figure 3.8. $G_2$ désigne le sous-graphe maximal connexe de $G$ incluant le sommet $s_2$ et ne comprenant pas l'arête $a$. $G_1$ est défini de manière similaire à $G_2$, mais en excluant le sommet $s_1$. $G_3$ correspond au sous-graphe maximal connexe de $G$ incluant $a$ et ne comprenant que des isthmes. Le sommet $s$ est défini comme étant l'extrémité de $G_3$ appartenant à $G_1$. Nous distinguons deux cas :
  - Il existe au moins un sommet vide (sans robot) dans le sous-graphe $G_2$. Considérons un chemin élémentaire de $s_2$ vers un sommet vide de $G_2$ (comme précédemment, le

[5]Les autres types de chemins d'exclusion mutuelle peuvent être traités de manière similaire et ne sont donc pas expliqués en détail.

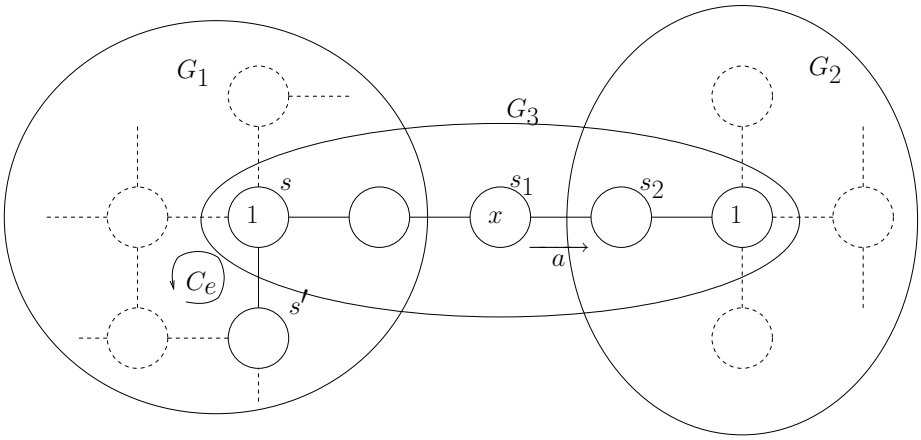



FIG. 3.8 – Configuration où l'arête $a$ n'appartient à aucun cycle

sommet et le chemin sont choisis de manière déterministe). Tous les robots situés sur ce chemin se déplacent simultanément afin de libérer le sommet $s_2$ de son (éventuel) robot. Le robot $x$ peut alors traverser l'arête $a$.

– Il n'existe aucun sommet vide dans le sous-graphe $G_2$. Puisque le nombre de robots présents est au plus de $p-q$, cela implique que le sous-graphe $G_1$ possède au moins $q$ sommets vides. De plus le sommet $s_1$ (où est situé le robot $x$) appartient à une chaîne d'isthmes (le sous-graphe $G_3$) de longueur maximale $q$ (cela provient des définitions de la longueur d'un chemin d'exclusion mutuelle et de celle des classes de graphes $\mathcal{G}(p,q)$). La séquence de mouvements suivante permet alors au robot $x$ de traverser l'arête $a$ :

  1. Les robots de $G_1$ se déplacent, de manière déterministe, pour libérer les sommets de $G_1 \cap G_3$ ainsi qu'un sommet $s'$ adjacent au sommet $s$, tel que les deux sommets $s$ et $s'$ appartiennent à un même cycle élémentaire $C_e$. Cette opération est réalisable car il y a au moins $p-q$ sommets vides dans $G_1$ et que le sommet $s$ est de type 1 ($s$ appartient à un cycle élémentaire).
  2. Le robot $x$ se déplace sur le sommet $s'$.
  3. Le robot situé sur le sommet $s_2$ se déplace sur le sommet $s$ ; ce qui libère tous les sommets situés entre $s$ non-inclus et $s_2$ inclus.
  4. Les robots situés sur le cycle $C_e$ se déplacent simultanément d'un sommet sur $C_e$ afin de positionner le robot $x$ sur le sommet $s$.
  5. Le robot $s$ se déplace sur le sommet $v_2$. Ces mouvements sont possibles puisque les sommets intermédiaires sont maintenant vides.

Dans tous les cas, le robot $x$ peut franchir l'arête $a$. Chaque robot est donc amené à parcourir successivement le cycle canonique $C$ ; l'algorithme proposé résout bien l'exploration perpétuelle du graphe lorsqu'il y a au plus $p-q$ robots dans un graphe de la classe $\mathcal{G}(p,q)$. □

**Remarques** Cet algorithme n'est pas optimal en nombre de mouvements. Le but est uniquement de montrer l'existence d'un algorithme. Nous n'avons pas cherché à trouver des algorithmes efficaces ; c'est néanmoins un problème probablement très intéressant.

Contrairement aux algorithmes présentés dans la section suivante, il n'est pas nécessaire dans ce cas particulier ($\rho = \infty$) que le graphe soit une grille partielle. Si le graphe est quelconque l'algorithme décrit permet aussi aussi de résoudre l'exploration.

La nécessité de considérer une grille partielle apparaît de manière claire pour un rayon de vision limité ($\rho = 1$). En effet lorsque la vision n'est plus totale, les robots ne connaissent pas initialement la topologie du graphe. Ils doivent alors, en premier lieu découvrir le graphe sur lequel ils se déplacent, et cela est impossible sans restriction sur le graphe[6].

## 3.6 Rayon de vision $\rho = 1$

Cette section étudie l'existence d'algorithmes $f$-résolvant le problème $EPCG$ pour une vision limitée $\rho = 1$. Contrairement aux deux cas extrêmes précédents, $\rho = 0$ et $\rho = \infty$, les résultats sont ici moins évidents et l'algorithme correspondant nécessite plusieurs mécanismes astucieux. Formellement :

**Théorème 3.10** *Il existe un algorithme $A$ qui $f$-résout le problème $EPCG$ pour une vision $\rho = 1$ si et seulement si $f(1,0) \leq 1$, $f(p,0) \leq p-1$ avec $p > 1$, et $f(p,q) \leq p-q$ pour toute paire $(p,q)$ avec $q > 0$.*

**Remarque** Ce théorème montre deux choses pertinentes. D'un coté, une vision très limitée ($\rho = 1$) permet de résoudre pratiquement autant de situations que le permet une vision globale ($\rho = \infty$) ; seuls les cas où le graphe est sans isthme ($q = 0$) impose une limite de $p-1$ robots, soit un de moins que la borne topologique. De l'autre coté, ce rayon de vision unité apparaît alors clairement comme la frontière rendant le problème $EPCG$ solvable ($\rho \geq 1$) ou insolvable ($\rho = 0$).

**Preuve** L'implication $\Leftarrow$ du théorème est démontrée par contradiction. Supposons qu'il existe un algorithme $A$ qui $f$-résout le problème $EPCG$ pour $\rho = 1$ telle que $f$ ne vérifie pas les hypothèses du théorème. Pour les classes de graphes $\mathcal{G}(1,0)$ et $\mathcal{G}(p,q)$ avec $q > 0$, la contradiction est immédiate puisque les valeur du théorème correspondent à la borne topologique ; il est impossible de résoudre le problème $EPCG$ avec plus de $p-q$ robots.

Considérons alors une classe de graphes $\mathcal{G}(p,0)$ avec $p > 1$ et supposons que $f(p,q) = p$. Initialement, le graphe est entièrement rempli de robots. Ainsi pour conserver les exclusions mutuelles, la seule possibilité consiste à coordonner les robots pour qu'ils se déplacent simultanément le long de cycles élémentaires. Cependant chaque robot, n'ayant qu'une vision locale du graphe, ne peut détecter l'existence de cycles. Par conséquent aucun robot ne peut

[6]La restriction à une grille partielle n'est pas nécessaire. Il est possible de montrer que des restrictions plus faibles sur les directions et les longueurs des arêtes suffisent. Dans le cas d'une grille partielle, les arêtes ne sont que horizontales ou verticales et toujours de même longueur.

décider de se déplacer et doit donc rester immobile ; une contradiction. Ce qui conclut la preuve de l'implication $\Leftarrow$.

Il reste à prouver l'implication opposée $\Rightarrow$. Pour cela, la suite de cette section décrit un algorithme $f$-résolvant le problème $EPCG$ pour $\rho = 0$ avec les contraintes du théorème. L'algorithme proposé ne tient pas compte du cas particulier où le graphe est réduit à un seul sommet. Ce cas trivial peut facilement être traité séparément où l'unique robot présent décide de rester immobile de manière permanente s'il détecte qu'il se situe sur ce graphe particulier à un seul sommet.

Notre algorithme se décompose en trois étapes distinctes. La première requiert d'avoir au moins un sommet libre ; c'est donc cette étape limitante qui impose les bornes du théorème.

*Étape* 1 *: Découverte du graphe* Chaque robot exécute un premier algorithme permettant (1) de découvrir la topologie du graphe, et (2) de se mettre implicitement d'accord avec les autres robots sur un temps d'arrêt pour passer à l'étape suivante. Si cette étape est couronnée de succès (ce qui est le cas lorsqu'il y a au moins un sommet vide), tous les robots peuvent calculer les valeurs $p$ et $q$ du graphe sur lequel ils se déplacent.

*Étape* 2 *: Estimation du nombre de robots* Chaque robot exécute un second algorithme permettant d'évaluer le prédicat *"Il y a au plus $p - q$ robots sur le graphe"*. Il n'est pas nécessaire de connaître le nombre exact de robots, mais uniquement de savoir si l'exploration est possible ou non, d'après la borne topologique de $p - q$ robots.

*Étape* 3 *: Exploration perpétuelle* Si le système comporte plus de $p - q$ robots, l'exploration est impossible, les robots décident donc de rester immobiles afin de satisfaire les contraintes d'exclusions mutuelles. Dans le cas contraire où le système comporte au plus $p-q$ robots, ceux-ci décident d'effectuer l'exploration perpétuelle du graphe selon un algorithme proche de celui utilisé lorsque $\rho = \infty$.

□

Suivant le nombre de robots $R$ initialement présents sur un graphe d'une classe $\mathcal{G}(p, q)$, l'algorithme peut comporter de trois manières différentes :

- Si $R = p$, l'algorithme reste bloqué dans la première étape. Aucun robot ne se déplace, mais tous les robots continuent l'exécution indéfiniment.
- Si $R < p$ et $R > p-q$, l'algorithme effectue les deux premières étapes, puis termine son exécution car l'exploration est impossible. Contrairement au cas précédent, les robots stoppent complètement l'exécution.
- Si $R < p$ et $R > p - q$, l'algorithme effectue les deux premières étapes, puis boucle indéfiniment (et volontairement) dans la troisième étape afin de permettre l'exploration perpétuelle du graphe par tous les robots.

### 3.6.1 Découverte du graphe

La méthode utilisée pour permettre aux robots de découvrir le graphe est totalement novatrice. En effet, nous montrons que des robots sont capables d'apprendre la topologie d'un graphe, sans pour autant visiter entièrement le graphe. Les entités ne possédant aucun

moyen direct de communication, nous utilisons une communication cachée qui permet aux robots d'apprendre des informations en observant le comportement des autres robots. Pour cela, nous introduisons le concept de *mouvement contextuel* qui repose sur l'existence d'une boussole et sur la restriction des graphes à des grilles partielles.

**Mouvement contextuel** Cela désigne un déplacement élémentaire d'un robot qui dépend de la position actuelle du robot. Nous distinguons, par exemple, un déplacement vers le Nord d'un robot situé sur un sommet ne possédant qu'une arête vers le Nord, d'un déplacement vers le Nord d'un robot situé sur un sommet possédant les quatre arêtes possibles (Nord, Est, Sud, et Ouest). Formellement un mouvement contextuel $mc$ est décrit par une paire composée (1) d'un sous-ensemble non-vide *contexte* de l'ensemble des arêtes $\{Nord, Est, Sud, Ouest\}$, et (2) d'une direction *direction* appartenant à cet ensemble contexte.

Il existe 32 mouvements contextuels différents ; 4 dont l'ensemble *contexte* est réduit à une seule direction, 12 dont l'ensemble *contexte* comprend deux directions, 12 dont l'ensemble *contexte* comprend trois directions, et 4 dont l'ensemble *contexte* comprend les quatre directions. Pour qu'un robot $x$ puisse effectuer un mouvement contextuel $mc$, il faut que deux conditions soient remplies : (1) $x$ se situe sur un sommet qui possède le contexte de $mc$, et (2) le sommet de destination indiqué par la direction de $mc$ est libre.

### Description de l'algorithme

L'algorithme permettant aux robots de découvrir le graphe est donné par le Figure 3.9. Le texte, volontairement bref, nécessite quelques explications, en particulier sur la ligne 3 et sur la valeur de $k_{max}$.

```
Algorithme decouverte1 () :
(1)  k ← 1 ; k_max ← +∞ ;
(2)  tant que (k < k_max)
(3)      Effectuer de manière déterministe
             toutes les séquences possibles de k mouvements contextuels
             en déduisant progressivement le graphe ;
(4)      si (le graphe est entièrement connu)
(5)          alors Calculer les paramètres p et q du graphe ;
(6)                   Calculer k_max ;
(7)      fin si ;
(8)      k ← k + 1
(9)  fin tant
```

FIG. 3.9 – Algorithme de découverte du graphe

**Déplacements des robots** Simultanément tous les robots essayent d'effectuer les mêmes séquences de mouvements contextuels. Cela signifie que pour une séquence donnée $S = m_1, m_2, \ldots, m_z$ ; lors de la première ronde, tous les robots pouvant effectuer le mouvement contextuel $m_1$ se déplacent (suivant la direction de $m_1$) ; puis lors de la seconde ronde tous

les robots pouvant effectuer le mouvement contextuel $m_2$ se déplacent, ... jusqu'au dernier mouvement de la séquence. Après cette séquence les robots rejoignent tous leurs positions initiales en effectuant la séquence inverse.

L'utilisation des mouvements contextuels permet de garantir les exclusions mutuelles. En effet lors d'une ronde donnée, tous les robots qui se déplacent le font suivant un même mouvement contextuel. Le fait que tous les robots se déplacent suivant le même direction permet de garantir qu'il n'y aura pas de collision. Le fait que les robots se déplacent uniquement si leur contexte actuel est spécifique permet aux autres robots de déduire les sommets auxquels ils n'ont pas accès ; en effet un robot qui voit un autre robot se déplacer lors d'une ronde $r$ apprend quel est le contexte du sommet de départ du robot en mouvement. Les détails du mécanisme de déduction sont expliqués un peu plus loin.

Admettant pour le moment que les robots arrivent à déduire la topologie du graphe, ils leur restent encore à s'entendre sur un arrêt de l'algorithme, autrement dit sur la valeur de $k_{max}$. Une fois qu'un robot connaît tout le graphe, il simule localement (sans déplacement réel) toutes les exécutions possibles de l'algorithme dans toutes les situations initiales contenant au plus $p-1$ robots. Dans toutes ces simulations, tous les robots découvrent à un moment donné la topologie du graphe ; il suffit alors de prendre pour $k_{max}$ la longueur de la plus longue séquence nécessaire, pour un robot dans une exécution, pour déduire entièrement le graphe. Cette valeur $k_{max}$ est calculée de manière identique pour tous les robots ; cela leur permet de s'accorder sur la fin de l'algorithme.

### Un exemple d'exécution

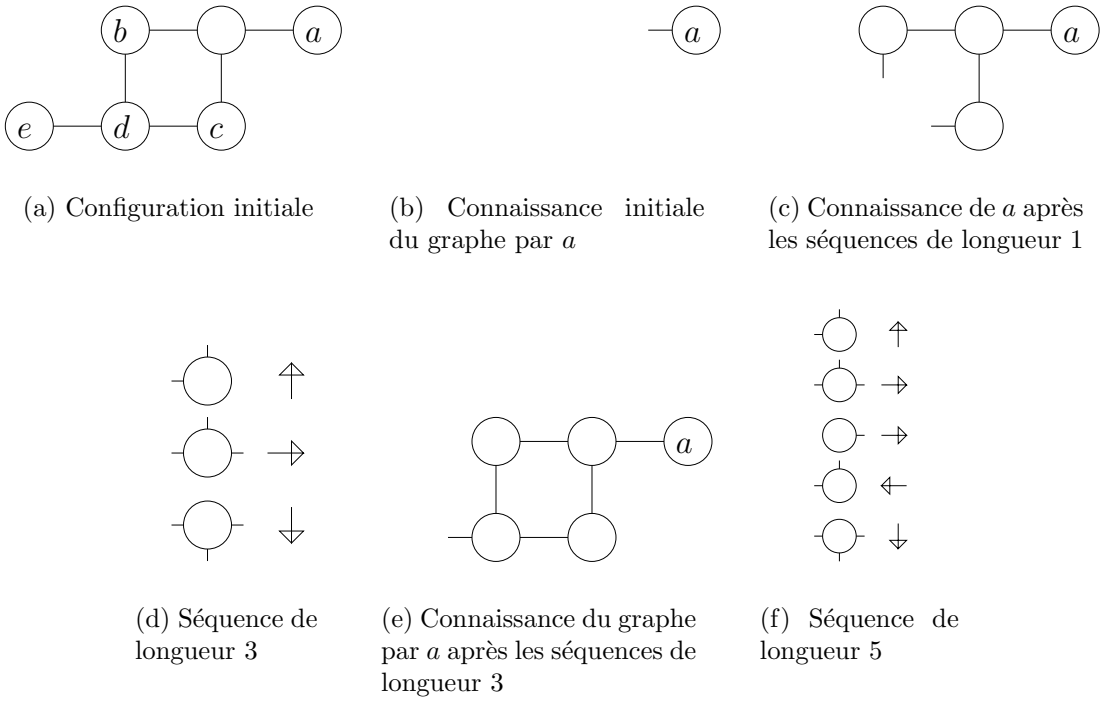
Nous prions le lecteur de bien vouloir nous excuser pour l'informalité de ce qui suit. Il serait trop complexe d'écrire explicitement l'algorithme de déduction et cela n'apporterait pas grand chose au propos. Nous décrivons alors, à l'aide d'un exemple, le fonctionnement du mécanisme de déduction de la topologie du graphe. L'exemple représenté sur la Figure 3.10(a) utilise un graphe de 6 sommets sur lequel 5 robots sont initialement placés[7]. Nous montrons l'apprentissage progressif du graphe par le robot $a$.

Initialement le robot $a$ ne connaît que son sommet et sait que celui-ci possède un voisin vers l'Ouest. Il ne connaît pas le contexte de ce voisin (ses arêtes incidentes), mais voit que ce sommet est actuellement vide. La connaissance initiale du graphe par $a$ est représentée sur la Figure 3.10(b).

**Séquences composées d'un seul mouvement contextuel** Parmi les 32 séquences composées d'un seul mouvement contextuel, il y en a trois qui permettent à $a$ d'apprendre/déduire une partie du graphe :

– Celle qui permet au robot $a$ de se déplacer vers l'Ouest. Lors de ce mouvement, $a$ apprend le contexte du sommet voisin de sa position initiale.

[7]Les noms donnés aux robots sont uniquement là pour les explications. Les robots sont anonymes et indistinguables.




(a) Configuration initiale

(b) Connaissance initiale du graphe par $a$

(c) Connaissance de $a$ après les séquences de longueur 1

(d) Séquence de longueur 3

(e) Connaissance du graphe par $a$ après les séquences de longueur 3

(f) Séquence de longueur 5

FIG. 3.10 – Un exemple de déduction sur un graphe avec 5 robots

- Celle qui permet au robot $b$ de se déplacer vers l'Est. Lors de ce mouvement contextuel, $a$ voit un robot arriver sur son sommet voisin. Il ne connaît pas l'identité $b$ du robot puisque les robots sont anonymes, mais de par l'arrivée d'un robot lors de ce mouvement contextuel particulier, il déduit le contexte du sommet initial de $b$.
- Celle qui permet au robot $c$ de se déplacer vers le Nord. De manière similaire au déplacement du robot $b$, $a$ peut déduire de l'arrivée de $c$ le contexte du sommet initial de $c$.

Suite aux séquences composées d'un seul mouvement contextuel, la connaissance du graphe par le robot $a$ a augmenté et est résumée sur la Figure 3.10(c).

**Séquences composées de deux mouvements contextuels** Ces séquences n'apportent aucune information supplémentaire pour $a$. Néanmoins elles sont nécessaires car elles apportent éventuellement des informations aux autres robots. (Dans ce cas particulier $e$ déduit des informations lorsque $c$ puis $d$ se déplacent.)

**Séquences composées de trois mouvements contextuels** Une séquence, parmi les $32^3$ séquences composées de trois mouvements contextuels, permet au robot $a$ d'augmenter sa connaissance du graphe. Cette séquence est représentée sur la Figure 3.10(d). Lors de l'exécution de cette séquence, le robot $c$ va tout d'abord se déplacer vers le Nord, puis le robot $d$ va se déplacer vers l'Est, puis aucun déplacement. Le fait qu'il n'y ait aucun

déplacement lors du troisième mouvement contextuel de cette séquence permet à $a$ de déduire le contexte du sommet initial de $d$ : en effet, l'absence de déplacement de $c$ lors du troisième mouvement contextuel implique qu'un robot ($d$ en l'occurrence) bloque le déplacement de $c$ vers le Sud. Ce blocage permet à $a$ de déduire le contexte du sommet initial de $d$, qui correspond obligatoirement au contexte du second mouvement contextuel.

La Figure 3.10(e) résume la connaissance du robot $a$ après les séquences composées de trois mouvements contextuels.

**Séquences composées de quatre mouvements contextuels** Ces séquences n'apportent aucune information supplémentaire pour $a$.

**Séquences composées de cinq mouvements contextuels** Grâce à la séquence représentée sur la Figure 3.10(f), le robot $a$ est capable de déduire le contexte du sommet initial de $e$. Le blocage du robot $c$ lors du cinquième mouvement contextuel implique un blocage du robot $d$ lors du quatrième mouvement contextuel, qui lui-même implique un déplacement d'un robot lors du troisième mouvement contextuel. À ce moment-là $a$ sait qu'il a découvert tout le graphe, car il n'existe plus aucune arête dont il ne connaît la destination.

**Après la découverte du graphe** Une fois que $a$ a déduit la topologie du graphe, il simule toutes les exécutions possibles de l'algorithme sur ce graphe en prenant toutes les configurations initiales comprenant de 1 à 5 robots. Il en déduit alors la valeur maximale $k_{max}$ nécessaire pour que tous les robots dans toutes les exécutions découvrent le graphe. $a$ continue alors à exécuter cet algorithme jusqu'à avoir tester toutes les séquences composées de $k_{max}$ mouvements contextuels. Tous les robots peuvent ensuite débuter la deuxième étape.

### Preuve de l'algorithme

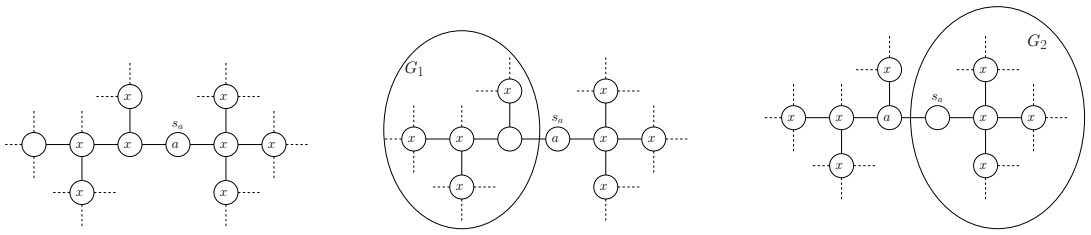
Avant de décrire le fonctionnement de la deuxième étape, nous donnons ici une "preuve" informelle de la validité de l'algorithme de découverte. Tout comme pour l'algorithme, une preuve formelle serait trop complexe à écrire et n'aurait pas, selon nous, un apport significatif.

**Preuve** La preuve est structurée en deux parties. Nous montrons d'abord que pour tout graphe, lorsqu'il y a exactement $p-1$ robots, il est possible de borner la longueur des séquences (de mouvements contextuels) nécessaires pour qu'un robot déduise tout le graphe. Nous généralisons ensuite lorsque le nombre de robots est inférieur à $p-1$. En effet étrangement, il est plus simple de déduire la topologie d'un graphe lorsque celui-ci est quasiment rempli de robots ; cela provient du fait que l'absence de robot entraîne une absence de mouvement, donc une absence d'observation, et finalement une absence de déduction.

1. Considérons une grille partielle d'une classe $\mathcal{G}(p,q)$ contenant exactement $p-1$ robots et soit $a$ un robot situé initialement sur le sommet $s_a$. Soit $d_v$ la distance séparant le sommet $s_a$ de l'unique sommet vide du graphe. Après avoir exécuté toutes les séquences de longueur inférieure à $d_v + k$, le robot $a$ a déduit au moins le sous-graphe contenant

les sommets situés à distance au plus $\lfloor \frac{k+1}{2} \rfloor$ de $s_a$. Le graphe étant fini, cela permet de borner (grossièrement) $k_{max}$ par $d_v + 2D$, $D$ étant le diamètre du graphe.

Étant donnée une configuration initiale (Figure 3.11(a)), soit $S_1$ une séquence de $d_v - 1$ mouvements contextuels permettant de "déplacer" le sommet vide à coté de $s_a$ (Figure 3.11(b)) et $S_2$ un prolongement de $S_1$ déplaçant le robot $a$ vers le sommet vide (Figure 3.11(c)). A partir de ces deux nouvelles configurations, il faut $2k$ mouvements contextuels supplémentaires (à ajouter aux séquences $S_1$ ou $S_2$) pour découvrir les sommets situés à distance $k$ du robot $a$. En effet comme détaillé dans l'exemple de la Figure 3.10, deux mouvements sont nécessaires à chaque fois pour successivement bloquer les robots et permettre à $a$ de déduire des informations. (Formellement la preuve nécessiterait une induction sur la valeur de $k$.)




(a) Configuration initiale

(b) Configuration après $S_1$

(c) Configuration après $S_2$

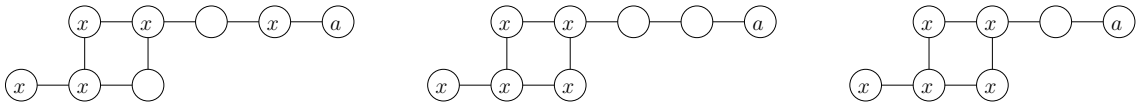
FIG. 3.11 – Exemple avec $p - 1$ robots et $d_v = 3$

2. Intuitivement, si il est possible pour un robot de déduire entièrement un graphe en temps fini lorsque ce graphe ne possède qu'un seul sommet vide, il en reste de même si ce graphe possède plusieurs sommets vides. Paradoxalement, la présence de multiples sommets vides ajoute de la complexité dans le mécanisme de déduction, néanmoins il est toujours possible de se ramener à une situation *comme si* il n'y avait qu'un seul sommet vide. La Figure 3.12(a) propose une situation initiale d'un graphe comprenant deux sommets vides et un robot $a$. Il est possible grâce à une séquence finie $S$ de mouvements contextuels de se ramener à la configuration de la Figure 3.12(b). À partir de cette nouvelle configuration, le robot $a$ est apte à déduire le graphe comme si il était dans la configuration de la Figure 3.12(c) ; les séquences utiles comportent cependant plus de mouvements contextuels puisque chaque robot devra effectuer deux déplacements pour se rapprocher de $a$.

□

### 3.6.2 Estimation du nombre de robots

Lors de l'étape précédente, les robots se sont coordonnés afin que chacun connaisse la topologie du graphe sur lequel ils évoluent. Par ailleurs, ils terminent simultanément l'étape



(a) Configuration initiale (b) Configuration après $S_1$ (c) Configuration après $S_2$

FIG. 3.12 – Exemple avec $p-2$ robots

précédente et peuvent donc commencer de manière synchrone cette étape. L'objectif consiste à déterminer si l'exploration est réalisable ou non, suivant le nombre de robots présents. La Section 3.4 a montré qu'il fallait au plus $p-q$ robots, $p$ et $q$ étant maintenant connus des robots puisque ces paramètres ne dépendent que de la topologie du graphe.

Les robots ne pouvant communiquer, l'estimation de leur nombre s'effectue une nouvelle fois par l'observation de leurs comportements. Les quatre grandes phases de notre algorithme apparaissent sur la Figure 3.13 et sont détaillées ci-dessous.

**Algorithme** Estimation_nombre_robots () :
(1) Numéroter les sommets avec un traitement postfix d'un parcours en profondeur ;
(2) Déplacer les robots de manière à occuper les sommets de 1 à $R$ ;
(3) Evaluer le prédicat $(R \leq p-q)$ ;
(4) Déplacer les robots de manière à occuper les sommets de 1 à $R$ ;

FIG. 3.13 – Algorithme d'estimation du nombre de robots

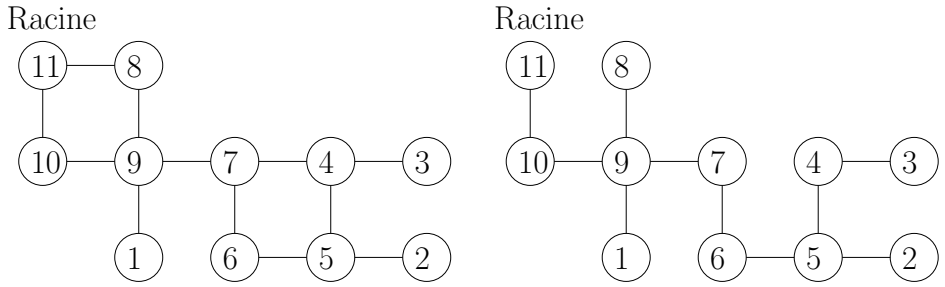
**Ligne 1 : Numérotation** Grâce au sens global de direction fourni par la boussole, les robots peuvent se mettre d'accord sur un sommet particulier du graphe, par exemple celui situé le plus au Nord-Ouest. À partir de ce sommet, chaque robot effectue localement une exploration en profondeur du graphe, en numérotant les sommets de manière postfix. Cette numérotation des sommets de 1 à $p$ s'effectue de manière déterministe et sans déplacement réel des robots.

La caractéristique postfix du parcours permet de garantir les deux propriétés suivantes qui sont illustrées sur la Figure 3.14 :

1. Pour toute valeur de $\ell$ comprise entre 1 et $p$, le sous-graphe contenant uniquement les sommets étiquetés de $\ell$ à $p$ est un graphe connexe.
2. Il est possible de créer un arbre dont la racine est le sommet étiqueté $p$ tel que chaque sommet choisit pour père son sommet voisin dont l'étiquette est la plus faible parmi celles qui sont plus grandes que sa propre étiquette. Le sommet 4 possède trois voisins 3, 5, et 7 ; il choisit pour père le sommet 5 qui est le plus petit parmi les étiquettes 5 et 7.

Les deux propriétés vont être utiles dans la suite de l'algorithme. La première permet aux robots de se déplacer "facilement" dans le graphe tout en laissant fixes certains robots

déjà positionnés. La seconde permettra de transmettre de l'information entre les robots, en occupant ou non les sommets pères.




(a) Numérotation postfix (b) Arbre correspondant

FIG. 3.14 – Exemple de parcours en profondeur avec une numérotation postfix

**Ligne 2 : Positionnement** Lors de cette phase, l'objectif est de déplacer les robots pour qu'ils occupent les sommets avec les identités les plus petites. $R$ désignant le nombre de robots présents, à la fin de la phase, les robots sont situés sur les sommets 1 à $R$. Pour aboutir à cette configuration, l'algorithme utilise $p-1$ sous-phases durant lesquelles tous les robots vont successivement essayer de rejoindre le sommet 1, puis le sommet 2, ..., puis le sommet $p-1$. Chaque sous-phase dure $4D$ rondes où $D$ correspond au diamètre du graphe. À chaque direction cardinale, nous associons un entier unique dans l'ensemble $\{0, 1, 2, 3\}$.

Au début de la première sous-phase, chaque robot $x$ détermine le plus court chemin $c_x$ (de longueur maximale $D$) dans le graphe lui permettant de rejoindre le sommet 1. Pour chaque ronde $r$ de la première sous-phase, un robot $x$ décide de progresser sur son chemin $c_x$ si et seulement si (a) le prochain sommet est dans une direction associée à l'entier ($r$ mod 4) et (b) ce prochain sommet est vide. Ces deux conditions garantissent les contraintes d'exclusions mutuelles pour les mêmes raisons que les mouvements contextuels introduits lors de la première étape.

Après les $4D$ rondes de la première sous-phase, un robot a rejoint le sommet 1. La seconde sous-phase débute alors durant laquelle tous les robots, excepté celui positionné en 1, tentent de rejoindre de manière similaire le sommet 2. Le processus continue pour les $p-1$ sous-phases : lors de la $i$-ème sous-phase, tous les robots, exceptés ceux déjà positionnés sur les sommets 1 à $i-1$, tentent de rejoindre le sommet $i$ (si il reste des robots non placés). Une fois les $R$ robots positionnés, les sous-phases restantes n'engendrent aucun déplacement, mais elles sont néanmoins effectuées afin de permettre à tous les robots de terminer cette phase et débuter la suivante de manière simultanée.

**Ligne 3 : Évaluation du prédicat** ($R \leq p-q$) L'objectif consiste à évaluer si le nombre de robots présents autorise l'exploration perpétuelle du graphe. Dans le cas où $q = 0$ la réponse

est trivialement positive[8] puisque tous les robots, connaissant le graphe, savent que $q = 0$. Dans le cas contraire ($q > 0$), nous expliquons ici comment les robots peuvent découvrir et partager cette information.

L'idée principale consiste à remarquer que, grâce au positionnement effectué précédemment, certains robots sont capables immédiatement d'évaluer le prédicat ($R \leq p - q$). Il suffit alors de diffuser l'information auprès des autres robots.

Au début de cette phase, les $R$ robots sont situés sur les sommets 1 à $R$. Si certains robots se situent sur des sommets $p - q + 1$ à $p$, ils savent immédiatement que $R > p - q$. La phase est alors découpées en $p-q$ sous-phases durant lesquels les autres robots vont successivement apprendre la valeur du prédicat. Chaque sous-phase dure $4D$ rondes. Durant la $i$-ème sous-phase, tous les robots (si il y en a) situés sur les sommets $p - q - i + 2$ connaissent la valeur du prédicat et se coordonnent pour informer le robot (si il y en a un) situé sur le sommet $p - q - i + 1$ de cette valeur. Par exemple, lors de la première sous-phase ($i = 1$), le robot qui apprend la valeur du prédicat est celui situé sur le sommet $p - q$, le premier sommet qui ne permet pas de déduire immédiatement la valeur du prédicat.

La transmission de la valeur du prédicat s'effectue grâce au mécanisme suivant : à la fin de la $i$-ème sous-phase, le robot situé sur le sommet $p - q - i + 1$ observe si son sommet père $s_p$, dans l'arbre obtenu par le parcours en profondeur, est occupé ou non par un robot. Si $s_p$ est vide, cela signifie que le prédicat est vrai, si au contraire un robot occupe ce sommet, cela signifie que le prédicat est faux. Suivant la valeur (connue) du prédicat, les robots situés sur les sommets $p - q - i + 1$ à $p$ se déplacent pour occuper ou libérer $s_p$ :

- Si le prédicat est faux, tous les robots des sommets $p-q-i+1$ à $p$ tentent de rejoindre le sommet $s_p$ en utilisant le même mécanisme que lors de la phase 2. En $4D$ rondes, un robot atteint ce sommet.
- Si le prédicat est vrai, tous les robots des sommets $p-q-i+1$ à $p$ tentent de s'éloigner le plus possible du sommet $s_p$ en utilisant un mécanisme similaire à celui de la phase 2 : lors de chaque ronde $r$ de la sous-phase, tout robot $x$ des sommets $p - q - i + 1$ à $p$ choisit de se déplacer si et seulement si (1) le sommet dont la direction associée vaut ($r \mod 4$) est vide et (2) ce déplacement éloigne le robot $x$ de $s_p$. Cela garantit qu'en $4D$ rondes aucun robot n'occupe le sommet $s_p$

Après les $p - q$ sous-rondes, tous les robots présents connaissent la valeur du prédicat ($R \leq p - q$) et savent donc si l'exploration du graphe est possible.

**Ligne 4 : Positionnement** Afin de permettre l'exploration les robots ont besoin de se positionner dans une configuration initiale "connue". Une phase de positionnement identique à celle de la Ligne 1 est donc effectuée. Après ce positionnement les robots se retrouvent sur les sommets 1 à $R$.

[8] Les deux phases précédentes sont d'ailleurs facultatives dans cette situation.

### 3.6.3 Exploration perpétuelle

L'étape précédente composée de quatre phases distinctes a permis aux robots (1) d'évaluer le prédicat ($R \leq p-q$) et (2) de se placer dans une configuration "connue". Si le prédicat est faux, les robots décident de ne plus se déplacer puisque l'exploration est impossible. Lorsque le prédicat est vrai, les robots effectuent leur exploration en se basant sur l'algorithme décrit Section 3.5.3 : l'astuce consiste à exécuter l'algorithme défini pour une vision $\rho = \infty$ en simulant le comportement de tous les robots. Pour cela il est nécessaire que chaque robot connaisse le graphe et la configuration initiale. Il suffit alors de considérer la configuration initiale particulière où il y a exactement $p-q$ robots positionnés sur les sommets 1 à $p-q$.

L'exécution de l'algorithme conçu pour $\rho = \infty$ consiste alors à déplacer physiquement les $R$ robots et virtuellement les $p-q-R$ robots imaginaires initialement situés sur les sommets $R+1$ à $p-q$. Cela conclut les explications pour l'algorithme lorsque $\rho = 1$.

## 3.7 Conclusion

Ce chapitre nous a permis de présenter plusieurs résultats traitant de l'exploration perpétuelle contrainte de graphes par des robots anonymes. D'un coté, nous démontrons un résultat fondamental : une limite topologique, pour tout graphe, bornant le nombre maximal de robots permettant la résolution du problème. D'un autre coté, nous proposons un ensemble d'algorithmes permettant effectivement à des robots d'explorer les graphes.

Les points importants à retenir sont les suivants :

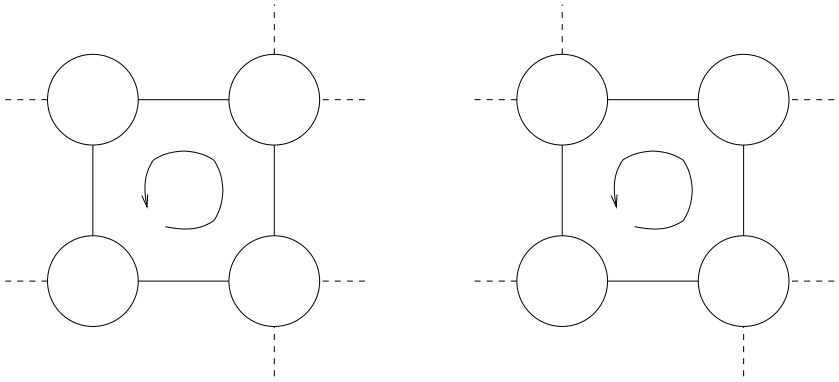
- Le problème de l'exploration perpétuelle contrainte est un problème intéressant et complexe. La présence de contraintes d'exclusions mutuelles rend le problème pertinent d'un point de vue pratique, et passionnant d'un point de vue théorique.
- La borne topologique obtenue n'est pas un résultat trivial ; la définition de l'arbre de mobilité permet de faire ressortir les composantes topologiques importantes des graphes.
- L'introduction de la notion de $f$-solvabilité est intéressante car elle permet (1) de garantir en toutes circonstances les propriétés de sûreté, et (2) de résoudre le problème de l'exploration lorsque les conditions initiales le permettent.
- Lorsque les robots possèdent une boussole et une vision globale du graphe, il est toujours possible d'atteindre la borne topologique, et ce quelque soit le type de graphe sur lequel évoluent les robots.
- Lorsque les robots ne possèdent qu'une vision très limitée et une boussole, il est possible en restreignant les graphes à des grilles partielles de :
  - Découvrir entièrement un graphe à partir du moment où celui-ci possède au moins un sommet vide. L'algorithme proposé utilise la nouvelle notion très séduisante de mouvement contextuel.
  - Compter le nombre de robots présents sur un graphe à partir du moment où celui-ci possède au moins un sommet vide[9].

[9]Une généralisation de l'algorithme présenté permet en effet de compter précisément le nombre de robots

– Résoudre le problème de l'exploration perpétuelle contrainte pour un nombre de robots très proche de la borne topologique.

Dans la même thématique, il reste néanmoins des sujets à aborder :

– Trouver des algorithmes plus efficace ; en particulier dans le cas $\rho = \infty$ où il est évident qu'il existe des algorithmes plus efficaces que celui proposé ici.
– Déterminer la complexité minimale, en nombre de rondes, pour effectuer une fois l'exploration pour tous les robots. Intuitivement, cette complexité doit pouvoir se calculer à partir de l'arbre de mobilité.
– Généraliser les résultats pour des valeurs intermédiaires de $\rho$. Nous conjecturons qu'il suffit de considérer d'autres mouvements contextuels. À partir de $\rho = 3$, il est possible de considérer des mouvements contextuels comprenant un cycle de quatre sommets ; suivant les arêtes incidentes à ces 4 sommets, on obtient ou non un mouvement contextuel. Deux exemples sont représentés sur la Figure 3.15 et le lecteur est invité à deviner pourquoi l'élément de gauche correspond à un mouvement contextuel alors que l'élément de droite n'en est pas un !



(a) Un mouvement contextuel lorsque $\rho = 3$

(b) Pas un mouvement contextuel lorsque $\rho = 3$

FIG. 3.15 – Mouvements contextuels lorsque $\rho = 3$ ?

présents. Néanmoins pour notre problème, cela n'est pas nécessaire, seul la valeur du prédicat $(R \leq p - q)$ est pertinente.

# Chapitre 4

# Détecteurs de fautes et consensus dans les systèmes anonymes

Les résultats présentés dans ce chapitre correspondent principalement à des traductions des publications [18, 19, 20] et des rapports techniques [22, 23, 24]. Des résultats supplémentaires de [18] sont évoqués mais ne figurent pas entièrement dans ce chapitre.

## 4.1 Introduction

**Systèmes asynchrones anonymes** Les systèmes asynchrones non-anonymes font l'objet de nombreuses études car ils permettent de représenter une grande variété de systèmes réels. D'un point de vue théorique, l'asynchronie empêche de distinguer une entité très lente d'une entité défaillante. L'hypothèse supplémentaire de l'anonymat dans ces systèmes généralise l'indistinguabilité entre entités, même entre processus corrects.

Un des tous premiers travaux (dont nous avons connaissance) étudiant des systèmes anonymes est celui de D. Angluin [2]. Dans ce papier, l'auteur s'intéresse à des problèmes de calculabilité tels que "quelles fonctions peut on calculer dans un système à passage de messages anonyme et asynchrone ?" L'élection d'un leader est un exemple simple de problème insolvable car il est impossible, dans de tels systèmes, de briser la symétrie en présence d'anonymat et d'asynchronie. Des systèmes anonymes sans panne ont aussi été étudiés dans [69, 70] où les auteurs donnent une caractérisation précise des problèmes solvables suivant les connaissances initiales du réseau par les participants.

**Détecteurs de fautes** La notion de détecteurs de fautes [26] est une des approches les plus populaires pour contourner les résultats d'impossibilité existant dans les systèmes répartis asynchrones. Informellement, cela consiste à fournir des informations supplémentaires à chaque participant afin de permettre la résolution d'un problème initialement insolvable. L'exemple le plus connu, et celui que nous avons étudié, est le problème du consensus. Dans ce problème, chaque processus propose une valeur et tous les processus corrects (qui ne tombent pas en panne durant l'exécution) doivent décider la même valeur, parmi celles pro-

posées au départ. Ce problème est prouvé comme insolvable dès que l'on considère un système asynchrone où au moins un processus peut défaillir [35]. Plusieurs détecteurs de fautes ont été introduits pour permettre la résolution du problème du consensus, deux méritent une attention particulière ; $\Omega$ [25] et $\Sigma$ [28]. En effet la combinaison des deux décrit le plus faible détecteur de fautes autorisant la résolution du problème du consensus [28, 29].

**Détecteurs de fautes dans les systèmes anonymes** La sortie du détecteur $\Omega$ est une identité de processus, celles de $\Sigma$ et $P$ [26] sont quant à elles des listes d'identités de processus. Ces détecteurs de fautes ne sont évidemment pas conçus pour les systèmes anonymes, néanmoins il est possible de considérer un système anonyme comme un système où (1) les processus possèdent des identités distinctes, mais (2) ces identités sont inconnues et inaccessibles par les processus eux-mêmes. Dans de tels systèmes, les détecteurs de fautes cités peuvent être utilisés et utiles. Certes $\Omega$ n'apporte aucune information pertinente[1], mais $P$ permet lui de résoudre le consensus : la liste des identités en tant que telle est inutile mais connaître le cardinal de celle-ci est suffisant pour résoudre le problème du consensus.

Contrairement à $\Omega$, $\Sigma$, et $P$, certains détecteurs de fautes, initialement proposés dans des contextes non-anonymes, semblent parfaitement conçus pour les systèmes anonymes. Par exemple, le détecteur $\mathcal{L}$ [30] donne une simple valeur booléenne à chaque processus. $\mathcal{L}$ a été introduit car il est le plus faible détecteur de fautes pour résoudre le problème du $(n-1)$-set agreement [30] dans un système asynchrone (non-anonyme) à passage de messages. Une généralisation de ce détecteur est proposé dans [12].

**Contributions** Ce chapitre concerne les détecteurs de fautes dans les systèmes asynchrones anonymes et leur utilisation pour résoudre le problème du consensus.

La première contribution consiste en l'introduction de nouveaux détecteurs de fautes spécialement conçus pour les systèmes anonymes (intuitivement, sans identité). De manière intéressante, nous montrons ensuite que ces détecteurs sont équivalents à leurs homologues classiques ($\Omega$, $\Sigma$, et $P$) dans les systèmes non-anonymes. Plus généralement nous proposons deux hiérarchies classant les détecteurs les uns par rapport aux autres ; une dans les systèmes non-anonymes, une dans les systèmes anonymes.

La seconde contribution consiste en deux algorithmes de consensus, dans les systèmes anonymes, basés sur des détecteurs de fautes ; d'une part avec un détecteur parfait et d'autre part avec la paire de détecteurs $(A\Sigma, A\Omega)$. L'existence de ces algorithmes résolvant le consensus pose la question de l'existence d'un plus faible détecteur de fautes pour le consensus dans les systèmes anonymes. Nous ne répondons pas entièrement à la question mais apportons quelques éléments de réfléxion.

**Plan du chapitre** Le chapitre comporte 6 sections. La Section 4.2 présente formellement les systèmes asynchrones (non-)anonymes ainsi que le problème du consensus. La Section 4.3

[1] En réalité $\Omega$ fournit tout de même l'identité d'un processus. $\Omega$ permet par exemple de résoudre l'inintéressant problème "retourner une identité de processus" qui est insolvable dans un système anonyme sans ajout de détecteur de fautes.

définit l'ensemble des détecteurs de fautes étudiés puis la Section 4.4 décrit les relations entre ceux-ci. La Section 4.5 propose deux algorithmes de consensus dans le système anonyme muni de détecteurs de fautes et étudie la notion de plus faible détecteur de fautes dans le cadre des système anonymes. Enfin la Section 4.6 conclut le chapitre en proposant quelques perspectives de recherche.

## 4.2 Modèles et problème

Nous définissons dans cette section les modèles asynchrones anonyme et non-anonyme. Afin d'étudier le réalisme des détecteurs de fautes, une notion définie plus tard, nous précisons aussi les équivalents synchrones de ces deux modèles. Quel que soit le modèle étudié, le problème du consensus quant à lui se définit toujours de la même manière.

### 4.2.1 Modèles

**Les processus** Le système est composé d'un nombre $n$ fixé et connu de processus, noté $p_1, \ldots, p_n$. $\Pi = \{1, \ldots, n\}$ représente l'ensemble des identités des processus. Lorsque le système n'est pas anonyme, chaque processus connaît sa propre identité et celles des autres processus. Lorsque le système est anonyme, chaque processus ignore sa propre identité ainsi que l'ensemble $\Pi$ des identités. Il est important de comprendre que nous considérons ici que les identités existent dans les systèmes anonymes mais sont inconnues et inaccessibles pour les processus.

Le modèle temporel sous-jacent est l'ensemble des naturels $\mathbb{N}$. Les instants sont notés $\tau$, $\tau'$, *etc.* Le temps n'est pas accessible par les processus (tout comme les identités pour le modèle anonyme). Il est utile uniquement d'un point de vue extérieur pour permettre d'énoncer ou prouver des propriétés. Les processus sont asynchrones dans les sens où aucune hypothèse n'est faite sur leurs vitesses relatives ; néanmoins tout processus, non défaillant, progresse. De manière plus générale, excepté la notion d'anonymat le modèle correspond à celui défini dans [26].

**Les pannes** Un processus exécute correctement son algorithme jusqu'à son éventuelle panne. Une panne d'un processus correspond, dans ce chapitre, à un arrêt complet de l'exécution par le processus : il n'y a pas de pannes byzantines, ou de fautes d'émission/réceptions. Un processus qui ne tombe pas en panne lors d'une exécution est appelé *correct* ; autrement il est *fautif*. Tant qu'il n'a pas défailli (si cela arrive), un processus est dit *vivant*. Dans une exécution, $Correct$ représente l'ensemble des processus corrects.

Un *environnement* est un ensemble de motifs de fautes, où un *motif de fautes* (“failure pattern” en anglais dans [26]) est une fonction $F$ de $\mathbb{N} \to 2^{\Pi}$ telle que $F(\tau)$ représente l'ensemble des processus tombés en panne entre l'instant initial et l'instant $\tau$. Nous considérons, dans la majorité de ce chapitre, des motifs de fautes dans lesquels tous les processus sauf un peuvent défaillir. Cet ensemble de motifs est appelé *environnement sans-attente* (“wait-free”

en anglais) car il implique de concevoir des algorithmes permettant à chaque processus de progresser sans attendre des messages des autres processus.

**Les communications** Les processus communiquent en échangeant des messages au moyen de canaux de communications. Ces canaux sont asynchrones mais fiables : aucune hypothèse sur le temps de transmission, mais tout message arrive en temps fini sans altération.

Les processus disposent d'une primitive de diffusion ("broadcast" en anglais) qui leur permet d'envoyer le même message à tous les processus. Cette diffusion n'est pas fiable au sens où, si l'émetteur subit une panne durant l'émission, certains processus peuvent recevoir le message envoyé alors que d'autres non. Dans les systèmes anonymes, la primitive de diffusion est le seul moyen de communication disponible ; les processus ne connaissant pas leurs identités, ils ne peuvent communiquer directement.

**Notations** Nous notons $\mathcal{SAA}[\emptyset]$ le modèle anonyme décrit ci-dessus. $\mathcal{SAA}$ est un acronyme pour Système Asynchrone Anonyme ; $\emptyset$ signifie qu'il n'y a aucune hypothèse supplémentaire. $\mathcal{SA}[\emptyset]$ est utilisé pour décrire le Système Asynchrone (non-anonyme) correspondant [3, 56].

Nous utilisons aussi les deux modèles synchrones associés aux modèles précédents. $\mathcal{SSA}[\emptyset]$ et $\mathcal{SS}[\emptyset]$ désignent, respectivement, le Système Synchrone Anonyme et le Système Synchrone non-anonyme. Dans ceux-ci, les communications et les processus sont synchrones ; cela signifie qu'il est possible de détecter la panne d'un processus lorsque ce dernier ne répond plus. Nous étudions alors dans ces systèmes synchrones le réalisme des détecteurs de fautes. Un détecteur est dit réaliste dans un système asynchrone donné, s'il existe une implémentation de ce détecteur dans le système synchrone associé.

### 4.2.2 Problème du consensus

Le problème du consensus est un problème d'accord dans lequel chaque processus propose une valeur puis décide une valeur, s'il ne subit pas de panne. Formellement, le problème est défini par les trois propriétés suivantes :

- (*Terminaison*) Tout processus correct doit décider une valeur,
- (*Validité*) Toute valeur décidée doit avoir été proposée,
- (*Accord*) Deux processus ne peuvent décider deux valeurs différentes.

Comme mentionné dans l'introduction, ce problème est insolvable dans un système asynchrone (non-anonyme) où la moindre panne peut arriver [35]. C'est la raison pour laquelle la notion de détecteur de fautes à été introduite.

## 4.3 Détecteurs de fautes

Cette section donne une définition formelle des détecteurs de fautes et explicite ceux que nous utilisons dans le chapitre.

### 4.3.1 Définition

Les définitions suivantes, basées sur $\Pi$ l'ensemble des processus et $\mathbb{N}$ l'ensemble des instants, sont issues de [26]. L'*histoire d'un détecteur de fautes de portée* $R$ décrit le comportement d'un détecteur de fautes durant une exécution. C'est une fonction $H : \Pi \times \mathbb{N} \rightarrow R$ où $H(i, \tau)$ correspond à la valeur retournée par le détecteur au processus $p_i$ à l'instant $\tau$. Un *détecteur de fautes* $D$ *de portée* $R$ est une fonction qui, à tout motif de fautes, associe un ensemble d'histoires de détecteur de fautes de portée $R$ ; $D(F)$ correspond à l'ensemble des comportements que le détecteur $D$ peut suivre lorsque le motif de fautes est $F$.

Soient $A$ et $B$ deux détecteurs de fautes. Nous noterons $\mathcal{SAA}[A]$ le système $\mathcal{SAA}[\emptyset]$ enrichi par un détecteur de fautes $A$ ; cela signifie qu'à tout instant les processus peuvent accéder à la valeur retournée par le détecteur de fautes $A$ (cette valeur peut être différente suivant les processus). De même, $\mathcal{SAA}[A, B]$ désigne le système $\mathcal{SAA}[\emptyset]$ enrichi par la paire de détecteurs de fautes $(A, B)$.

### 4.3.2 Détecteurs classiques

Nous présentons ici trois détecteurs de fautes bien connus ; $P$, $\Omega$, et $\Sigma$, ainsi qu'une variante $\overline{P}$. Le premier est réellement un détecteur de fautes au sens littéral ; il propose une information directement reliée aux fautes. $\Omega$ et $\Sigma$ sont aussi appelés des détecteurs de fautes mais le terme *oracle*, parfois utilisé, peut sembler plus pertinent. En effet ces détecteurs fournissent des informations sur le système qui ne sont pas directement liées aux fautes.

**Le détecteur de fautes parfait** $P$ Le détecteur de fautes parfait $P$ [26] fournit à chaque processus $p_i$ un ensemble $suspects_i$ qui contient une liste d'identités de processus suspectés d'être en panne. Ces ensembles doivent vérifier deux propriétés :

- (*Sûreté*) Un processus vivant ne peut être suspecté,
- (*Vivacité*) Ultimement, un processus fautif est suspecté pour toujours par tous les processus corrects.

**Le détecteur de fautes parfait** $\overline{P}$ Le détecteur de fautes parfait $\overline{P}$ correspond intuitivement au complémentaire de $P$. Il fournit à chaque processus $p_i$ un ensemble $vivants_i$ qui contient une liste d'identités de processus "suspectés" d'être vivants. Ces ensembles doivent vérifier deux propriétés :

- (*Sûreté*) Un processus vivant est nécessairement suspecté (d'être vivant),
- (*Vivacité*) Ultimement, un processus fautif n'est jamais plus suspecté (d'être vivant) par tous les processus corrects.

**Le détecteur de fautes** $\Omega$ Le détecteur de fautes $\Omega$ [25], appelé "Leader ultime", fournit à chaque processus $p_i$ une variable $leader_i$ qui contient l'identité d'un processus. Les identités retournées doivent satisfaire la propriété suivante :

- (*Vivacité*) Ultimement, la même identité est, pour toujours, retournée à tous les processus corrects. Cette identité est celle d'un processus correct.

**Le détecteur de fautes $\Sigma$** Le détecteur de fautes $\Sigma$ [44, 60], appelé "Détecteur quorum", fournit à chaque processus $p_i$ un ensemble $quorum_i$ qui contient une liste d'identités de processus. Ces ensembles doivent vérifier deux propriétés :

- (*Vivacité*) Ultimement, tout quorum ne contient que des identités de processus corrects,
- (*Sûreté*) Toute paire de quorums retournés un jour par $\Sigma$ s'intersectent.

### 4.3.3 Détecteurs "anonymes"

Les quatre détecteurs présentés ci-dessus ont une caractéristique commune, leurs sorties contiennent des identités de processus du système. De part cette caractéristiques, ces détecteurs ne sont, a priori, pas destinés à être utilisés dans des systèmes anonymes. Nous proposons ici des détecteurs spécialement conçus pour les systèmes anonymes. Dans ce sens là, ces détecteurs de fautes sont "anonymes" ; néanmoins rien, dans leurs définitions, ne les distingue réellement des détecteurs classiques. Ils peuvent être ajoutés aussi bien à un système anonyme qu'à un système non-anonyme.

Les trois premiers correspondent à des détecteurs déjà étudiés. Le dernier, quant à lui, constitue une nouveauté intéressante. C'est la raison pour laquelle nous le détaillons un peu plus.

**Le détecteur de fautes parfait $AP$** Le détecteur de fautes parfait $AP$ (introduit sous différents noms dans [18, 58, 59] fournit à chaque processus $p_i$ un entier $nb\_suspects_i$ qui donne une estimation du nombre de processus en panne. Intuitivement cela correspond au cardinal de l'ensemble $suspect_i$ retourné par le détecteur $P$. Les entiers $nb\_suspects$ doivent vérifier deux propriétés :

- (*Sûreté*) Le nombre de suspects $nb\_suspects$ est toujours inférieur ou égal au nombre réel de processus en panne,
- (*Vivacité*) Ultimement, le nombre de suspects égale, pour toujours, le nombre de processus fautifs.

**Le détecteur de fautes parfait $\overline{AP}$** Tout comme pour $\overline{P}$, le détecteur de fautes parfait $\overline{AP}$ correspond intuitivement au complémentaire de $AP$. Il fournit à chaque processus $p_i$ un entier $nb\_vivants_i$ qui donne une estimation du nombre de processus vivants. Les entiers $nb\_vivants$ doivent vérifier deux propriétés :

- (*Sûreté*) Le nombre de suspectés vivants $nb\_vivants$ est toujours supérieur ou égal au nombre réel de processus vivants,
- (*Vivacité*) Ultimement, le nombre de suspectés vivants égale, pour toujours, le nombre de processus vivants.

**Le détecteur de fautes $A\Omega$** Le détecteur de fautes $A\Omega$ [46] fournit à chaque processus $p_i$ une variable booléenne $est\_leader_i$ qui indique au processus s'il est ou non leader. La propriété suivante doit être satisfaite :

– (*Vivacité*) Ultimement, il existe un processus correct dont le booléen *est_leader* vaut *vrai* pour toujours et les booléens des autres processus valent $faux$ pour toujours.

**Le détecteur de fautes** $A\Sigma$ Le détecteur de fautes $A\Sigma$ fournit à chaque processus $p_i$ un ensemble $liste\_quorums_i$ de couples d'entiers. Intuitivement et informellement, lorsque la couple $(x, y)$ apparaît dans l'ensemble $liste\_quorums_i$ d'un processus $p_i$, cela signifie que le détecteur $A\Sigma$ informe $p_i$ qu'il appartient à un quorum étiqueté $x$ dans lequel $y$ processus participent. Formellement, les ensembles *liste_quorums* doivent satisfaire les propriétés ci-dessous ($liste\_quorum_i^\tau$ correspond à la sortie du détecteur $A\Sigma$ pour le processus $p_i$ à l'instant $\tau$).

**Définition préliminaire** $S(x) = \{i \mid \exists \tau \in \mathbb{N}, (x, -) \in liste\_quorums_i^\tau\}$[2].

**Définition formelle de** $A\Sigma$

– (*Validité*) La sortie du détecteur de fautes $A\Sigma$ est un ensemble de couples d'entiers dans lequel il n'existe pas deux éléments ayant la même première composante :
$\forall\ i \in \Pi \quad \forall\ \tau \in \mathbb{N} \quad liste\_quorums_i^\tau = \{(x_1, y_1), \ldots, (x_m, y_m)\}$
où $\forall\ 1 \leq a, b \leq m \quad (x_a, y_b \in \mathbb{N})\ \wedge\ \big((a \neq b) \Rightarrow (x_a \neq x_b)\big)$.

– (*Monotonie*) Dans un ensemble *liste_quorums*, étant donnée une première composante ; lorsqu'un couple apparaît avec cette composante, il apparaît pour toujours et la deuxième composante ne peut que décroître :
$\forall i \in \Pi \quad \forall \tau \in \mathbb{N} \quad (x, y) \in liste\_quorums_i^\tau \Rightarrow \big(\forall \tau' \geq \tau \quad \exists y' \leq y, (x, y') \in liste\_quorums_i^{\tau'}\big)$.

– (*Vivacité*) Ultimement, il existe au moins un "bon" quorum dans l'ensemble *liste_quorums* :
$\forall i \in Correct \quad \exists (x, y), \exists \tau, \forall\ \tau' \geq \tau$
$\big((x, y) \in liste\_quorums_i^{\tau'}\big) \wedge \big(|S(x) \cap Correct| \geq y\big)$.

– (*Sûreté*) Toute paire de "bons" quorums s'"intersectent" :
$\forall\ i_1, i_2 \in \Pi \quad \forall\ \tau_1, \tau_2 \in \mathbb{N} \quad \forall\ (x_1, y_1) \in liste\_quorums_{i_1}^{\tau_1} \quad \forall\ (x_2, y_2) \in liste\_quorums_{i_2}^{\tau_2}$
$\forall\ T_1 \subseteq S(x_1) \quad \forall\ T_2 \subseteq S(x_2) \quad \big((|T_1| = y_1)\ \wedge\ (|T_2| = y_2)\big) \Rightarrow (T_1 \cap T_2 \neq \emptyset)$.

**Explications** Dans les deux dernières propriétés, nous faisons référence à des "bons" quorums. Un "bon" quorum pour le détecteur $A\Sigma$ désigne un quorum étiqueté $x$, issu d'un couple $(x, y)$ donné par $A\Sigma$, tel qu'il existe au moins $y$ processus à qui le détecteur fournit le couple $(x, -)$. De manière similaire au détecteur classique $\Sigma$, toute paire de "bons" quorums doit s'intersecter, ce qui est exprimé formellement par la propriété de sûreté.

[2]Ces ensembles $S(x)$ ne sont pas connus et accessibles par les processus ; ils servent uniquement à la définition du détecteur de fautes $A\Sigma$.

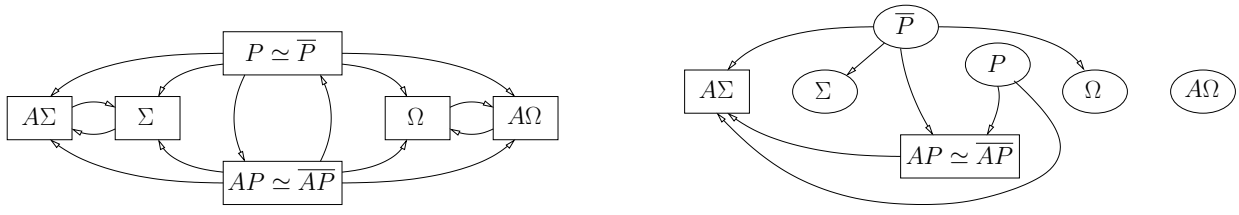
## 4.4 Réductions entre détecteurs de fautes

**Définitions** Ces définitions proviennent de [26]. Étant donnés deux détecteurs de fautes $D_1$ et $D_2$, et un modèle $\mathcal{C}$ du système ($\mathcal{SAA}$ ou $\mathcal{SA}$), nous disons que $D_1$ est *plus faible* que $D_2$ dans $\mathcal{C}$ (noté $D_1 \preceq_{\mathcal{C}} D_2$) s'l existe un algorithme capable d'émuler la sortie du détecteur $D_1$ dans $\mathcal{C}[D_2]$. Si la réduction existe dans les deux sens (*i.e.*, $D_1 \preceq_{\mathcal{C}} D_2$ et $D_2 \preceq_{\mathcal{C}} D_1$), $D_1$ et $D_2$ sont dit *équivalents* dans $\mathcal{C}$ ; cela est noté $D_1 \simeq_{\mathcal{C}} D_2$. Si la réduction n'existe que dans un seul sens, (*i.e.*, $D_1 \preceq_{\mathcal{C}} D_2$ et $D_2 \not\preceq_{\mathcal{C}} D_1$), $D_1$ est *strictement plus faible* que $D_2$ ; cela est noté $D_1 \prec_{\mathcal{C}} D_2$. De manière similaire nous définissons les notions de *plus fort* et de *strictement plus fort* qui correspondent aux relations opposées. Enfin les détecteurs $D_1$ et $D_2$ sont *incomparable* s'il n'existe aucune réduction de l'un vers l'autre.

**Remarques** Il est important de relever que l'existence de réductions entre deux détecteurs de fautes dépend du modèle. C'est la raison pour laquelle nous avons choisi de ne pas utiliser la notion habituelle de *classe de détecteurs de fautes*. En effet deux détecteurs de fautes peuvent être équivalents dans un modèle donné (donc appartenir à la même classe), mais incomparables dans un autre modèle. Plus généralement, comme la seule différence entre $\mathcal{SAA}$ et $\mathcal{SA}$ est la non-connaissance des identités, nous avons :

- $\forall D_1, D_2 \quad (D_1 \preceq_{\mathcal{SAA}} D_2) \Rightarrow (D_1 \preceq_{\mathcal{SA}} D_2)$,
- $\exists D_1, D_2, \quad (D_1 \preceq_{\mathcal{SA}} D_2) \wedge (D_1 \not\preceq_{\mathcal{SAA}} D_2)$.

**Résultats** La suite de cette section consiste à montrer les réductions représentés sur la Figure 4.1. Un arc d'un détecteur $D_1$ vers un détecteur $D_2$ est dessiné si et seulement s'il existe une réduction de $D_1$ vers $D_2$, autrement dit si $D_2$ est plus faible que $D_1$. L'absence d'arc entre deux sommets signifie explicitement que les deux détecteurs sont incomparables. Les résultats présentés ici sont génériques ; nous ne considérons pas les cas pathologiques pour lequel il peut exister d'autres relations entre détecteurs de fautes : typiquement lorsque $n = 1$, ces détecteurs sont presque tous équivalents (et inutiles).




(a) dans $\mathcal{SA}$ (b) dans $\mathcal{SAA}$

FIG. 4.1 – Réductions entre détecteurs de fautes

### 4.4.1 Réductions simples dans le modèle non-anonyme

Les relations entre les détecteurs classiques sont déjà connues dans le système non-anonyme. Nous rappelons simplement les résultats suivants [25, 28] :

- $\Omega$ est strictement plus faible que $P$ dans $\mathcal{SA}$,
- $\Sigma$ est strictement plus faible que $P$ dans $\mathcal{SA}$,
- $\Omega$ et $\Sigma$ ne sont pas comparables dans $\mathcal{SA}$.

Par ailleurs, les détecteurs $P$ et $\overline{P}$ d'une part, et $AP$ et $\overline{AP}$ d'autre part, sont trivialement équivalents. Les sorties des détecteurs $P$ et $\overline{P}$ vérifient des propriétés similaires : étant donné un de ces deux détecteurs, il est possible d'émuler le second en prenant simplement pour sortie le complémentaire de la sortie du premier, par rapport à l'ensemble $\Pi$ des processus. De manière identique, il est possible d'émuler la sortie de $\overline{AP}$ (resp. $AP$) en soustrayant la sortie de $AP$ (resp. $\overline{AP}$) au nombre $n$ de processus.

Pour le modèle non-anonyme, il reste à démontrer les trois équivalences entre les détecteurs classiques et leurs homologues "anonymes".

### 4.4.2 Équivalence entre $P$ et $AP$ dans $\mathcal{SA}$

**Construction de $AP$ dans $\mathcal{SA}[P]$** La réduction est triviale dans cette direction et ne nécessite aucun échange de message. Localement, chaque processus $p_i$ calcule simplement le cardinal de l'ensemble $suspects_i$ retourné par $P$. Ce cardinal émule la sortie $nb\_suspects_i$ du détecteur $AP$.

**Construction de $P$ dans $\mathcal{SA}[AP]$** La Figure 4.2 propose une réduction de $AP$ vers $P$ dans le modèle non-anonyme. L'idée principale consiste à échanger des messages afin de détecter l'identité des processus en panne. Le détecteur $AP$ renseigne chaque processus sur le nombre de pannes ; il suffit de demander périodiquement aux autres processus une réponse. L'algorithme présenté est légèrement plus astucieux, puisqu'il est silencieux ; au bout d'un certain temps, lorsque toutes les pannes ont eu lieu, les processus stoppent tout échange de message. L'algorithme se décompose en quatre tâches exécutées en parallèle :

1. La première tâche détecte les mises à jour du nombre de suspects fourni par le détecteurs de fautes $AP$. Lorsque le processus relève une augmentation du nombre de suspects, il initie une nouvelle phase qui consiste à envoyer une requête à tous les processus.
2. Lorsque $p_i$ reçoit une requête d'un processus $p_j$, il répond à $p_j$ qu'il est toujours vivant.
3. Lorsque $p_i$ reçoit une réponse de la part de $p_j$, il met à jour son tableau de réponses. Cela signifie qu'au moment de cette requête, le processus $p_j$ était encore vivant.
4. La dernière tâche calcule réellement l'ensemble $suspects_i$ qui émule la sortie du détecteur $P$. Elle consiste simplement à chercher les identités des processus n'ayant pas répondu à la dernière requête lancée. Si ce nombre égale exactement le nombre de pannes, cela signifie que les identités des processus en panne sont précisément connues. Cette tâche $T4$ peut nécessiter plusieurs exécutions successives. Il est possible qu'après une

exécution, le prédicat ($|suspects_i| < k_i$) soit toujours *vrai* ; cela signifie que tous les processus vivants n'ont pas encore répondu à la dernière requête.

**Init :** $k_i \leftarrow 0$ ; $reponses_i \leftarrow [0, \cdots, 0]$ ; $suspects_i \leftarrow \emptyset$.

(1) $T1$ : **lorsque** ($nb\_suspects_i > k_i$) : $k_i \leftarrow nb\_suspects_i$ ; diffuser REQUETE($k_i$).

(2) $T2$ : **lorsque** REQUETE($k$) **est reçu de** $p_j$ : envoyer VIVANT($k$) à $p_j$.

(3) $T3$ : **lorsque** VIVANT($k$) **est reçu de** $p_j$ : $reponses_i[j] \leftarrow \max(reponses_i[j], k)$.

(4) $T4$ : **lorsque** ($|suspects_i| < k_i$) :
(5) $k'_i \leftarrow k_i$ ; % $k'_i$ est local à $T4$, contrairement à $k_i$ %
(6) $X_i \leftarrow \{x \mid reponses_i[x] < k'_i\}$ ;
(7) **si** ($|X_i| = k'_i$) **alors** $suspects_i \leftarrow X_i$ **fin si**.

FIG. 4.2 – Construction de $P$ dans $\mathcal{SA}[AP]$ (code pour $p_i$)

**Théorème 4.1** *L'algorithme présenté en Figure 4.2 est une construction silencieuse de $P$ dans $\mathcal{SA}[AP]$.*

**Preuve** L'algorithme présenté en Figure 4.2 construit un ensemble $suspects_i$ contenant des identités de processus. Il faut vérifier que cet ensemble respecte les propriétés de sûreté et de vivacité définissant $P$.

*Sûreté :* Soit $p_i$ un processus. Nous devons montrer qu'aucun processus n'est ajouté à l'ensemble $suspects_i$ avant de défaillir. L'ensemble $suspects_i$ est mis à jour uniquement à la Ligne 7. Ce changement ne peut avoir lieu que si l'ensemble $X_i$ est de cardinal $k'_i$. Or l'ensemble $X_i$ correspond à l'ensemble des processus n'ayant pas (encore) répondu à la $k'_i$-ème requête du processus $p_i$. Cette requête n'étant effectuée que lorsqu'il y a déjà eu $k'_i$ pannes (sûreté de $AP$), l'ensemble $X_i$ ne peut être de cardinal $k'_i$ que lorsqu'il ne contient que les $k'_i$ processus ayant défailli avant l'envoi de la requête par $p_i$. Cela implique que tout processus appartenant à l'ensemble $suspects_i$ est nécessairement en panne.

*Vivacité :* Soit $p_i$ un processus correct. Nous devons montrer qu'un processus fautif apparaît ultimement de manière permanente dans l'ensemble $suspects_i$. Considérons un instant $\tau$ après lequel tous les processus fautifs ont subit leur panne. Soit $f$ le nombre de processus fautifs. Par sa propriété de vivacité, le détecteur $AP$ doit, ultimement, informer $p_i$ qu'il y a $nb\_suspects_i = f$ processus fautifs. La tâche $T1$ met alors à jour la variable $k_i$ à la valeur $f$ puis diffuse une nouvelle requete avec la valeur $k_i = f$. Les tâches $T2$ et $T3$ garantissent que $p_i$ reçoit une réponse à sa requete de la part des $n - f$ processus corrects ; le tableau $reponses_i$ contient alors la valeur $k_i$ pour toutes les entrées des processus corrects et une valeur inférieure pour les entrées des processus fautifs. Finalement, la tâche $T4$ détecte précisément les $f$ processus fautifs et les ajoute de manière permanente dans l'ensemble $suspects_i$.

*Construction silencieuse :* Lorsque tous les processus fautifs sont tombés en panne et lorsque tous les processus corrects ont initié puis répondu à toutes les requêtes, plus aucun message n'est échangé par l'algorithme. Cet algorithme est silencieux, mais ne termine néanmoins jamais : les processus stoppent tout envoi de messages, mais ne sont pas capables de le détecter. □

### 4.4.3 Équivalence entre $\Omega$ et $A\Omega$ dans $\mathcal{SA}$

**Construction de $A\Omega$ dans $\mathcal{SA}[\Omega]$** La réduction est triviale dans cette direction et ne nécessite aucun échange de message. Localement, chaque processus évalue le prédicat $leader_i = i$ où $leader_i$ désigne la sortie du détecteur $\Omega$. La valeur de ce prédicat émule la sortie $est\_leader_i$ du détecteur $A\Omega$.

**Construction de $\Omega$ dans $\mathcal{SA}[A\Omega]$** La réduction est simple dans cette direction mais nécessite néanmoins des échanges de messages. Excepté pour le leader ultime, cette construction est silencieuse ; seul le leader ultime envoie un nombre infini de messages.

Chaque processus $p_i$ exécute deux tâches : (1) $p_i$ observe périodiquement le booléen $est\_leader_i$ fourni par $A\Omega$ et s'il vaut $vrai$, il diffuse un message LEADER$(i)$ ; (2) lorsque $p_i$ reçoit un message LEADER$(k)$, il met à jour sa variable $leader_i$ à $k$. Cette variable émule la sortie du détecteur $\Omega$. La preuve (simple) est laissée aux éventuels lecteurs consciencieux.

### 4.4.4 Équivalence entre $\Sigma$ et $A\Sigma$ dans $\mathcal{SA}$

**Construction de $A\Sigma$ dans $\mathcal{SA}[\Sigma]$** Puisque nous considérons le modèle non-anonyme, tous les processus connaissent l'ensemble des processus $\Pi$. Il est alors possible de définir une fonction $\mathsf{nom}$ qui associe un unique entier à tout sous-ensemble de $\Pi$. Formellement cette fonction $\mathsf{nom}$ est une bijection et vérifie donc les deux propriétés suivantes :

- $\forall Q \subseteq \Pi \quad \mathsf{nom}(Q) \in \{1, \ldots, 2^{|\Pi|}\}$,
- $\forall Q, Q' \subseteq \Pi \quad (Q \neq Q') \Rightarrow (\mathsf{nom}(Q) \neq \mathsf{nom}(Q'))$.

Avec l'aide de cette fonction $\mathsf{nom}$, il est maintenant possible de construire un détecteur $A\Sigma$ à partir d'un détecteur $\Sigma$. Une telle réduction est proposée sur la Figure 4.3. Celle-ci se décompose en deux tâches :

1. La première tâche détecte si le quorum $quorum_i$ fourni par $\Sigma$ est un nouveau quorum. Si tel est le cas, le processus $p_i$ diffuse l'existence de ce quorum. L'utilisation de l'ensemble $envois_i$ permet de se rappeler des quorums précédents et d'éviter les envois multiples des mêmes quorums. Cela permet par ailleurs à l'algorithme de réduction d'être silencieux ; le nombre de quorums différents étant fini, le nombre de messages envoyés l'est aussi.
2. La deuxième tâche permet l'émulation de la sortie $liste\_quorums_i$ du détecteur $A\Sigma$. À chaque réception d'un quorum $qr$, le processus $p_i$ vérifie s'il appartient à ce quorum. Si tel est le cas, il ajoute le couple $(\mathsf{nom}(qr), |qr|)$ à son ensemble $liste\_quorums_i$. Du

point de vue du détecteur de fautes émulé $A\Sigma$, cela signifie que le processus $p_i$ participe au quorum nommé $\mathsf{nom}(qr)$ formé de $|qr|$ processus.

**Init :** $liste\_quorums_i \leftarrow \{(\mathsf{nom}(\Pi), |\Pi|)\}$ ; $envois_i = \{\Pi\}$ ;

(1) $T1$ : **répéter**
(2) $qr_i \leftarrow quorum_i$ ;
(3) **si** $(qr_i \notin envois_i)$ **alors** diffuser QUORUM$(qr_i)$ ; $envois_i \leftarrow envois_i \cup \{qr_i\}$ **fin si**
(4) **fin répéter**.

(5) $T2$ : **lorsque** QUORUM$(qr)$ **est reçu** :
(6) **si** $(i \in qr)$ **alors** $liste\_quorums_i \leftarrow liste\_quorums_i \cup \{(\mathsf{nom}(qr), |qr|)\}$ **fin si**.

FIG. 4.3 – Construction de $A\Sigma$ dans $\mathcal{SA}[\Sigma]$ (code pour $p_i$)

**Théorème 4.2** *L'algorithme présenté en Figure 4.3 est une construction silencieuse de $A\Sigma$ dans $\mathcal{SA}[\Sigma]$.*

**Preuve** L'algorithme présenté en Figure 4.3 construit un ensemble $liste\_quorums_i$ contenant des couples d'entiers. Il faut vérifier que cet ensemble respecte les propriétés de validité, monotonie, sûreté, et vivacité définissant $A\Sigma$.

*Validité et Monotonie :* Ces deux propriétés découlent immédiatement de celles définissant la fonction $\mathsf{nom}$ : pour chaque entier $x$, il n'existe qu'un seul sous-ensemble $Q$ de $\Pi$ tel que $\mathsf{nom}(Q) = x$.

*Vivacité :* Soit $p_i$ un processus correct. Nous devons montrer $\exists(x,y), \exists\tau, \forall\ \tau' \geq \tau\ \big((x,y) \in liste\_quorums_i^{\tau'}\big) \wedge \big(|S(x) \cap Correct| \geq y\big)$. Pour cela considérons un instant $\tau$ après lequel le détecteur de fautes $\Sigma$ ne fournit que des ensembles contenant uniquement des processus corrects (cet instant existe grâce à la propriété de vivacité de $\Sigma$). Soit $qr_i$ un quorum particulier obtenu par $p_i$ du détecteur $\Sigma$ après l'instant $\tau$. La tâche $T1$ garantit que ce quorum $qr_i$ est diffusé (s'il ne l'a pas déjà été) à tous les processus. Puisque $qr_i$ ne contient que des processus corrects (hypothèse sur $\tau$), tous les processus de $qr_i$ reçoivent le message QUORUM$(qr_i)$ et leur tache $T2$ assure que le couple $(\mathsf{nom}(qr_i), |qr_i|)$ termine dans leur ensemble $liste\_quorums$. Ainsi $S(\mathsf{nom}(qr_i)) = qr_i \subseteq Correct$ et par conséquent le couple $(x,y) = (\mathsf{nom}(qr_i), |qr_i|)$ vérifie la propriété de vivacité pour le processus $p_i$.

*Sûreté :* Nous devons montrer que toute paire de "bons" quorums s'"intersectent", comme énoncé dans la propriété de sûreté de $A\Sigma$. Soient $(x_1, y_1)$ et $(x_2, y_2)$ deux paires de couples appartenant, un jour, à un ensemble $liste\_quorums$ d'un processus (potentiellement, à des instants et des processus différents). Puisque le couple $(x_1, y_1)$ a été ajouté dans un ensemble $liste\_quorums$, cela signifie que, pour au moins un processus, la tâche $T2$ a reçu un message QUORUM$(qr_1)$ tel que $\mathsf{nom}(qr_1) = x_1$, cela signifie que le quorum $qr_1$ a, un jour, été fournit par le détecteur $\Sigma$ à un processus. Il en est de même pour un quorum $qr_2$ tel que $\mathsf{nom}(qr_2) = x_2$. Par ailleurs, la tâche $T2$ garantir que seuls les processus appartenant à $qr_1$ ajoutent le couple $(x_1, y_1)$ à leur ensemble $liste\_quorums$ ; ce qui assure que $S(x_1) \subseteq qr_1$. Par conséquent, tel

que défini dans la propriété de sûreté de $A\Sigma$, tout sous-ensemble $T_1$ de $S(x_1)$ de cardinal $y_1 = |qr_1|$ ne peut être que $T_1 = qr_1$. Il en est de même pour un sous-ensemble $T_2$ de $S(x_2)$ de cardinal $y_2 = |qr_2|$ ; s'il en existe un, c'est nécessairement $T_2 = qr_2$. La propriété de sûreté de $\Sigma$ garantit que les deux ensembles $qr_1$ et $qr_2$ s'intersectent ; il en est donc de même pour $T_1$ et $T_2$, ce qui valide la propriété de sûreté.

*Construction silencieuse :* Le nombre de processus est fini, égal à $n$. Le nombre d'ensemble de processus est donc lui aussi fini. Puisque chaque processus envoie au maximum un message pour chaque ensemble de processus, le nombre de messages envoyés est fini. L'algorithme est silencieux. □

**Construction de $\Sigma$ dans $\mathcal{SA}[A\Sigma]$** Cette réduction est la plus complexe parmi celles que nous présentons ; elle est aussi très intéressantes par les différents mécanismes qui entrent en jeu. Une des principales difficultés consiste à passer d'un ensemble de couples (désignant des quorums) donné par $A\Sigma$ à un quorum unique, comme cela est requis pour émuler $\Sigma$. L'algorithme de réduction présenté sur la Figure 4.4 repose sur deux structures de données :

- $vivants_i$ est une file ordonnant les processus suivant la dernière fois où $p_i$ a reçu un message de leur part : quand $p_i$ reçoit un message de $p_j$ il supprime l'identité $j$ de la file et la replace en première position. Ce mécanisme permet de garantir que, ultimement, les identités des processus corrects apparaîssent toujours dans la file avant les identités des processus fautifs.
- $files_i$ est un tableau de files tel que $files_i[x]$ contient les identités des processus qui, selon $p_i$, participent au quorum désigné par le couple $(x, -)$ fourni par le détecteur $A\Sigma$. ($files_i$ contient donc un sous-ensemble de $S(x)$.)

Avec l'aide de ces deux structures de données, notre algorithme se décompose en trois tâches exécutées simultanément par tous les processus :

1. La tâche $T1$ consiste en une boucle infinie dans laquelle chaque processus $p_i$ diffuse périodiquement un message VIVANT$(i, etiquettes_i)$, indiquant d'une part qu'il est toujours vivant, et fournissant d'autre part la liste des étiquettes des quorums donnés par $A\Sigma$.
2. La tâche $T2$ traitent les messages envoyés par les tâches $T1$. Lorsque $p_i$ reçoit un message VIVANT$(j, etiquettes)$, il met à jour la file $vivants_i$ en plaçant $j$ en tête car c'est le dernier processus dont il a reçu un message. $p_i$ actualise aussi toutes les files $files_i[x]$ pour les étiquettes $x$ de quorums qui (i) appartiennent à la liste $etiquettes$ du message reçu, et (ii) correspondent à des quorums qu'il reçoit directement du détecteur $A\Sigma$. Cette actualisation consiste, comme pour la file $vivants_i$ à placer l'identité $j$ en tête des files $files_i[x]$.
3. La tâche $T3$ constitue le cœur de l'émulation. Elle consiste en une boucle infinie qui détermine l'émulation de la sortie $quorum_i$ du détecteur $\Sigma$. Cette objectif est atteint en deux étapes :
   - La première phase (Ligne 13) définit une liste de candidats potentiels pouvant permettre l'émulation de $\Sigma$. Quelque soit le candidat choisi pour émuler $\Sigma$, la propriété

de sûreté de ce dernier sera garantie. Les candidats correspondent aux couples $(x, y)$, fournis par $A\Sigma$, tels que le processus $p_i$ connaît au moins $y$ identités de processus participant au quorum nommé $x$, c'est-à-dire $|files_i[x]| \geq y$.

– La deuxième phase (Lignes 15 à 17) permet de choisir le meilleur candidat pour garantir la propriété de vivacité de $\Sigma$. L'idée est de sélectionner le candidat $(x, y)$ qui apparaît globalement devant les autres dans la files $vivants_i$. Une fois le meilleurs candidat sélectionné, il suffit de prendre les $y$ processus participant au candidat $(x, y)$ dont $p_i$ a reçu des messages le plus récemment (Ligne 18).

Afin de comprendre plus facilement la tâche $T3$, nous proposons au lecteur un exemple. Supposons que $vivants_i = [7, 1, 3, 9, 4, 8, 2, 5, 6]$ ; cela signifie que le processus $p_i$ a reçu en dernier un message de $p_1$ puis un message de $p_7$. Considérons ensuite que $liste\_quorum_i = \{(5, 4), (7, 3), (2, 5)\}$ et que les seules entrés utilisées du tableau $files$ sont les suivantes : $files_i[5] = [1, 3, 4, 2, 5]$, $files_i[7] = [1, 8, 5]$, et $files_i[2] = [1, 5]$. Parmi les trois couples donnés par $A\Sigma$, seuls les couples $(5, 4)$ et $(7, 3)$ sont des candidats potentiels ; en effet $|files_i[2]| < 5$, donc $(2, 5)$ n'est pas un candidat valide. Parmi les deux candidats, nous avons $rang(files_i[5][4]) = rang(2) = 7$ et $rang(files_i[7][3]) = rang(5) = 8$ ; le meilleur candidat est alors le couple $(5, 4)$. L'émulation de $\Sigma$ s'obtient finalement en prenant les 4 premiers éléments de la files $files_i[5]$ c'est-à-dire $quorum_i \leftarrow \{1, 3, 4, 2\}$.

```
Init : vivants_i ← toutes les identités de processus, dans un ordre arbitraire ;
       pour tout x faire queue_i[x] ← file vide fin pour.

(01) T1 : répéter
(02)         etiquettes_i ← {x | (x, −) ∈ liste_quorums_i} ;
(03)         diffuser VIVANT(i, etiquettes_i)
(04)       fin répéter.

(05) T2 : lorsque VIVANT(j, etiquettes) est reçu :
(06)         supprimer j de vivants_i ; ajouter j en tête de vivants_i ;
(07)         pour tout x ∈ etiquettes tel que ((x, −) ∈ liste_quorums_i) faire
(08)               si (j ∈ files_i[x]) alors supprimer j de files_i[x] fin si ;
(09)               ajouter j en tête de files_i[x]
(10)         fin pour.

(11) T3 : répéter
(12)         soit candidats = { (x, y) | (x, y) ∈ liste_quorums_i ∧ |files_i[x]| ≥ y } ;
(13)         si (candidats = ∅)
(14)            alors quorum_i ← Π
(15)            sinon soit r_min = min_{(x,y)∈candidats} (rang(files_i[x][y]))
(16)                                  où rang(ℓ) = rang de ℓ dans la file vivants_i ;
(17)                  soit (x, y) ∈ candidats tel que rang(files_i[x][y]) = r_min ;
(18)                  quorum_i ← les y premiers éléments de files_i[x]
(19)         fin si
(20)       fin répéter.
```

FIG. 4.4 – Construction de $\Sigma$ dans $\mathcal{SA}[A\Sigma]$ (code pour $p_i$)

**Théorème 4.3** *L'algorithme présenté en Figure 4.4 est une construction de* $\Sigma$ *dans* $\mathcal{SA}[A\Sigma]$.

**Preuve** L'algorithme présenté en Figure 4.4 construit un ensemble $quorums_i$ contenant des identités de processus. Il faut vérifier que cet ensemble respecte les propriétés de sûreté et de vivacité définissant $\Sigma$.

*Sûreté :* Nous devons montrer que toute paire de quorums $quorums_i$ et $quorums_j$ s'intersectent, ces ensembles étant calculés par deux processus $p_i$ et $p_j$ (éventuellement identiques) à deux instants quelconques. Remarquons tout d'abord que l'algorithme garantit que les ensembles $quorums_i$ calculés ne sont jamais vides ; si aucun candidat n'est disponible (Ligne 13) alors l'émulation du détecteur $\Sigma$ consiste à prendre $\Pi$ pour quorum. Ce cas ne pose pas de problème car $\Pi$ s'intersecte obligatoirement avec tout quorum non vide.

Considérons alors deux quorums $quorums_1$ et $quorums_2$ calculés à la Ligne 18. Le quorum $quorum_1$ est calculé par un processus $p_{i_1}$ et provient des $y_1$ premiers éléments d'une file $files_{i_1}[x_1]$ pour un couple $(x_1, y_1)$ donné par le détecteur $A\Sigma$. La construction de la file $files_{i_1}[x_1]$ par $p_{i_1}$ (tâches $T1$ et $T2$) garantit que $files_{i_1}[x_1] \subseteq S(x_1)$ et donc par conséquent $quorum_1 \subseteq S(x_1)$. De manière similaire, il existe un couple $(x_2, y_2)$ tel que $quorum_2 \subseteq S(x_2)$ et tel que le cardinal de $quorum_2$ vaut $y_2$. D'après la propriété de sûreté de $A\Sigma$, en prenant $T_1 = quorum_1$ et $T_2 = quorum_2$, nous obtenons la propriété recherché $quorum_1 \cap quorum_2 \neq \emptyset$.

*Vivacité :* Nous devons montrer qu'ultimement, tout $quorum_i$ calculé par des processus corrects ne contient que des processus corrects. Considérons un instant $\tau_0$ après lequel (i) tous les processus fautifs ont subit leur panne, (ii) tous les messages envoyés par les processus fautifs ont été reçus, et (iii) tout processus correct a reçu au moins un message VIVANT de chaque processus correct après avoir reçu ceux des processus fautifs. Un tel temps $\tau_0$ existe car les processus fautifs diffusent un nombre fini de messages VIVANT alors que les processus corrects en diffusent un nombre infini. Après cet instant $\tau_0$, pour tout processus correct $p_i$, les processus corrects apparaissent donc tous avant les processus fautifs dans la file $vivants_i$ (Lignes 3 et 6).

De plus, grâce à la propriété de vivacité du détecteur $A\Sigma$, il existe un instant $\tau_1$ après lequel, pour tout processus correct $p_i$, il existe un couple $(x_i, y_i) \in liste\_quorums_i$ tel que $|S(x_i) \cap Correct| \geq y_i$. Considérons alors un instant $\tau \geq \max(\tau_0, \tau_1)$, un processus correct $p_i$, et un couple $(x_i, y_i)$ vérifiant les conditions précédentes.

Puisqu'il y a au moins $y_i$ processus dans l'ensemble $S(x_i) \cap Correct$, ultimement, au moins $y_i$ processus envoient périodiquement un message VIVANT$(-, etiquettes)$ où l'ensemble $etiquettes$ contient la valeur $x_i$. Ces $y_i$ messages sont reçus par $p_i$ et traités par la tâche $T2$ : la file $files_i[x_i]$ contient, ultimement, au moins $y_i$ processus et les $y_i$ premiers sont des processus corrects. Le couple $(x_i, y_i)$ devient alors ultimement un candidat (Ligne 12) pour le processus $p_i$. Une fois devenu un candidat, ce couple $(x_i, y_i)$ ne peut plus perdre ce statut ; la condition (Ligne 12) reste vraie pour toujours[3].

Si le meilleur candidat correspond au couple $(x_i, y_i)$, le $quorum_i$ généré ne contient que des processus corrects ; en effet nous avons vu que les $y_i$ premiers processus de la file $files_i[x_i]$

[3]Éventuellement la valeur de $y_i$ peut décroître, d'après la propriété de monotonie de $A\Sigma$, mais il restera toujours un couple $(x_i, -)$ candidat.

sont des processus corrects. Si un autre candidat est sélectionné, cela signifie que ce dernier est "globalement situé avant" dans la liste $vivants_i$ (Lignes 14 à 16) ; or nous avons vu que la file $vivants_i$ contient en premières positions les processus corrects. Cela implique donc que le quorum obtenu à partir du candidat sélectionné contient uniquement des processus corrects, tout comme le quorum obtenu à partir du candidat $(x_i, y_i)$. Nous obtenons ainsi la propriété de vivacité recherchée. □

### 4.4.5 Réductions dans le modèle anonyme

Certaines relations existant dans le modèle non-anonyme (sur la Figure 4.1(a)) ne sont plus valides dans le modèle anonyme. Intuitivement l'absence d'identités limite l'émulation des processus classiques $P$, $\Sigma$, ou $\Omega$. Cependant certaines réductions demeurent possibles, même si les processus ne connaissent pas leurs identités. Il est par ailleurs intéressant de remarquer que des détecteurs étant trivialement équivalents dans le modèle non-anonyme ($P$ et $\overline{P}$) ne le sont plus dans le modèle anonyme : ils ne se réduisent pas non plus aux mêmes détecteurs comme il est résumé sur la Figure 4.1(b).

**Réductions possibles** Les réductions suivantes étant relativement simples, nous donnons simplement quelques explications permettant de les comprendre :

- $AP \prec_{\mathcal{SAA}} P$. La réduction de $P$ vers $AP$ est identique à celle du modèle non-anonyme. Localement chaque processus calcule le cardinal de l'ensemble $suspects_i$ retourné par $P$.
- $\overline{AP} \prec_{\mathcal{SAA}} \overline{P}$. La réduction de $\overline{P}$ vers $\overline{AP}$ est identique à celle du modèle non-anonyme. Localement chaque processus calcule le cardinal de l'ensemble $vivants_i$ retourné par $\overline{P}$.
- $AP \simeq_{\mathcal{SAA}} \overline{AP}$. L'équivalence est identique à celle du modèle non-anonyme. Il suffit de soustraire la sortie de l'un au nombre $n$ de processus pour obtenir la sortie de l'autre.
- $A\Sigma \prec_{\mathcal{SAA}} AP$. La réduction de $AP$ vers $A\Sigma$ consiste à prendre le couple $(0, n - nb\_suspects)$ pour émuler la sortie de $A\Sigma$, l'entier $nb\_suspects$ étant la sortie de $AP$. Ce couple vérifie les quatre propriétés définissant le détecteur $A\Sigma$.
- $A\Sigma \prec_{\mathcal{SAA}} \overline{AP}$, $A\Sigma \prec_{\mathcal{SAA}} P$, et $A\Sigma \prec_{\mathcal{SAA}} \overline{P}$. Ces réductions se déduisent des précédentes par transitivité.
- $\Sigma \prec_{\mathcal{SAA}} \overline{P}$. La réduction consiste à émuler la sortie $quorum_i$ de $\Sigma$ par l'ensemble $vivants_i$ retourné par le détecteur $\overline{P}$.
- $\Omega \prec_{\mathcal{SAA}} \overline{P}$. La réduction consiste à émuler la sortie $leader_i$ de $\Omega$ en prenant la plus petite identité de l'ensemble $vivants_i$ retourné par le détecteur $\overline{P}$. Les propriété de sûreté et vivacité de $\overline{P}$ garantissent immédiatement la propriété de vivacité de $\Omega$.

**Réductions impossibles** Nous devons prouver ici l'absence de réduction entre certains détecteurs. Tout comme pour les réductions ci-dessus, nous donnons de brèves explications permettant de comprendre chacune d'entre elles :

- $D_1 \not\succ_{\mathcal{SAA}} D_2$ où $D_1 \in \{P, \overline{P}, AP, \overline{AP}\}$ et $D_2 \in \{A\Sigma, \Sigma, \Omega, A\Omega\}$. Ces réductions n'existent pas dans le modèle non-anonyme ; elles ne peuvent pas non plus exister dans le modèle anonyme qui est plus faible.
- $D_1 \not\succ_{\mathcal{SAA}} D_2$ et $D_2 \not\succ_{\mathcal{SAA}} D_1$ où $D_1 \in \{\Sigma, A\Sigma\}$ et $D_2 \in \{\Omega, A\Omega\}$. Ces réductions n'existent pas dans le modèle non-anonyme ; elles ne peuvent pas non plus exister dans le modèle anonyme qui est plus faible.
- $D_1 \not\succ_{\mathcal{SAA}} D_2$ où $D_1 \in \{P, \overline{P}, \Sigma, \Omega\}$ et $D_2 \in \{AP, \overline{AP}, A\Sigma, A\Omega\}$. Le système étant anonyme (les processus ne connaissent pas les identités) et un détecteur de fautes $D_2$ ne fournissant aucune indication sur ces identités, il est impossible d'émuler un détecteur $D_1$ dont la sortie contient des identités de processus.
- $\overline{P} \not\succ_{\mathcal{SAA}} P$, $\Omega \not\succ_{\mathcal{SAA}} P$, et $\Sigma \not\succ_{\mathcal{SAA}} P$. Il est impossible pour un processus de deviner des identités de processus vivants en connaissant uniquement celles de processus en panne.
- $P \not\succ_{\mathcal{SAA}} \overline{P}$. Il est impossible pour un processus de deviner des identités de processus en panne en connaissant uniquement celles de processus vivants.
- $A\Omega \not\succ_{\mathcal{SAA}} D$, où $D$ est n'importe quel autre détecteur défini dans ce chapitre. Les sorties de tous les autres détecteurs peuvent être identiques pour tous les processus. Les processus ne connaissant pas leurs propres identités, il n'existe aucun moyen permettant de briser la symétrie entre eux. Il est impossible d'atteindre la propriété de vivacité de $A\Omega$.
- $A\Sigma \not\succ_{\mathcal{SAA}} \Sigma$. Cette relation est la moins évidente à démontrer : en effet, une première intuition serait qu'il est possible d'émuler $A\Sigma$ à partir de $\Sigma$ en générant, de manière unique et déterministe, un couple $(x, y)$ pour tout quorum $Q$ donné par le détecteur $\Sigma$ (tout comme dans le système non-anonyme). Cependant cela ne fonctionne pas car, contrairement au modèle non-anonyme, un processus ne peut pas vérifier s'il appartient à un quorum $(x, y)$ donné. Afin de prouver l'absence de réduction, considérons, par exemple, que le détecteur $\Sigma$ fournit à tous les processus le même quorum $\{1\}$ (c'est possible si le processus 1 est correct). Puisqu'il est impossible pour les processus de se différencier, ils doivent alors émuler $A\Sigma$ de manière identique, or il n'existe aucune émulation valable respectant la symétrie.

### 4.4.6 Réalisme des détecteurs de fautes

Un lecteur attentif aura probablement remarqué que sur la Figure 4.1 certains détecteurs sont entourés par un rectangle alors que d'autres par des ellipses. Ce choix n'est pas dû au hasard, mais représente la caractéristique réaliste du détecteur de faute dans le modèle considéré. Un rectangle signifie que le détecteur est réaliste, tandis qu'une ellipse indique qu'il ne l'est pas.

**Définition 4.4** *(Détecteur de fautes réaliste) [27] Un détecteur de fautes $D$ est dit réaliste dans le système asynchrone anonyme $\mathcal{SAA}$ (resp. le système asynchrone non-anonyme $\mathcal{SA}$) si et seulement s'il existe une implémentation de ce détecteur $D$ dans le système synchrone anonyme $\mathcal{SSA}$ (resp. le système synchrone non-anonyme $\mathcal{SS}$).*

Cette notion de réalisme est intéressante car elle donne un (léger) sens pratique au concept théorique de détecteur de fautes. Il est en effet possible d'envisager un système réel asynchrone contrôlé par un système supposé synchrone : ce dernier étant plus coûteux à maintenir, il n'est utilisé que pour surveiller le système asynchrone qui lui constitue le réseau où circulent les données applicatives.

**Réalisme dans le système non-anonyme** Dans le modèle non-anonyme, tous les détecteurs de fautes présentés sont réalistes. En effet, pour les implémenter en synchrone, il suffit d'utiliser un mécanisme où chaque processus diffuse périodiquement un message VIVANT contenant son identité. Dans un système synchrone, de tels échanges de messages permettent de détecter les processus défaillants lorsque les processus vivants arrêtent de recevoir leurs messages : contrairement à la situation asynchrone, cela ne peut être dû au délai de transmission.

Il existe cependant des détecteurs de fautes non-réaliste, y compris dans les modèles non-anonymes. Un détecteur fournissant, au bout d'un temps fini, l'identité d'un processus correct. Puisqu'il est impossible, y compris dans un système synchrone, de prévoir le futur ; un tel détecteur ne peut être implémenter.

**Réalisme dans le système anonyme** Dans le modèle anonyme, les détecteurs de fautes dont la sortie inclut des identités de processus ne sont trivialement pas réalistes. Les détecteurs $A\Sigma$, $AP$, et $\overline{AP}$ sont réalistes : il est possible de les implémenter dans un système synchrone anonyme en utilisant un mécanisme de diffusion comme précédemment. Cependant au lieu d'inclure leurs identités, les processus ajoutent un numéro de séquence aux messages qu'ils diffusent. Ainsi si lors de la ronde $k$ un processus reçoit $n_k$ messages VIVANT$(k)$, cela signifie qu'il y a $n_k$ processus vivants à cet instant, ce qui permet d'implémenter les trois détecteurs :

- $\overline{AP}$ en prenant $nb\_vivants = n_k$,
- $AP$ en prenant $nb\_suspects = n - n_k$,
- $A\Sigma$ en prenant $liste\_quorums = (0, n_k)$.

Le détecteur $A\Omega$ n'est par contre pas réaliste. En effet même si le système est parfaitement synchrone, il est impossible de briser la symétrie entre les processus. En supposant que tous les processus envoient et reçoivent tous leurs messages aux mêmes instants, les sorties du détecteur de fautes seront nécessairement identiques, ce qui rend empêche d'atteindre la propriété de vivacité de $A\Omega$.

La détecteur de fautes $A\Omega$ est néanmoins intéressant car, comme nous le montrons en Section 4.5.3, il entre probablement en jeu dans la définition du plus faible détecteur de fautes permettant de résoudre le problème du consensus dans les systèmes anonymes.

## 4.5 Consensus avec des détecteurs de fautes

Dans les deux sections précédentes, nous avons défini des détecteurs de fautes et montré quelless sont les relations entre ceux-ci. Cependant l'objectif réel des détecteurs de fautes est de permettre la résolution de problèmes insolvables dans les modèles pures (sans ajout de

détecteur). Comme introduit dans la Section 4.2, nous nous intéressons ici au problème du consensus.

Nous proposons deux algorithmes qui résolvent le problème du consensus dans le modèle anonyme en utilisant des détecteurs de fautes.

- Le premier algorithme utilise le détecteur parfait $\overline{AP}$ et utilise un nombre borné de messages pour résoudre le problème du consensus.
- Le second utilise la combinaison des détecteurs $A\Sigma$ et $A\Omega$. Il n'existe par contre aucune borne sur le nombre de messages envoyés.

### 4.5.1 Algorithme de consensus avec $\overline{AP}$

**Description de l'algorithme** Cet algorithme issu de nos travaux [18] utilise un modèle légèrement plus générique que celui défini dans la Section 4.2. Dans le reste de ce chapitre, nous considérons uniquement l'environnement sans-attente dans lequel au maximum $n-1$ processus peuvent subir une panne. Nous considérons ici plus généralement les environnements tolérant au maximum $t$ pannes ($0 \leq t < n$). Le système $\mathcal{SAA}_t$ désigne alors le système anonyme asynchrone dans lequel au plus $t$ pannes peuvent arriver. Le système $\mathcal{SAA}$ correspond au cas particulier $t = n-1$, c'est-à-dire au système $\mathcal{SAA}_{n-1}$.

Nous introduisons ces modèles car l'algorithme proposé en Figure 4.5 s'adapte au nombre de fautes que peut subir un système. Plus précisément, l'algorithme nécessite $2t+1$ envois de messages (par processus) lorsqu'il est utilisé dans un système où $t$ pannes peuvent apparaître[4].

Le processus $p_i$ débute l'algorithme par un appel à l'opération propose($v_i$) (où $v_i$ correspond à la valeur que $p_i$ propose) puis décide une valeur lorsqu'il exécute l'instruction décider de la Ligne 10. Le principe de l'algorithme est relativement classique et consiste en une succession de rondes durant lesquelles les processus s'échangent des messages. Les rondes sont asynchrones et chaque processus progresse à la ronde suivante lorsqu'il a reçu suffisamment de messages, ce nombre de messages correspondant au nombre de processus vivants fourni par le détecteur $\overline{AP}$. Au début de chaque ronde, les processus envoient une valeur *est* qui représente une estimation de la décision finale du consensus (Ligne 4) ; à la fin de chaque ronde, chaque processus détermine sa nouvelle estimation en sélectionnant la plus petite estimation qu'il a reçue durant la ronde (Ligne 6).

**Preuve de l'algorithme** Afin de prouver que l'algorithme de la Figure 4.5 résout le problème du consensus, nous prouvons tout d'abord quelques lemmes intermédiaires.

**Lemme 4.5** *Toute valeur décidée par un processus a été proposée (validité).*

**Preuve** La preuve de validité se déduit immédiatement du code de l'algorithme : une valeur décidée provient d'une variable locale $est_i$ or ces variables sont toutes initialisées par des valeurs proposées (Ligne 1) puis uniquement mises à jour à partir d'autres variables *est* (Ligne 6). □

[4]En réalité, dans le cas particulier où $t = n-1$, $2t$ rondes suffisent.

```
opération propose(v_i) :
(01) est_i ← v_i ; r_i ← 1 ;
(02) tant que (r_i ≤ 2t + 1) faire
(03)    début ronde asynchrone
(04)    diffuser EST(r_i, est_i) ;
(05)    attendre que (nb_vivants_i messages EST(r_i, −) ont été reçus) ;
(06)    est_i ← min(est valeurs reçues à la ligne précédente) ;
(07)    r_i ← r_i + 1 ;
(08)    fin ronde asynchrone
(09) fin tant ;
(10) décider(est_i).
```

FIG. 4.5 – Algorithme de consensus dans $\mathcal{SAA}_t[\overline{AP}]$

**Lemme 4.6** *Tout processus correct décide (terminaison).*

**Preuve** La propriété de vivacité du détecteur de fautes $\overline{AP}$ garantit qu'un processus ne peut pas rester bloquer indéfiniment à la Ligne 5. Ainsi tout processus qui ne subit pas de panne avant la fin de la ronde $2t + 1$ décide. □

**Lemme 4.7** *Si aucun processus ne subit de panne durant deux rondes consécutives $r$ et $r + 1$, alors tous les processus terminant la ronde $r$ possède la même estimation est à la fin de cette ronde.*

**Preuve** Soient $r$ et $r + 1$ deux rondes telles qu'aucun processus $p_i$ ne subit de panne lorsque localement $r_i$ vaut $r$ ou $r + 1$. Soit $E_r$ l'ensemble des processus qui exécutent ces deux rondes et soit $p_i$ le premier processus qui termine la ronde $r + 1$ à un instant $\tau$ donné. $p_i$ a nécessairement reçu $|E_r|$ messages lors de cette ronde $r + 1$, puisque la propriété de sûreté du détecteur $\overline{AP}$ garantit que $nb\_suspects_i$ est, à tout instant, supérieure au nombre de processus vivants. Cela signifie qu'à l'instant $\tau$ tous les processus de $E_r$ ont déjà débuté la ronde $r+1$ ; par conséquent tous les processus de $E_r$ ont terminé la ronde $r$ alors qu'il y avait au moins $|E_r|$ processus vivants. Or seuls $|E_r|$ processus ont envoyé un message lors de la ronde $r$ ; cela implique que tous les processus de $E_r$ ont reçu exactement le même ensemble de $|E_r|$ messages lors de la ronde $r$ ; ils ont donc nécessairement mis à jour leur estimation *est* de manière identique. □

**Lemme 4.8** *Deux processus ne décide pas de valeurs différentes (accord).*

**Preuve** La preuve considère deux cas :

- Cas 1 : Dans la séquence des $(2t + 1)$ rondes, il existe deux rondes consécutives sans panne. Soient $r$ et $r + 1$ ces deux rondes avec $1 \leq r \leq 2t$. Le Lemme 4.7 implique que tous les processus terminant la ronde $r$ possède la même estimation *est* à la fin de cette ronde. Ainsi, une seule valeur, pouvant être décidée, existe dans le système à partir de cet instant.

– Cas 2 : Dans la séquence des $(2t+1)$ rondes, il n'existe pas deux rondes consécutives sans panne. Cela implique qu'il y a au moins une panne lors des rondes 1 ou 2, au moins une panne lors des rondes 3 ou 4, ..., au moins une panne lors des rondes $2t-1$ ou $2t$. Les $t$ pannes possibles ont nécessairement lieu avant le début de la ronde $2t+1$. Il ne peut donc y avoir que $n-t$ processus qui débutent cette dernière ronde et cette dernière est sans panne ; tous ces processus reçoivent le même ensemble de $n-t$ estimations et décide alors la même valeur (la plus petite de ces estimations).

□

**Théorème 4.9** *L'algorithme décrit à la Figure 4.5 résout le problème du consensus en* $(2t+1)$ *rondes dans le modèle* $\mathcal{SAA}_t[\overline{AP}]$.

**Preuve** La preuve découle directement des Lemmes 4.5, 4.6, 4.8 démontrant respectivement les propriétés de validité, de terminaison, et d'accord du problème du consensus. □

**Remarque** Nous montrons dans [18] que $(2t+1)$ est une borne inférieure sur le nombre de rondes nécessaires pour résoudre le consensus dans le modèle $\mathcal{SAA}_t[\overline{AP}]$. Cette borne reste valide en remplaçant le détecteur $\overline{AP}$ par un autre détecteur parfait, $P$, $\overline{P}$, ou $AP$. Il est par ailleurs prouvé [47] que dans un modèle asynchrone non-anonyme, $(t+1)$ rondes sont nécessaires pour résoudre le consensus avec le détecteur parfait $P$. C'est intéressant puisque cela montre que l'utilisation d'un détecteur parfait dans un système anonyme entraîne un doublement du nombre de rondes par rapport à l'utilisation du même détecteur dans un système non-anonyme.

### 4.5.2 Algorithme de consensus avec $(A\Sigma, A\Omega)$

**Description de l'algorithme** Cet algorithme utilise la paire de détecteurs $(A\Sigma, A\Omega)$ dans le modèle anonyme. Nous proposons un tel algorithme parce que (1) la paire $(\Sigma, \Omega)$ correspond au plus faible détecteur de fautes permettant de résoudre le consensus dans le modèle non-anonyme, (2) la paire $(\Sigma, \Omega)$ est inutile dans le modèle anonyme, et (3) la paire $(A\Sigma, A\Omega)$ constitue l'homologue "anonyme" de la paire $(\Sigma, \Omega)$.

L'algorithme, représenté sur la Figure 4.6, emprunte la structure de rondes divisées en trois phases des algorithmes de consensus pour les modèles non-anonymes basés sur la paire de détecteurs $(\Sigma, \Omega)$ [23]. Cependant à l'intérieur de chaque phase, les mécanismes utilisés sont différents ; en particulier les deux dernières phases de chaque ronde nécessitent plusieurs sous-rondes.

Le processus $p_i$ débute l'algorithme par un appel à l'opération propose$(v_i)$ (où $v_i$ correspond à la valeur que $p_i$ propose) puis décide une valeur lorsqu'il exécute l'instruction décider des Lignes 10 ou 38. Avant de décider une valeur, un processus diffuse un message DECISION contenant cette valeur afin d'éviter les éventuels blocages et pour permettre ainsi à tous les processus corrects de décider.

Le détecteur $A\Omega$ est utilisé dans la première phase et permet d'affecter une valeur à la variable $est_1$ diffusée lors de la deuxième phase. Le détecteur $A\Sigma$ est quant à lui utilisé dans les deux phases suivantes et permet aux processus, d'une part de définir une variable $est_2$ diffusée lors de la troisième phase, et d'autre part de s'accorder éventuellement sur une décision.

- Durant la première phase, un processus $p_i$ qui se considère comme leader (grâce à la variable $est\_leader_i$ fournie par le détecteur $A\Omega$) diffuse un message PHASE1 contenant sa propre estimation $est1_i$. Si $p_i$ ne se considère pas comme leader, il attend de recevoir un message de la part d'un leader, adopte la valeur reçue puis la diffuse (pour éviter des éventuels blocages).
- De manière similaire à ce qui est présenté dans [60], le but de la seconde phase est d'affecter des valeurs aux variables $est2$ tel que la propriété $P(r)$ suivante soit toujours satisfaite ($est2_i[r]$ désigne la valeur de la variable $est2$ du processus $p_i$ après son affectation lors de la ronde $r$) :

  $$P(r) \ : \ \big[(est2_i[r] \neq \bot) \ \wedge \ (est2_j[r] \neq \bot)\big] \ \Rightarrow \ \big(est2_i[r] = est2_j[r]\big).$$

  Pour atteindre cet objectif dans le modèle non-anonyme, chaque processus envoie un message puis attend de recevoir des messages de tout un quorum (fourni par $\Sigma$). Ici les processus ne connaissent pas leurs identités et ne peuvent taguer leurs messages qu'avec ce que leur fourni le détecteur $A\Sigma$. Plus précisément, nous incluons dans chaque message (Ligne 8) l'ensemble des étiquettes issues de $A\Sigma$ (comme lors de la réduction de $A\Sigma$ vers $\Sigma$ sur la Figure 4.4). Cependant contrairement aux identités, ces ensembles *etiquettes* changent au cours du temps. C'est la raison pour laquelle nous utilisons des sous-rondes ; une sous-ronde débutant (Lignes 15 et 16) lorsqu'un processus $p_i$ détecte un changement dans son ensemble $etiquettes_i$ ou si un autre processus a déjà envoyé des messages dans une nouvelle sous-ronde (Ligne 14).
  L'idée est qu'il existe obligatoirement une sous-ronde permettant au processus de progresser, mais ce n'est pas nécessairement la dernière ; chaque processus vérifie donc si, parmi toutes les sous-rondes (Ligne 11), une permet de progresser. Une sous-ronde $sr$ permet au processus $p_i$ de progresser s'il existe un couple $(x, y)$ fourni par $A\Sigma$ au processus $p_i$ tel que $p_i$ a reçu $y$ messages contenant l'étiquette $x$ lors de la sous-ronde $sr$. Si cette condition est vérifiée, le processus $p_i$ peut débuter la troisième phase en affectant la variable $est2_i$ (Ligne 12) : si tous les messages reçus contiennent la même estimation $est1 = v$ alors la variable est initialisée à cette valeur ; dans le cas contraire elle prend la valeur par défaut $\bot$ (qui ne peut pas être une valeur proposée).
  Le test de la Ligne 14 permet de rattraper les processus ayant déjà progressé à la phase suivante de l'algorithme. Ce simple mécanisme évite les blocages que pourraient sinon rencontrer les processus les plus lents.
- La troisième phase utilise le même mécanisme de sous-rondes. La différence réside uniquement dans le traitement effectué lorsque suffisamment de messages ont été reçus (Lignes 24 à 28). Si tous les messages reçus contiennent la même valeur (différente de $\bot$) alors cette valeur est décidée ; le mécanisme des quorums garantissant (comme nous

le verrons dans la preuve) qu'il n'existe alors plus aucune autre valeur dans le système. Si tous les messages reçus ne contiennent pas uniquement la même valeur, les processus ne peuvent pas décider et entament donc une nouvelle ronde.

```
opération propose(v_i) :
(01) est1_i ← v_i ; r_i ← 0 ;
(02) répéter
(03)   début ronde asynchrone
(04)   r_i ← r_i + 1 ;
       Phase 1 : affecter une valeur à est1_i grâce au détecteur AΩ
(05)   attendre ((est_leader_i) ∨ (PHASE1(r_i, v) reçu)) ;
(06)   si (PHASE1(r_i, v) reçu) alors est1_i ← v fin si ;
(07)   diffuser PHASE1(r_i, est1_i) ;
       Phase 2 : affecter une valeur ou ⊥ à est2_i
(08)   sr_i ← 1 ; etiquettes_i ← {x | (x, −) ∈ liste_quorums_i} ; diffuser PHASE2(r_i, sr_i, etiquettes_i, est1_i) ;
(09)   répéter
(10)     si (PHASE3(r_i, −, −, est2) reçu) alors est2_i ← est2 ; sortir de la boucle fin si ;
(11)     si (∃ (x, y) ∈ liste_quorums_i ∧ ∃ sr ∈ ℕ
               tels que y msgs PHASE2(r_i, sr, etiquettes_j, −) ont été reçus avec x ∈ etiquettes_j pour chacun d'eux)
(12)     alors si (tous les y messages précédents contiennent la même valeur v) alors est2_i ← v sinon est2_i ← ⊥ fin si ;
(13)           sortir de la boucle
(14)     sinonsi (etiquettes_i ≠ {x | (x, −) ∈ liste_quorums_i}) ∨ (PHASE2(r_i, sr, −, −) reçu avec sr > sr_i)
(15)           alors sr_i ← sr_i + 1 ; etiquettes_i ← {x | (x, −) ∈ liste_quorums_i} ;
(16)                 diffuser PHASE2(r_i, sr_i, etiquettes_i, est1_i)
(17)           fin si
(18)     fin si
(19)   fin répéter ;
       Phase 3 : essayer de décider une valeur à partir des valeurs est2
(20)   sr_i ← 1 ; etiquettes_i ← {x | (x, −) ∈ liste_quorums_i} ; diffuser PHASE3(r_i, sr_i, etiquettes_i, est2_i) ;
(21)   répéter
(22)     si (PHASE1(r_i + 1, −) reçu ) alors sortir de la boucle fin si ;
(23)     si (∃ (x, y) ∈ liste_quorums_i ∧ ∃ sr ∈ ℕ
               tels que y msgs PHASE3(r_i, sr, etiquettes_j, −) ont été reçus avec x ∈ etiquettes_j pour chacun d'eux)
(24)     alors soit rec_i = l'ensemble des est2 contenus dans les y messages précédents ;
(25)           cas (rec_i = {v})      alors diffuser DECISION(v) ; décider(v)
(26)               (rec_i = {v, ⊥})   alors est1_i ← v
(27)               (rec_i = {⊥})      alors skip
(28)           fin cas ;
(29)           sortir de la boucle
(30)     sinonsi (etiquettes_i ≠ { x | (x, −) ∈ liste_quorums_i }) ∨ (PHASE3(r_i, sr, −, −) reçu avec sr > sr_i)
(31)           alors sr_i ← sr_i + 1 ; etiquettes_i ← {x | (x, −) ∈ liste_quorums_i} ;
(32)                 diffuser PHASE3(r_i, sr_i, etiquettes_i, est2_i)
(33)           fin si
(34)     fin si
(35)   fin répéter
(36)  fin ronde asynchrone
(37) fin répéter.

(38) lorsque DECISION(v) est reçu : diffuser DECISION(v) ; décider(v).
```

FIG. 4.6 – Algorithme de consensus dans $\mathcal{SAA}[A\Sigma, A\Omega]$

**Preuve de l'algorithme** Afin de prouver que l'algorithme de la Figure 4.6 résout le problème du consensus, nous prouvons tout d'abord quelques lemmes intermédiaires.

**Lemme 4.10** *Toute valeur décidée par un processus a été proposée (validité).*

**Preuve** La preuve de validité se déduit immédiatement du code de l'algorithme : une valeur décidée provient des variables locales $est1$ et $est2$ or ces variables sont toutes initialisées par des valeurs proposées (Ligne 1) puis uniquement mises à jour à partir d'autres variables $est1$ ou $est2$ (Lignes 6, 10, 12, et 26). □

**Lemme 4.11** *Si aucun processus ne décide, les processus corrects éffectuent un nombre infini de rondes.*

**Preuve** La preuve est par contradiction. Supposons qu'aucun processus ne décide une valeur, et qu'il existe au moins un processus correct qui ne termine qu'un nombre fini de rondes. Soit $r$ la première ronde dans laquelle un processus correct $p_i$ reste bloquer de manière permanente. Nous distinguons les cas suivant la phase dans laquelle $p_i$ reste bloquer.

- Phase 1. Si $p_i$ est le leader ultime désigné par le détecteur $A\Omega$, il ne peut rester bloquer indéfiniment car le booléen $est\_leader_i$ passe nécessairement à $vrai$ (vivacité de $A\Omega$). Sinon, soit $p_\ell$ le leader ultime désigné par $A\Omega$. $p_\ell$ ne peut pas bloquer indéfiniment. Durant cette première phase, il envoie un message qui est reçu par $p_i$ ; $p_i$ progresse alors à la phase suivante. C'est une contradiction.
- Phase 2. Par hypothèse $p_i$ boucle indéfiniment dans les Lignes 9 à 19. $p_i$ ne reçoit aucun message provenant de la phase 3 car cela lui permettrait de progresser à cette phase 3 (Ligne 10) ; nous pouvons en déduire qu'aucun processus correct ne participe à cette phase 3 et que par conséquent tous les processus corrects sont également bloqués dans cette phase 2 de la ronde $r$.
  La propriété de vivacité de $A\Sigma$ garantit qu'après un certain instant $\tau$, il existe un couple $(x, y) \in liste\_quorums_i$ tel que $|S(x) \cap Correct| \geq y$. Nous pouvons déduire (1) de la définition de $S(x)$, (ii) du fait que $S(x)$ contienne au moins $y$ processus corrects après $\tau$, et (ii) du fait que chaque processus correct envoie un nouveau message PHASE3$(r, -, -, -)$ lorsque les étiquettes des couples fournis par $A\Sigma$ changent, qu'il existe nécessairement une sous-rondes $sr$ dans laquelle au moins $y$ processus corrects envoient un message contenant l'étiquette $x$. Lorsque $p_i$ reçoit ces messages de la sous-ronde $sr$, les conditions de la Ligne 11 sont remplies et il progresse vers la phase 3. C'est une contradiction.
- Phase 3. Le même raisonnement que pour la phase 2 s'applique car la seule différence entre les deux phases tient dans le traitement effectué une fois reçu le bon ensemble de messages. C'est une contradiction.

□

**Lemme 4.12** *Tout processus correct décide (terminaison).*

**Preuve** Avant de décider une valeur (Ligne 25), tout processus diffuse cette valeur. Ainsi, si un processus décide, tous les processus corrects décident par la reception de ce message. Supposons alors, par contradiction, qu'aucun processus ne décide.

La propriété de vivacité de $A\Omega$ garantit qu'il existe un instant $\tau_0$ à partir duquel il existe un et un seul processus correct dont le booléen $est\_leader$ vaut $vrai$ ; appelons $p_\ell$ ce processus.

Par ailleurs soit $\tau_1$, un instant après lequel tous les processus fautifs ont déjà subit leur panne et considérons alors un instant $\tau$ tel que $\tau \geq \max(\tau_0, \tau_1)$.

Nous sommes dans les conditions du Lemme 4.11 et par conséquent il existe une ronde $r$ telle qu'aucun processus n'a entamé la ronde $r$ avant l'instant $\tau$ et aucun processus correct n'est bloqué dans cette ronde. Lors de la première phase de cette ronde, seul $p_\ell$ peut imposer son estimation $est1_\ell = v$ car c'est le seul processus dont le booléen $est\_leader$ vaut $vrai$. Par conséquent tous les processus débutent la phase 2 avec $est1 = v$. Tous les processus s'échangent cette même estimation et donc tous les processus débutent la ronde suivante avec $est2 = v$ (Lignes 12 et 10). Dans la troisième phase, seul le cas de la Ligne 25 peut se produire et nous pouvons alors en déduire que tous les processus corrects décident une valeur à la fin de cette ronde. □

**Lemme 4.13** *Deux processus ne décide pas de valeurs différentes (accord).*

**Preuve** Si au plus un processus décide à la Ligne 25 alors la propriété d'accord est satisfaite puisque les autres processus décident uniquement à la Ligne 38 après réception d'un message DECISION contenant la valeur décidée. Considérons alors qu'au moins deux processus décident à la Ligne 25. Soit $r$ la première ronde durant laquelle un processus décide et soit $p_i$ un tel processus décidant la valeur $v$. Soit $p_j$ un autre processus qui décide la valeur $v'$ à la Ligne 25 lors d'une ronde $r' \geq r$. Deux cas sont à distinguer :

- Cas 1 : $p_j$ décide durant la même ronde $r' = r$ que $p_i$. Puisque $p_i$ décide la valeur $v$ lors de la ronde $r$, le prédicat de la Ligne 23 est satisfait à un instant donné, tout en vérifiant la condition de la Ligne 25 : $\exists (x, y) \in liste\_quorums_i$ tel que $p_i$ a reçu $y$ messages PHASE3$(r, sr, etiquettes_k, v)$ où $x \in etiquettes_k$ pour chacun des messages. Soit $T \in S(x)$ l'ensemble des $y$ processus dont $p_i$ a reçu ces messages. De manière similaire, puisque $p_j$ décide valeur $v'$ lors de la ronde $r$, $\exists (x', y') \in liste\_quorums_j$ tel que $p_j$ a reçu $y'$ messages PHASE3$(r, sr, etiquettes_k, v')$ où $x' \in etiquettes_k$ pour chacun des messages. Soit $T' \in S(x')$ l'ensemble des $y'$ processus dont $p_j$ a reçu ces messages. La propriété de sûreté de $A\Sigma$ garantit que les ensembles $T$ et $T'$ s'intersectent. Par conséquent au moins un processus a envoyé le même message PHASE3$(r, sr, etiquettes_k, v)$ à $p_i$ et PHASE3$(r, sr, etiquettes_k, v')$ à $p_j$ ; cela signifie que $v = v'$. $p_i$ et $p_j$ décident la même valeur.
- Cas 2 : $p_j$ décide durant une ronde $r' > r$. Cela signifie que certains processus (au moins $p_j$) débutent la ronde $r + 1$ alors que $p_i$ décide lors de la ronde $r$. Considérons un processus $p_k$ qui termine la ronde $r$ sans utiliser la Ligne 22 (il se peut que $p_j$ ne soit pas dans cette situation).
  Le même raisonnement que pour le cas précédent peut s'appliquer pour l'ensemble des messages reçu par $p_i$ et par $p_k$. Il existe au moins un même message reçu par ces deux processus. Or, puisque $p_i$ décide il n'a reçu que des messages contenant la valeur $v$, cela signifie que le processus $p_k$ a lui aussi reçu la valeur $v$, et par conséquent il exécute nécessairement la Ligne 26. La valeur $v$ est la seule valeur qui reste dans le système après la fin de la ronde $r$ et par conséquent tout processus, tel $p_j$ qui décide après la ronde $r$ décide obligatoirement cette même valeur $v$.

□

**Théorème 4.14** *L'algorithme décrit à la Figure 4.6 résout le problème du consensus dans le modèle* $\mathcal{SAA}_t[A\Sigma, A\Omega]]$.

**Preuve** La preuve découle directement des Lemmes 4.10, 4.12, 4.13 démontrant respectivement les propriétés de validité, de terminaison, et d'accord du problème du consensus. □

### 4.5.3 Notion(s) de plus faible détecteur de fautes

**Algorithmes de consensus basés sur des détecteurs de fautes** Nous avons proposé dans les pages précédentes deux algorithmes résolvant le consensus. Ces deux algorithmes n'utilisent pas les mêmes détecteurs de fautes et il est alors naturel de se demander quel est l'algorithme utilisant le détecteur le plus faible. Malheureusement $\overline{AP}$ et $(A\Sigma, A\Omega)$ sont incomparables dans le modèle anonyme.

**Notion de plus faible détecteur de fautes** D'après [25], étant donné un problème $\mathcal{P}$ et un détecteur de fautes $D$ ; nous disons que $D$ est *le plus faible détecteur de fautes* pour $\mathcal{P}$ dans $\mathcal{SA}$ (ou de manière similaire dans $\mathcal{SAA}$) si (i) il existe un algorithme résolvant le problème $\mathcal{P}$ dans le modèle $\mathcal{SA}[D]$, et (ii) tout détecteur de fautes $D'$, permettant de résoudre le problème $\mathcal{P}$, est plus fort que $D$ : $D \preceq D'$. Il est montré dans [50] que tout problème possède un plus faible détecteur dans le modèle non-anonyme. Qu'en est il pour le problème du consensus dans le modèle anonyme ?

**Nouveau détecteur de fautes** Étant donnés deux détecteurs $D_1$ et $D_2$, nous définissons un nouveau détecteur $D_1 \oplus D_2$ : pendant une période de temps arbitrairement longue, mais finie, la sortie de $D_1 \oplus D_2$ vaut $\perp$ pour chaque processus ; ensuite $D_1 \oplus D_2$ se comporte soit comme $D_1$, soit comme $D_2$, mais de manière identique pour tous les processus. Observons que si $D_1$ et $D_2$ sont deux détecteurs de fautes incomparables, le détecteur $D_1 \oplus D_2$ est strictement plus faible que $D_1$ d'une part, et $D_2$ d'autre part.

**Plus faible détecteur pour le consensus** Parmi les détecteurs de fautes que nous avons présentés, aucun ne correspond (s'il existe) au plus faible détecteur pour le consensus. Nous pensons que le détecteur $(A\Sigma, A\Omega) \oplus AP$ est le plus faible détecteur de fautes pour résoudre le consensus dans le modèle asynchrone anonyme $\mathcal{SAA}$. Il est possible et facile de définir un algorithme résolvant le consensus à l'aide de ce détecteur, en se basant sur les deux algorithmes détaillés précédemment.

Cette conjecture est de plus motivée par le fait que $\big((\Sigma, \Omega) \oplus P\big) \simeq_{\mathcal{SA}} (\Sigma, \Omega)$ où $(\Sigma, \Omega)$ est connu pour être le plus faible détecteur pour le consensus dans le modèle non-anonyme $\mathcal{SA}$. Si notre conjecture est vraie, cela signifierait que $(A\Sigma, A\Omega) \oplus AP$ est réellement *le* plus faible détecteur de fautes pour le consensus, aussi bien dans le modèle anonyme que non-anonyme.

## 4.6 Conclusion

Ce chapitre nous a permis de présenter plusieurs résultats concernant les détecteurs de fautes et le problème du consensus dans le modèle asynchrone anonyme. D'un coté, nous décrivons plusieurs détecteurs de fautes spécialement adaptés au contexte anonyme et étudions leurs puissances relatives. D'un autre coté, nous montrons qu'il est effectivement possible de résoudre le problème du consensus dans les systèmes anonymes à l'aide de détecteurs de fautes.

Les points importants à retenir sous les suivants :

- La puissance d'un détecteur de faute n'est pas intrinsèque ; elle dépend du modèle dans lequel il est utilisé.
- Les réductions entre détecteurs de fautes changent elles-aussi suivant le modèle. Nous démontrons ainsi une nouvelle hiérarchie des détecteurs de fautes, dans les systèmes anonymes.
- L'utilisation d'un détecteur parfait autorise la résolution du consensus en un nombre borné de messages dans le système anonyme. Cette borne dépend du nombre de fautes que tolère le système. Même si cela n'est pas montré explicitement dans ce manuscrit, ce nombre de messages est deux fois plus élevé que dans un système non-anonyme.
- L'utilisation de la paire de détecteurs $(A\Sigma, A\Omega)$ permet aussi la résolution du consensus, en utilisant le nouveau mécanisme de sous-ronde.

Il demeure néanmoins un problème ouvert majeur concernant le plus faible détecteur de fautes pour le problème du consensus. Nous proposons une conjecture, mais nous n'avons, pour le moment, pas réussi à démontrer ce résultat. Une première étape consisterait déjà à montrer l'existence d'un plus faible détecteur...

# Chapitre 5

# Conclusion

Ce dernier chapitre termine cette thèse sur les systèmes répartis. Comme annoncé dans l'introduction, trois thématiques très différentes ont été abordées. Le point commun entre ces trois travaux réside dans la présence de multiples entités devant coopérer pour résoudre un problème commun. Nous rappelons tout d'abord succinctement les contributions que nous avons apportées dans ce manuscrit. Nous présentons ensuite brièvement les autres travaux effectués lors de cette thèse ; ceux que nous avons choisi de ne pas présenter en détail. Enfin nous terminerons par des perspectives un peu plus globales.

## 5.1 Travaux présentés

Le manuscrit est composé, outre l'introduction et la conclusion, de trois chapitres présentant les résultats de nos recherches.

**Routage dans les réseaux petits-mondes** Le Chapitre 2 étudie le problème du routage dans les réseaux petits-mondes. Nous proposons un mélange entre des résultats théoriques et des résultats pratiques : une nouvelle méthode d'analyse du routage en utilisant des équations récursives et un nouvel algorithme permettant la construction de réseaux petits-mondes performants.

**Exploration de graphes par des robots anonymes** Le Chapitre 3 s'intéresse à l'exploration perpétuelle de graphes par des robots synchrones anonymes. L'objectif est de permettre à tous les robots de visiter infiniment souvent chaque sommet du graphe. Nous démontrons une borne topologique bornant le nombre de robots pouvant être déployés. Nous proposons ensuite différents algorithmes en fonction de la vision du graphe dont dispose chaque robot.

**Détecteurs de fautes et consensus dans les systèmes anonymes** Le Chapitre 4 étudie les détecteurs de fautes et leurs applications pour le problème du consensus dans le contexte des systèmes anonymes. Nous présentons des détecteurs conçus pour les systèmes

anonymes et les comparons avec leurs homologues classiques conçus pour les systèmes non-anonymes. Nous proposons ensuite deux algorithmes résolvant le consensus en utilisant des détecteurs incomparables ; ce qui nous amène alors à rechercher l'éventuel plus faible détecteur de fautes pour le consensus.

## 5.2 Autres travaux

Afin de meubler un peu ce chapitre de conclusion, nous avons décidé de présenter brièvement les autres travaux effectués durant cette thèse, en espérant que cela pourra amener le lecteur curieux à découvrir de nouvelles choses.

**Généralisation aux autres problèmes d'accord dans les systèmes anonymes** Dans le Chapitre 4, nous présentons un algorithme résolvant le consensus avec un détecteur parfait en $2t+1$ rondes. Dans [18], nous généralisons ce résultat au problème du $k$-accord, dans lequel les processus peuvent décider au plus $k$ valeurs différentes. Nous montrons qu'il existe une famille $\{\psi_\ell\}_{0\leq\ell<n-1}$ de détecteurs de fautes, telle qu'il est possible de résoudre le problème du $k$-accord avec un détecteur $\psi_\ell$ en $2\lfloor\frac{t}{k-\ell}\rfloor+1$ rondes lorsque $\ell < k$.

**Notion de décision précoce** Nous concevons des algorithmes pouvant tolérer $t$ fautes, mais dans les situations idéales, il n'y a pas ou peu de fautes. Ainsi lorsque le nombre réel $f$ de fautes est moindre ($f < t$), il est parfois possible de terminer l'exécution des algorithmes de manière prématurée. Dans [18] et [21] nous proposons des algorithmes utilisant des décisions précoces pour résoudre le consensus à l'aide de détecteurs de fautes parfaits, respectivement dans les systèmes anonymes et non-anonymes. Dans les systèmes non-anonymes, le nombre de rondes est borné par $\min(f+2, t+1)$ alors que dans les systèmes anonymes, la borne est de $\min(2f+2, 2t+1)$.

**Quête vers le plus faible détecteur de fautes pour le $k$-accord** Dans le Chapitre 4, nous évoquons la notion de plus faible détecteur de fautes pour le consensus. Dans les systèmes non-anonymes la réponse est connue ; la paire $(\Sigma, \Omega)$ constitue le plus faible détecteur de fautes. Pour le $n-1$-accord, le plus faible détecteur de fautes est aussi connu ; il s'agit de $\mathcal{L}$. La question reste néanmoins ouverte pour le problème du $k$-accord. Dans [17], nous proposons des pistes de recherche pouvant mener à ces détecteurs ; nous montrons en particulier que le détecteur $\Sigma_k$ introduit est nécessaire pour résoudre le problème du $k$-accord.

**Conditions pour résoudre les problèmes d'accord** Nous souhaitons préciser que l'approche des détecteurs de fautes n'est pas la seule approche permettant de contourner les impossibilités liées à l'asynchronie des systèmes. Dans [16], nous généralisons l'approche par condition aux problèmes du $k$-accord. L'idée des détecteurs de fautes consiste en une augmentation de la puissance du système pour permettre la résolution d'un problème ; au contraire l'idée des conditions consiste en une réduction de la complexité du problème pour

le rendre solvable dans un système donné. Plus précisément, utiliser des conditions revient à limiter les entrées possibles du problèmes ; n'importe quelle valeur ne peut pas être proposée.

## 5.3 Perspectives

Dans les conclusions respectives des trois chapitres précédents, nous avons présenté les différentes perspectives de recherche correspondantes. Nous les résumons ici de manière brève afin de conclure cette thèse par une ouverture vers de futures recherches.

**Routage dans les réseaux petits-mondes** Il serait intéressant d'utiliser la notion d'équations récursives pour d'autres problèmes que celui du routage dans les réseaux petits-mondes. Il serait pertinent de réaliser des simulations et expérimentations plus poussées des algorithmes épidémiques permettant la conception de réseaux petits-mondes.

**Exploration de graphes par des robots anonymes** Les cas extrêmes de rayon de vision ($\rho = 1$ et $\rho = 1$) sont maintenant résolus ; il serait intéressant de poursuivre la recherche pour les valeurs intermédiaires. Dans nos travaux actuels nous ne cherchons pas des algorithmes optimaux, vis-à-vis de la complexité, mais optimaux vis-à-vis de la calculabilité ; il pourrait être pertinent d'étudier ces notions de complexité et de chercher des algorithmes plus efficaces.

**Détecteurs de fautes et consensus dans les systèmes anonymes** La question la plus intéressante, selon nous, consiste à trouver le plus faible détecteur de fautes pour le consensus. Ensuite, les études sur les systèmes anonymes étant relativement rares, pour le moment, il est possible de continuer la recherche dans ce domaine en étudiants d'autres problèmes, ou d'autres types de fautes ; byzantines, faute de réception, duplication de messages, ...

# Bibliographie


[1] S. Albers and M.R. Henzinger. Exploring unknown environments. *SIAM Journal on Computing*, 29(4) :1164–1188, 2000.

[2] D. Angluin. Local and global properties in networks of processes. In *Proceedings of the 12th Symposium on Theory of Computing (STOC'80)*, pages 82–93. ACM Press, 1980.

[3] H. Attiya and J. Welch. *Distributed Computing, Fundamentals, Simulation and Advanced Topics (Second edition)*. Wiley Series on Parallel and Distributed Computing, 2004.

[4] B. Awerbuch, M. Betke, R.L. Rivest, and M. Singh. Piecemeal graph exploration by a mobile robot. *Information and Computation*, 152(2) :155–172, 1999.

[5] R. Baldoni, F. Bonnet, A. Milani, and M. Raynal. Anonymous graph exploration without collision by mobile robots. *Information Processing Letters*, 109(2) :98–103, 2008.

[6] R. Baldoni, F. Bonnet, A. Milani, and M. Raynal. On the solvability of anonymous partial grids exploration by mobile robots. In *Proceedings of the 12th International Conference on Principles of Distributed Systems (OPODIS'08)*, pages 428–445, 2008.

[7] R. Baldoni, F. Bonnet, A. Milani, and M. Raynal. Anonymous graph exploration without collision by mobile robots. Technical report, Research Report IRISA #1886, February 2008.

[8] R. Baldoni, F. Bonnet, A. Milani, and M. Raynal. On the solvability of anonymous partial grids exploration by mobile robots. Technical report, Research Report IRISA #1892, May 2008.

[9] L. Barrière, P. Fraigniaud, E. Kranakis, and D. Krizanc. Efficient routing in networks with long range contacts. In *Proceedings of the 15th International Symposium on Distributed Computing (DISC'01)*, pages 270–284, 2001.

[10] M.A. Bender and D. Slonim. The power of team exploration : Two robots can learn unlabeled directed graphs. In *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science (FOCS'94)*, pages 75–85, 1974.

[11] M. Bertier, F. Bonnet, A.-M. Kermarrec, V. Leroy, S. Peri, and M. Raynal. D2ht : the best of both worlds integrating rps and dht. In *Proceedings of the 8th European Dependable Computing Conference (EDCC'10)*, 2010.

[12] M. Biely, P. Robinson, and U. Schmid. Weak synchrony models and failure detectors for message-passing ($k$)set agreement. In *Proceedings of the 13rd International Conference on Principles of Distributed Systems (OPODIS'09)*, pages 285–299, 2009.

[13] K.P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and V. Minsky. Bimodal multicast. *ACM Transactions on Computer Systems*, 17(2) :41–88, 1999.

[14] F. Bonnet, A.-M. Kermarrec, and M. Raynal. Small-world networks : From theoretical bounds to systems. In *Proceedings of the 11th International Conference on Principles of Distributed Systems (OPODIS'07)*, pages 372–385, 2007.

[15] F. Bonnet, A.-M. Kermarrec, and M. Raynal. Small-world networks : Is there a mismatch between theory and practice. Technical report, Research Report IRISA #1849, May 2007.

[16] F. Bonnet and M. Raynal. Conditions for set agreement with an application to synchronous systems. *Journal of Computer Science and Technology*, 24(3) :418–433, 2009.

[17] F. Bonnet and M. Raynal. Looking for the weakest failure detector for $k$-set agreement in message-passing systems : Is $\pi_k$ the end of the road ? In *Proceedings of the 11st International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'09)*, pages 149–164, 2009.

[18] F. Bonnet and M. Raynal. The price of anonymity : Optimal consensus despite asynchrony, crash and anonymity. In *Proceedings of the 23rd International Symposium on Distributed Computing (DISC'09)*, pages 341–355, 2009.

[19] F. Bonnet and M. Raynal. Anonymous asynchronous systems : the case of failure detectors. In *Proceedings of the 24th International Symposium on Distributed Computing (DISC'10), (to appear)*, 2010.

[20] F. Bonnet and M. Raynal. Consensus in anonymous distributed systems : Is there a weakest failure detector ? In *Proceedings of the 24th International Conference on Advanced Information Networking and Applications (AINA 2010)*, pages 206–213, 2010.

[21] F. Bonnet and M. Raynal. Early consensus in message-passing systems enriched with a perfect failure detector and its application in the theta model. In *Proceedings of the 8th European Dependable Computing Conference (EDCC'10)*, pages 107–116, 2010.

[22] F. Bonnet and M. Raynal. The price of anonymity : Optimal consensus despite asynchrony, crash and anonymity. Technical report, Research Report IRISA #1918, December 2008.

[23] F. Bonnet and M. Raynal. Anonymous asynchronous systems : The case of failure detectors. Technical report, Research Report IRISA #1945, January 2010.

[24] F. Bonnet and M. Raynal. Consensus in anonymous distributed systems : Is there a weakest failure detector ? Technical report, Research Report IRISA #1938, October 2009.

[25] T. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4) :685–722, 1996.

[26] T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2) :225–267, 1996.

[27] C. Delporte-Gallet, H. Fauconnier, and R. Guerraoui. A realistic look at failure detectors. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks (DSN'02)*, pages 345–352, 2002.

[28] C. Delporte-Gallet, H. Fauconnier, and R. Guerraoui. Shared memory *vs* message passing. Technical report, Tech Report IC/2003/77, EPFL, December 2003.

[29] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, V. Hadzilacos, P. Kouznetsov, and S. Toueg. The weakest failure detectors to solve certain fundamental problems in distributed computing. In *Proceedings of the 23rd ACM Symposium on Principles of Distributed Computing (PODC'04)*, pages 338–346, 2004.

[30] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, and A. Tielmann. The weakest failure detector for message passing set-agreement. In *Proceedings of the 22nd International Symposium on Distributed Computing (DISC'08)*, pages 109–120, 2008.

[31] A.J. Demers, D.H. Greene, C. Hauser, W. Irish, and J. Larson. Epidemic algorithms for replicated database maintenance. In *Proceedings of the 6th ACM Symposium on Principles of Distributed Computing (PODC'87)*, pages 1–12, 1987.

[32] S. Dobrev, J. Jansson, K. Sadakane, and W.K. Sung. Finding short right-hand-on-the-wall walks in undirected graphs. In *Proceedings of the 12th Colloquium on Structural Information and Communication Complexity (SIROCCO'05)*, pages 127–139, 2005.

[33] P.T. Eugster, R. Guerraoui, B. Handurukande, A.-M. Kermarrec, and P. Kouznetsov. Lightweight probabilistic broadcast. *ACM Transactions on Computer Systems*, 21(1) :341–374, 2003.

[34] P.T Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulié. Epidemic information dissemination in distributed systems. *IEEE Computer*, 37(5) :60–67, 2004.

[35] M.J. Fischer, N.A. Lynch, and M.S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2) :374–382, 1985.

[36] P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. Computing without communicating : Ring exploration by asynchronous oblivious robots. In *Proceedings of the 11th International Conference on Principles of Distributed Systems (OPODIS'07)*, pages 105–118, 2007.

[37] P. Fraigniaud, P. Gauron, and M. Latapy. Combining the use of clustering and scale-free nature of user exchanges into a simple and efficient p2p system. In *Proceedings of European Conference on Parallelism (EUROPAR'05)*, pages 1163–1172, 2005.

[38] P. Fraigniaud, C. Gavoille, A. Kosowski, E. Lebhar, and Z. Lotker. Universal augmentation schemes for network navigability : Overcoming the $\sqrt{n}$ barrier. In *Proceedings of the 19th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'07)*, pages 1–7, 2007.

[39] P. Fraigniaud, C. Gavoille, and C. Paul. Eclecticism shrinks even small worlds. In *Proceedings of the 23th ACM Symposium on Principles of Distributed Computing (PODC'04)*, pages 169–178, 2004.

[40] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science*, 345(2-3) :331–344, 2005.

[41] P. Fraigniaud, D. Ilcinkas, S. Rajsbaum, and S. Tixeuil. Space lower bounds for graph exploration via reduced automata. In *Proceedings of the 12th Colloquium on Structural Information and Communication Complexity (SIROCCO'05)*, pages 140–154, 2005.

[42] A. Franchi, L. Freda, G. Oriolo, and M. Vendittelli. A randomized strategy for cooperative robot exploration. In *International Conference on Robotics and Automation (ICRA'07)*, pages 768–774, 2007.

[43] L. Gasieniec, R. Klasing, R.A. Martin, A. Navarra, and X. Zhang. Fast periodic graph exploration with constant memory. In *Proceedings of the 14th Colloquium on Structural Information and Communication Complexity (SIROCCO'07)*, pages 26–40, 2007.

[44] D.K. Gifford. Weighted voting for replicated data. In *Proceedings of the 7th ACM Symposium on Operating System Principles (SOSP'79)*, pages 150–172, 1979.

[45] R. Grossi, A. Pietracaprina, and G. Pucci. Optimal deterministic protocols for mobile robots on a grid. *Information and Computation*, 173(2) :132–142, 2002.

[46] R. Guerraoui and E. Ruppert. Anonymous and fault-tolerant shared memory computing. *Distributed Computing*, 20(3) :165–177, 2007.

[47] J.-M. Helary, M. Hurfin, A. Mostefaoui, M. Raynal, and F. Tronel. Computing global functions in asynchronous distributed systems with perfect failure detectors. *IEEE Transactions on Parallel and Distributed Systems*, 11(9) :897–909, 2000.

[48] Suzuki I. and Yamashita M. Distributed anonymous mobile robots : Formation of geometric patterns. *SIAM Journal on Computing*, 28(4) :1347–1363, 1999.

[49] D. Ilcinkas. Setting port numbers for fast graph exploration. In *Proceedings of the 13th Colloquium on Structural Information and Communication Complexity (SIROCCO'06)*, pages 59–69, 2006.

[50] P. Jayanti and S. Toueg. Every problem has a weakest failure detector. In *Proceedings of the 27th ACM Symposium on Principles of Distributed Computing (PODC'08)*, pages 75–84, 2008.

[51] M. Jelasity and O. Babaoglu. T-man : Gossip-based overlay topology management. In *Proceedings of the 3rd International Workshop on Engineering Self-Organising Applications (ESOA'05)*, pages 1–15, 2005.

[52] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. The peer sampling service : Experimental evaluation of unstructured gossip-based implementations. In *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware (Middleware'04)*, pages 79–98, 2004.

[53] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems*, 23(3) :219–252, 2005.

[54] J. Kleinberg. Navigation in a small world. *Nature*, 406 :845, 2000.

[55] J. Kleinberg. The small-world phenomenon : an algorithmic perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing (STOC'00)*, pages 163–170, 2000.

[56] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publisher, 1996.

[57] S. Milgram. The small-world problem. *Psychology Today*, 61(2) :60–67, 1967.

[58] A. Mostefaoui, S. Rajsbaum, M. Raynal, and C. Travers. The combined power of conditions and information on failures to solve asynchronous set agreement. *SIAM Journal of Computing*, 38(4) :1574–1601, 2008.

[59] A. Mostefaoui, S. Rajsbaum, M. Raynal, and C. Travers. On the computability power and the robustness of set agreement-oriented failure detector classes. *Distributed Computing*, 21(3) :201–222, 2008.

[60] A. Mostefaoui and M. Raynal. Solving consensus using chandra-toueg's unreliable failure detectors : a general quorum-based approach. In *Proceedings of the 13rd International Symposium on Distributed Computing (DISC'99)*, pages 49–63, 1999.

[61] M. Naor and U. Wieder. Know the neighbor's neighbor : Better routing for skip-graphs and small worlds. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS'04)*, pages 269–277, 2004.

[62] P. Panaite and A. Pelc. Impact of topographic information on graph exploration efficiency. *Networks*, 36(2) :96–103, 2000.

[63] H.A. Rollik. Automaten in planaren graphen. *Acta Informatica*, 13 :287–298, 1980.

[64] `http://peersim.sourceforge.net/`.

[65] S. Voulgaris, D. Gavidia, and M. van Steen. Cyclon : Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2) :197–217, 2005.

[66] S. Voulgaris, E. Rivière, A.-M. Kermarrec, and M. van Steen. Sub-2-sub : Self-organizing content-based publish and subscribe for dynamic and large scale collaborative networks. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS'06)*, 2006.

[67] S. Voulgaris and M. van Steen. Epidemic-style management of semantic overlays for content-based searching. In *Proceedings of the International Conference on Parallel and Distributed Computing (Euro-Par'05)*, pages 1143–1152, 2005.

[68] D.J. Watts and S.H. Strogatz. Collective dynamics of small-world networks. *Nature*, 393 :440–442, 1998.

[69] M. Yamashita and T. Kameda. Computing on anonymous networks : Part i-characterizing the solvable cases. *IEEE Transactions on Parallel Distributed Systems*, 7(1) :69–89, 1996.

[70] M. Yamashita and T. Kameda. Computing on anonymous networks : Part ii-decision and membership problems. *IEEE Transactions on Parallel Distributed Systems*, 7(1) :90–96, 1996.

[71] A.C.C. Yao. On constructing minimum spanning trees in $k$-dimensional space and related problems. *SIAM Journal of Computing*, 11 :721–736, 1982.

[72] R. Yared, X. Defago, and M. Wiesmann. Collision prevention using group communication for asynchronous cooperative mobile robots. In *Proceedings of the 21st International Conference on Advanced Information Networking and Applications (AINA 2007)*, pages 244–249, 2007.

VU :
**Le directeur de thèse**
(Nom et prénom)

VU :
**Le responsable de l'école doctorale**
(Nom et Prénom)

**VU pour autorisation de soutenance**

**Rennes, le**
**Le président de l'Université de Rennes 1**
**Guy CATHELINEAU**

**VU après soutenance pour autorisation de publication :**
**Le Président de Jury,**
(Nom et Prénom)