



Université Aix Marseille

Ecole Doctorale 184

UFR 7279

Laboratoire d'informatique fondamentale

Thèse présentée pour obtenir le grade universitaire de docteur

Discipline : Informatique

Antoine NAUDIN

Impact des connaissances initiales sur la calculabilité distribuée

Soutenue le JJ/MM/AAAA devant le jury :

Maria POTOP-BUTUCARU	LIP6/UPMC	Rapporteur
Nicolas NISSE	I3S/INRIA	Rapporteur
David ILCINKAS	LaBRI/CNRS	Examineur
Yann VAXES	LIF/AMU	Examineur
Emmanuel GODARD	LIF/AMU	Directeur de thèse
Jérémie CHALOPIN	LIF/CNRS/AMU	Directeur de thèse

Sommaire

I	Calculabilité distribuée du modèle des Participants inconnus	9
1	Préliminaires	15
1.1	Définitions et notations	15
1.1.1	Graphe et Graphe dirigé	15
1.1.2	Graphe dirigé symétrique	16
1.1.3	Chemin, longueur et distance dirigée	16
1.1.4	Sommets accessibles	17
1.1.5	Graphe dirigé étiqueté	18
1.1.6	Réétiquetage de graphes	18
1.2	Système distribué à passage de messages	18
1.2.1	Réseau de communication	19
1.2.2	Étiquetage des ports	19
1.2.3	Représentation d'un algorithme distribué	20
1.2.4	Représentation d'une exécution d'un algorithme distribué	20
1.2.5	Spécification d'un problème	21
1.2.6	Propriétés des exécutions	22
1.2.7	Propriétés d'un algorithme	22
1.2.8	Identité des processus	23
1.3	Modèle des participants inconnus	23
1.3.1	Présentation du modèle	23
1.3.2	Graphe de communication	24
1.3.3	Étiquetage : Codage de l'état des processus	26
2	Cartographie des sommets accessibles	29
2.1	Description du problème $\mathcal{REACH}$	29
2.2	Description de l'algorithme d'inondation	29
2.2.1	Boîte aux lettres	30

2.2.2	Déroulement de l'algorithme	30
2.3	Correction de l'algorithme	31
2.4	Algorithme pour $\mathcal{REACH}$	33
2.4.1	Correction de l'algorithme 2	34
3	Cartographie complète du réseau	35
3.1	Description du problème $\mathcal{CARTO}$	35
3.2	Lemme d'isolement	36
3.3	Condition nécessaire $\mathfrak{C}_{\mathcal{CARTO}}$	38
3.4	Condition suffisante	39
3.4.1	Présentation de l'algorithme	39
3.4.2	Correction de l'algorithme	40
3.5	Exemples	41
4	Élection	43
4.1	Description du problème $\mathcal{ELEC}$	43
4.2	Condition nécessaire $\mathfrak{C}_{\mathcal{ELEC}}$	43
4.3	Condition suffisante	45
4.3.1	Présentation de l'algorithme	45
4.3.2	Correction de l'algorithme	46
4.4	Exemples	49
5	k-Élection de leader	51
5.1	Description du problème $k - \mathcal{ELEC}$	51
5.2	Condition nécessaire $\mathfrak{C}_{k-\mathcal{ELEC}}$	51
5.3	Condition suffisante	52
5.4	Exemples	53
	Conclusion	53
6	Élection dans le modèle des participants inconnus anonymes	55
	Introduction	55
6.1	Définitions	57
6.1.1	Fonction simpliciale, Morphisme et Revêtement dirigé	57
6.2	Modèle des participants inconnus et anonymes	60
6.2.1	Codage de l'état des processus	60
6.3	Problème de l'élection	61
6.4	Premiers résultats d'impossibilité et conditions nécessaires	61
6.4.1	Lemme de relèvement	61

6.4.2	Lemme d'isolement	63
6.5	Algorithme d'élection pour les familles de graphes de taille bornée	65
6.5.1	Présentation de l'algorithme d'élection	65
6.5.2	États internes des processus	65
6.5.3	Les connaissances globales	66
6.5.4	Les messages	67
6.5.5	Un ordre sur les vues locales des processus	67
6.5.6	Les primitives et fonctions	67
6.5.7	Présentation de l'algorithme	68
6.6	Correction de l'algorithme	69
	Conclusion	81
II	Exploration de graphes anonymes	83
7	Exploration de graphes anonymes par un agent mobile	85
	Introduction	85
7.1	Définitions et Notations	87
7.1.1	Graphes	87
7.1.2	Graphes dirigés	87
7.1.3	Chemins, Cycles et Boucles	88
7.1.4	Étiquetage des ports	89
7.1.5	Graphes anonymes et avec homonymes	89
7.1.6	Homomorphisme, Revêtement et Quasi-revêtement	89
7.2	Modèle standard d'agent mobile	92
7.2.1	Problèmes et Algorithmes à base d'agent mobile	93
7.2.2	Propriétés des exécutions et détection de la terminaison	93
7.2.3	État de l'art des agents mobiles	93
7.3	Calculabilité de l'exploration sans connaissance initiale	94
7.3.1	Le problème de l'exploration sans connaissance	94
7.3.2	Calculabilité de l'exploration	97
	Conclusion	100
8	Exploration de graphes anonymes par un agent mobile avec jumelles	103
	Introduction	103
8.1	Agents mobiles avec jumelles	104
8.1.1	Modèle	104
8.1.2	Étiquetage de jumelles	104

8.2	Définitions	105
8.2.1	Vues des sommets	105
8.2.2	Vues avec étiquetage de jumelles	106
8.2.3	Complexes Simpliciaux	106
8.2.4	Complexe de cliques	108
8.2.5	Homotopie et Contractibilité	110
8.2.6	Simple Connexité	113
8.3	Caractérisation des graphes explorables avec jumelles	113
8.4	Exploration de $\mathcal{FC}$	117
8.4.1	Présentation de l'algorithme	117
8.4.2	Correction de l'algorithme	118
8.5	Complexité du problème d'exploration	122
8.5.1	L'algorithme	123
8.5.2	Correction	123
	Conclusion	126
9	Exploration efficace avec jumelles	127
	Introduction	127
9.1	Préliminaires	128
9.1.1	Exploration à base d'agent mobile avec jumelles	128
9.1.2	Définitions et Notations	128
9.1.3	Graphes de Weetman	129
9.2	Algorithme d'exploration efficace pour les graphes de Weetman	132
9.2.1	La carte $\mathcal{M}$	133
9.2.2	Étiquetage de jumelles	134
9.2.3	Structures utilisées	135
9.2.4	Description d'une phase	136
9.2.5	Construction et mise à jour de la carte $\mathcal{M}$	137
9.3	Correction de l'algorithme	139
9.4	Complexité de l'algorithme	145
9.4.1	Exécution sur des graphes qui ne sont pas des graphes de Weetman	146
	Conclusion	148
	Conclusion générale et perspective	151
	Bibliographie	153

Introduction

Un algorithme est une suite d'instructions décrivant la résolution d'une fonction ou d'une tâche. Comme une recette de cuisine, un algorithme permet, à partir d'ingrédients de base, les données d'entrée, de produire un résultat final voulu. En informatique, les algorithmes permettent de décrire formellement une série d'instructions qui seront exécutées par une entité de calcul autonome appelée *processus*. Pour exécuter n'importe quel type d'algorithme et réaliser des fonctions arbitraires, les processus sont capables de traiter et de stocker de l'information. En pratique, un processus peut prendre la forme d'un processeur, mais peut tout aussi être un homme ou un animal.

Pour partager une même ressource et réaliser des tâches collaboratives, plusieurs processus peuvent être connectés entre eux. Cette mise en relation de plusieurs processus au sein d'un même environnement est appelée un système distribué. Les exemples de systèmes distribués sont multiples : les bases de données distribuées, les processus partageant une mémoire (multi-processeurs), les grilles de calculs, les radars, les drones ou encore les robots. Les applications de tels systèmes sont nombreuses et variées comme par exemple répartir la charge de travail entre différents processus, aider à découvrir des environnements inconnus en déployant des réseaux de capteurs, surveiller un environnement comme le plus grand système d'information existant : le réseau internet.

Au sein d'un réseau, chaque processus exécute localement son algorithme et il est primordial que les processus interagissent entre eux via un médium de communication : mémoire, envoi/réception de messages via des canaux dédiés. Notons que sans communication, tous les processus exécuteraient leur algorithme individuellement sans prendre en compte les autres processus. La mise en réseau serait donc inutile.

Dans un réseau, il est nécessaire que les processus s'accordent sur un *algorithme distribué* ou *protocole distribué*. Dès les premiers réseaux informatiques, des protocoles ou algorithmes distribués ont été implémentés pour répondre aux contraintes des différents systèmes. Par exemple, il a été nécessaire de concevoir des algorithmes d'exclusion mutuelle (problème introduit par Edsger Dijkstra en 1965) pour synchroniser les accès à une mémoire partagée ou un canal de communication afin que deux processus communiquant en même temps dans le réseau ne puissent pas se perturber mutuellement. Ces problèmes d'un genre nouveau sont appelés problèmes distribués. De nos jours, les protocoles distribués sont couramment utilisés avec par exemple le routage du réseau internet, la synchronisation des exécutions entre processus, la validation d'une transaction distribuée (commerce en ligne) ou le partage et l'indexation des données réparties dans un réseau comme internet (moteur de recherche).

Dans un système distribué, chaque processus effectue ses calculs à partir des données qui lui sont accessibles localement. Une configuration du réseau regroupe l'ensemble des états locaux de chaque processus du réseau à un moment donné. Un algorithme distribué réalise une fonction qui, à partir d'une configuration initiale du système, modifie les états locaux (chaque processus exécute le même algorithme et modifie son propre état local) pour obtenir une configuration

finale du système. Comme chaque processus exécute son algorithme à sa propre vitesse, plusieurs ordonnancements des exécutions sont possibles. Par conséquent, différentes configurations finales peuvent être atteintes à partir d'une même configuration initiale donnée en fonction de l'ordre dans lequel les processus mettent à jour leur état local. Ces différents ordonnancements possibles peuvent avoir de multiples origines en pratique, comme des messages perdus ou retardés un temps arbitrairement long par un routeur mal configuré, des processus qui soudainement deviennent très lents ou tombent en panne, ... Par conséquent, un problème distribué est spécifié par une relation liant, un ensemble donné de configurations initiales, les réseaux pour lesquels l'algorithme a été conçu, et l'ensemble des configurations finales admissibles, les configurations correctes vis-à-vis du problème à résoudre et de la configuration initiale du réseau. Un problème distribué dépend donc de l'état global du système et non de l'état des machines individuellement.

Plusieurs paramètres influent sur la calculabilité au sein des différents systèmes distribués : la topologie du réseau, les canaux de communication comme les différents ordres possibles pour acheminer un paquet ou la vitesse d'acheminement des paquets dans les canaux. La synchronisation "globale" de l'exécution est aussi déterminante. Dans un système *synchrone*, les processus ont accès à une horloge globale leur permettant de synchroniser leurs calculs, calculer des intervalles de temps. Dans un système *asynchrone*, aucune hypothèse n'est faite ni sur le temps d'exécution des processus, ni sur le temps de transmission des messages. Lors de la conception d'un algorithme, des hypothèses sont émises sur les paramètres des systèmes distribués où l'algorithme sera exécuté. Ces hypothèses émises peuvent être vues comme des informations données, plus ou moins indirectement, aux processus du réseau. Un algorithme supposant que le réseau possède n sommets ou que chaque processus connaît son degré, se basera sur ces propriétés, intrinsèques du réseau, pour résoudre son problème.

Nous distinguerons plusieurs types d'informations : les connaissances structurelles et les informations structurelles. Les connaissances structurelles sont liées aux aspects topologiques du réseau. Elles sont des propriétés globales partagées par tous les processus. Par exemple, supposer que les processus connaissent le nombre de processus total du réseau ou savoir le réseau est un anneau est une connaissance structurelle donnée aux processus.

Les informations structurelles sont informations locales avec des propriétés globales mais spécifiques à chaque processus. Par exemple, supposer dans un algorithme distribué que tous les processus possèdent une identité unique implique que les processus possèdent une identité qui leur est propre et la connaissent. De plus, la correction de l'algorithme reposera sur cette hypothèse forte pour résoudre son problème. Exécuter cet algorithme dans un réseau où plusieurs processus possèdent une même identité pourrait mener à des exécutions incorrectes. En effet, les processus sont des machines et exécuteront leur algorithme de manière arbitraire dans n'importe quel graphe, et ce, même si les informations données aux processus via l'algorithme ne correspondent pas au réseau. Notons que, dans certains réseaux, certaines propriétés sont indécidables localement pour un processus.

L'ensemble des connaissances initiales peut se formaliser par une famille de graphes. L'idée est de faire correspondre, à toute connaissance initiale (ex. le réseau a moins de n sommets ou possède des identités uniques), une famille précise de graphes (ex. l'ensemble des graphes ayant moins de n sommets ou tous les graphes dotés d'identités uniques). Par définition, un algorithme résolvant un problème en supposant que les processus connaissent le nombre n de processus dans le réseau implique que cet algorithme devra résoudre le problème sur tout réseau à n processus, peu importe sa topologie. Ainsi, la connaissance initiale restreint l'application des algorithmes distribués et impose aux algorithmes de résoudre le problème dans l'ensemble des graphes satisfaisant la connaissance. Le problème devient donc le suivant : étant donné un problème à résoudre, quelles sont les familles de graphes pour lesquelles il existe un algorithme

résolvant le problème dans tous les graphes de cette famille ? Plus généralement, existe-t-il une famille maximale rassemblant tous ces réseaux ? Notons que donner des informations structurales aux processus nécessite une intervention logicielle pour faire parvenir les connaissances aux processus. Modifier un système distribué déjà en application, comme modifier l'infrastructure, la topologie, les primitives de communication ou la synchronisation des processus, peut être extrêmement coûteux, voire impossible à réaliser sans perturber les protocoles en cours d'exécution. Il peut être donc préférable dans certains cas de diffuser un nouvel algorithme ou juste une connaissance globale plutôt que de modifier l'environnement des processus.

Pour concevoir des protocoles distribués robustes et corrects dans un réseau donné (ou un ensemble de réseaux), il est nécessaire d'étudier théoriquement les systèmes distribués et d'identifier les paramètres des réseaux, nécessaires et suffisants, pour résoudre une tâche donnée. Plusieurs questions émergent vis-à-vis des protocoles et des différents réseaux. Peut-on réutiliser un même algorithme dans différents réseaux ? Quels sont les paramètres du réseau impactant la calculabilité distribuée du système ?

Répondre à de telles questions permet d'accélérer et de simplifier la conception des réseaux en identifiant les paramètres importants à prendre en compte lors de la conception des algorithmes distribués ; par exemple, en identifiant les primitives nécessaires (ou superflues) à l'accomplissement de certaines tâches. De plus, grâce aux études théoriques de ces systèmes, il est possible de prouver que des protocoles distribués déjà implémentés sont corrects (ou non) dans le système où ils sont exécutés.

Un modèle distribué abstrait les détails techniques des réseaux et de leurs implémentations pour se concentrer sur leur calculabilité et pouvoir mieux étudier les différents paramètres et primitives pouvant exister. Un modèle distribué est défini par une primitive de communication ; par les différents ordonnancements possibles des processus ; par les connaissances initiales données aux processus et par le type de terminaison des exécutions considéré.

Plusieurs modèles théoriques, plus ou moins proches des réseaux implémentés, ont été proposés. Nous définissons ci-dessous les modèles étudiés dans cette thèse en détail, puis, nous présenterons succinctement quelques autres pour avoir un aperçu de la diversité des modèles existants. Nous nous intéresserons principalement au modèle dit "à passage de messages" et au modèle dit "à agents mobiles" présentés ci-dessous.

- le modèle à passages de messages : le support de communication est composé de canaux de communication reliant les processus deux à deux. Les processus communiquent par envois et réceptions de messages via ces canaux. Dans ce modèle, les données sont acheminées aux entités de calculs dispersés dans le réseau. Le modèle peut être synchrone ou asynchrone et plusieurs types de terminaisons ont été étudiés pour ce modèle.
- le modèle à agent mobile : ce modèle se place dans un paradigme différent. Au lieu d'envoyer les données aux entités de calculs, ce sont les entités de calculs qui se déplacent directement aux données. Ici, le réseau est toujours constitué de noeuds et de canaux les reliant deux à deux. La différence est que les calculs sont réalisés par une entité autonome se déplaçant dans le réseau via les canaux. Les noeuds du réseau, quant à eux, ne produisent aucun calcul. L'agent étant seul, pour un graphe donné, un seul ordonnancement des exécutions est possible et différents types de terminaisons existent aussi pour ce modèle.

D'autres modèles existent, comme le modèle à mémoire partagée qui est très proche de ce qu'on observe sur une seule machine. Dans ce modèle, le support de communication est une mémoire partagée par tous les processus du réseau. Les processus communiquent par lecture et écriture (synchrone ou asynchrone) de cette mémoire. Le modèle à calculs locaux est un modèle plus théorique que les trois premiers modèles présentés ci-dessus. Dans ce modèle, les processus ne

communiquent pas directement, mais accèdent à l'état interne des processus voisins (de manière synchrone ou asynchrone en fonction des modèles) pour calculer leur nouvel état.

Des problèmes distribués sont dits, *fondamentaux*, dans le sens où ils capturent la difficulté des modèles et permettent (souvent) d'obtenir des techniques pour résoudre n'importe quelle tâche dans le modèle. De plus, ils servent de briques de base nécessaire pour résoudre des problèmes beaucoup plus complexes. Ces problèmes peuvent aussi être utilisés pour comparer certains modèles et les hiérarchiser en fonction de la capacité de résolution. Les problèmes de consensus entre processus sont fondamentaux en algorithmique distribuée due à la décentralisation des calculs et des données. Le problème de l'élection consiste à ce qu'un unique processus décide *élu* et à ce que tous les autres décident *non-élu*. Notons que les processus sont au courant de leur propre statut final (*élu* ou *non-élu*) sans forcément être au courant des statuts pris par les autres. L'élection est un problème indispensable à résoudre pour de nombreuses tâches de plus haut niveau comme les transactions dans une base de données distribuée.

Il apparaît que, en fonction des systèmes, certains problèmes fondamentaux deviennent impossibles à résoudre. Par exemple, en présence de processus possédant une identité unique dans un modèle à passage de messages asynchrone, il est possible de réaliser une élection (par exemple en élisant le processus ayant la plus petite identité). À l'inverse, si les processus ne possèdent pas de telles identités ou si leurs identités ne sont pas uniques, alors l'élection devient impossible à résoudre, en général, sans hypothèses additionnelles. Le problème du consensus consiste à ce que tous les processus décident une même valeur, elle-même proposée par un processus participant. D'autres problèmes existent comme par exemple la cartographie du réseau ou calculer un arbre couvrant pour diffuser des messages à l'ensemble des noeuds du réseau. Le problème de renommage des processus est utilisé pour réduire l'espace mémoire nécessaire au nommage des processus dans des contextes où la mémoire est critique. Un problème plus difficile à résoudre est l'énumération des processus. Il s'agit de renommer les processus avec des entiers compris dans l'intervalle $[1 \dots n]$ où n est le nombre de processus. Résoudre ce problème permet par exemple de résoudre l'élection ou de compter le nombre de processus présents dans le réseau.

Universalité des algorithmes et connaissances.

Deux difficultés majeures apparaissent lors de la conception d'algorithmes distribués. Premièrement, identifier les paramètres d'un système nécessaires et/ou suffisants au bon fonctionnement de l'algorithme. Par exemple, est-il nécessaire que tous les processus possèdent une identité unique ? Que le réseau soit un cycle ? Peut-on concevoir un algorithme fonctionnant dans l'ensemble des réseaux, peu importe sa topologie ? Est-il nécessaire que les processus connaissent le nombre total de processus pour que les exécutions de l'algorithme terminent correctement ? Notons que des algorithmes corrects dans un maximum de réseaux peuvent ne pas être efficaces pour résoudre une tâche distribuée et qu'il peut être préférable de concevoir un algorithme plus efficace, mais spécialisé pour certains réseaux.

La deuxième problématique majeure en algorithmique distribuée est la détection de la terminaison. Dans un système totalement distribué, les processus n'ont qu'une vision locale du système et n'ont aucune connaissance globale. Le problème de la détection de la terminaison est un problème en soi et ne dépend pas tout le temps du problème à résoudre. Il est possible qu'un processus possède un état final correct vis-à-vis du problème à résoudre, mais ne le détecte jamais. Il continuera donc à envoyer des messages et à en attendre bien après avoir fini son calcul. Cette non-terminaison des exécutions peut perturber le réseau avec une activité permanente des processus, avec une inondation du réseau par des messages inutiles (pouvant ralentir l'exécution d'autres protocoles distribués exécutés en parallèle) ...

Pour terminer une exécution, un processus doit décider une valeur finale. Plusieurs types de terminaisons existent. La terminaison explicite consiste à contraindre les processus à décider une et une seule fois lors d’une exécution. Une terminaison plus faible est la terminaison implicite consistant à laisser les processus changer leur décision plusieurs fois lors d’une exécution. Cette terminaison est intéressante pour concevoir des algorithmes auto stabilisant. Dans cette thèse, nous nous concentrerons sur la terminaison explicite. De par la *dynamacité* du réseau et l’asynchronicité des interactions entre processus, un processus ne peut jamais arrêter totalement son exécution sans prendre le risque d’empêcher de couper toute communication entre deux parties du réseau. Par exemple, un processus peut séparer le réseau en deux parties et tout message à destination de l’une ou de l’autre partie devra passer par ce processus. Notons que pouvoir arrêter tous les processus en fin d’exécution est intéressant mais, malheureusement, est impossible à obtenir dans la plupart des modèles asynchrones. Nous considérerons donc une détection locale et explicite de la terminaison. Cette détection consiste, pour tout processus, à identifier une étape de l’exécution où le processus à terminer localement son calcul et à décider une valeur finale, i.e., il ne la changera plus jamais. Une telle détection ne permet pas aux processus de détecter si les autres processus ont aussi terminé leur exécution ou non. Bien qu’elle ne permette pas d’arrêter complètement les processus en fin d’exécution, la détection locale de la terminaison est nécessaire pour obtenir une terminaison explicite et n’est pas triviale à obtenir dans certains modèles. Comme nous le verrons, même si les processus n’arrêtent pas complètement leur exécution, il est possible de concevoir des algorithmes silencieux qui ont la particularité de ne plus envoyer de message et de ne plus effectuer de calcul au bout d’un temps fini après la décision des processus. Les processus seront alors en attente permanente de message (qui n’arriveront peut-être jamais) mais ne perturberont pas le réseau inutilement.

But de cette thèse

Dans cette thèse, nous montrons l’impact des connaissances initiales sur la calculabilité de deux systèmes distribués particuliers : le modèle, à passage de messages, des participants inconnus (avec identités et anonymes) et un nouveau modèle d’agents mobiles, appelé agent mobile avec jumelles.

La méthodologie utilisée est une approche algébrique consistant à capturer la symétrie des réseaux et de leurs exécutions grâce à des structures algébriques reposant sur des morphismes adaptés au modèle étudié. Ces structures nous permettent, par la suite, de mieux comprendre les difficultés liées au modèle et d’obtenir des preuves d’impossibilités grâce à un ”lemme de relèvement des exécutions”. Un problème peut être impossible à résoudre dans un modèle donné pour différentes raisons. Les impossibilités dites *spécifiques* induites par problème qui est impossible à résoudre sur un graphe en particulier et les impossibilités *universelles* où, dans ce cas, le problème est impossible à résoudre sur un ensemble de graphes avec un même algorithme. Notons que dans certains cas, le problème peut être résolu individuellement sur chacun des graphes mais être impossible à résoudre sur l’ensemble. Cela revient à dire que les processus peuvent résoudre le problème en connaissant la topologie du réseau. Pour étudier la calculabilité d’un problème distribué dans un modèle donné, il est donc important d’identifier les réseaux pour lesquels il existe une impossibilité spécifique et les relations liant les réseaux où il existe une impossibilité universelle. L’approche algébrique s’avère très adaptée pour résoudre ces questions dans un nombre très important de modèles courants.

L’approche algébrique susdite est décrite en détail dans [God15]. L’approche algébrique a été introduite par Angluin [Ang80]. Le résultat obtenu est très fort et montre que l’élection est impossible à résoudre dans un réseau anonyme où les processus exécutent leur algorithme

de manière synchrone. Cette approche a été généralisée dans [BLMS90] pour d'autres modèles de calculs tels les systèmes de réétiquetage de graphes. Puis, Yamashita et Kameda [YK96c] l'appliquent aux passages de messages et caractérisent la calculabilité de différents problèmes fondamentaux en présence de connaissances structurelles particulières. Par exemple, ils étudient si l'élection est réalisable en connaissant le diamètre ou une borne sur le diamètre d'un graphe ? Si la cartographie du réseau est réalisable sans connaissance ? Pour chacun de ces problèmes, ils donnent une condition nécessaire sur les réseaux où le problème peut être résolu en présence de la connaissance donnée. Par la suite, dans [BV99], Boldi et Vigna caractérisent les tâches distribuées calculables sans connaissance pour un certain nombre de modèles. Maintenant, il est reconnu que l'approche algébrique est adaptée à ce genre d'étude et est, maintenant, systématiquement utilisée pour étudier la calculabilité d'un grand nombre de modèles [God02, GMM00, Cha06, CDK10b] et permet de manipuler/étudier aisément les différentes connaissances et hypothèses émises dans la spécification des modèles

Dans cette thèse, nous utiliserons cette approche algébrique pour des modèles particuliers qui n'ont jamais été étudiés sous cet angle et pour qui, à première vue, les outils algébriques utilisés jusqu'alors ne sont pas adaptés.

Plan.

Dans la partie I, nous étudions un modèle à passage de messages particulier, appelé modèle des participants inconnus. Ce modèle permet d'étudier des réseaux où des processus peuvent découvrir de nouveaux voisins grâce aux messages reçus. Dans ce modèle, nous caractériserons les connaissances globales, nécessaires et suffisantes, permettant de résoudre les problèmes fondamentaux suivant : cartographie des sommets accessibles (chapitre 2), cartographie du réseau (chapitre 3), élection (chapitre 4) et k -élection (chapitre 5).

Dans le chapitre 6, nous étendons le modèle des participants inconnus aux réseaux anonymes. Avec la même méthodologie que pour les chapitres précédents, nous caractérisons les connaissances globales nécessaires nous permettant de résoudre le problème de l'élection dans ce modèle pour des familles de graphes de taille bornée. Les résultats présentés dans ces chapitres ont été en partie publiés lors de la conférence SIROCCO en 2014 [CGN14] et une version journal contenant tous les résultats est en cours de rédaction.

Dans la partie II, nous nous placerons dans un modèle à agent mobile et étudions le problème de l'exploration.

Dans le chapitre 7, nous introduisons le modèle à agent mobile et le problème de l'exploration de graphes anonymes sans connaissance. Le problème de l'exploration consiste à explorer, avec un agent mobile, tous les sommets du graphe et s'arrêter. Nous présentons une caractérisation de la calculabilité de l'exploration sans connaissance initiale par un agent mobile. Nous montrons que la famille des arbres est la famille de graphes maximale pour l'exploration, i.e., la famille maximale pouvant être explorée par un même algorithme sans connaissance initiale. Cette caractérisation, issue du folklore, est exprimée de manière algébrique (via les revêtements de graphe) pour montrer la similitude avec les chapitres suivants.

Dans le chapitre 8, nous étendons le modèle en dotant l'agent d'un capteur, noté jumelles, lui permettant d'élargir sa vision locale du réseau. Cette connaissance locale additionnelle donnée à l'agent permet à l'agent de percevoir le graphe induit par son voisinage dans le réseau. Nous montrons qu'augmenter les informations locales de l'agent améliore considérablement la calculabilité de l'exploration des graphes anonymes en permettant la détection de la terminaison des exécutions sur un ensemble de graphes bien plus vaste que les arbres. Dans cette étude, nous caractérisons l'ensemble maximal des graphes pouvant être explorés par un agent mobile avec

jumelles. De plus, nous montrons que tout algorithme d'exploration pour cette famille (contenant tous les graphes explorables) requiert un nombre de mouvements très coûteux : ce nombre ne peut être borné par aucune fonction calculable. Ce résultat est étonnant car sans jumelles, la famille maximale (les arbres) peut être explorée en un nombre de mouvements linéaire. Les résultats présentés dans ce chapitre ont été publiés lors de la conférence DISC en 2015 [CGN15a].

Dans le chapitre 9, nous montrons comment les jumelles permettent d'explorer de grandes familles de graphes telles les graphes triangulés, les graphes "bridged" et certaines triangulations planaires, en un nombre de mouvements linéaire en la taille du graphe exploré, même si les graphes possèdent un nombre quadratique d'arêtes (ce qui n'est pas possible sans jumelles). Les résultats présentés dans ce chapitre sont en cours de rédaction. Une version courte a été présentée lors de la conférence Algotel en 2016 [CGN16b] et a reçu le prix de la meilleure communication étudiante. Une version intermédiaire est accessible sur Arxiv [CGN16a].

Première partie

Calculabilité distribuée du modèle des Participants inconnus

Introduction

Nous nous intéresserons au modèle dit “à passage de messages”. Dans ce modèle, les processus sont reliés deux à deux par des canaux de communications. Pour communiquer, les processus envoient et reçoivent des messages via les canaux de communication. Le réseau se modélise par un graphe $G = (V, E)$ où l’ensemble des sommets V correspond à l’ensemble des processus et l’ensemble des arêtes E correspond aux canaux de communication. Deux processus reliés par un canal dans le réseau sont des processus “voisins”. Pour différencier les processus, chaque noeud du réseau peut être identifié de façon unique.

Les réseaux distribués, par définition, ne possèdent pas de processus centraux permettant une synchronisation globale des calculs effectués par les processus. Il est donc nécessaire de définir des moyens de synchronisation garantissant la correction du calcul global effectué. Des protocoles distribués tels des algorithmes d’élection ([CSS77]) ou d’exclusion mutuelle (introduite par Edsger Dijkstra en 1965) permettent de synchroniser les calculs. Ces protocoles seront utilisés comme briques de bases “solides” pour des applications plus importantes comme les transactions sur des bases de données réparties et permettent aux applications les utilisant de s’abstraire des difficultés liées aux calculs distribués.

Pour garantir la correction des protocoles, il est nécessaire de prendre en compte les différents paramètres du réseau ainsi que des processus. Parmi la multitude de paramètres et donc de modèles distribués à passage de messages, nous allons étudier un modèle particulier appelé le modèle des participants inconnus. Ce modèle est intéressant, car il permet de modéliser l’aspect décentralisé et dynamique de certains réseaux actuels tel le réseau internet. En effet, de tels réseaux ne possèdent aucun utilisateur fixe centralisant les données telles que le nombre d’utilisateurs connectés ou l’emplacement dans le réseau. Ce modèle, introduit par Cavin, Sasson et Schiper dans [CSS04], est asynchrone (délai fini mais non borné sur les temps de transmission de messages et sur les temps de calcul des processus). Chaque processus du réseau est fiable, possède sa propre identité unique. De plus, les processus et les canaux de communication sont fiables.

La particularité du modèle est que chaque processus ne connaît, initialement, qu’un sous-ensemble de ses voisins dans le réseau. Au fil de l’exécution, les processus pourront ainsi recevoir des messages de voisins (ou participants) inconnus, i.e., d’un voisin que le processus ne connaît pas initialement. La découverte d’un nouveau voisin permet au processus de communiquer avec ce nouveau participant via le port de réception du message. Ce mécanisme de découverte de nouveaux voisins induit un réseau évolutif au fil de l’exécution. Ce modèle peut être comparé à un modèle “téléphonique” où recevoir un appel d’un numéro inconnu permet de rappeler l’émetteur ultérieurement par le même numéro.

Ce modèle est intéressant à étudier en présence de processus avec identités car, contrairement aux modèles à passage de messages classiques où les processus connaissent l’intégralité de leurs voisins initialement, ce modèle met en évidence des impossibilités liées aux communications asynchrones et non aux exécutions synchrones comme dans le cas classique. De plus, les

outils algébriques utilisés dans des modèles plus classiques comme les revêtements ne sont pas adaptés dans ce modèle. Il nous faudra identifier de nouvelles relations permettant d'exprimer les impossibilités propres au modèle des participants inconnus.

État de l'art

Dans [CSS04], Cavin, Sasson et Schiper introduisent le modèle des participants inconnus et caractérisent la calculabilité du problème du consensus lorsque les processus ne possèdent aucune information globale sur le réseau. Ils caractérisent (en fournissant une condition nécessaire et suffisante sur la topologie du graphe de communication initial), les graphes de communication initiaux où le consensus peut être résolu sans connaissance globale. Ils montrent que le consensus est impossible à résoudre si le graphe de communication initial comporte plus d'une composante puits (condition nécessaire) et donnent un algorithme résolvant le consensus dans tout graphe ayant au plus une composante puits (condition suffisante). À notre connaissance, seul le problème du consensus a été étudié pour ce modèle. Les participants inconnus ont été étudiés dans des versions tolérantes aux pannes : dans [GT07], le lien précis entre la synchronie des communications et la tolérance aux pannes est établi. Dans [ABFG08], le modèle est étudié en présence de fautes byzantines. Dans [GSAS12, ABGdSF16], le modèle est enrichi d'un détecteur de fautes. Seulement [CSS04] considère un réseau fixé avec participants inconnus sûr et asynchrone.

La caractérisation présentée dans [CSS04] ne considérant pas des connaissances globales données aux processus, nous étudions dans ce chapitre leur impact sur la calculabilité du modèle. Ainsi, le modèle des participants inconnus en présence de connaissances globales n'a jamais été étudié.

Plan et contributions

Nous allons caractériser la calculabilité du modèle des participants inconnus grâce à l'approche algébrique introduite par Angluin [Ang80]. Cette approche va être notre guide pour cette étude et va nous permettre de caractériser exactement les exécutions problématiques liées au modèle grâce à une relation particulière d'inclusion de graphes et grâce à un lemme d'isolement "à la Angluin".

Il a été prouvé dans [CSS04] que le consensus ne pouvait être résolu dans le modèle des participants inconnus sans information partielle sur le réseau (i.e. impossible à résoudre dans tout réseau). Dans cette partie, nous étendons et généralisons l'étude réalisée dans [CSS04] pour un certain nombre de problèmes donnés : cartographie du réseau, élection et k -élection. Cette partie reprend et complète les résultats présentés lors de la conférence SIROCCO en 2014 [CGN14]. Pour obtenir ces résultats, nous utilisons l'approche algébrique décrite dans l'introduction. Pour chacun de ces problèmes, nous caractérisons, par une condition sur les familles de graphes, les connaissances nécessaires et suffisantes à donner aux processus pour résoudre le problème dans ce modèle. Nos preuves sont constructives et pour chacun des problèmes étudiés, nous donnons soit un contre-exemple montrant que le problème est impossible à résoudre sans la connaissance donnée, soit un algorithme correct vis-à-vis du problème considéré qui utilise la connaissance donnée.

Dans la section 1.1 de ce chapitre, nous introduisons les notations et les différents objets mathématiques utilisés. Dans la section 1.2, nous définissons formellement le modèle à passage de messages ; un problème distribué et les propriétés assurant un résultat correct pour toute exécution produite par un algorithme telles la correction et la terminaison des exécutions. Pour

cela, nous suivrons la modélisation de ces systèmes, à partir de réétiquetage de graphes, proposée dans le livre [Tel00] et reprise par la suite dans [Cha06]. Pour finir, nous définirons formellement le modèle des participants inconnus dans la section 1.3.

Dans le chapitre 2, nous débutons cette étude avec le problème de "cartographie des sommets accessibles". Ce problème est introduit et étudié car il ne nécessite aucune connaissance initiale pour être résolu, i.e., il admet un algorithme universel.

Dans le chapitre 3, nous étudions le problème de cartographie complète du réseau. Dans cette partie, nous introduisons le principal outil caractérisant les impossibilités spécifiques au modèle des participants inconnus : le lemme d'"isolement". Ce lemme, "à la Angluin" exprime la possibilité d'avoir une sous partie des processus exécutant leur algorithme sans avoir connaissance du reste du réseau, comme s'ils étaient seuls dans le réseau. En effet, un ensemble de processus peuvent être suffisamment lents pour permettre aux autres processus (rapides) d'exécuter leur algorithme sans détecter la présence des processus lents, entraînant de ce fait, des résultats localement valides, mais globalement incorrects (erronés vis-à-vis de la spécification du problème). Comme nous allons le voir, les connaissances globales vont nous permettre de retarder les décisions des processus pour éviter que plusieurs processus décident sans avoir découvert l'ensemble des processus.

Dans le chapitre 4, nous étudions le problème de l'élection. Nous montrons que dans ce modèle, le problème de l'élection est moins difficile à résoudre que la cartographie complète du réseau dans le sens où les connaissances nécessaires et suffisantes pour résoudre le problème de l'élection contiennent celles permettant de résoudre le problème de la cartographie du réseau. Ce problème est plus difficile à résoudre que le problème précédent, mais se résout sur de plus vastes familles de graphes.

Dans le chapitre 5, nous étudions le problème de l' k -élection. Ce problème est intéressant à étudier ici car un algorithme d'élection résout la k -élection en présence de connaissances globales plus faibles. Ce problème illustre parfaitement la difficulté du modèle due à l'asynchronicité des calculs et à la terminaison explicite. Grâce aux exécutions asynchrones, nous résolvons la k -élection en utilisant un algorithme d'élection et en donnant des connaissances adaptées aux processus, permettant à au plus k processus de décider "élu" lors d'une exécution.

Chapitre 1

Préliminaires

Dans ce chapitre, nous définissons les objets utilisés dans cette thèse et nous présentons de manière formelle le modèle distribué à passage de messages ainsi que la variante considérée, le modèle à passage de messages avec participants inconnus.

1.1 Définitions et notations

Les définitions utilisées sont standards [Ros00].

1.1.1 Graphe et Graphe dirigé

Définition 1.1. Un graphe dirigé $G = (V, A)$, ou digraphe, est composé d'un ensemble de sommets V et d'un ensemble d'arcs $A \subseteq V \times V$. Un arc entre un sommet u et un sommet v de $V(G)$ est noté uv .

Notons que la relation $A \subseteq V \times V$ n'est pas forcément symétrique.

Un graphe $G = (V, E)$ est composé d'un ensemble V de sommets et d'un ensemble E d'arêtes reliant les sommets deux à deux où la relation $E \subseteq V \times V$ est symétrique.

Remarque 1.2. Étant donné un graphe G (resp. digraphe G), $V(G)$ désignera l'ensemble des sommets de G et $E(G)$ (resp. $A(G)$) désignera l'ensemble des arêtes de G (resp. arcs de G).

Dans ce chapitre, nous considérerons uniquement des graphes dirigés simples, i.e., sans boucle, ni arc multiple entre deux sommets.

Un graphe (resp. digraphe) est dit *simple* s'il ne possède aucune boucle simple, i.e., une arête (resp. arc) reliant un même sommet par ses deux bouts et s'il ne possède aucune arête multiple (resp. arc multiple), i.e., des arêtes (resp. arcs) possédant les mêmes extrémités. Dans ce chapitre, nous manipulerons exclusivement des graphes dirigés simples.

Définition 1.3 (Sous graphe dirigé). Étant donnés deux digraphes H et G , H est un *sous-graphe dirigé* de G si $V(H) \subseteq V(G)$ et $A(H) \subseteq A(G)$.

Définition 1.4 (Sous graphe dirigé induit). Étant donnés deux digraphes H et G , H est un *sous-graphe dirigé induit* de G si H est un sous-graphe dirigé de G et que pour tout $uv \in A(G)$, si $u, v \in V(H)$ alors $uv \in A(H)$.

Définition 1.5 (Sous graphe dirigé restreint). Étant donnés un digraphe G et un sous-ensemble de sommets $U \subseteq V(G)$, nous noterons $G[U]$, le sous graphe induit de G par l'ensemble de sommets U .

La relation suivante de clôture par successeurs est primordiale dans ce modèle et sera utilisée dans toutes les preuves d'impossibilités montrées par la suite.

Définition 1.6 (Sous-graphes clos par successeurs). Soit H et G deux digraphes. Nous noterons $H \subseteq \downarrow G$ si et seulement si H est un sous-graphe induit de G et si, pour tout arc $uv \in A(G)$, $u \in V(H)$ alors $v \in V(H)$.

1.1.2 Graphe dirigé symétrique

Définition 1.7 (Graphe dirigé symétrique). Un *digraphe dirigé symétrique* est un graphe dirigé G doté d'une involution $\sigma : A(G) \rightarrow A(G)$ qui pour tout arc $a \in A(G)$, associe son arc symétrique $\sigma(a)$ tel que $s_G(\sigma(a)) = t_G(a)$.

Remarque 1.8. Étant donné un graphe dirigé symétrique $\overset{\leftrightarrow}{G}$, $\sigma(\sigma(a)) = a$

Pour tout digraphe G , le graphe dirigé symétrique de G , noté $\overset{\leftrightarrow}{G}$, est un digraphe symétrisant les arcs de $A(G)$.

Définition 1.9. Étant donné un digraphe G , le digraphe symétrique $\overset{\leftrightarrow}{G}$ est défini par

- $V(\overset{\leftrightarrow}{G}) = V(G)$,
- $A(\overset{\leftrightarrow}{G}) \subseteq V(G) \times V(G)$ tel que pour tout arc $a \in A(G)$, $(s(a), t(a)), (t(a), s(a)) \in A(\overset{\leftrightarrow}{G})$.

Notons que pour tout arc $a \in A(\overset{\leftrightarrow}{G})$, $\sigma(a)$ est bien défini.

1.1.3 Chemin, longueur et distance dirigée

Définition 1.10 (Chemin dirigé). Un *chemin dirigé* p dans un digraphe G reliant un sommet u à un sommet v est un ensemble de sommets $s_1 \dots s_k \subseteq V(G)$ tel que pour tout $1 \leq i < k$, $s_i s_{i+1} \in A(G)$, $s_1 = u$ et $s_k = v$.

Un chemin reliant un sommet u à un sommet v est noté $p : u \rightarrow v$. Un sommet v est *accessible* à partir d'un sommet u s'il existe un chemin dirigé reliant u à v dans G .

Définition 1.11 (Chemin non dirigé dans un graphe dirigé). Un chemin *non dirigé* p reliant un sommet u à un sommet v est un ensemble de sommets $\{s_1 \dots s_k\} \subseteq V(G)$ tel que $s_1 = u$ et $s_k = v$ et pour tout $1 \leq i < k$, $s_i s_{i+1} \in A(G)$ ou $s_{i+1} s_i \in A(G)$.

Remarque 1.12. Notons que tout chemin p non dirigé dans un graphe dirigé induit un chemin dirigé dans le digraphe symétrique $\overset{\leftrightarrow}{G}$ de G .

Définition 1.13 (Longueur dirigée). La *longueur* d'un chemin dirigé p est le nombre d'arcs qui le compose. La distance dirigée entre deux sommets u et v d'un digraphe G est égale au nombre d'arcs du plus court chemin dirigé reliant u à v dans G lorsqu'il existe un chemin dirigé reliant u à v .

Définition 1.14 (Longueur non dirigée). La distance non dirigée, notée $\bar{d}$, entre deux sommets u et v d'un digraphe G est égale au nombre d'arcs du plus court chemin reliant u à v dans le digraphe symétrique $\overset{\leftrightarrow}{G}$.

Par la suite, un chemin dirigé et une distance dirigée seront appelés simplement chemin et distance. En fonction du contexte, nous préciserons si le chemin et la distance sont non dirigés.

Définition 1.15 (Forte connexité d'un graphe dirigé). Un graphe dirigé G est *fortement connexe* si pour toute paire de sommets u et v de $V(G)$, il existe un chemin dirigé reliant u à v .

Définition 1.16 (Connexité d'un graphe dirigé). Un digraphe G est *connexe* si $\overset{\leftrightarrow}{G}$ est fortement connexe.

Définition 1.17 (Composante fortement connexe). Une *composante fortement connexe* dans un digraphe G est un sous-graphe induit de G fortement connexe et maximal pour l'inclusion.

Remarque 1.18. Les composantes fortement connexes décomposent un digraphe en composantes disjointes.

1.1.4 Sommets accessibles

Nous présentons maintenant l'ensemble $\text{REACH}_G(u)$ qui est l'ensemble des sommets accessibles à partir d'un sommet u dans un digraphe G . Cet ensemble est illustré dans la figure 1.1. Nous introduisons cet ensemble car, comme nous allons le voir par la suite, tout processus v dans un digraphe G est capable de calculer le digraphe $G[\text{REACH}_G(v)]$ sans connaissance initiale particulière dans le modèle des participants inconnus.

Définition 1.19. Étant donné un digraphe G et un sommet $u \in V(G)$,

$$\text{REACH}_G(u) = \{v \in V(G) \mid \text{il existe un chemin dirigé } p : u \rightarrow v \text{ inclus dans } G\}$$

Ci-dessous, nous présentons quelques propriétés intéressantes de cet ensemble dans les digraphes (ou composantes) fortement connexes.

Proposition 1.20. *Les conditions suivantes sont équivalentes*

- (1) G est fortement connexe
- (2) Pour tout $u, v \in V(G)$, $\text{REACH}_G(u) = \text{REACH}_G(v)$
- (3) Pour tout $u, v \in V(G)$, $v \in \text{REACH}_G(u)$
- (4) Pour tout $u \in V(G)$, $\text{REACH}_G(u) = V(G)$

Proposition 1.21. *Pour tout graphe G, H , si $H \subseteq\downarrow G$ alors pour tout $v \in V(H)$, $\text{REACH}_H(v) = \text{REACH}_G(v)$.*

Démonstration. Considérons deux graphes H et G tels que $H \subseteq\downarrow G$. Raisonnons par l'absurde et considérons un sommet $v \in V(G)$ tel que $\text{REACH}_H(v) \neq \text{REACH}_G(v)$.

Comme H est un sous graphe induit de G , les chemins dirigés de H sont des chemins dirigés de G et donc $\text{REACH}_H(v) \subseteq \text{REACH}_G(v)$. Comme $\text{REACH}_H(v) \neq \text{REACH}_G(v)$, il existe un sommet $w \in \text{REACH}_G(v) \setminus \text{REACH}_H(v)$ et par définition de $\text{REACH}_G(v)$, il existe un chemin dirigé $c = vv_1 \dots v_\ell \dots v_k = w \subseteq \text{REACH}_G(v)$. Sans perte de généralité, considérons le sommet v_ℓ de c minimisant ℓ ($\ell > 1$) et n'appartenant pas à $\text{REACH}_H(v)$. Par le choix de v_ℓ , $v_{\ell-1} \in \text{REACH}_H(v)$ et comme $v_{\ell-1}v_\ell \in A(G)$, $v_\ell \in \text{REACH}_H(v)$ par définition. Nous obtenons une contradiction avec le choix de v_ℓ et nous prouvons cette proposition. $\square$

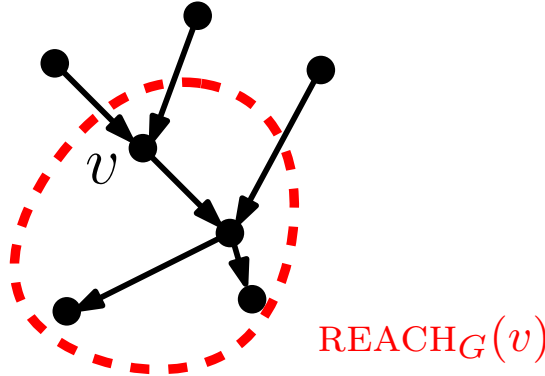


FIGURE 1.1 – Illustration de l'ensemble $\text{REACH}_G(v)$ (cercle en pointillés), l'ensemble des sommets accessibles dans un graphe dirigé G à partir du sommet v .

1.1.5 Graphe dirigé étiqueté

Définition 1.22 (Graphe dirigé étiqueté). Un digraphe étiqueté $\mathbf{G} = (G, \lambda_G)$ est un digraphe $G = (V, A)$ doté d'un étiquetage $\lambda_G : V \cup A \rightarrow \Lambda$ des sommets et des arcs où Λ est l'ensemble des étiquettes possibles.

Nous étendons maintenant les définitions présentées ci-dessus aux digraphes étiquetés.

Définition 1.23 (Sous-graphe étiqueté induit). Étant donnés deux digraphes étiquetés $\mathbf{H} = (H, \lambda_H)$ et $\mathbf{G} = (G, \lambda_G)$, $\mathbf{H}$ est un sous graphe induit de $\mathbf{G}$ ($\mathbf{H} \subseteq \mathbf{G}$) si et seulement si $H \subseteq G$ et si pour tout sommet $v \in V(H)$, $\lambda_H(v) = \lambda_G(v)$. La notion de sous-graphe dirigé étiqueté induit sera notée $\subseteq$.

Nous définissons maintenant une relation de clôture propre aux graphes dirigés, la clôture par successeurs.

Définition 1.24 (Sous-graphe étiqueté clos par successeurs $\subseteq\downarrow$). Étant donnés deux digraphes étiquetés $\mathbf{H} = (H, \lambda_H)$ et $\mathbf{G} = (G, \lambda_G)$, $\mathbf{H} \subseteq\downarrow \mathbf{G}$ si et seulement si $H \subseteq\downarrow G$ et si pour tout sommet $v \in V(H)$, $\lambda_H(v) = \lambda_G(v)$.

Remarque 1.25. Notons que par isomorphisme, pour tout étiquetage λ_G d'un digraphe G , pour tout sous-graphe H de G , λ_G induit un étiquetage λ_H de H .

1.1.6 Réétiquetage de graphes

Définition 1.26 (Réétiquetage de graphes [Cha06]). Une relation de réécriture (ou réétiquetage) de graphes est une relation binaire $\mathcal{R}$ sur les graphes étiquetés close par isomorphisme, i.e., si $\mathbf{G}\mathcal{R}\mathbf{H}$ et $\mathbf{G}'$ est isomorphe à $\mathbf{G}$ alors il existe $\mathbf{H}'$ tel que $\mathbf{G}'\mathcal{R}\mathbf{H}'$ et $\mathbf{H}'$ est isomorphe à $\mathbf{H}$.

Remarque 1.27. Les réétiquetages de graphes considérés peuvent avoir différentes propriétés comme modifier uniquement les étiquettes ou modifier les arcs du réseau.

1.2 Système distribué à passage de messages

Dans cette section, nous définissons formellement le modèle distribué dit "à passage de messages", un algorithme distribué et son exécution. Ces définitions sont données par Tel dans [Tel00]. Nous préciserons plus tard le modèle de cette partie, dit des participants inconnus.

1.2.1 Réseau de communication

Les systèmes distribués à passages de message permettent de modéliser des réseaux de communication entre divers processus. Dans un réseau, les processus seront reliés deux à deux grâce à des canaux de communication. Un réseau se formalise naturellement par un graphe $G = (V, E)$ où l'ensemble des sommets V correspond à l'ensemble des processus et l'ensemble des arêtes E correspond aux canaux de communication. Deux processus reliés par un canal dans le réseau sont des processus "voisins".

1.2.2 Étiquetage des ports

Par la suite, nous considérerons des réseaux où les processus peuvent identifier les canaux qui leur sont adjacents. Pour modéliser cette "connaissance", nous considérerons des réseaux dotés d'un étiquetage des ports.

Définition 1.28 (Étiquetage des ports). Un étiquetage des ports pour un graphe G est une fonction injective d'étiquetage des sommets $\delta : V(G) \rightarrow \mathbb{N}^{\mathbb{N}}$ qui, pour tout sommet $u \in V(G)$ donné, assigne à chaque arête $uv \in E$ une identité unique.

Soit v un sommet d'un graphe G muni d'un étiquetage des ports δ et soit vw une arête de G . Nous noterons $\delta_v(w)$ le numéro de port qui, à partir du noeud v , mène au noeud w , i.e., correspondant à l'extrémité de l'arc vw attaché à v (voir Figure 1.2). De plus, nous noterons $\lambda(uv) = (\delta_u(v), \delta_v(u))$, l'étiquetage d'un arc induit par les numéros de port qui lui sont attachés.

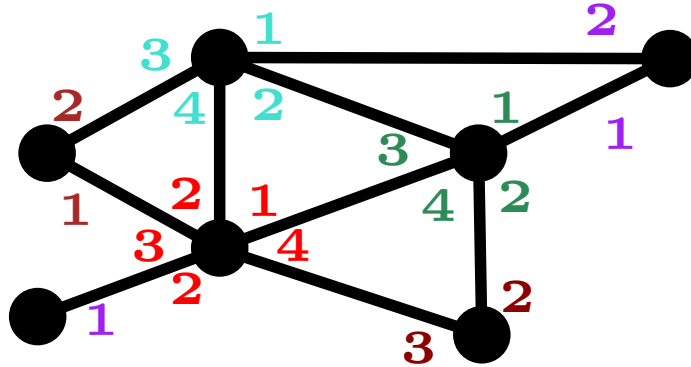


FIGURE 1.2 – Exemple de graphe doté d'un étiquetage des ports

Par définition, notons que pour tout arc $vu, vw \in E(G)$ si $u \neq w$ alors $\delta_v(u) \neq \delta_v(w) \in \mathbb{N}$. Nous noterons par (G, δ) le graphe G doté d'un étiquetage des ports $\delta = \{\delta_v\}_{v \in V(G)}$.

Remarque 1.29. Un tel étiquetage permet aux processus de distinguer les différents canaux adjacents à sa position dans le réseau. Cette information permet par exemple de distinguer le degré des processus pour, par exemple, les distinguer entre eux (autant que possible).

Nous noterons $\mathcal{G}^\delta$, l'ensemble des graphes dotés d'un étiquetage des ports quelconque. Pour être le plus général possible, nos algorithmes devront être corrects pour n'importe quel couple $(G, \delta) \in \mathcal{G}^\delta$. Notons que lors d'une exécution donnée, les numéros de port sont fixes. Par la suite, nous considérerons uniquement des graphes dotés d'un étiquetage des ports $(G, \delta) \in \mathcal{G}^\delta$ et les noterons simplement par G .

Définition 1.30 (Étiquetage induit par un arc et un chemin). Étant donné un graphe $\mathbf{G} \in \mathcal{G}^\delta$, l'étiquetage des ports d'une arête uv , noté $\lambda(uv)$, est composé du couple $(\delta_u(v), \delta_v(u))$. Étant donné un chemin $c = \{u_1 \dots u_k\}$ dans un graphe $\mathbf{G} \in \mathcal{G}^\delta$, l'étiquetage des ports du chemin c , noté $\lambda(c)$, est composé par l'ensemble de numéros de port $\{\delta_{u_i}(u_{i+1}) \mid 1 \leq i < k\}$

1.2.3 Représentation d'un algorithme distribué

Nous présentons maintenant un algorithme distribué de manière formelle et générale. Le modèle des participants inconnus sera introduit en détail dans la section suivante.

Tout processus v dans un réseau G possède un état interne (une mémoire) noté $\lambda_G(v)$ ou λ_v quand le contexte nous le permettra. Par la suite, nous noterons $\mathbf{G}$ pour désigner le couple (G, λ_G) composé d'un réseau G doté d'un étiquetage λ_G encodant les états internes des processus. Un processus v va voir évoluer son état interne $\lambda_G(v)$ différemment en fonction des algorithmes. Un algorithme est représenté par un ensemble de règles de transition d'états fonction de l'état courant du système. Pour représenter la communication entre processus (envoi et réception de message), nous attribuons un tampon pour chaque canal de communication, notée $B(v, w)$. Ce tampon contient les messages circulant sur le canal à destination de w et émis par v . Nous supposons des canaux de communication préservant l'ordre d'émission des messages lors de la réception. Ce type de canaux est appelé *First In First Out* ou *FIFO* plus communément. Précisément, lorsque deux messages m et m' sont émis à destination d'un processus v , si m est émis avant m' alors m sera reçu par v avant m' .

Nous pouvons ainsi définir une relation récursive de transition d'état $\vdash_v$ (état suivant fonction de l'état courant) pour tout processus v : on notera $(\lambda(v), in, m) \vdash (\lambda'(v), out, m')$ une telle transition. Nous observons trois types de transitions spécifiques au modèle à passage de messages : transition interne, réception de message et envoi de message. Chaque transition est définie comme suit :

- Si $\lambda(v) \neq \lambda'(v)$, $in = out = 0$ et $m = m' = \perp$, une transition interne est effectuée, ceci est possible uniquement lorsque l'état de v est $\lambda(v)$. L'état de v devient alors $\lambda'(v)$.
- Si $in \neq 0$ et $out = 0$ et $m' = \perp$, une transition de réception de message est effectuée, possible uniquement lorsque l'état de v est $\lambda(v)$ et lorsqu'il existe un message m dans le tampon $B(w, v)$ tel que $in = \delta_v(w)$. L'état de v devient $\lambda'(v)$, et m est retiré de $B(w, v)$.
- Si $out \neq 0$ et $in = 0$ et $m = \perp$, la transition représente une transition d'envoi de message, possible uniquement lorsque l'état de v est $\lambda(v)$. Une fois cette transition effectuée, un message m' est inséré dans le tampon $B(v, w)$ tel que $\delta_v(w) = out$. L'état de v devient alors $\lambda'(v)$.

Observons que dans notre cas, les canaux étant FIFO, tout tampon $B(v, w)$ peut être représenté par une file. Un algorithme distribué $\mathcal{A}$ à passage de messages peut donc se définir comme une collection d'algorithmes $\mathcal{A}_p$ paramétrés par l'état initial de chaque processus, on note $\mathcal{A} = (\mathcal{A}_{\lambda(v)})_{v \in \mathcal{V}(G)}$ cet ensemble. Une transition de cet algorithme correspond à une transition d'un des processus sur son algorithme local et les files de messages en transit.

1.2.4 Représentation d'une exécution d'un algorithme distribué

Une exécution d'un algorithme distribué peut être vue comme une séquence de réétiquetages de sommets, notée ρ .

Les états étant codés par des étiquettes sur les sommets, tout processus exécutant un pas de calcul (transition interne) exécute un réétiquetage de son état en fonction des "règles de

transition" données par son algorithme et de son état courant. Une exécution ρ d'un algorithme $\mathcal{A}$ sur un graphe étiqueté $\mathbf{G}$ avec étiquetage des ports est représentée par une suite de couples $\left[(\lambda^0, B^0), (\lambda^1, B^1), \dots, (\lambda^i, B^i), \dots \right]_\rho$ où :

- Pour toute étape i , pour tout processus v , $\lambda^i(v)$ représente l'état du processus p à l'étape i de ρ .
- Pour toute étape i , pour tout processus v, w , $B^i(v, w)$ représente l'ensemble des messages en transition entre v et w à l'étape i de ρ .
- Pour toute étape i , il existe un processus p tel que $\lambda^i(v) \neq \lambda^{i+1}(v)$ et pour tout processus $w \neq v$, $\lambda^i(w) = \lambda^{i+1}(w)$.
- λ_v^{i+1} et $B^{i+1}(v, w)$ sont obtenus à partir de $\lambda^i(v)$ et $B^i(v, w)$ en effectuant une transition citée ci-dessus.

Une configuration finale de l'algorithme est atteinte lorsque plus aucune transition ne peut être effectuée. Initialement, tous les tampons B^0 sont vides (i.e. $B^0(v, w) = B^0(w, v) = \emptyset$, aucun message en transit). Chaque transition effectuée par un processus est appelée une "étape" ou un "pas".

Pour maintenir la cohérence des transitions locales effectuées par chaque processus dans la représentation séquentielle globale ρ , il faut que ρ respecte la transitivité des transitions locales et que toute transition d'envoi de message soit située avant sa réception. Pour toute exécution ρ , la suite $\left[(\lambda^0, B^0), (\lambda^1, B^1), \dots, (\lambda^i, B^i), \dots \right]_\rho$ obtenue en fin d'exécution définit un ordonnancement des processus.

Nous dotons chaque processus d'une variable locale de résultat, notée $out_{\mathbf{G}}(v)$, et considérons uniquement l'ensemble de ces variables pour représenter une configuration ou état final du graphe :

Définition 1.31 (variable locale de résultat $out_{\mathbf{G}}(v)$). Nous représentons un étiquetage final $\lambda_{out} \in \mathcal{O}$ par l'ensemble des variables "résultats" $out_{\mathbf{G}}(v)$ des processus. Si le contexte nous le permet, nous la noterons $out(v)$

Remarque 1.32. Plusieurs exécutions d'un même algorithme sur un même graphe étiqueté peuvent mener à plusieurs états finaux suivant l'ordonnancement total des processus au fil de l'exécution.

1.2.5 Spécification d'un problème

Pour savoir si un problème donné a été résolu lors d'une exécution, nous comparons les états initiaux des processus et leurs états finaux obtenus en fin d'exécution. Cette comparaison permet d'assurer que toute décision, prise localement par un processus, est admissible et est globalement correcte vis-à-vis du problème à résoudre (réponse cohérente de l'ensemble des processus). Chaque problème sera donc exprimé par une spécification $\mathcal{S}$ définissant une relation de réétiquetage de graphes.

Définition 1.33 (Spécification). Une spécification $\mathcal{S} \subset: \mathcal{I} \times \mathcal{O}$ est une relation de réétiquetage de graphes entre un ensemble de graphes étiquetés initiaux $\mathcal{I}$ et finaux $\mathcal{O}$.

Étant donnés deux graphes étiquetés $\mathbf{G}^{in} \in \mathcal{I}$ et $\mathbf{G}^{out} \in \mathcal{O}$, nous noterons $\mathbf{G}^{in} \mathcal{S} \mathbf{G}^{out}$ si $\mathbf{G}^{out}$ est légal pour la spécification $\mathcal{S}$ avec le graphe en entrée $\mathbf{G}^{in}$.

1.2.6 Propriétés des exécutions

Pour être correcte, toute exécution d'un algorithme distribué doit se terminer dans un état légal pour la spécification donnée. Les propriétés de terminaison ne sont donc pas équivalentes (ni même comparables) aux propriétés de corrections des exécutions.

Pour montrer qu'une exécution se termine, une exécution devra atteindre un état stable et silencieux, i.e., où les processus ne changent plus leur état interne et où plus aucun message ne circule dans le réseau.

Définition 1.34 (Stabilisation de la décision). Une exécution ρ est *stable pour la décision* s'il existe une étape i_0 de ρ telle que pour tout $v \in V(\mathbf{G})$ et pour tout $i \geq i_0$, $out(v)^i = out(v)^{i_0} \neq \perp$.

Définition 1.35 (Silence). Une exécution ρ est *silencieuse* (ou satisfait la propriété de silence) s'il existe une étape i_0 de ρ telle que pour tout processus $v, v' \in V(\mathbf{G})$ et pour toute étape $i > i_0$, $B^i(v, v') = \emptyset$

Plusieurs types de terminaisons existent. Nous nous focaliserons sur la terminaison explicite présentée ci-dessous.

Définition 1.36 (Terminaison explicite). Une exécution ρ vérifie la terminaison explicite si ρ est stable pour la décision, si ρ vérifie la propriété de silence et si tous les processus écrivent une seule fois la variable *out*.

D'autres formes de terminaison existent. Par exemple, la terminaison implicite permet à un processus de changer sa décision en cours d'exécution.

Définition 1.37 (Terminaison implicite). Une exécution vérifie la terminaison implicite si celle-ci est stable pour la décision et vérifie la propriété de silence.

La terminaison explicite est plus contraignante du point de vue algorithmique que la terminaison implicite : les processus ne peuvent pas revenir sur leur décision. Une décision prise à un moment donné sera définitive et ne pourra plus être changée pour tout le reste de l'exécution. En travaillant avec des communications asynchrones, cette unique prise de décision est souvent difficile à satisfaire sans connaissance globale sur le réseau.

Les exécutions se terminant devront être correctes vis-à-vis de la spécification.

Définition 1.38 (Correction d'une exécution). Étant donné un ensemble de graphes étiquetés $\mathcal{I}$ et un graphe $\mathbf{G}^{in} = (\mathbf{G}, \lambda_{in}) \in \mathcal{I}$, une exécution ρ d'un algorithme $\mathcal{A}$ sur $\mathbf{G}^{in}$ est correcte pour une spécification $\mathcal{S}$ si ρ est stable pour la décision, silencieuse, et si le graphe étiqueté $\mathbf{G}^{out} = (\mathbf{G}, \lambda_{out})$ obtenu suite à l'exécution ρ satisfait $\mathbf{G}^{in} \mathcal{S} \mathbf{G}^{out}$.

1.2.7 Propriétés d'un algorithme

Nous définissons les propriétés définissant un algorithme distribué valide.

Définition 1.39 (Algorithme valide). Un algorithme $\mathcal{A}$ est valide pour une spécification $\mathcal{S}$ sur un graphe étiqueté $\mathbf{G} \in \mathcal{I}$, si pour toute exécution ρ de $\mathcal{A}$ sur $\mathbf{G}$, ρ vérifie les propriétés de terminaison explicite et de correction et si l'étiquetage final $\mathbf{G}^{out}$ obtenu en fin d'exécution est correct pour $\mathcal{S}$.

Si un tel algorithme existe pour une spécification $\mathcal{S}$, alors nous noterons que l'algorithme $\mathcal{A}$ *résout* $\mathcal{S}$ sur $\mathbf{G}$.

Définition 1.40 (Algorithme $\mathcal{F}$ -restreint). Étant donnée une famille de graphes $\mathcal{F}$, un algorithme résolvant une spécification $\mathcal{S}$ est dit $\mathcal{F}$ -restreint, si cet algorithme résout $\mathcal{S}$ pour tout graphe $\mathbf{G} \in \mathcal{F}$. Nous noterons alors que $\mathcal{A}$ résout $\mathcal{S}$ sur $\mathcal{F}$.

Définition 1.41 (Algorithme universel). Un algorithme résolvant une spécification $\mathcal{S}$ est dit universel si cet algorithme résout $\mathcal{S}$ pour tout graphe $\mathbf{G} \in \mathcal{G}^\delta$.

1.2.8 Identité des processus

Nous commençons l'étude du modèle des participants inconnus dans des réseaux avec identités. Pour commencer, les processus sont dotés d'une identité unique. Notons qu'il existe plusieurs assignations d'identités possibles pour un même graphe (par exemple, en permutant les identités des processus). Une assignation d'identités se représente par une fonction $id_{\mathbf{G}} : V(\mathbf{G}) \rightarrow \mathbb{N}$ attribuant une identité unique à tout sommet du graphe $\mathbf{G}$. Nous noterons $(\mathbf{G}, id_{\mathbf{G}})$, le couple formé par un graphe $\mathbf{G}$ doté d'une assignation d'identités représentée par $id_{\mathbf{G}}$. Nous noterons $\mathcal{G}_{id}$ pour représenter l'ensemble des graphes connexes dotés de toutes les assignations d'identités possibles.

Définition 1.42. $\mathcal{G}_{id} = \{(\mathbf{G}, id_{\mathbf{G}}) \mid \mathbf{G} \text{ connexe et } id_{\mathbf{G}}, \text{ assignation d'identités de } \mathbf{G}\}$

Pour être le plus général possible et résoudre un problème indépendamment des identités prises par les processus, nous considérerons uniquement des graphes contenus dans $\mathcal{G}_{id}$. Ainsi pour toute famille de graphes $\mathcal{F} \subseteq \mathcal{G}_{id}$, nous considérerons la famille $\mathcal{F}_{id}$ contenant tous les étiquetages "valides" pour $\mathcal{F}$.

Définition 1.43. Pour toute famille de graphes $\mathcal{F} \subseteq \mathcal{G}_{id}$, $\mathcal{F}_{id} = \{(\mathbf{G}, id_{\mathbf{G}}) \mid \mathbf{G} \in \mathcal{F} \text{ et } (\mathbf{G}, id_{\mathbf{G}}) \in \mathcal{G}_{id}\}$.

Comme nous voulons résoudre un problème $\mathcal{S}$ dans une famille de graphes $\mathcal{F}_{id} \subseteq \mathcal{G}_{id}$ possédant un étiquetage valide, nous noterons simplement par $\mathcal{F}$, les familles contenues dans $\mathcal{F}_{id}$.

1.3 Modèle des participants inconnus

Le modèle des participants inconnus défini par la suite a été introduit dans [CSS04] et est une extension du modèle à passage de messages "classique" présenté ci-dessus.

1.3.1 Présentation du modèle

Dans le modèle des participants inconnus, chaque processus v du réseau possède une identité unique, notée $id(v)$. La particularité de ce modèle est qu'un processus ne connaît initialement qu'une sous-partie de ses voisins et ne peut donc pas, initialement, communiquer avec tous. Pour concrétiser cette "connaissance", chaque processus possède sa propre liste de *contacts*. Cette liste se complétera lorsque le processus recevra un message de la part d'un voisin qui n'est pas dans sa liste. On peut voir ce modèle comme un modèle de *liste de contacts téléphoniques* où un processus Alice ne pourra pas communiquer par téléphone avec Bob tant que Alice ne connaît pas le numéro de Bob. Par contre, une fois que Bob aura appelé Alice, Alice aura accès au numéro de Bob et pourra le contacter par la suite. L'empreinte de communication d'un réseau donné représente la structure physique du réseau, i.e., un lien de communication entre deux sommets correspond à un lien physique entre deux machines.

Remarque 1.44. Dans [CSS04], l’empreinte de communication est un graphe complet permettant une communication directe entre chaque paire de processus du réseau. Dans ce document, nous considérerons que l’empreinte de communication est un graphe arbitraire (symétrique) et qu’un processus peut uniquement communiquer avec ses voisins dans l’empreinte. Si une communication directe entre deux processus qui ne sont pas voisins dans le réseau est requise, alors des protocoles de plus haut niveau peuvent être implémentés comme le protocole IPv4 pour internet, permettant une communication ”point à point” dans n’importe quelle empreinte de communication. De tels protocoles abstraient les distances physiques entre tout processus.

1.3.2 Graphe de communication

Chaque processus v est doté d’une liste $\text{CONTACTS}(v)$ contenant des identités de processus. Initialement, cet ensemble sera noté $\text{CONTACTS}^0(v)$ et contiendra les identités d’un sous ensemble de processus voisins de v dans $\overset{\leftrightarrow}{G}$ avec qui v peut communiquer *initialement*, i.e., ses voisins connus initialement. Nous noterons $\text{CONTACTS}^i(v)$ et $\text{out}^i(v)$, le contenu des registres au début d’une étape i d’une exécution.

Définition 1.45 (Relation ”connaître”). Pour tout processus $v, w \in V(\mathbf{G})$, pour toute étape i , si $\text{id}(w) \in \text{CONTACTS}^i(v)$ alors nous noterons que v connaît w .

Les listes de contacts des processus encodent une relation d’adjacence entre les sommets. Nous noterons $\mathbf{G}$ (ou de façon équivalente $\mathbf{G}^0$), le graphe dirigé de communication initial $\mathbf{G}$ induit par le contenu des listes de contacts des processus.

Définition 1.46 (Graphe de communication initial). Le graphe dirigé $\mathbf{G} = (V, A)$ est le graphe de communication initial composé par un ensemble de processus V correspondant aux différents processus, un ensemble d’arcs $A = \{uv \mid \text{id}(v) \in \text{CONTACTS}^0(u)\}$.

Le réseau possédant des canaux bidirectionnels et étant ”statiques”, i.e., seules les connaissances des processus évoluent et aucun canal de communication n’est créé lors d’une exécution. L’empreinte du réseau (appelé aussi support de communication), notée $\overset{\leftrightarrow}{\mathbf{G}}$, encode l’infrastructure du réseau et détermine les relations d’adjacence entre processus. Notons que ce graphe n’indique pas avec qui un processus peut communiquer initialement, il indique seulement quels processus sont reliés entre eux par un canal de communication.

Définition 1.47 (Empreinte de communication). L’empreinte de communication, notée $\overset{\leftrightarrow}{\mathbf{G}}$ est le graphe dirigé symétrique correspondant à la clôture symétrique du graphe de communication initial $\mathbf{G}$.

Définition 1.48. L’empreinte de communication $\overset{\leftrightarrow}{\mathbf{G}}$ est dotée d’un étiquetage des ports symétrique tel que pour tout arc $a \in A(\overset{\leftrightarrow}{\mathbf{G}})$, $\delta_{s(a)}(t(a)) = \delta_{t(\sigma(a))}(s(\sigma(a)))$ et $\delta_{t(a)}(s(a)) = \delta_{s(\sigma(a))}(t(\sigma(a)))$. De plus, le graphe de communication initial $\mathbf{G}$ étant un sous graphe de $\overset{\leftrightarrow}{\mathbf{G}}$, $\mathbf{G}$ sera doté d’un étiquetage des ports induit par l’étiquetage des ports de $\overset{\leftrightarrow}{\mathbf{G}}$.

Par la suite, nous considérerons toujours des graphes de communication simples, dirigés, connexes et dotés d’un étiquetage des ports.

La particularité intéressante de ce modèle est que le graphe de communication évolue au fil d’une exécution. En effet, si lors d’une étape i , un processus v reçoit un message en provenance d’un processus voisin w ”inconnu” (i.e., $\text{id}(w) \notin \text{CONTACTS}^{i-1}(v)$), alors v met à jour sa liste de contacts avec cette nouvelle identité et peut ainsi communiquer avec w .

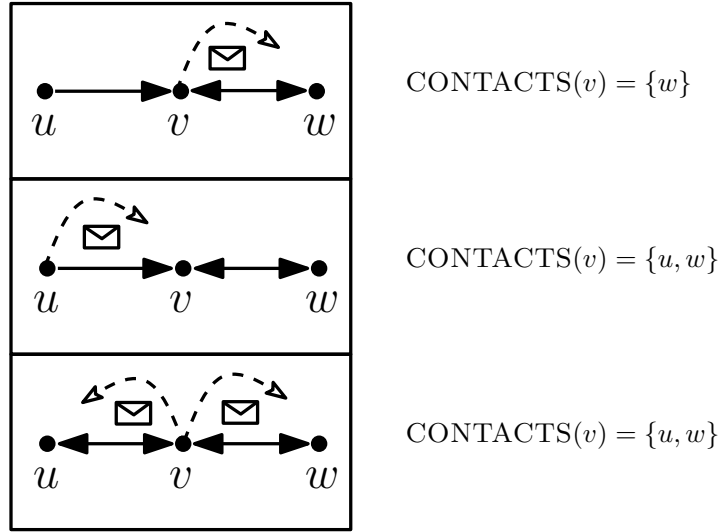


FIGURE 1.3 – Illustration de l'évolution du graphe de communication au cours de l'exécution : Initialement, v et w peuvent communiquer l'un avec l'autre (ils se connaissent). Premièrement, le processus v exécute la procédure I et envoie un message à w , son seul voisin initialement connu. Lors de la réception du message, w exécute la procédure R mais ne met pas à jour sa liste de contacts car il connaît déjà l'identité de v . Lorsque le processus u exécutera la procédure I à son tour (image centrale), il enverra un message à son seul voisin initialement connu v . Lors de la réception du message (image du bas), v met à jour sa liste de contacts et enregistre l'identité de u . Comme v a changé d'état, il envoie un message à tous ses contacts (ici, w et u).

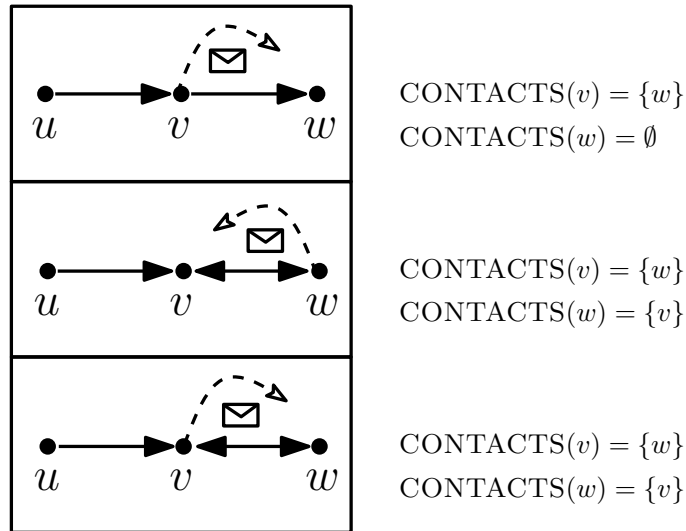


FIGURE 1.4 – Illustration de l'évolution du graphe de communication : En haut, le processus v envoie un message à ses voisins sortants (ici, w). Lors de la réception du message (image du milieu), w met à jour sa liste de contacts avec l'identité de v . Comme w a changé d'état, w envoie un message à tous ses voisins (ici, v). À la réception du message de w , v enverra un message à l'ensemble de ses voisins connus (ici, seulement w) s'il a changé d'état.

Nous définissons ci-dessous $\mathbf{G}^i$, le graphe de communication obtenu lors d'une étape i d'une exécution.

Définition 1.49. Pour toute étape i d'une exécution sur un graphe $\mathbf{G}$, le graphe de communication $\mathbf{G}^i$ est un graphe dirigé avec

- un ensemble de sommets $V(\mathbf{G}^i) = V(\mathbf{G})$,
- un ensemble d'arcs $A(\mathbf{G}^i) = \{vw \in A(\overset{\leftrightarrow}{\mathbf{G}}) \mid id(w) \in \text{CONTACTS}^i(v)\}$.
- pour tout arc $uv \in A(\mathbf{G}^i)$, $\lambda_{\mathbf{G}^i}(uv) = \lambda_{\overset{\leftrightarrow}{\mathbf{G}}}(uv)$.

Dans les figures 1.3 et 1.4, nous illustrons quelques cas de figure possibles pouvant arriver lorsqu'un processus reçoit un message avec les évolutions possibles du graphe de communication.

Nous présentons une propriété du graphe de communication $\mathbf{G}^i$ invariante pour toute étape d'une exécution. Pour prouver la proposition suivante, il suffit de remarquer que lorsqu'un processus reçoit un message, celui-ci peut soit découvrir un nouveau voisin et mettre à jour CONTACTS^i , soit ne rien découvrir et ne pas mettre à jour CONTACTS^i . Ainsi, le contenu de la liste de contacts d'un processus ne peut que croître au cours d'une exécution. De plus, comme le graphe de communication évolue en fonction du contenu des listes de contacts des processus, nous obtenons la proposition suivante.

Proposition 1.50. Pour toute étape i d'une exécution sur un graphe $\mathbf{G}$, pour tout processus v ,

$$\text{CONTACTS}^i(v) \subseteq \text{CONTACTS}^{i+1}(v) \text{ et } \mathbf{G}^0 \subseteq \mathbf{G}^i \subseteq \mathbf{G}^{i+1} \subseteq \overset{\leftrightarrow}{\mathbf{G}}$$

.

Nous présentons maintenant quelques remarques à propos du graphe de communication.

Remarque 1.51. L'ensemble des voisins potentiels d'un processus est fixé par l'empreinte de communication $\overset{\leftrightarrow}{\mathbf{G}}$ et est invariant tout au long de l'exécution.

Remarque 1.52. Si un sommet ne possède initialement aucun arc sortant, ce sommet ne pourra pas communiquer tant que celui-ci n'aura pas reçu de message de la part d'un de ses voisins "inconnus".

Remarque 1.53. Nous considérerons des graphes de communication initiaux connexes. En effet, même si le modèle permet aux processus de découvrir leurs voisins, si deux processus u et v relié par un canal de communication dans le réseau ne se connaissent pas initialement (i.e. $id(u) \notin \text{CONTACTS}^0(v)$ et $id(v) \notin \text{CONTACTS}^0(u)$) alors ils ne pourront jamais communiquer ensemble. Formellement, il n'existe aucun arc reliant les deux processus dans l'empreinte de communication $\overset{\leftrightarrow}{\mathbf{G}}$ du graphe de communication initial $\mathbf{G}$.

1.3.3 Étiquetage : Codage de l'état des processus

L'état interne d'un processus dans ce modèle est encodé par l'étiquette suivante.

Définition 1.54. Pour tout processus v dans un réseau $\overset{\leftrightarrow}{\mathbf{G}}$, soit

$$\lambda_{\mathbf{G}}(v) = \langle \text{CONTACTS}(v), in(v), out(v), MEM(v) \rangle$$

l'étiquette attachée au processus v où

- $\text{CONTACTS}(v) \subseteq \mathcal{P}(\mathcal{I})$ est la liste d'identités de processus présentée dans la sous-section 1.3.2.
- $\text{in}(v)$ est un ensemble de paramètres initiaux propres à chaque processus. Il contient notamment l'identité unique du processus v , notée $\text{id}(v)$.
- $\text{out}(v)$ est la variable dans laquelle le processus écrira sa décision.
- $\text{MEM}(v)$ est le reste de la mémoire allouée au processus.

Remarque 1.55. Comme nous nous intéressons à la calculabilité distribuée d'un problème et non à optimiser la complexité d'un algorithme comme le nombre de messages envoyés ou l'espace mémoire utilisé par les processus, nous supposons que les processus possèdent une mémoire suffisamment grande pour leur besoin et qu'ils accèdent à leur mémoire en temps constant.

Chapitre 2

Cartographie des sommets accessibles

Dans ce chapitre, nous étudions le problème de cartographie des sommets accessibles dans le modèle des participants inconnus. Ces résultats ont été publiés lors de la conférence SIROCCO en 2014 [CGN14].

2.1 Description du problème $\mathcal{REACH}$

Étant donné un graphe de communication initial $\mathbf{G}$ et un processus $v \in V(\mathbf{G})$, le graphe $\mathbf{G}[\text{REACH}(v)]$ désignera le sous-graphe du graphe de communication initial $\mathbf{G}$ induit par l'ensemble de sommets $\text{REACH}_{\mathbf{G}}(v)$. Nous nous intéressons au problème suivant,

Définition 2.1 (Spécification de $\mathcal{REACH}$). Pour tout graphe $\mathbf{G}$, pour tout graphe $\mathbf{G}^{out}$, nous noterons $\mathbf{GREACH}_{\mathbf{G}^{out}}$ si pour tout processus $v \in V(\mathbf{G})$, $out_{\mathbf{G}^{out}}(v)$ est un graphe isomorphe au graphe $\mathbf{G}[\text{REACH}(v)]$.

Il s'agit donc de calculer localement, pour tout processus $v \in V(\mathbf{G})$, la carte induite par l'ensemble $\mathbf{G}[\text{REACH}(v)]$, les sommets initialement accessibles à partir de v dans le graphe de communication initial $\mathbf{G}$.

Notons que comme, par définition, le graphe de communication $\mathbf{G}$ évolue au fil des étapes d'une exécution, il est nécessaire que tout processus v stocke et propage ses contacts initiaux afin de pouvoir reconstruire le sous-graphe $\text{REACH}_{\mathbf{G}}(v) (\simeq \text{REACH}_{\mathbf{G}^0}(v))$.

Nous nous intéressons à ce problème car il servira de brique de base pour résoudre des problèmes plus complexes dans la suite de cette étude. De plus, ce problème ne nécessitant aucune connaissance initiale pour être résolu, il admet donc un algorithme correct pour tout graphe, i.e., il admet un algorithme universel.

2.2 Description de l'algorithme d'inondation

Nous introduisons maintenant les variables et structures utilisées dans l'algorithme. Chaque processus v possède une identité unique $id(v)$ dans le réseau et peut y accéder.

2.2.1 Boîte aux lettres

Tout processus v possède une boîte aux lettres $M(v)$ lui permettant de stocker son état ainsi que l'état des processus du réseau. Chaque élément de $M(v)$ est un couple (id, C) où $id \in \mathbb{N}$ est l'identité d'un processus, C est la liste de contacts initiale CONTACTS^0 du processus. Initialement, la boîte aux lettres $M^0(v)$ d'un processus v est égale à l'ensemble $\{(id(v), \text{CONTACTS}^0(v))\}$.

Quand un processus enverra un message, il enverra l'intégralité de sa boîte aux lettres ainsi que son identité.

Définition 2.2. Un message, envoyé par un processus v sera de la forme $\langle id(v), M(v) \rangle$ avec

- $id(v)$, l'identité du processus v ,
- $M(v)$, la boîte aux lettres du processus v lors de l'envoi.

Pour cela, nous définissons une primitive **SendAll** utilisée par les processus pour envoyer des messages à l'ensemble de leurs voisins connus.

Définition 2.3 (SendAll). Étant donné un graphe $\mathbf{G}$, un processus $v \in V(\mathbf{G})$ et une étape i d'une exécution d'un algorithme sur $\mathbf{G}$, **SendAll** est une primitive qui, exécutée par un processus v , enverra un message m via l'ensemble de ports $\{id \mid id \in \text{CONTACTS}^i\}$.

2.2.2 Déroulement de l'algorithme

L'algorithme est un algorithme *d'inondation* ("flooding"), i.e., à chaque changement d'état, un processus enverra son nouvel état à tous ses voisins. Nous étudions en premier une première version de cet algorithme où les processus collectent les contacts initiaux des processus du réseau sans décider. Puis, nous présenterons une deuxième version qui contiendra une condition de décision permettant aux processus de résoudre le problème $\mathcal{REACH}$.

L'algorithme 1 est composé de deux procédures : **I** et **R**. Initialement, chaque processus exécute la procédure **I** et envoie son état initial à tous ses voisins initiaux. La procédure **R** est exécutée par un processus v lorsque celui-ci reçoit un message $\langle id(u), M(u) \rangle$ de la part d'un processus voisin u . Le processus mettra alors à jour sa boîte aux lettres $M(v)$ s'il existe des entrées $(id, \text{CONTACTS}) \in M(u)$ non présentes ou différentes dans $M(v)$. De plus, si $id(u)$ n'appartient pas à la liste $\text{CONTACTS}(v)$, v ajoutera $id(u)$ dans $\text{CONTACTS}(v)$. Pour finir, si v a modifié sa boîte aux lettres ou sa liste de contacts $\text{CONTACTS}(v)$ durant cette étape, alors v enverra un message $\langle id(v), M(v) \rangle$ à tous ses voisins.

Remarque 2.4. Notons que la procédure de mise à jour de CONTACTS pourrait être automatisée par une primitive du type **getMessageAndUpdate** mais nous préférons l'explicitier dans les algorithmes pour plus de compréhension.

À partir de l'algorithme, quand un processus découvre un nouveau voisin, il met uniquement à jour sa liste locale de contacts et non sa liste de contacts présente dans sa boîte aux lettres. Nous obtenons la proposition suivante.

Proposition 2.5. *Pour tout graphe $\mathbf{G}$, toute étape i d'une exécution et tout processus $v \in V(\mathbf{G})$, $(id(v), \text{CONTACTS}^0(v)) \in M^i(v)$.*

Il est évident que l'ensemble des boîtes aux lettres initiales des processus forme un tableau d'adjacences permettant de calculer le réseau de communication initial. En s'échangeant des informations au fil d'une exécution, les processus élargiront leurs connaissances et auront un aperçu plus global du réseau. Nous définissons donc une fonction $\mathbb{T}$ permettant d'extraire le graphe dirigé induit par une boîte aux lettres.

Algorithme 1 : Algorithme d'inondation pour les participants inconnus

```

Procédure I() : /* Procédure initiale */
1  |   $M \leftarrow \{(id, \text{CONTACTS}^0)\};$ 
2  |  SendAll<id, M>;

Procédure R(id', M') : /* Réception d'un message */
3  |  updated = faux;
4  |  si  $M' \setminus M \neq \emptyset$  alors
5  |  |   $M \leftarrow M \cup M';$ 
6  |  |  updated = vrai;
7  |  si  $id' \notin \text{CONTACTS}$  alors
8  |  |   $\text{CONTACTS} \leftarrow \text{CONTACTS} \cup \{id'\};$ 
9  |  |  updated = vrai;
10 |  si updated = vrai alors
11 |  |  SendAll<id, M>;

```

Définition 2.6. Étant donnée une boîte aux lettres M , le graphe $\mathbb{T}(M)$ est défini comme suit :

- $V(\mathbb{T}(M)) = \{id \mid (id, C) \in M \vee ((id', C') \in M \wedge id \in C')\}$
- $A(\mathbb{T}(M)) = \{idid' \mid (id, C) \in M \wedge id' \in C\}$

La propriété suivante s'obtient directement à partir des définitions.

Proposition 2.7. Étant donnés une boîte aux lettres M et un graphe $\mathbf{G}$, $\mathbb{T}(M) \subseteq \mathbf{G}$.

2.3 Correction de l'algorithme

Nous prouvons en premier que l'algorithme 1 converge vers un état stable. Pour cela, observons que par définition, l'union des contacts initiaux des processus du réseau induit un graphe isomorphe au graphe de communication initial.

Lemme 2.8. Pour tout graphe $\mathbf{G}$,

$$\mathbb{T}\left(\bigcup_{v \in V(\mathbf{G})} (id(v), C)\right) \simeq \mathbf{G}$$

Le lemme suivant montre que le contenu des boîtes aux lettres ne peut prendre qu'un nombre fini de valeurs. Il est obtenu directement à partir de l'algorithme et grâce à l'unicité des identités.

Lemme 2.9. Pour toute étape i d'une exécution de l'algorithme 1, si un processus v exécute une transition lors de cette étape i alors $M^i(v) \subseteq M^{i+1}(v)$ et $\text{CONTACTS}^i(v) \subseteq \text{CONTACTS}^{i+1}(v)$.

Nous montrons maintenant que la correction de l'inondation, i.e., tout processus recevra les contacts initiaux de tous les processus du réseau.

Lemme 2.10. Pour tout processus $v, w \in V(\mathbf{G})$, il existe une étape i telle que $(id, C) \in M^i(w)$.

Démonstration. Nous montrons ce lemme par récurrence sur la distance non dirigée entre processus. L'hypothèse de récurrence $HR(k)$ est la suivante. Pour tout processus $v, w \in V(G)$ tel que $\bar{d}(v, w) \leq k$, pour toute étape i , pour tout $(id, C) \in M^i(v)$, il existe une étape j telle que $(id, C) \in M^j(w)$.

Prenons un graphe $\mathbf{G}$ et deux processus $v, w \in V(\mathbf{G})$ tels que $\bar{d}(v, w) = 1$.

Si $id(w) \in \text{CONTACTS}^0(v)$ alors, pour toute étape i et pour tout $(id, C) \in M^i(v)$, soit $(id, C) \in M^0(v)$, soit v a reçu un message $m = \langle id', M' \rangle$ lors d'une étape $h \leq i$ avec $(id, C) \in M'$.

Si $(id, C) \in M^0(v)$, alors quand v exécute la procédure I , w recevra un message en provenance de v (canaux fiables) contenant $M^0(v)$ lors d'une étape j et mettra à jour sa boîte aux lettres. Par conséquent, $(id, C) \in M^j(w)$.

Sinon, $(id, C) \notin M^0(v)$ et donc v a reçu un message $m = \langle id', M' \rangle$ lors d'une étape $h \leq i$ contenant (id, C) . Comme $(id, C) \in M' \setminus M^{h-1}(v)$, en exécutant la procédure R , $M^{h-1}(v) \neq M^h(v)$ et donc, v envoie un message $\langle id, M^h(v) \rangle$ à tous ses voisins (dont w car $id(w) \in \text{CONTACTS}^0(v)$). Ainsi, w recevra un message en provenance de v (canaux fiables) contenant $M^h(v)$ lors d'une étape j et donc, $(id, C) \in M^j(w)$.

Si $id(w) \notin \text{CONTACTS}^0(v)$, alors $id(v) \in \text{CONTACTS}^0(w)$ ($\bar{d}(v, w) = 1$) et comme w exécute la procédure I , il existe une étape h où v reçoit un message $m = \langle id', M' \rangle$ en provenance de w . Comme $id(w) \notin \text{CONTACTS}^0(v)$, v mettra à jour $\text{CONTACTS}^{h-1}(v)$ avec $id(w)$ et, à la fin de l'étape h , $id(w) \in \text{CONTACTS}^h(v)$. En appliquant le cas précédent à partir de cette étape, pour toute étape i , pour tout $(id, C) \in M^i(v)$, il existe une étape j telle que $(id, C) \in M^j(w)$. Nous prouvons ce premier cas.

Supposons notre hypothèse de récurrence $HR(k)$ vérifiée pour tout couple de sommets à distance (non dirigée) inférieure ou égale à k . Montrons ce lemme pour tout couple de sommets à distance $k+1$. Soit $v, w \in V(\mathbf{G})$ deux processus à distance $k+1$, soit i une étape de l'exécution. Comme $\overset{\leftrightarrow}{\mathbf{G}}$ est fortement connexe (i.e. $\mathbf{G}$ est connexe), il existe un processus u tel que $\bar{d}(v, u) = k$ et $\bar{d}(u, w) = 1$. Par hypothèse de récurrence, pour tout $(id, C) \in M^i(v)$, il existe une étape j telle que $(id, C) \in M^j(u)$. Par $HR(1)$, pour tout $(id, C) \in M^j(u)$, il existe une étape h telle que $(id, C) \in M^h(w)$. Par conséquent, pour tout $(id, C) \in M^i(v)$, il existe une étape h telle que $(id, C) \in M^h(w)$, ce lemme est prouvé. $\square$

Par les lemmes 2.9 et 2.10, nous obtenons le corollaire suivant,

Corollaire 2.11. *Pour toute exécution de l'algorithme 1 sur un graphe $\mathbf{G}$, il existe une étape i telle que tout processus possédera une boîte aux lettres $M = \{\bigcup_{v \in V(G)} (id(v), \text{CONTACTS}^0(v))\}$ et $\mathbb{T}(M) \simeq \mathbf{G}$.*

Remarque 2.12. L'algorithme 1 peut être utilisé pour calculer, avec terminaison implicite, la carte intégrale du réseau $\mathbf{G}$. Dans une telle exécution, les processus ne seront pas capables de détecter la fin de leur exécution mais posséderont tous une même boîte aux lettres isomorphe au graphe de communication initial. De plus, les processus n'enverront plus de message après une telle étape. Notons que les processus sont aussi capables de calculer la carte du graphe $\overset{\leftrightarrow}{\mathbf{G}}$ s'ils mettent à jour leur boîte aux lettres avec leur liste CONTACTS locale mise à jour.

Ainsi, l'inondation de données (avec terminaison implicite) dans le modèle des participants inconnus est réalisable, peu importe le réseau.

2.4 Algorithme pour $\mathcal{REACH}$

Nous montrons maintenant que pour résoudre le problème $\mathcal{REACH}$, il nous suffit d'ajouter une condition de décision à l'algorithme 1. Pour cela, nous introduisons deux fonctions, notées VU et $REÇU$, utilisées par l'algorithme pour extraire des informations d'une boîte aux lettres et permettre aux processus de décider.

La fonction VU extrait l'ensemble des identités de processus présentes dans une boîte aux lettres donnée.

Définition 2.13. Étant donnée une boîte aux lettres M , $VU(M) = \{id \mid (id, C) \in M \vee ((id', C') \in M \wedge id \in C')\}$

La fonction $REÇU$ extrait toutes les identités d'une boîte aux lettres telles que la boîte aux lettres contient une liste de contacts pour cette identité. En d'autres termes, une identité id appartient à $REÇU(M)$ si le processus v a reçu un message $\langle id', M' \rangle$ avec $(id, C) \in M'$.

Définition 2.14. Étant donnée une boîte aux lettres M , $REÇU(M) = \{id \mid (id, C) \in M\}$

Les fonctions VU et $REÇU$ vont permettre aux processus de détecter un état stable dans l'exécution. Ainsi, pour décider, un processus v devra attendre d'obtenir une boîte aux lettres satisfaisant la condition $VU(M(v)) = REÇU(M(v))$. La décision sera effective lorsque le processus v écrira le graphe restreint $\mathbb{T}(M)[REACH_{\mathbb{T}(M)}(id)]$ dans sa variable $out(v)$.

Notons que la condition $VU(M(v)) = REÇU(M(v))$ ne permet pas aux processus de savoir que leur boîte aux lettres contient toutes les identités du réseau. Comme nous allons le voir, cette condition permet à tout processus de savoir si sa boîte aux lettres *contient au moins* toutes les identités des processus présents dans l'ensemble $REACH_{\mathbf{G}}(v)$.

Algorithme 2 : Algorithme pour $\mathcal{REACH}$

```

Procédure I():                                     /* Procédure initiale */
1  |   $M \leftarrow \{(id, CONTACTS^0)\};$ 
2  |   $out \leftarrow \perp;$ 
3  |   $SendAll \langle id, M \rangle;$ 

Procédure R(id', M'):                             /* Réception d'un message */
4  |   $updated = \text{faux};$ 
5  |  si  $M' \setminus M \neq \emptyset$  alors
6  |  |   $M \leftarrow M \cup M';$ 
7  |  |   $updated = \text{vrai};$ 
8  |  si  $id' \notin CONTACTS$  alors
9  |  |   $CONTACTS \leftarrow CONTACTS \cup \{id'\};$ 
10 |  |   $updated = \text{vrai};$ 
11 |  si  $VU(M) = REÇU(M) \wedge out = \perp$  alors
12 |  |   $out \leftarrow \mathbb{T}(M)[REACH_{\mathbb{T}(M)}(id)];$ 
13 |  si  $updated = \text{vrai}$  alors
14 |  |   $SendAll \langle id, M \rangle;$ 

```

2.4.1 Correction de l'algorithme 2

Nous montrons maintenant que tout processus v satisfaisant la condition de décision pour la première fois de l'exécution possédera dans sa boîte aux lettres toutes les informations nécessaires à la reconstruction du sous-graphe induit par l'ensemble $\text{REACH}_{\mathbf{G}}(v)$.

Lemme 2.15. *Pour tout processus v , si lors d'une étape i , $\text{VU}(M^i(v)) = \text{RECU}(M^i(v))$ alors pour tout processus $w \in \text{REACH}_{\mathbf{G}}(v)$, $\text{id}(w) \in \text{RECU}(M^i(v))$.*

Démonstration. Nous prouvons ce lemme par contradiction. Soit v , un processus et i une étape de l'exécution telle que $\text{VU}(M^i(v)) = \text{RECU}(M^i(v))$. Soit $w \in \text{REACH}_{\mathbf{G}}(v)$, le processus le plus proche de v ne satisfaisant pas le lemme, i.e., $\text{id}(w) \notin \text{RECU}(M^i(v))$ et $\bar{d}(v, w)$ est minimal parmi tous les autres processus appartenant à $\text{REACH}_{\mathbf{G}}(v)$ dans cette situation. Notons que la distance dirigée entre w et v est bien définie par définition de l'ensemble $\text{REACH}_{\mathbf{G}}(v)$.

Soit $w' \in \text{REACH}_{\mathbf{G}}(v)$ un processus tel que w' est un prédécesseur de w (i.e., $\text{id}_{w'} \in \text{CONTACTS}^0(w')$) et w' est situé sur un plus court chemin dirigé entre v et w . Par notre choix de w , $\text{id}(w') \in \text{RECU}(M^i(v))$ et donc $(\text{id}_{w'}, \text{CONTACTS}^0(w')) \in M^i(v)$. Comme $\text{id}(w) \in \text{CONTACTS}^0(w')$, nous obtenons que $\text{id}(w) \in \text{VU}(M^i(v))$. Comme par hypothèse, $\text{VU}(M^i(v)) = \text{RECU}(M^i(v))$, $\text{id}(w) \in \text{RECU}(M^i(v))$ et par conséquent, nous obtenons une contradiction et prouvons ce lemme. $\square$

Le corollaire suivant est obtenu par construction et par définition de $\text{REACH}_{\mathbf{G}}(v)$.

Corollaire 2.16. *Pour tout processus v et toute étape i , si $\text{VU}(M^i(v)) = \text{RECU}(M^i(v))$ alors $\mathbb{T}(M^i(v)) \subseteq \downarrow \mathbf{G}$ et $\text{REACH}_{\mathbb{T}(M^i(v))}(\text{id}(v)) \simeq \text{REACH}_{\mathbf{G}}(v)$.*

Démonstration. Soit $v \in V(G)$, un processus et i une étape de l'exécution telle que $\text{VU}(M^i(v)) = \text{RECU}(M^i(v))$. Nous montrons que pour tout arc $uw \in A(\mathbf{G})$, si $u \in \text{REACH}_{\mathbf{G}}(v)$, alors il existe $\text{id}(u), \text{id}(w) \in V(\mathbb{T}(M^i(v)))$ et $\text{id}(u)\text{id}(w) \in A(\mathbb{T}(M^i(v)))$.

Par définition, pour tout arc $uw \in A(\mathbf{G})$, $\text{id}(w) \in \text{CONTACTS}^0(u)$. Par le lemme 2.15, si $u \in \text{REACH}_{\mathbf{G}}(v)$ alors $\text{id}(u) \in \text{RECU}(M^i(v))$ et donc $(\text{id}(u), \text{CONTACTS}^0(u)) \in M^i(v)$ par définition. Par conséquent, $\text{id}(w) \in V(\mathbb{T}(M^i(v)))$, $\text{id}(u)\text{id}(w) \in A(\mathbb{T}(M^i(v)))$ et donc $\mathbb{T}(M^i(v)) \subseteq \downarrow \mathbf{G}$.

Pour montrer le dernier point, il suffit d'appliquer la proposition 1.21 entre $\mathbb{T}(M^i(v))$ et $\mathbf{G}$ pour obtenir un isomorphisme entre l'ensemble $\text{REACH}_{\mathbb{T}(M^i(v))}(\text{id}(v))$ et l'ensemble $\text{REACH}_{\mathbf{G}}(v)$. $\square$

À partir des lemmes 2.10 et 2.15, pour tout processus v , il existe une étape i telle que v vérifie (pour la première fois) la condition de décision. Par le corollaire 2.16, il décidera donc un sous graphe isomorphe à $G[\text{REACH}_{\mathbf{G}}(v)]$. Comme v ne reviendra jamais sur sa première décision (d'après la condition de décision), nous obtenons que l'algorithme 2 résout $\mathcal{REACH}$ pour tout graphe $\mathbf{G}$.

Théorème 2.1. *L'algorithme 2 est un algorithme universel pour le problème $\mathcal{REACH}$.*

Chapitre 3

Cartographie complète du réseau

Dans ce chapitre, nous étudions le problème de cartographie complète du réseau dans le modèle des participants inconnus. Ces résultats ont été publiés lors de la conférence SIROCCO en 2014 [CGN14].

Dans la section 3.1, nous présentons le problème $\mathcal{CARTO}$. Dans la section 3.2, nous introduisons le *lemme d'isolement*, principal outil permettant d'obtenir des résultats d'impossibilité dans ce modèle. Ce lemme à la "Angluin" capture le fait que, à cause des communications asynchrones et des participants inconnus, une sous partie stricte des processus peuvent exécuter leur algorithme et terminer leur exécution de façon isolée, i.e., sans aucune interaction avec le reste du réseau.

Dans la section 3.3, nous prouvons que la condition $\mathfrak{C}_{\mathcal{CARTO}}$ est nécessaire en montrant que s'il existe un algorithme $\mathcal{F}$ -restreint résolvant $\mathcal{CARTO}$ alors la famille $\mathcal{F}$ satisfait $\mathfrak{C}_{\mathcal{CARTO}}$. Dans la section 3.4, nous prouvons que la condition $\mathfrak{C}_{\mathcal{CARTO}}$ est suffisante. Pour cela, nous proposons un algorithme $\mathcal{F}$ -restreint résolvant $\mathcal{CARTO}$ pour toute famille $\mathcal{F}$ satisfaisant $\mathfrak{C}_{\mathcal{CARTO}}$ et nous prouvons sa correction. Dans la section 3.5, nous donnons des exemples de familles de graphes vérifiant et ne vérifiant pas $\mathfrak{C}_{\mathcal{CARTO}}$.

3.1 Description du problème $\mathcal{CARTO}$

Le problème de cartographie complète du réseau est défini par la spécification ci-dessous.

Définition 3.1 (Spécification $\mathcal{CARTO}$). Pour tout graphe $\mathbf{G}$, pour tout graphe $\mathbf{G}^{out}$, nous noterons $\mathbf{G} \mathcal{CARTO} \mathbf{G}^{out}$ si pour tout processus $v \in V(\mathbf{G})$, $out_{\mathbf{G}^{out}}(v)$ est un graphe isomorphe au graphe de communication initial $\mathbf{G}$.

Comme remarqué dans le chapitre précédent, il est possible de construire une carte du réseau de communication initial avec terminaison implicite. Le problème de cartographie complète du réseau est donc intéressant car, avec terminaison explicite, il n'admet pas d'algorithme universel, i.e., il n'existe pas d'algorithme résolvant ce problème pour tout graphe donné. Comme nous allons le montrer, il peut néanmoins être résolu avec terminaison explicite en dotant les processus de connaissances additionnelles.

Pour cela, nous allons nous intéresser à des algorithmes plus spécifiques, dits $\mathcal{F}$ -restreints. Ces algorithmes (présentés dans la définition 1.40) sont spécialisés pour résoudre le problème dans une famille de graphes donnée $\mathcal{F}$.

Dans ce chapitre, nous caractérisons les familles de graphes $\mathcal{F}$ pour lesquelles il existe un algorithme $\mathcal{F}$ -restreint résolvant $\mathcal{CARTO}$ au travers d'une condition structurelle $\mathfrak{C}_{\mathcal{CARTO}}$. Nous

montrons que cette condition est satisfaite par une famille de graphes $\mathcal{F}$ si et seulement s'il existe un algorithme $\mathcal{F}$ – *restreint* pour $\mathcal{CARTO}$.

3.2 Lemme d'isolement

Pour montrer l'inexistence d'un algorithme universel pour ce problème, nous introduisons un lemme clé dans l'étude de ce modèle, dit lemme d'isolement. Ce lemme est un lemme à la "Angluin" et sera réutilisé dans tous les cas d'étude qui suivent pour ce modèle. Il précise que dans un graphe $\mathbf{G}$, une exécution isolée peut apparaître dans une sous-partie $\mathbf{H} \subseteq \downarrow \mathbf{G}$ si tous les messages à destination de $\mathbf{H}$ et en provenance de $\mathbf{G} \setminus \mathbf{H}$ sont bloqués suffisamment longtemps (communication asynchrone malgré des canaux fiables). Un tel sous-ensemble de processus de $\mathbf{G}$ sera dit *isolé du reste du réseau*. En effet, chaque processus de $\mathbf{H}$ exécutera son algorithme comme s'il était dans $\mathbf{H}$ et non dans $\mathbf{G}$. Les figures 3.1 et 3.2 illustrent ces exécutions.

Par la suite, un graphe étiqueté $\mathbf{G}$, et ses tampons de messages en transit $B_{\mathbf{G}}$ seront notés $\underline{\mathbf{G}} = (G, \lambda_G, B_G)$. Nous présentons maintenant une définition de la clôture par successeurs étendue aux graphes étiquetés dotés de tampons de messages en transit.

Définition 3.2. Soit $\underline{\mathbf{G}} = (G, \lambda_G, B_G)$, un graphe étiqueté $\mathbf{G}$, et ses tampons de messages en transit $B_{\mathbf{G}}$. Pour tout graphe $\underline{\mathbf{G}} = (G, \lambda_G, B_G)$ et $\underline{\mathbf{H}} = (H, \lambda_H, B_H)$, nous noterons $\underline{\mathbf{H}} \subseteq \downarrow \underline{\mathbf{G}}$ si les propriétés suivantes sont vérifiées :

- $\mathbf{H} \subseteq \downarrow \mathbf{G}$
- $\forall uv \in A(\mathbf{H}), B_{\mathbf{H}}(u, v) = B_{\mathbf{G}}(u, v)$

Le lemme suivant est immédiat à partir de la définition des états initiaux des processus.

Lemme 3.3 (Initialisation). *Pour tout graphe $\mathbf{G}$ et $\mathbf{H}$ si $\mathbf{H} \subseteq \downarrow \mathbf{G}$ alors pour tout étiquetage initial $\lambda_{\mathbf{G}}$ de $\mathbf{G}$, il existe un étiquetage $\lambda_{\mathbf{H}}$ tel que $(\mathbf{H}, \lambda_{\mathbf{H}}, B_{\mathbf{H}}) \subseteq \downarrow (\mathbf{G}, \lambda_{\mathbf{G}}, B_{\mathbf{G}})$.*

Nous prouvons maintenant que toute étape d'une exécution dans un graphe $\mathbf{H}$ peut être exécutée dans un graphe $\mathbf{G}$ si $\mathbf{H} \subseteq \downarrow \mathbf{G}$.

Lemme 3.4 (Une étape). *Pour tout graphe $\underline{\mathbf{H}} = (H, \lambda_H, B_H)$ et $\underline{\mathbf{G}} = (G, \lambda_G, B_G)$, si $\underline{\mathbf{H}} \subseteq \downarrow \underline{\mathbf{G}}$ alors toute transition exécutée dans $\underline{\mathbf{H}}$ peut être exécutée dans $\underline{\mathbf{G}}$ et les graphes $\underline{\mathbf{G}}'$ et $\underline{\mathbf{H}}'$ obtenus après la transition vérifient $\underline{\mathbf{H}}' \subseteq \downarrow \underline{\mathbf{G}}'$.*

Démonstration. Nous prouvons ce lemme en construisant une exécution similaire dans $\mathbf{G}$ et dans $\mathbf{H}$ préservant la relation $\subseteq \downarrow$ pour les graphes résultants. Pour un processus $v \in V(\mathbf{H})$, nous noterons $\tilde{v} \in V(\mathbf{G})$ pour désigner le processus de $\mathbf{G}$ correspondant à v par la relation $\subseteq \downarrow$. Une étape de l'exécution correspond à une transition locale d'un processus v du réseau.

Si le processus v envoie un message m_0 à ses voisins dans $\mathbf{H}$ alors pour tout voisin w connu de v et donc $id(w) \in \text{CONTACTS}_{\mathbf{H}}(v)$, $\tilde{v}$ possède un voisin $\tilde{w}$ tel que $\delta_v(w) = \delta_{\tilde{v}}(\tilde{w})$ et $id(w) = id_{\tilde{w}} \in \text{CONTACTS}_{\mathbf{G}}(\tilde{v})$ (car $\text{CONTACTS}_{\mathbf{H}}(v) = \text{CONTACTS}_{\mathbf{G}}(\tilde{v})$ et $\lambda_{\mathbf{H}}(v) = \lambda_{\mathbf{G}}(\tilde{v})$ par définition de $\subseteq \downarrow$ et par hypothèse). Ainsi, $\tilde{v}$ calcul les mêmes valeurs et envoie le même message m_0 dans $\mathbf{G}$ via les mêmes numéros de port que v dans $\mathbf{H}$. Comme pour tout $vw \in A(\mathbf{H})$, le message m_0 est ajouté aux tampons $B_{\mathbf{H}}(v, w)$ et $B_{\mathbf{G}}(\tilde{v}, \tilde{w})$, et comme par hypothèse, $B_{\mathbf{H}}(v, w) = B_{\mathbf{G}}(\tilde{v}, \tilde{w})$, nous obtenons que les états de v et $\tilde{v}$ et les tampons de messages en transit $B'_{\mathbf{H}}$ et $B'_{\mathbf{G}}$ obtenus en fin de transition sont identiques.

Si le processus v reçoit un message $m_1 \in B_{\mathbf{H}}(w, v)$ d'un de ses voisins w dans $\mathbf{H}$ par un port in alors comme $B_{\mathbf{H}}(w, v) = B_{\mathbf{G}}(\tilde{w}, \tilde{v})$, $\tilde{v}$ reçoit le même message m_1 dans $\mathbf{G}$. Comme le

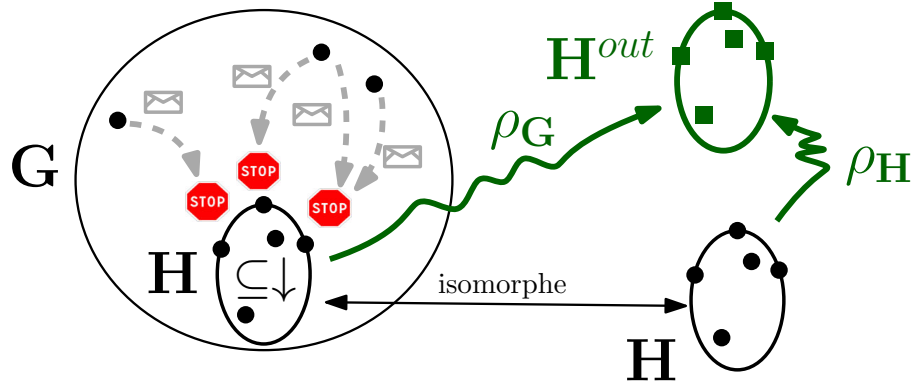


FIGURE 3.1 – Illustration du lemme d'isolement.

Dans l'exécution ρ_G , les messages à destination du sous-graphe H sont retardés suffisamment longtemps pour que les processus de G inclus dans H exécutent ρ_G comme s'ils étaient dans le graphe H (en bas à droite). Ainsi, ils décideront une valeur prise dans un étiquetage final de H , noté H^{out} (graphe en haut à droite). Les sommets représentés par un carré sont des sommets ayant décidé.

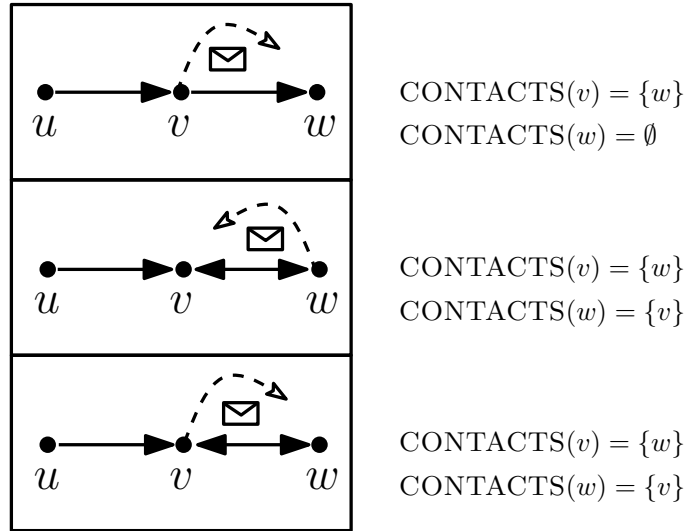


FIGURE 3.2 – Illustration d'un cas particulier du lemme d'isolement.

Dans cette exécution, v et w ne pourront pas détecter la présence de u dans le réseau tant que u ne se sera pas manifesté, i.e., n'aura exécuté ni la procédure I , ni la procédure R .

message est retiré des tampons $B_{\mathbf{H}}(w, v)$ et $B_{\mathbf{G}}(\tilde{w}, \tilde{v})$, nous obtenons que v et $\tilde{v}$ achèvent leur transition dans le même état et avec les mêmes messages en transit : $B_{\mathbf{H}}(w, v)' = B_{\mathbf{G}}(\tilde{w}, \tilde{v})'$. Notons que v peut découvrir l'existence de w au travers m_1 . Un tel cas implique que $\tilde{v}$ découvre aussi l'existence de $\tilde{w}$ et comme $id(w) = id_{\tilde{w}}$ et $\delta_v(w) = \delta_{\tilde{v}}(\tilde{w})$ (par construction), en fin de transition, $\text{CONTACTS}_{\mathbf{H}}(v)' = \text{CONTACTS}_{\mathbf{G}}(\tilde{v})'$. Comme v est le seul processus actif durant cette étape, nous obtenons que $\mathbf{H}' \subseteq \downarrow \mathbf{G}'$. $\square$

À partir du lemme précédent, pour tout graphe $\mathbf{G}, \mathbf{H}$ tel que $\mathbf{H} \subseteq \downarrow \mathbf{G}$, pour tout algorithme $\mathcal{A}$ et toute exécution $\rho_{\mathbf{H}}$ de $\mathcal{A}$ sur $\mathbf{H}$, nous pouvons construire une exécution $\rho_{\mathbf{G}}$ dans $\mathbf{G}$ en itérant le lemme précédent. Au début de cette exécution, les seuls processus actifs seront les processus de $\mathbf{H}$ et tous les autres seront inactifs. Par conséquent, les processus actifs de $\mathbf{G}$ exécuteront leur algorithme comme s'ils exécutaient leur algorithme dans $\mathbf{H}$ et non dans $\mathbf{G}$. Une telle exécution est appelée une *exécution isolée* de $\mathcal{A}$ dans un sous graphe $\mathbf{H}$ de $\mathbf{G}$. En considérant des exécutions isolées qui terminent, nous obtenons le lemme d'isolement suivant,

Lemme 3.5 (Lemme d'isolement). *Pour tout graphe $\mathbf{G}$ et $\mathbf{H}$ tel que $\mathbf{H} \subseteq \downarrow \mathbf{G}$, pour toute exécution $\rho_{\mathbf{H}}$ d'un algorithme $\mathcal{A}$ sur $\mathbf{H}$ qui termine, il existe une exécution $\rho_{\mathbf{G}}$ de $\mathcal{A}$ sur $\mathbf{G}$ telle que $\text{out}_{\rho_{\mathbf{H}}}(v) = \text{out}_{\rho_{\mathbf{G}}}(v)$, pour tout $v \in V(\mathbf{H})$.*

3.3 Condition nécessaire $\mathfrak{C}_{\text{CARTO}}$

Par le lemme d'isolement présenté ci-dessus, pour tout graphe $\mathbf{H} \subseteq \downarrow \mathbf{G}$ et pour toute exécution d'un algorithme sur $\mathbf{G}$, un adversaire peut retarder les messages suffisamment longtemps pour que les processus du sous-graphe $\mathbf{H}$ exécutent leur algorithme dans $\mathbf{G}$ et terminent sans avoir eu connaissance de l'ensemble des processus du réseau $\mathbf{G}$. Pour un algorithme tentant de résoudre CARTO , les processus de $\mathbf{H}$ pourront donc décider la carte de $\mathbf{H}$ et non celle de $\mathbf{G}$, contredisant CARTO . Pour qu'un algorithme puisse résoudre CARTO dans tout graphe d'une famille $\mathcal{F}$, il faut donc empêcher les processus de décider tant qu'ils ne connaissent pas l'intégralité des processus du réseau. La condition $\mathfrak{C}_{\text{CARTO}}$ est donc la suivante.

Définition 3.6 ($\mathfrak{C}_{\text{CARTO}}$). Une famille de graphes $\mathcal{F}$ satisfait $\mathfrak{C}_{\text{CARTO}}$ si pour tout couple de graphes $\mathbf{G}, \mathbf{H} \in \mathcal{F}$ tels que $\mathbf{H} \subseteq \downarrow \mathbf{G}$ alors $\mathbf{H} \simeq \mathbf{G}$.

En d'autres termes, pour qu'une famille vérifie $\mathfrak{C}_{\text{CARTO}}$, aucun graphe de la famille $\mathcal{F}$ ne doit être clos par successeurs dans un graphe (différent du premier) appartenant aussi à la famille $\mathcal{F}$. Nous prouvons maintenant que cette condition est nécessaire.

Lemme 3.7. *Pour toute famille $\mathcal{F}$ de graphes, s'il existe un algorithme $\mathcal{F}$ -restreint pour le problème CARTO alors $\mathcal{F}$ satisfait $\mathfrak{C}_{\text{CARTO}}$.*

Démonstration. Supposons par l'absurde une famille $\mathcal{F}$ admettant un algorithme $\mathcal{F}$ -restreint $\mathcal{A}$ pour CARTO telle que $\mathcal{F}$ ne vérifie pas $\mathfrak{C}_{\text{CARTO}}$. Par définition, il existe deux graphes $\mathbf{G}$ et $\mathbf{H}$ appartenant à $\mathcal{F}$ tels que $\mathbf{H} \subseteq \downarrow \mathbf{G}$ et $\mathbf{H} \not\simeq \mathbf{G}$. Comme $\mathcal{A}$ est $\mathcal{F}$ -restreint pour CARTO et que $\mathbf{H} \in \mathcal{F}$, toute exécution ρ de $\mathcal{A}$ sur $\mathbf{H}$ s'achève et tout processus $v \in V(\mathbf{H})$ aura écrit dans sa variable $\text{out}(v)$ une carte isomorphe à $\mathbf{H}$. Comme $\mathcal{A}$ est $\mathcal{F}$ -restreint pour CARTO et que $\mathbf{G} \in \mathcal{F}$, nous pouvons appliquer le lemme d'isolement entre $\mathbf{H}$ et $\mathbf{G}$ pour obtenir une exécution isolée de $\mathcal{A}$ sur $\mathbf{G}$ où les processus $V(\mathbf{H})$ décideront exactement les mêmes valeurs que s'ils exécutaient leur algorithme dans $\mathbf{H}$. Par conséquent, cette exécution dans $\mathbf{G}$ admet des processus qui décident un graphe isomorphe à $\mathbf{H}$, une contradiction pour CARTO . $\square$

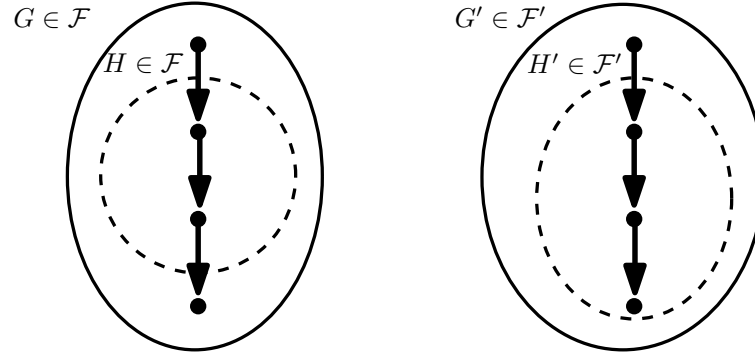


FIGURE 3.3 – Illustration de la condition $\mathcal{C}_{\text{CARTO}}$. Dans cet exemple, la famille $\mathcal{F} = \{G, H\}$ à gauche vérifie $\mathcal{C}_{\text{CARTO}}$ car $H \not\subseteq G$. La famille $\mathcal{F}' = \{G', H'\}$ à droite ne vérifie pas $\mathcal{C}_{\text{CARTO}}$ car $H' \subseteq G$, $H' \neq G'$ et $H' \in \mathcal{F}'$.

Par exemple, comme illustrée dans la figure 3.3, la famille des chemins dirigés $\mathcal{P}$ ne vérifie pas $\mathcal{C}_{\text{CARTO}}$. Par conséquent, nous pouvons en déduire grâce au lemme précédent qu'il n'existe pas d'algorithme $\mathcal{P}$ – *restreint* pour le problème CARTO . Nous obtenons le lemme suivant,

Lemme 3.8. *Il n'existe pas d'algorithme universel résolvant CARTO .*

3.4 Condition suffisante

Nous montrons maintenant que $\mathcal{C}_{\text{CARTO}}$ est suffisante en proposant un algorithme $\mathcal{F}$ –restreint résolvant CARTO pour toute famille $\mathcal{F}$ satisfaisant $\mathcal{C}_{\text{CARTO}}$.

Définition 3.9. Soit $\chi_{\mathcal{F}} : \mathbf{G} \rightarrow \{\text{vrai}, \text{faux}\}$ la fonction caractéristique de la famille de graphes $\mathcal{F}$ telle que $\chi_{\mathcal{F}}(\mathbf{G}) = \text{vrai}$ si et seulement si $\mathbf{G} \in \mathcal{F}$.

Cette fonction permet au processus d'obtenir une information "globale" sur le contexte d'exécution. Comme nous allons le voir, cette connaissance supplémentaire permettra de retarder les décisions des processus suffisamment longtemps pour permettre à l'exécution de se stabiliser et aux processus de faire découvrir.

Notons que cette manière de donner des connaissances globales aux processus est équivalente à créer un algorithme spécifique pour chaque famille de graphes (utilisant explicitement les propriétés de la famille considérée).

3.4.1 Présentation de l'algorithme

L'algorithme est quasi identique à l'algorithme 2 et seules les lignes 9 et 10 changent. La stratégie est donc la même. Initialement, chaque processus exécute la procédure **I** et envoie son identité ainsi que les identités de ses voisins initiaux. Puis, à chaque réception de message, les processus exécutent la procédure **R**. La différence se situe au niveau de la prise de décision. Un processus décidera s'il n'a jamais décidé lors d'une étape précédente, si l'exécution a atteint un état stable ($\text{Vu}(M) = \text{Reçu}(M)$) et si le graphe induit par la boîte aux lettres M appartient à la famille $\mathcal{F}$. Cette dernière condition donne une information cruciale au processus.

Nous savons à partir du lemme 2.16 que tout processus exécutant la ligne 10 possédera une boîte aux lettres M telle que $\text{T}(M) \subseteq G$. En considérant une exécution de l'algorithme dans un graphe appartenant à une famille satisfaisant $\mathcal{C}_{\text{CARTO}}$, si $\chi_{\mathcal{F}}$ retourne **vrai** alors, par définition

de $\mathcal{C}_{\text{CARTO}}$, $\mathbb{T}(M)$ est isomorphe à $\mathbf{G}$. Si $\mathbb{T}(M)$ n'est pas isomorphe à $\mathbf{G}$, alors le processus sait qu'il ne possède pas toutes les identités du réseau et qu'il doit attendre pour les recevoir car $\mathbb{T}(M) \notin \mathcal{F}$. Une fois ces conditions satisfaites, le processus écrira la carte induite par sa boîte aux lettres dans sa variable de décision `out`.

3.4.2 Correction de l'algorithme

Nous montrons donc que tout processus décide une carte isomorphe au graphe de communication initial $\mathbf{G}$.

Lemme 3.10. *Pour toute famille $\mathcal{F}$ satisfaisant $\mathcal{C}_{\text{CARTO}}$, pour toute exécution de l'algorithme 3 sur un graphe $\mathbf{G} \in \mathcal{F}$, pour tout processus $v \in V(\mathbf{G})$, il existe une étape i telle que v décide un graphe $\mathbb{T}(M)$ isomorphe à $\mathbf{G}$.*

Démonstration. Soit $\mathcal{F}$ une famille satisfaisant $\mathcal{C}_{\text{CARTO}}$ et soit $G \in \mathcal{F}$. À partir des lemmes 2.9 et 2.10 et par le corollaire 2.16, pour tout processus $v \in V(\mathbf{G})$, il existe une étape i telle que $VU(M^i(v)) = \text{RECU}(M^i(v))$ et donc $\mathbb{T}(M^i(v)) \subseteq \downarrow \mathbf{G}$. Si le processus décide lors de cette étape, alors $\mathbb{T}(M^i(v)) \in \mathcal{F}$, et donc $\mathbb{T}(M)$ est isomorphe à $\mathbf{G}$ car il n'existe aucun graphe $\mathbf{H} \in \mathcal{F}$ tel que $\mathbf{H} \not\cong \mathbf{G}$ et $\mathbf{H} \subseteq \downarrow \mathbf{G}$ (condition $\mathcal{C}_{\text{CARTO}}$). Ce lemme est prouvé. $\square$

Nous venons de montrer que si un processus décide, alors il décide une carte isomorphe à $\mathbf{G}$. Comme nous avons montré que pour tout processus, il existe une étape où ce processus décide, nous venons donc de prouver que l'algorithme 3 résout CARTO pour toute famille $\mathcal{F}$ satisfaisant $\mathcal{C}_{\text{CARTO}}$ et donc $\mathcal{C}_{\text{CARTO}}$ est suffisante.

Conjointement avec le lemme 3.7 ($\mathcal{C}_{\text{CARTO}}$ est nécessaire), nous obtenons le théorème suivant,

Théorème 3.1. *Il existe un algorithme $\mathcal{F}$ -restreint pour CARTO si et seulement si $\mathcal{F}$ satisfait $\mathcal{C}_{\text{CARTO}}$.*

Algorithme 3 : Calcul de la carte du réseau

Input : $\chi_{\mathcal{F}} : \mathcal{G}_{id} \rightarrow \{\text{true}, \text{false}\}$, fonction caractéristique de $\mathcal{F}$

Procédure I() : /* Procédure initiale */

1 $\lfloor \text{SendAll} \langle id, \{(id(v), \text{CONTACTS}^0(v))\} \rangle;$

Procédure R(id', M') : /* Réception d'un message */

```

2   updated = faux;
3   si  $M' \setminus M \neq \emptyset$  alors
4        $M \leftarrow M \cup M'$ ;
5       updated = vrai;
6   si  $id' \notin \text{CONTACTS}$  alors
7        $\text{CONTACTS} \leftarrow \text{CONTACTS} \cup \{id'\}$ ;
8       updated = vrai;
9   si  $VU(M) = \text{RECU}(M) \wedge out = \perp \wedge \chi_{\mathcal{F}}(\mathbb{T}(M)) = \text{vrai}$  alors
10       $out \leftarrow \mathbb{T}(M)$ ;
11   si updated = vrai alors
12       $\text{SendAll} \langle id, M \rangle;$ 
```

3.5 Exemples

Nous présentons plusieurs familles satisfaisant et ne satisfaisant pas $\mathfrak{C}_{\text{CARTO}}$.

Familles satisfaisant $\mathfrak{C}_{\text{CARTO}}$. La famille des graphes comportant n sommets satisfait $\mathfrak{C}_{\text{CARTO}}$. Pour montrer cela, posons $\chi_{\mathcal{F}} = \text{vrai}$ si et seulement si la boîte aux lettres d'un processus contient n identités distinctes, i.e., la condition d'arrêt devient $|V(\mathbb{T}(M))| = n$. Trivialement, pour tout $\mathbf{H} \subseteq \downarrow \mathbf{G}$, si $\mathbf{H}$ n'est pas isomorphe à $\mathbf{G}$ alors $\mathbf{H} \notin \mathcal{F}$ car $|V(\mathbf{H})| < |V(\mathbf{G})|$ par définition de sous-graphe. De la même manière, la famille des graphes comportant n composantes fortement connexes satisfait aussi $\mathfrak{C}_{\text{CARTO}}$.

Familles ne satisfaisant pas $\mathfrak{C}_{\text{CARTO}}$. La famille $\mathcal{F}$ des graphes comportant au plus n sommets ne satisfait pas $\mathfrak{C}_{\text{CARTO}}$ car pour tout graphe $\mathbf{G} \in \mathcal{F}$, pour tout sous-graphe strict $\mathbf{H} \subseteq \downarrow \mathbf{G}$, le graphe $\mathbf{H}$ appartient à la famille $\mathcal{F}$, contredisant directement $\mathfrak{C}_{\text{CARTO}}$. Pour les mêmes raisons, les familles des chemins dirigés ou des graphes comportant au plus n composantes fortement connexes ou avec un diamètre borné ne satisfont pas $\mathfrak{C}_{\text{CARTO}}$.

Chapitre 4

Élection

Dans ce chapitre, nous étudions le problème de l'élection dans le modèle des participants inconnus. Ces résultats ont été publiés lors de la conférence SIROCCO en 2014 [CGN14].

Dans la section 4.1, nous présentons le problème $\mathcal{ELEC}$. Dans la section 4.2, nous définissons la condition $\mathfrak{C}_{\mathcal{ELEC}}$ et montrons que $\mathfrak{C}_{\mathcal{ELEC}}$ est nécessairement satisfaite par toute famille $\mathcal{F}$ admettant un algorithme $\mathcal{F}$ -restreint pour $\mathcal{ELEC}$. Pour cela, nous réutiliserons les mêmes méthodes et techniques que celles utilisées dans le chapitre précédent. Dans la section 4.3, nous prouvons que la condition $\mathfrak{C}_{\mathcal{ELEC}}$ est suffisante. Pour cela, nous proposons un algorithme $\mathcal{F}$ -restreint résolvant $\mathcal{ELEC}$ pour toute famille satisfaisant $\mathfrak{C}_{\mathcal{ELEC}}$ et nous prouvons sa correction. Dans la section 4.4, nous donnons des exemples de familles de graphes vérifiant et ne vérifiant pas $\mathfrak{C}_{\mathcal{ELEC}}$.

4.1 Description du problème $\mathcal{ELEC}$

Le problème de l'élection est un problème fondamental en algorithmique distribuée. C'est une brique centrale à résoudre pour de nombreux problèmes d'accords dans un réseau distribué. Par exemple, les transactions sur des bases de données réparties se basent sur une élection. Nous considérerons le problème de l'élection suivant,

Définition 4.1 (Spécification $\mathcal{ELEC}$). Pour tout graphe $\mathbf{G}$, pour tout graphe $\mathbf{G}^{out}$, nous noterons $\mathbf{G} \mathcal{ELEC} \mathbf{G}^{out}$ s'il existe un unique processus $v \in V(\mathbf{G})$ tel que $out_{\mathbf{G}^{out}}(v) = \text{élu}$ et pour tout processus $w \neq v$, $out_{\mathbf{G}^{out}}(w) = \text{non-élu}$.

Comme pour le problème $\mathcal{CARTO}$ précédent, ce problème n'admet pas d'algorithme universel.

Remarque 4.2. Ce problème est plus complexe à résoudre que $\mathcal{CARTO}$ sur le plan algorithmique mais, au contraire des cas classiques où il est équivalent, il se résout sur de plus grandes familles de graphes que pour $\mathcal{CARTO}$.

4.2 Condition nécessaire $\mathfrak{C}_{\mathcal{ELEC}}$

Nous montrons maintenant qu'il est impossible de résoudre une élection dans l'ensemble des graphes. L'idée de cette impossibilité est que, dans certains graphes, plus d'une exécution isolée et distincte peut s'exécuter en parallèle et se terminer avec un processus élu dans chaque

exécution isolée. Nous avons montré que la condition $\mathfrak{C}_{\mathcal{CARTO}}$, présentée dans le chapitre 3, contraint les processus à retarder leur décision pour éviter que des exécutions isolées intempestives empêchent la résolution du problème. Néanmoins, comme nous allons le voir, cette condition n'est pas nécessaire pour le problème, $\mathcal{EL}\mathcal{EC}$ car une seule exécution isolée laissant les processus participants décider une valeur cohérente avec $\mathcal{EL}\mathcal{EC}$, i.e., un processus décide **élu** et tous les autres décident **non-élu**, ne compromet pas directement la spécification $\mathcal{EL}\mathcal{EC}$. Si aucun autre processus n'est élu par la suite, l'étiquetage final obtenu vérifiera $\mathcal{EL}\mathcal{EC}$.

Définition 4.3 ($\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$). Une famille de graphes $\mathcal{F}$ satisfait $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$ si pour tout graphe $\mathbf{G}$, $\mathbf{H}_1$ et $\mathbf{H}_2$ appartenant à $\mathcal{F}$ tels que $\mathbf{H}_1 \subseteq \downarrow \mathbf{G}$ et $\mathbf{H}_2 \subseteq \downarrow \mathbf{G}$ alors $V(\mathbf{H}_1) \cap V(\mathbf{H}_2) \neq \emptyset$.

La figure 4.1 illustre cette condition. Nous prouvons maintenant que $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$ est une condition nécessaire à l'existence d'un algorithme $\mathcal{F}$ -restreint pour le problème $\mathcal{EL}\mathcal{EC}$.

Lemme 4.4. Pour toute famille $\mathcal{F}$ de graphes, s'il existe un algorithme $\mathcal{F}$ -universel pour le problème $\mathcal{EL}\mathcal{EC}$ alors $\mathcal{F}$ satisfait $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$.

Démonstration. Considérons une famille $\mathcal{F}$ admettant un algorithme $\mathcal{F}$ -restreint pour $\mathcal{EL}\mathcal{EC}$ et supposons que cette famille $\mathcal{F}$ ne vérifie pas $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$. Par définition, il existe $\mathbf{G}$, $\mathbf{H}_1$ et $\mathbf{H}_2$ appartenant à $\mathcal{F}$ tels que $\mathbf{H}_1 \subseteq \downarrow \mathbf{G}$, $\mathbf{H}_2 \subseteq \downarrow \mathbf{G}$ et $V(\mathbf{H}_1) \cap V(\mathbf{H}_2) = \emptyset$.

Comme $\mathcal{A}$ est $\mathcal{F}$ -restreint pour $\mathcal{EL}\mathcal{EC}$ et que $\mathbf{H}_1 \in \mathcal{F}$, pour toute exécution ρ_1 de $\mathcal{A}$ sur $\mathbf{H}_1$, il existe une étape i telle que l'exécution s'achève avec un unique processus $v_1 \in V(\mathbf{H}_1)$ ayant décidé **élu**, i.e., $out(v_1) = \text{élu}$ et tous les autres ayant décidé **non-élu**. De la même façon, comme $\mathbf{H}_2 \in \mathcal{F}$, pour toute exécution ρ_2 de $\mathcal{A}$ sur $\mathbf{H}_2$, il existe une étape i telle que l'exécution s'achève avec un unique processus $v_2 \in V(\mathbf{H}_2)$ ayant décidé **élu**, i.e., $out(v_2) = \text{élu}$ et tous les autres ayant décidé **non-élu**.

Comme $\mathbf{G} \in \mathcal{F}$, $\mathbf{H}_1, \mathbf{H}_2 \subseteq \downarrow \mathbf{G}$ et $V(\mathbf{H}_1) \cap V(\mathbf{H}_2) = \emptyset$, nous pouvons appliquer le lemme d'isolement sur $\mathbf{H}_1$ (resp. $\mathbf{H}_2$) et $\mathbf{G}$. Comme ces exécutions terminent dans $\mathbf{G}$, les processus isolés de $\mathbf{H}_1$ (resp. $\mathbf{H}_2$) décideront. Nous obtenons donc dans une même exécution sur $\mathbf{G}$, un processus élu dans $\mathbf{H}_1 \subseteq \downarrow \mathbf{G}$ et un processus élu dans $\mathbf{H}_2 \subseteq \downarrow \mathbf{G}$, une contradiction avec $\mathcal{EL}\mathcal{EC}$. $\square$

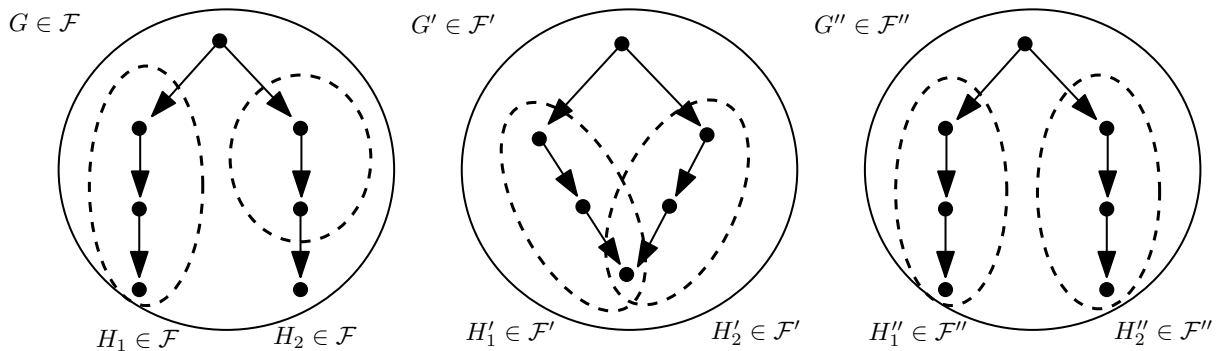


FIGURE 4.1 – Illustration de la condition $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$. Dans cet exemple, la famille $\mathcal{F} = \{G, H_1, H_2\}$ à gauche vérifie $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$ car $H_2 \not\subseteq \downarrow G$. La famille $\mathcal{F}' = \{G', H_1', H_2'\}$ à droite vérifie $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$ car $V(H_1' \cap H_2') \neq \emptyset$. La famille $\mathcal{F}'' = \{H_1'', H_2'', G\}$ ne vérifie pas $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$ car $V(H_1'' \cap H_2'') = \emptyset$, $H_1'', H_2'' \subseteq \downarrow G$.

4.3 Condition suffisante

Nous montrons maintenant que $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$ est suffisante en proposant un algorithme $\mathcal{F}$ -restreint résolvant $\mathcal{EL}\mathcal{EC}$ pour toute famille $\mathcal{F}$ satisfaisant $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$.

Algorithme 4 : Élection

Input : $\chi_{\mathcal{F}} : \mathcal{G}_{id} \rightarrow \{true, false\}$, fonction caractéristique de $\mathcal{F}$

Procédure I() : /* Procédure initiale */

```

1   $M^0 = \{(id(v), \text{CONTACTS}^0(v))\};$ 
2   $\mathcal{V}^0 = \{(id(v), M^0)\};$ 
3   $\text{SendAll} \langle id(v), \mathcal{V}^0(v) \rangle;$ 

```

Procédure R(id', $\mathcal{V}'$) : /* Message reçu */

```

4  updated = faux;
5  si  $\mathcal{V}' \setminus \mathcal{V} \neq \emptyset$  alors
6     $M \leftarrow M \cup \bigcup_{(id', M', statut') \in \mathcal{V}'} M';$ 
7     $\mathcal{V} \leftarrow \mathcal{V} \cup \mathcal{V}' \cup \{(id, M, statut)\};$ 
8    updated = vrai;
9  si  $id' \notin \text{CONTACTS}$  alors
10    $\text{CONTACTS} \leftarrow \text{CONTACTS} \cup \{id'\};$ 
11   updated = vrai;
12  si  $\text{VU}(M) = \text{REQU}(M)$ ,  $out = \perp$ ,  $\chi_{\mathcal{F}}(\mathbb{T}(M)) = \text{vrai}$  et pour tout
     $(id', M', statut') \in \mathcal{V}$ ,  $M' = M$  alors
13   updated = vrai;
14   si  $\nexists (id'', C'', \text{élu}) \in M \wedge id = \min\{id'' \mid id'' \in \text{VU}(M)\}$  alors
15      $out \leftarrow \text{élu};$ 
16   sinon
17      $out \leftarrow \text{non-élu};$ 
18  si updated = vrai alors
19    $\text{SendAll} \langle id, M \rangle;$ 

```

4.3.1 Présentation de l'algorithme

L'algorithme 4 présenté dans cette section est une extension de l'algorithme 3 précédent. À la différence de l'algorithme 3, l'algorithme 4 retarde la décision des processus en attendant un état de stabilisation "avancé", i.e., un état où les processus partageront les mêmes connaissances du réseau. Un processus décidera si de son point de vue (local), tous les processus à sa connaissance possèdent une même boîte aux lettres que lui.

Cette stabilisation est obtenue grâce à une nouvelle structure donnée aux processus : une *super boîte aux lettres* permettant à chaque processus d'enregistrer les boîtes aux lettres des autres processus pour se comparer.

Chaque entrée d'une super boîte aux lettres est composée d'une identité, d'un statut (la décision du processus) et d'une boîte aux lettres (la même que celle présentée dans les sections précédentes).

Définition 4.5 (Super boîte aux lettres $\mathcal{V}$). On appelle $\mathcal{V} \subset \mathbb{N} \times (\mathbb{N} \times \mathbb{N}^{\mathbb{N}}) \times \{\text{élu}, \text{non-élu}\}$, une super boîte aux lettres. Chaque élément de $\mathcal{V}$ est de la forme $(id, M, statut)$ avec id , l'identité d'un processus, M une boîte aux lettres et $statut \in \{\text{élu}, \text{non-élu}\}$.

Les processus enverront des messages de la forme $\langle id, \mathcal{V} \rangle$ avec id l'identité du processus émetteur du message et $\mathcal{V}$, la super boîte aux lettres du processus au moment de l'envoi. Les processus s'échangeront leur super boîte aux lettres pour obtenir les dernières boîtes aux lettres connues des processus du réseau.

Pour décider, un processus devra avoir une boîte aux lettres satisfaisant les conditions suivantes (Ligne 12) :

- $VU(M) = \text{RECU}(M)$. Cette condition nous assure que le graphe reconstruit $\mathbb{T}(M)$ est clos par successeurs dans $\mathbf{G}$.
- $\mathbb{T}(M) \in \mathcal{F}$. Cette condition nous permet d'utiliser les propriétés de la famille $\mathcal{F}$.
- $out = \perp$. Cette condition nous assure que les processus décident de manière explicite.
- $\forall (id(v), M(v), statut(v)) \in \mathcal{V}, M(v) = M$. Cette condition permet à un processus v d'attendre un état suffisamment stable de l'exécution pour décider, i.e., tel que tout processus w connu de v possède (ou ait possédé) la même boîte aux lettres que v .

Une fois ces conditions satisfaites, un processus décidera **élu** si son identité est minimale parmi toutes celles présentes dans sa boîte aux lettres et si, à sa connaissance, il n'existe pas de processus ayant déjà décidé **élu** (Ligne 14). Si une de ces deux conditions n'est pas respectée, alors le processus décidera **non-élu** à la ligne 17.

Remarque 4.6. Notons qu'un processus qui décide **élu** possède l'identité minimale parmi toutes les identités apparaissant dans sa boîte aux lettres et que celle-ci peut ne pas être minimale dans le réseau (en cas d'exécutions isolées).

4.3.2 Correction de l'algorithme

Nous montrons maintenant que toute exécution de l'algorithme 4 satisfait la condition de terminaison explicite et est correcte vis-à-vis de $\mathcal{ELC}$. Tout d'abord, nous montrons certaines propriétés montrant la consistance d'une super boîte aux lettres tout au long d'une exécution de l'algorithme 4.

Comme les boîtes aux lettres sont identiques à celles utilisées dans les chapitres 2 et 3 précédents, nous ne reprouvons pas leurs propriétés invariantes déjà prouvées dans le lemme 2.9. Ci-dessous, nous montrons que le contenu des super boîtes aux lettres est cohérent tout au long d'une exécution de l'algorithme 4. Pour cela, nous prouvons que les boîtes aux lettres présentes dans une super boîte aux lettres ne sont pas "corrompues" par l'algorithme, i.e., qu'elles ont bien toutes appartenues à un processus du réseau à un moment donné de l'exécution.

Lemme 4.7. *Pour toute exécution ρ de l'algorithme 4 sur un graphe $\mathbf{G}$, pour toute étape i , pour tout processus $v \in V(\mathbf{G})$ et pour tout $(id, M, statut) \in \mathcal{V}^i(v)$, il existe un processus u et une étape $i' < i$ telle que $id(u) = id$, $M^{i'}(u) = M$ et $out^{i'}(u) = statut$.*

Démonstration. Soit j et i deux étapes d'une exécution sur un graphe $\mathbf{G}$ telles que $i < j$. Soit $V^j = \{\bar{\mathcal{V}}^i(v) \mid \bar{\mathcal{V}}^i(v) = \mathcal{V}^i(v) \setminus \{(id(v), M(v), out(v))\}, \forall v, \forall i \leq j\}$ l'ensemble contenant la super boîte aux lettres de chaque processus v du réseau lors d'une étape $i \leq j$ sans sa propre entrée $(id(v), M(v), out(v))$.

Nous prouvons ce lemme par induction sur les éléments contenus dans cet ensemble V^j . Initialement, avant toute transmission de message, la boîte aux lettres de chaque processus

possède seulement les identités des successeurs du processus dans $\mathbf{G}$. Donc V^0 est vide. Par l'algorithme, pour tout $(id(w), M, statut)$ qui apparaît pour la première fois dans V^i où $i > 0$, un processus a reçu un message contenant $(id(w), M, statut)$. Comme aucun processus ne possède une telle entrée $(id(w), M, statut)$ initialement (excepté w), nous obtenons que w a envoyé ce message. Pour toute étape i , pour toute entrée $(id, M, statut) \in V^i$, si $M^0(w) \neq M$, alors il existe une étape $h < j$ telle que $(id, M, statut) \notin V^{j-1}$ et $(id, M, statut) \in V^j$ (V^0 est vide). Par notre hypothèse d'induction, il existe donc une étape telle que w a envoyé le message contenant $(id, M, statut)$. Nous prouvons donc ce lemme. $\square$

Nous montrons maintenant que le contenu d'une super boîte aux lettres est croissant lors d'une exécution.

Lemme 4.8. *Pour tout processus v et pour toute étape $i \geq 0$ d'une exécution de l'algorithme 4, $\mathcal{V}^i(v) \subseteq \mathcal{V}^{i+1}(v)$ et v envoie un message lors de l'étape $i + 1$ si et seulement si $\mathcal{V}^i(v) \subset \mathcal{V}^{i+1}(v)$.*

Démonstration. Les super boîtes aux lettres contiennent une identité invariante dans l'exécution. Elles contiennent aussi une étiquette *statut* qui par construction ne peut changer de valeur qu'une et une seule fois par exécution. Concernant les boîtes aux lettres contenues dans une super boîte aux lettres, celles-ci ne prennent qu'un nombre fini de valeurs (Lemme 2.7). Comme lors d'une étape i , si v reçoit un message contenant une super boîte aux lettres $\mathcal{V}$, par le lemme 2.9 et 4.7, pour tout $(id, M, statut) \in \mathcal{V}$, si $(id, M', statut) \in \mathcal{V}_v^{i+1}$ alors $M \subset M'$. Comme $\mathcal{V}^i \subset \mathcal{V}^{i+1}$, nous prouvons que $\mathcal{V}^i$ est croissant. $\square$

Le lemme suivant est similaire au lemme 2.10 présenté dans le chapitre précédent et se prouve de manière similaire. Nous omettons la preuve.

Lemme 4.9. *Pour toute étape i d'une exécution de l'algorithme 4 sur un graphe $\mathbf{G}$, pour tout processus $v, w \in V(\mathbf{G})$, pour tout $(id, M, statut) \in \mathcal{V}^i(v)$, il existe une étape $j \geq i$ telle que $(id, M, statut) \in \mathcal{V}^j(w)$.*

Notons aussi que par les lemmes 2.7, 4.8 et 4.9, il existe une étape de stabilisation dans toute exécution où les processus posséderont tous une même super boîte aux lettres. Nous obtenons le lemme suivant.

Lemme 4.10. *Pour toute étape i d'une exécution de l'algorithme 4 sur un graphe $\mathbf{G}$, pour tout processus $v \in V(\mathbf{G})$, il existe une étape i telle que pour tout $(id, M, statut) \in \mathcal{V}^i(v)$, $M = M^i(v)$.*

Nous sommes maintenant en mesure de montrer la correction de l'algorithme 4. Notre but est de montrer que pour toute exécution de l'algorithme 4, il existe au moins un et au plus un processus élu. Nous montrons, en premier, qu'il existe toujours au moins un processus élu dans toute exécution.

Lemme 4.11. *Pour toute exécution ρ de l'algorithme 4 sur un graphe $\mathbf{G}$, il existe une étape i et un processus v tels que $out(v)^i = \mathbf{él}u$.*

Démonstration. Prenons une famille $\mathcal{F}$ satisfaisant $\mathcal{C}_{\mathcal{ELC}}$, un graphe $\mathbf{G} \in \mathcal{F}$ et une exécution de l'algorithme 4 sur $\mathbf{G}$. Soit v_{min} , le processus ayant l'identité id_{min} minimale parmi tous les processus de $\mathbf{G}$. Par les lemmes 4.9 et 4.10, il existe une étape i telle que pour tout $v, w \in V(G)$, $M^i(v) = M^j(v) = M^i(w) = M^j(w)$ et $\mathcal{V}_v^i = \mathcal{V}_v^j = \mathcal{V}_w^i = \mathcal{V}_w^j$, pour tout $j \geq i$. Lors d'une telle étape, si un processus a déjà décidé élu, alors ce cas est prouvé. Supposons qu'aucun processus soit élu et que le processus v_{min} soit actif lors de l'étape i . Comme tout processus

possède une boîte aux lettres identique, toutes les identités sont présentes dans la boîte aux lettres ainsi que tous les contacts initiaux. Par conséquent, $VU(M^i(v_{min})) = REGU(M^i(v_{min}))$ et $T(M^i(v_{min})) \simeq \mathbf{G}$. Comme $\mathbf{G} \in \mathcal{F}$, $T(M^i(v_{min})) \simeq \mathbf{G}$ retourne **vrai** et comme aucun processus n'est élu, $out(v_{min}) = \perp$. Ainsi, v_{min} satisfait toutes les conditions de décisions lors d'une telle étape et exécute la ligne 14. Comme $id(v_{min})$ est la plus petite identité parmi tous les processus du réseau, v_{min} décidera **élu**. Nous prouvons ce lemme. $\square$

Nous montrons maintenant qu'il y a toujours au plus un processus élu dans chaque exécution.

Lemme 4.12. *Pour toute exécution ρ de l'algorithme 4 sur un graphe $\mathbf{G}$, pour toute étape i et pour tout processus $u, v \in V(\mathbf{G})$ tel que $u \neq v$, si $out(u)^i = \mathbf{élu}$ alors $out(v)^i = \mathbf{non-élu}$.*

Démonstration. Supposons par l'absurde que deux processus u et v soit élus lors d'une exécution de l'algorithme 4 dans un graphe $\mathbf{G}$. Soit i , l'étape où u est élu et j est l'étape où v est élu. Sans perte de généralité, supposons que $i < j$.

Soit $\mathbf{H}_u = T(M^i(u))$ et $\mathbf{H}_v = T(M^j(v))$ les graphes induits par les boîtes aux lettres de u et v . Comme u et v sont élus, nous obtenons que

- $VU(M^i(u)) = REGU(M^i(u))$ et $VU(M^j(v)) = REGU(M^j(v))$
- $\mathbf{H}_u, \mathbf{H}_v \in \mathcal{F}$
- $\forall (id, M, statut) \in \mathcal{V}(u), M = M^i(u)$ et $\forall (id, M, statut) \in \mathcal{V}(v), M = M^i(v)$.
- $\mathbf{H}_u, \mathbf{H}_v \subseteq \downarrow \mathbf{G}$ (Lemme 2.16).

Nous distinguons deux cas possibles lors de l'étape i où u est élu.

Si $id(v) \in VU(M^i(u))$, alors par la Ligne 14, $id(u) < id(v)$. De plus, par les lignes 12, 14 et par le lemme 4.7, il existe une étape $h < i$ telle que $M^h(v) = M^i(u)$ et $out^h(v) = \perp$. Comme $id(u) \in VU(M^i(u))$, $VU(M^i(u)) = VU(M^h(v))$ et comme le contenu de $M(v)$ est croissant, $id(u) \in VU(M^j(v))$ ($h \leq i \leq j$). Par conséquent, lorsque v exécutera la ligne 14 lors de l'étape j , il ne sera pas élu car il ne possède pas l'identité minimale. Nous obtenons une contradiction.

Si $id(v) \notin VU(M^i(u))$ alors comme $\mathbf{H}_u, \mathbf{H}_v \subseteq \downarrow \mathbf{G}$ et comme $\mathcal{F}$ satisfait $\mathcal{CELEC}$, il existe un processus w tel que $id(w) \in V(\mathbf{H}_u) \cap V(\mathbf{H}_v)$, i.e., les deux exécutions ne sont pas sommet disjoint. Par le lemme 4.7, nous savons qu'il existe une étape $i' \leq i$ telle que $M^{i'}(w) = M^i(u)$. Pour les mêmes raisons, il existe une étape $j' \leq j$ telle que $M^{j'}(w) = M^j(v)$. Comme $id(v) \notin VU(M^{i'}(w))$ et que $id(v) \in VU(M^{j'}(w))$, par le lemme 2.9, nous obtenons que $i' < j'$ et donc que $M^i(u) = M^{i'}(w) \subsetneq M^{j'}(w) = M^j(v)$. Par conséquent, $id(u) \in VU(M^j(v))$ et comme v est élu lors de l'étape j , il existe $(id(u), M^j(u), statut) \in \mathcal{V}_v^j$. Par le lemme 4.7, il existe une étape $i'' < j$ telle que $M^{i''}(u) = M^j(v)$ et $out^{i''}(u) = statut$. Comme $id(v) \in VU(M^{i''}(u))$ et $id(v) \notin VU(M^i(u))$ alors $M^i(u) \subset M^{i''}(u)$. Par conséquent, comme $(id(u), M^{i-1}(u), \perp) \in \mathcal{V}_u^i$ et $(id(u), M^i(u), \mathbf{élu}) \in \mathcal{V}_u^i$, il existe $(id(u), M^{i''}(u), \mathbf{élu}) \in \mathcal{V}_u^j$ et donc, v ne sera pas élu lors l'étape j car il existe une étiquette **élu** dans $M^j(v)$ (Ligne 14). Nous obtenons une contradiction. $\square$

Comme pour toute exécution de notre algorithme dans une famille $\mathcal{F}$ satisfaisant $\mathcal{CELEC}$, il y aura toujours exactement un processus élu en fin d'exécution, nous obtenons le théorème suivant,

Théorème 4.1. *Il existe un algorithme $\mathcal{F}$ -restreint pour $\mathcal{ELEC}$ si et seulement si $\mathcal{F}$ satisfait $\mathcal{CELEC}$.*

4.4 Exemples

Nous présentons plusieurs familles satisfaisant et ne satisfaisant pas $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$. Une borne serrée B pour un ensemble S est une borne supérieure sur le nombre d'éléments de S telle que $|S| \leq B < 2 \times |S|$.

Familles satisfaisant $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$. Premièrement, par définition, toute famille satisfaisant $\mathfrak{C}_{\mathcal{CARTO}}$ satisfera $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$. La famille des chemins dirigés satisfait $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$. La famille $\mathcal{F}$ des graphes contenant exactement n composantes puits satisfait $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$. En effet, tout sous-graphe $\mathbf{H}, \mathbf{H}' \subseteq \downarrow \mathbf{G} \in \mathcal{F}$ tel que $\mathbf{H}, \mathbf{H}' \in \mathcal{F}$ partageront les mêmes n composantes puits par définition. La famille $\mathcal{F}$ des graphes ayant une taille bornée par B avec B une borne serrée satisfait $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$. En effet, tout sous-graphe $\mathbf{H}$ de $\mathbf{G}$ tel que $\mathbf{H} \subseteq \downarrow \mathbf{G}$ et $\mathbf{H} \in \mathcal{F}$ possède au moins $B/2$ sommets et possède moins de $|V(\mathbf{G})|$ sommets. Ainsi, pour tout graphe $\mathbf{H} \in \mathcal{F}$ et $\mathbf{H}' \in \mathcal{F}$ tels que $\mathbf{H} \subseteq \downarrow \mathbf{G}$, $\mathbf{H}' \subseteq \downarrow \mathbf{G}$ et $\mathbf{H} \cap \mathbf{H}' = \emptyset$ possèdent au moins $2 \times \frac{B}{2}$ sommets disjoints. Comme $B \leq |V(\mathbf{H}) \cup V(\mathbf{H}')| < |V(\mathbf{G})|$ (par définition de $\subseteq \downarrow$) et $|V(\mathbf{G})| \leq B$ par définition des bornes serrées, nous obtenons une contradiction.

Notons que par le même raisonnement, la famille $\mathcal{F}$ des graphes ayant un nombre de composantes fortement connexes borné par B avec B une borne serrée satisfait $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$.

Familles ne satisfaisant pas $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$. La famille $\mathcal{F}$ des graphes comportant au plus n sommets ne satisfait pas $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$ car il peut exister les graphes $\mathbf{G}, \mathbf{H}, \mathbf{H}' \in \mathcal{F}$ tels que $\mathbf{H}, \mathbf{H}' \subseteq \downarrow \mathbf{G}$ et $\mathbf{H} \cap \mathbf{H}' = \emptyset$. Comme $|V(\mathbf{H})| < |V(\mathbf{G})|$ et $|V(\mathbf{H}')| < |V(\mathbf{G})|$, $\mathbf{H}$ et $\mathbf{H}'$ appartiennent à $\mathcal{F}$, contredisant $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$. Pour les mêmes raisons, les familles des graphes comportant au plus n composantes fortement connexes ou avec un diamètre borné ne satisfont pas $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$.

La famille contenant tous les arbres dirigés et enracinés (dirigé depuis la racine) ne vérifie pas $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$ non plus. En effet, il existe une infinité d'arbres (possédant au moins trois feuilles et) contenant trois sous arbres sommet disjoint et clos par successeurs. Les trois sous arbres étant des arbres clos par successeurs dans un arbre non isomorphe, cette famille ne satisfait pas $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$.

Chapitre 5

k -Élection de leader

Dans ce chapitre, nous étudions le problème de la k -élection dans le modèle des participants inconnus.

Dans la section 5.1, nous présentons le problème k - $\mathcal{EL}\mathcal{EC}$. Dans la section 5.2, nous prouvons que la condition $\mathfrak{C}_{k-\mathcal{EL}\mathcal{EC}}$ est nécessaire en montrant qu'il n'existe pas d'algorithme $\mathcal{F}$ -restreint résolvant k - $\mathcal{EL}\mathcal{EC}$ pour toute famille $\mathcal{F}$ ne satisfaisant pas $\mathfrak{C}_{k-\mathcal{EL}\mathcal{EC}}$. Pour cela, nous étendons la condition $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$ pour admettre au plus k processus élus. Dans la section 5.3, nous prouvons que la condition $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$ est suffisante. Pour cela, nous prouvons que l'algorithme 4 présenté dans la section 4.3 résout le problème k - $\mathcal{EL}\mathcal{EC}$ dans les familles satisfaisant $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$. Dans la section 5.4, nous donnons des exemples de familles de graphes vérifiant $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$.

5.1 Description du problème k - $\mathcal{EL}\mathcal{EC}$

Le problème de la k -élection se définit comme suit,

Définition 5.1 (Spécification k - $\mathcal{EL}\mathcal{EC}$). Pour tout graphe $\mathbf{G}$, pour tout graphe $\mathbf{G}^{out}$, nous noterons $\mathbf{G}$ k - $\mathcal{EL}\mathcal{EC}$ $\mathbf{G}^{out}$ s'il existe au moins un et au plus k processus $\{v_1, \dots, v_k\} \subseteq V(\mathbf{G})$ tels que pour tout $1 \leq i \leq k$, $out_{\mathbf{G}^{out}}(v_i) = \text{élu}$ et pour tout processus $w \notin \{v_1, \dots, v_k\}$, $out_{\mathbf{G}^{out}}(w) = \text{non-élu}$.

Nous étudions ce problème car, il généralise le problème de l'élection. De plus, nous résolvons ce problème très simplement en adaptant les connaissances données aux processus et en réutilisant l'algorithme 4 d'élection.

5.2 Condition nécessaire $\mathfrak{C}_{k-\mathcal{EL}\mathcal{EC}}$

L'idée de la condition nécessaire est que l'algorithme 4 d'élection exécuté sur n'importe quel graphe $\mathbf{G}$ peut élire au plus un processus par exécution isolée. Cette idée de base nous a permis de construire la condition $\mathfrak{C}_{k-\mathcal{EL}\mathcal{EC}}$. Pour résoudre le problème k - $\mathcal{EL}\mathcal{EC}$ dans une famille $\mathcal{F}$, nous autorisons au plus k exécutions isolées et disjointes à se terminer avec chacune, un processus élu. Par conséquent, la condition $\mathfrak{C}_{k-\mathcal{EL}\mathcal{EC}}$ est une extension de $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$ pour k sous-parties disjointes. La condition $\mathfrak{C}_{k-\mathcal{EL}\mathcal{EC}}$ est définie ci-dessous,

Définition 5.2 ($\mathfrak{C}_{k-\mathcal{EL}\mathcal{EC}}$). Une famille satisfait $\mathfrak{C}_{k-\mathcal{EL}\mathcal{EC}}$ si pour tout graphe $\mathbf{G}, \mathbf{H}_1, \dots, \mathbf{H}_{k+1} \in \mathcal{F}$ tel que $\mathbf{H}_1, \dots, \mathbf{H}_{k+1} \subseteq \mathbf{G}$, alors il existe $1 \leq i < j \leq k+1$, $V(\mathbf{H}_i) \cap V(\mathbf{H}_j) \neq \emptyset$.

La figure 5.1 illustre cette condition. Nous prouvons maintenant que $\mathfrak{C}_{k-\mathcal{EL}\mathcal{EC}}$ est une condition nécessaire à l'existence d'un algorithme $\mathcal{F}$ -restreint pour le problème $k - \mathcal{EL}\mathcal{EC}$.

Lemme 5.3 (Condition nécessaire). *Pour toute famille $\mathcal{F}$ de graphes, s'il existe un algorithme $\mathcal{F}$ -universel pour le problème $k - \mathcal{EL}\mathcal{EC}$ alors $\mathcal{F}$ satisfait $\mathfrak{C}_{k-\mathcal{EL}\mathcal{EC}}$.*

Démonstration. Supposons par l'absurde et prenons un algorithme résolvant la k -élection pour une famille $\mathcal{F}$ ne vérifiant pas $\mathfrak{C}_{k-\mathcal{EL}\mathcal{EC}}$. Il existe donc un graphe $\mathbf{G} \in \mathcal{F}$ et un ensemble contenant $k + 1$ graphes $\{\mathbf{H}_1, \dots, \mathbf{H}_{k+1}\} \subseteq \mathcal{F}$ tel que $\mathbf{H}_i \in \mathcal{F}$, $\mathbf{H}_i \subseteq \downarrow \mathbf{G}$ et $V(\mathbf{H}_1) \cap \dots \cap V(\mathbf{H}_{k+1}) = \emptyset$.

Comme $\mathcal{A}$ est un algorithme $\mathcal{F}$ -restreint et comme pour tout $1 \leq i \leq k + 1$, $\mathbf{H}_i \in \mathcal{F}$, $\mathbf{H}_i \subseteq \downarrow \mathbf{G}$, il existe une exécution ρ_i de $\mathcal{A}$ sur $\mathbf{H}_i$ se terminant avec au moins un processus v_i élu, c.-à-d. $out_{\mathbf{H}_i}(v_i) = \text{élu}$. Comme pour tout $1 \leq i \neq j \leq k + 1$, $\mathbf{H}_i \cap \mathbf{H}_j = \emptyset$, par le lemme d'isolement (Lemme 3.5) appliqué à toute exécution ρ_i , il existe une exécution ρ de $\mathcal{A}$ dans $\mathbf{G}$ telle que pour tout $v \in V(\mathbf{H}_i)$, $out_{\mathbf{G}}(v) = out_{\mathbf{H}_i}(v)$. Ainsi, il existe une exécution de $\mathcal{A}$ sur $\mathbf{G}$ se terminant avec strictement plus de k processus élus. Nous prouvons ce lemme. $\square$

5.3 Condition suffisante

Les propriétés de l'algorithme concernant la stabilisation et la terminaison ont déjà été prouvées dans le chapitre 4. Par le lemme 4.11, nous sommes assurés qu'il y aura toujours au moins un processus élu. Il ne nous reste donc plus qu'à prouver que dans toute exécution, il existe toujours au plus k processus élus dans le réseau.

Par le lemme 3.5, dit d'isolement, appliqué aux familles satisfaisant $\mathfrak{C}_{k-\mathcal{EL}\mathcal{EC}}$, nous savons qu'il y aura au plus k exécutions isolées disjointes. Comme dans chacune de ses exécutions, par les lemmes 4.11 et 4.12, il y a au plus un processus élu (algorithme résolvant $\mathcal{EL}\mathcal{EC}$), il y a au plus de k processus élus en fin d'exécution et au moins un. Nous obtenons le lemme suivant.

Lemme 5.4. *Pour toute famille $\mathcal{F}$ satisfaisant $\mathfrak{C}_{k-\mathcal{EL}\mathcal{EC}}$, pour toute exécution de l'algorithme 4 sur un graphe $\mathbf{G} \in \mathcal{F}$, il y aura au moins un et au plus k processus ayant décidé élu.*

Nous avons prouvé que $\mathfrak{C}_{k-\mathcal{EL}\mathcal{EC}}$ est nécessairement satisfaite par toute famille $\mathcal{F}$ admettant un algorithme $\mathcal{F}$ -restreint pour $k - \mathcal{EL}\mathcal{EC}$ (Lemme 5.3). De plus, nous venons de prouver que, pour toute famille $\mathcal{F}$ satisfaisant $\mathfrak{C}_{k-\mathcal{EL}\mathcal{EC}}$, il existe un algorithme $\mathcal{F}$ -restreint pour $k - \mathcal{EL}\mathcal{EC}$. Nous obtenons donc le théorème suivant,

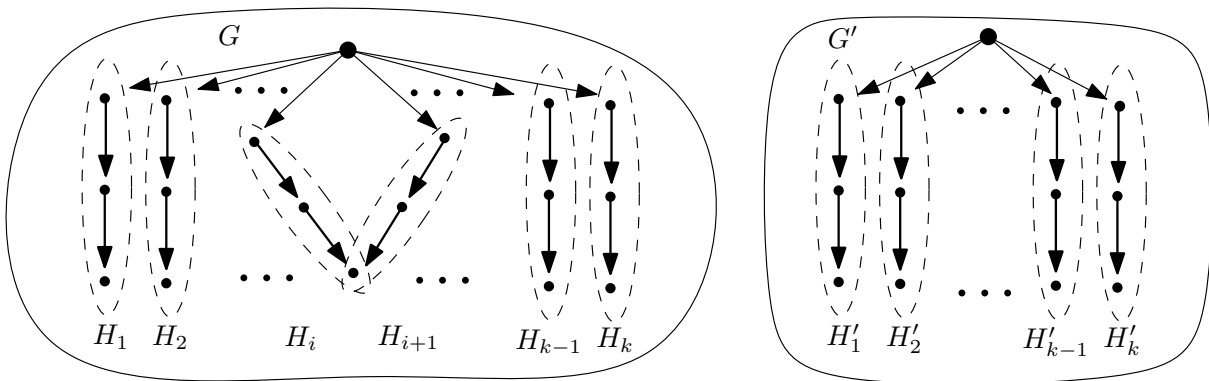


FIGURE 5.1 – Illustration de la condition $\mathfrak{C}_{k-\mathcal{EL}\mathcal{EC}}$. Dans cet exemple, la famille $\mathcal{F} = \{G, H_1, \dots, H_k\}$ à gauche vérifie $\mathfrak{C}_{k-\mathcal{EL}\mathcal{EC}}$ car $V(H_i \cap H_{i+1}) \neq \emptyset$. La famille $\mathcal{F}' = \{G', H'_1, \dots, H'_k\}$ à droite ne vérifie pas $\mathfrak{C}_{k-\mathcal{EL}\mathcal{EC}}$ pour tout $1 \leq i < j \leq k$, $V(H'_i \cap H'_j) = \emptyset$, $H'_i \subseteq \downarrow G'$ et $H'_i \in \mathcal{F}'$.

Théorème 5.1. *Il existe un algorithme $\mathcal{F}$ -restreint pour k - $\mathcal{EL}\mathcal{EC}$ si et seulement si $\mathcal{F}$ satisfait $\mathfrak{C}_{k-\mathcal{EL}\mathcal{EC}}$.*

5.4 Exemples

Nous appellerons une borne k -serrée B_k pour un ensemble S une borne sur le nombre d'éléments de S telle que $|S| \leq B \leq k \times |S|$.

Familles satisfaisant $\mathfrak{C}_{k-\mathcal{EL}\mathcal{EC}}$. Premièrement, par définition, toute famille satisfaisant $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}}$ satisfera $\mathfrak{C}_{k-\mathcal{EL}\mathcal{EC}}$. La famille $\mathcal{F}$ des graphes avec un nombre de sommets borné par B_k (borne k -serrée) satisfait $\mathfrak{C}_{k-\mathcal{EL}\mathcal{EC}}$. En effet, par l'absurde, supposons une famille $\mathcal{F}$ satisfaisant $\mathfrak{C}_{k-\mathcal{EL}\mathcal{EC}}$ telle que pour tout graphe $\mathbf{G} \in \mathcal{F}$ et pour tout ensemble de graphes $\mathbf{H}_1, \dots, \mathbf{H}_{k+1} \in \mathcal{F}$ tel que $\mathbf{H}_1, \dots, \mathbf{H}_{k+1} \subseteq \downarrow \mathbf{G}$, et pour tout $1 \leq i < j \leq k+1$, $\mathbf{H}_i \cap \mathbf{H}_j = \emptyset$. Comme, par définition, $|V(\mathbf{H}_i)| \geq \frac{B_k}{k}$, nous obtenons par hypothèse que

$$|V(\mathbf{G})| \geq \sum_{i=1}^{k+1} |V(\mathbf{H}_i)| \geq (k+1) \frac{B_k}{k} > B_k$$

Or, comme $\mathbf{G} \in \mathcal{F}$ et B_k est une borne serrée sur la taille des graphes de la famille $\mathcal{F}$, $|V(\mathbf{G})| \leq B_k$. Nous obtenons une contradiction.

Conclusion des chapitres 2, 3, 4 et 5

Dans ces chapitres, nous avons étudié l'impact des connaissances partielles sur la calculabilité distribuée de quatre problèmes dans le modèle des participants inconnus [CSS04] : cartographie des sommets accessibles, cartographie complète du réseau, élection et k -élection. Pour chacun de ces problèmes, nous avons caractérisé les connaissances partielles nécessaires et suffisantes à apporter aux processus pour le résoudre au travers une condition sur les familles de graphes. L'approche algébrique utilisée s'avère adaptée à l'étude de nouveaux modèles en nous permettant de caractériser la calculabilité d'un problème donné en fonction des connaissances données (vues comme des familles de graphes). Par exemple, nous avons montré que le problème de cartographie des sommets accessibles ne nécessite aucune connaissance pour être résolu. À l'inverse, le problème de cartographie complète ou de l'élection peut être résolu dans les familles de graphes satisfaisant $\mathfrak{C}_{\mathcal{CARTO}}$. Pour obtenir ces résultats, nous avons montré que si le problème $\mathcal{CARTO}$ pouvait être résolu dans une famille alors cette famille satisfaisait notre condition $\mathfrak{C}_{\mathcal{CARTO}}$. Pour montrer que la connaissance est suffisante, nous proposons un algorithme $\mathcal{F}$ -restreint résolvant $\mathcal{CARTO}$ dans toute famille satisfaisant $\mathfrak{C}_{\mathcal{CARTO}}$.

Une particularité du modèle est que les morphismes, la relation principalement utilisée pour décrire les impossibilités de manière algébrique, ne sont pas adaptés aux modèles. La relation appropriée au modèle des participants inconnus est une relation d'inclusion de graphes : les sous-graphes clos par successeurs. Grâce à cette relation, nous appliquer l'approche algébrique en développant un lemme d'isolement, à la Angluin, caractérisant exactement les exécutions problématiques pouvant apparaître dans ce modèle.

De plus, cette approche peut nous permettre de généraliser ces résultats pour caractériser la calculabilité d'un problème distribué arbitraire $\mathcal{S}$ en fonction d'une famille de graphes $\mathcal{F}$ arbitraire. À l'avenir, nous aimerions proposer une condition $\mathfrak{C}_{\mathcal{UNTV}}$ et un méta-algorithme $(\mathcal{S}, \mathcal{F})$ -universel réalisant $\mathcal{S}$ sur $\mathcal{F}$ si le couple $(\mathcal{S}, \mathcal{F})$ satisfait la condition $\mathfrak{C}_{\mathcal{UNTV}}$. Dans le chapitre

suivant, nous étendons ce modèle aux réseaux anonymes. En suivant la même méthodologie, nous généralisons nos résultats et caractérisons les connaissances nécessaires permettant de résoudre le problème de l'élection dans des familles de graphes de taille bornée.

Chapitre 6

Élection dans le modèle des participants inconnus anonymes

Les résultats présentés dans cette partie sont une extension de [CGN14]. Une version journal regroupant l'intégralité des résultats des chapitres 2, 3, 4, 5 et 6 est en cours de rédaction.

Introduction

Dans le modèle anonyme à passage de messages, les processus ne possèdent aucune identité pouvant être utilisée pour se distinguer des autres processus du réseau. Dans de tels réseaux, les processus ne peuvent plus se distinguer de façon unique et donc, ne peuvent plus signer leurs messages par exemple. Cependant, les processus peuvent quand même s'identifier (autant que cela est possible) grâce aux informations locales à leur disposition : étiquetage des ports, degré. Nous verrons que dans certains réseaux, ce type d'identification n'assure pas l'unicité des identités pouvant être prises par les processus.

Nous nous intéresserons à une variante du modèle anonyme, dit modèle avec homonymes, où les processus sont dotés d'une "identité" mais chaque identité peut être prise par un nombre arbitraire de processus. Notons que le modèle avec homonymes s'étend de façon naturelle au modèle des participants inconnus avec identités, car seule l'unicité des identités diffère.

L'étude de la calculabilité des réseaux anonymes/avec homonymes est intéressante pour, exécuter un algorithme dans de grands réseaux, dans des réseaux requérant une certaine confidentialité des processus ou ne divulguant pas les identités pour des raisons de sécurité. Dans des réseaux vastes et denses, l'unicité globale des identités peut être non seulement coûteuse mais surtout difficile (voir impossible) à maintenir et à vérifier. Les protocoles distribués ne nécessitant aucune identification a priori des processus sont par définition, plus robuste aux attaques par usurpation d'identité et aux différentes pannes occasionnelles pouvant impacter les identités des processus : redémarrage d'une machine, erreur de configuration humaine, corruption de mémoire, ...

État de l'art.

La correction des algorithmes dans un système anonyme est critique. Dana Angluin [Ang80] a prouvé que dans certains réseaux (et notamment tous les cycles étiquetés de façon symétrique), il était impossible de résoudre le problème de l'élection, i.e., il n'existe pas d'algorithme résolvant le problème de l'élection dans l'ensemble des réseaux. De plus, elle montre aussi que même en concevant des algorithmes spécifiques pour les réseaux problématiques, le problème reste impos-

sible à résoudre. Ce résultat s'étend à de nombreux problèmes fondamentaux tels l'énumération, la cartographie du réseau ou le consensus et impacte fortement la calculabilité distribuée de ces systèmes.

La détection locale de la terminaison est aussi problématique dans un système anonyme. Cette propriété non triviale est indépendante de la correction de l'algorithme et des identités prises par les processus. En générale, elle est d'ailleurs impossible à obtenir sans connaissance partielle sur le réseau comme une borne sur la taille du réseau ou son diamètre.

Il est évident que pour résoudre un problème distribué dans un ensemble de réseaux, il est nécessaire d'identifier et interdire les réseaux où le problème est impossible à résoudre.

L'approche algébrique permet de caractériser l'impact précis de connaissances spécifiques sur la calculabilité distribuée des réseaux anonymes. Dans [YK96c], Yamashita et Kameda caractérisent, pour certains problèmes distribués et certaines connaissances spécifiques, les classes de graphes où le problème peut être résolu si les processus possèdent la connaissance donnée. Les problèmes étudiés sont des problèmes fondamentaux en algorithmique distribuée : élection de leader, élection d'une arête, construction d'arbre couvrant et cartographie du réseau. Les connaissances étudiées sont "pas de connaissance", une borne supérieure sur le nombre de processus, le nombre exact de processus et la carte du réseau (une carte non étiquetée et sans sommet distingué).

Par la suite, Boldi et Vigna [BV99] proposent une caractérisation générale des relations calculables en présence de connaissances arbitraires dans un certain nombre de modèles distribués asynchrones et anonymes. Ils caractérisent, pour un problème donné, les connaissances nécessaires et suffisantes permettant aux processus de résoudre le problème sur l'ensemble des graphes considérés. Pour un problème donné et une connaissance donnée, leur technique de preuve permet soit d'obtenir un algorithme soit une preuve d'impossibilité.

Dans [CGM12], une première caractérisation des connaissances arbitraires permettant de résoudre l'élection dans un modèle à passage de messages anonyme est proposée. Pour cela, les auteurs utilisent une technique de quasi-simulation des exécutions introduite dans [MMW97] : les quasi-revêtements.

Notons que le modèle des participants inconnus est différent de tous les autres modèles étudiés cités ci-dessus, car la difficulté provient des exécutions asynchrones alors que dans [YK96c, BV99, CGM12], la difficulté provient des exécutions synchrones.

Dans ce chapitre, nous continuons l'étude du modèle des participants inconnus présentée dans les chapitres précédents. Pour ce faire, nous allons utiliser la même approche algébrique que précédemment pour caractériser les connaissances nécessaires permettant aux processus de résoudre le problème de l'élection dans la version anonyme du modèle des participants inconnus.

En premier lieu, nous montrons une condition nécessairement satisfaite par toute famille de graphes $\mathcal{F}$ admettant un algorithme d'élection $\mathcal{F}$ -restreint. Cette condition s'appuie sur les résultats d'impossibilité de Angluin et des résultats d'impossibilité spécifiques aux modèles des participants inconnus présentés dans les chapitres précédents.

Puis, nous montrons que cette condition est suffisante pour résoudre le problème de l'élection dans des familles de graphes de taille bornée. Pour cela, étant donné un entier B , nous proposons un algorithme d'élection $\mathcal{F}$ -restreint, pour toute famille $\mathcal{F}$ satisfaisant la condition et possédant uniquement des graphes possédant moins de B sommets.

Pour réaliser l'élection dans ce modèle, nous adapterons l'algorithme gSSP [CGM12] au modèle des participants inconnus. Cet algorithme d'élection est composé de deux algorithmes fondamentaux : l'algorithme d'énumération de Mazurkiewicz [Maz97] et l'algorithme de Szymanski, Shy et Prywes [SSP85] permettant d'obtenir la détection locale de la terminaison.

Le problème de l'énumération réalisé par l'algorithme de Mazurkiewicz consiste à attribuer à

tous les processus d'un réseau G possédant n sommets, une identité unique prise dans l'ensemble des entiers de 1 à n . Cependant, celui-ci ne garantit pas la détection locale de la terminaison. C'est pourquoi l'algorithme SSP [SSP85] est utilisé pour permettre la détection d'un état suffisamment stable du système pour décider, i.e., un état où les processus seront en accord sur l'identité prise par chacun. Cet accord est nécessaire pour que les processus décident globalement de façon cohérente. Par exemple, les processus pourront élire le processus ayant l'identité numéro 1. Cependant, l'algorithme SSP nécessite certaines connaissances globales telle une borne sur la taille ou le diamètre du réseau pour terminer dans le cas général.

Plan.

Dans la section 6.1, nous introduisons les notations et les définitions nécessaires pour la suite de ce chapitre. Dans la section 6.2, nous introduisons le modèle des participants inconnus dans les réseaux anonymes et nous formalisons les états internes des processus au sein d'un tel réseau. Dans la section 6.4, nous présentons une Condition $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}-\mathcal{AN}\mathcal{O}}$ et montrons qu'elle est nécessairement satisfaite par toute famille de graphes $\mathcal{F}$ admettant un algorithme $\mathcal{F}$ -restreint d'élection dans ce modèle. L'idée de cette condition est de caractériser séparément les impossibilités liées au modèle des participants inconnus grâce à un lemme d'isolement (introduit dans le chapitre 3) et les impossibilités liées à l'anonymat des processus avec un lemme de revêtement des exécutions "à la Angluin". Dans la section 6.5, nous présentons un algorithme $\mathcal{F}$ -restreint d'élection pour toute famille contenant des graphes de taille bornée et satisfaisant notre Condition $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}-\mathcal{AN}\mathcal{O}}$. Dans la section 6.6, nous montrerons sa correction.

6.1 Définitions

Définition 6.1. Étant donné un graphe dirigé $\mathbf{G}$ doté d'un étiquetage des ports, nous noterons $a_{u,v,p,q} \in A(\mathbf{G})$ un arc reliant un sommet u à un sommet v (possiblement égaux) étiqueté (p, q) .

Les graphes dirigés considérés dans ce chapitre peuvent posséder des arcs multiples et des boucles simples.

Définition 6.2 (Arcs multiples et boucle simple). Étant donnés un graphe dirigé $\mathbf{G}$ et deux sommets $u, v \in V(\mathbf{G})$, des arcs multiples sont des arcs de la forme $a_{u,v,p,q}, a_{u,v,p',q'} \in A(\mathbf{G})$ avec $p \neq p'$ et $q \neq q'$. Une boucle simple est un arc de la forme $a_{u,u,p,q}$.

Les arcs multiples (ici, double) et les boucles simples sont illustrés dans la figure 6.1. Notons que les définitions ci-dessus permettent d'obtenir des graphes dirigés possédant des boucles simples multiples.

Étant donné un arc $a_{u,v,p,q} \in A(\mathbf{G})$, le sommet u est la *source* de a et sera notée $s_G(a)$. Le sommet v est la *destination* de a et sera notée $t_G(a)$. Si le contexte nous le permet, nous les noterons simplement $t(a)$ et $s(a)$. Dans ce chapitre, nous considérerons uniquement des graphes dirigés simples.

6.1.1 Fonction simpliciale, Morphisme et Revêtement dirigé

Les définitions présentées ci-dessous prennent en compte des graphes dirigés non simples car nous aurons besoin de les manipuler par la suite. Notons qu'en considérant des graphes dirigés simples, les définitions tiennent toujours.

Remarque 6.3. Notons que pour tout graphe dirigé symétrique, pour tout arc $a_{u,v,p,q} \in A_{u,v}(G)$, $a_{v,u,q,p} = \sigma(a_{u,v,p,q})$

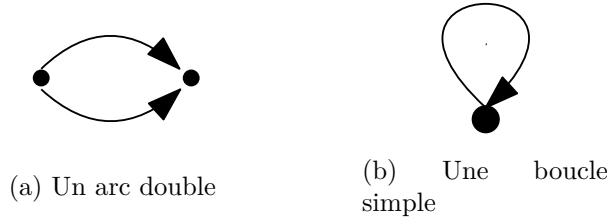


FIGURE 6.1 – Illustration d'une boucle simple et d'un arc multiple, ici, un arc double.

Les morphismes, introduits ci-dessous, sont des fonctions préservant la structure des graphes. En théorie des graphes, un morphisme est appelé une *fonction simpliciale rigide* [HKR14].

Définition 6.4 (Morphisme dirigé). Un morphisme $\varphi : G \rightarrow H$ entre deux graphes dirigés G et H est une fonction de $V(G)$ dans $V(H)$ et de $A(G)$ dans $A(H)$ tel que pour tout arc $a \in A(G)$, $s_H(\varphi(a)) = \varphi(s_G(a))$ et $t_H(\varphi(a)) = \varphi(t_G(a))$.

Définition 6.5 (Isomorphisme). Un isomorphisme entre deux digraphes $G, H \in \mathcal{G}^\delta$ est un morphisme bijectif de G dans H .

Un isomorphisme sera noté par le symbole $\simeq$.

Les revêtements dirigés, introduits ci-dessous, sont des morphismes dirigés préservant, non seulement la structure, mais aussi le voisinage de chaque sommet.

Définition 6.6 (Revêtement dirigé). Soit deux digraphes $G, H \in \mathcal{G}^\delta$. Un revêtement dirigé est un morphisme $\varphi : G \rightarrow H$ tel que pour tout arc $a' \in A(H)$ et pour tout sommet $u \in \varphi^{-1}(t(a'))$ (resp. $u \in \varphi^{-1}(s(a'))$), il existe un unique arc $a \in A(G)$ tel que $t(a) = u$ (resp. $s(a) = u$) et $\varphi(a) = a'$.

Définition 6.7 (Revêtement dirigé symétrique). Soit deux digraphes $G, H \in \mathcal{G}^\delta$. Un revêtement dirigé symétrique est un revêtement dirigé $\varphi : G \rightarrow H$ préservant la fonction σ , i.e., tel que pour tout arc $a \in A(G)$, $\sigma(\varphi(a)) = \varphi(\sigma(a))$.

Remarque 6.8. Si un graphe dirigé G est un revêtement dirigé symétrique d'un graphe H alors G est un revêtement dirigé de H .

Définition 6.9. [Ang80] Soit deux digraphes $G, H \in \mathcal{G}^\delta$, un revêtement dirigé $\varphi : G \rightarrow H$ est un revêtement à n -feuilles si pour tout élément $x \in V(H)$ ou $x \in A(H)$, $\varphi^{-1}(x) = n$.

Proposition 6.10. Soit deux digraphes $G, H \in \mathcal{G}^\delta$. S'il existe un revêtement $\varphi : G \rightarrow H$ alors il existe un entier n tel que φ est un revêtement à n -feuilles.

Proposition 6.11. Pour tout digraphe $G, H \in \mathcal{G}^\delta$, si G est un revêtement dirigé de H et $|V(G)| = |V(H)|$ alors G et H sont isomorphes.

La relation de revêtement étant transitive, elle induit un ordre partiel sur les graphes étiquetés dirigés. Nous définissons maintenant les éléments minimaux et maximaux pour cette relation.

Définition 6.12 (Base minimale). Un digraphe $G \in \mathcal{G}^\delta$ est minimal pour la relation de revêtement si pour tout digraphe $H \in \mathcal{G}^\delta$, tel que G est un revêtement de H , G et H sont isomorphes.

Définition 6.13 (Revêtement universel). Un digraphe $G \in \mathcal{G}^\delta$ (possiblement infini) est maximal pour la relation de revêtement si, pour tout digraphe $H \in \mathcal{G}^\delta$ tel que H est un revêtement de G , G et H sont isomorphes.

Un élément maximal est appelé un *revêtement universel*. Dans la proposition suivante, il est montré que tout digraphe possède un revêtement universel unique.

Proposition 6.14 ([Hat02]). *Pour tout digraphe G , il existe un unique digraphe $\hat{G}$ (possiblement infini) tel que pour tout digraphe G' , si G' est un revêtement de G alors $\hat{G}$ est revêtement de G' .*

Définition 6.15 (Construction des revêtements de Reidmeister [Rei32], 1932). Étant donné un graphe dirigé G et un entier n , un revêtement dirigé à q feuillets de G est obtenu comme suit.

1. Calculer un arbre couvrant T de G (pas forcément dirigé),
2. Créer q copies de cet arbre T ,
3. Connecter les s copies de l'arbre avec les arcs de $E(G)$ n'appartenant pas à l'arbre T selon un ensemble de permutations pris dans l'ensemble $[1 \dots q]$ associé à chaque arc exclu de l'arbre T .

La figure 6.2 illustre cette construction.

Remarque 6.16. Notons que cette technique peut amener à construire des revêtements non connexes et que certaines permutations de l'ensemble $[1, q]$ aboutissent à un même revêtement (à isomorphisme près).

Pour énumérer tous les revêtements dirigés à q feuillets d'un graphe G donné, il est suffisant d'exécuter l'algorithme ci-dessus pour toute permutation de l'ensemble $[1 \dots q]$. L'ensemble des permutations d'un ensemble fini étant fini, cette méthode aboutira. Cette construction nous autorise à construire des revêtements finis arbitrairement grands (possédant un nombre de feuillets arbitraire) à partir du moment où il existe un arc exclu de l'arbre couvrant, i.e., où le graphe considéré n'est pas un arbre. Nous obtenons la caractérisation suivante.

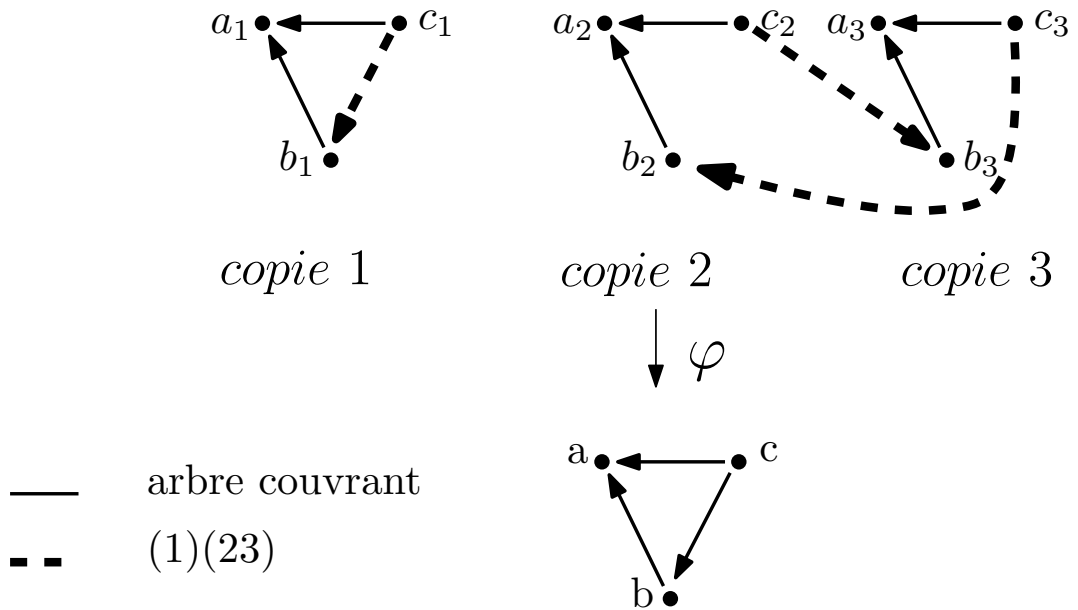


FIGURE 6.2 – Construction de Reidmeister pour les revêtements dirigés.

Proposition 6.17 (Corollaire du lemme 6.8.3¹ de [GR01]). *Un graphe G simple est isomorphe à son revêtement universel $\hat{G}$ si et seulement si G est un arbre.*

Corollaire 6.18. *Un graphe G possède un revêtement universel $\hat{G}$ infini si et seulement si G n'est pas un arbre.*

Comme nous travaillerons avec des graphes étiquetés, nous définissons maintenant une généralisation des revêtements préservant les étiquetages des graphes.

Définition 6.19 (Revêtement dirigé de graphes étiquetés). Soit deux digraphes $\mathbf{G} = (G, \lambda_G)$ et $\mathbf{H} = (H, \lambda_H)$ de $\mathcal{G}^\delta$. Un revêtement dirigé $\varphi : \mathbf{G} \rightarrow \mathbf{H}$ entre les deux graphes étiquetés $\mathbf{G}$ et $\mathbf{H}$ est un revêtement de graphes dirigés $\varphi : G \rightarrow H$ tel que pour tout $v \in V(G)$, $\lambda_{\mathbf{G}}(v) = \lambda_{\mathbf{H}}(\varphi(v))$.

De plus, comme nous travaillerons exclusivement avec des graphes dirigés, un revêtement dirigé sera simplement noté *revêtement*.

6.2 Modèle des participants inconnus et anonymes

Nous nous plaçons dans le modèle à passage de messages des participants inconnus introduit dans le chapitre 1. Dans le contexte de réseaux anonymes, les identités ne garantissent plus une identification unique des processus (plusieurs processus peuvent partager une même identité). Nous nous appuyons donc sur l'étiquetage local des processus, comme l'étiquetage des ports du réseau, pour discerner les processus entre eux autant que cela est possible.

6.2.1 Codage de l'état des processus

Le codage de l'état des processus présenté ci-dessous est une adaptation de celui proposé dans le chapitre 1 pour le modèle anonyme. La principale différence est que, dorénavant, la liste de contacts `CONTACTS` d'un processus ne contient plus une liste d'identités mais une liste de numéros de port. En effet, dans le contexte des réseaux anonymes, ce sont les seules informations disponibles permettant aux processus d'identifier leurs voisins dans le réseau.

Définition 6.20. Dans le modèle des participants inconnus et anonymes, pour tout réseau $\mathbf{G}$, l'état d'un processus v est codé par une étiquette $\lambda_{\mathbf{G}}(v)$ présentée ci-dessous.

$$\lambda_{\mathbf{G}}(v) = \langle in(v), \text{CONTACTS}(v), out(v), MEM(v) \rangle$$

avec

- $in(v) \subset \Lambda$ est un ensemble de paramètres propre à chaque processus. Cet ensemble peut contenir par exemple une identité pour le sommet v . Λ est l'ensemble de toutes les étiquettes pouvant être prises par les processus.
- $\text{CONTACTS}(v) \subset \mathbb{N}$ est un sous-ensemble *arbitraire* de numéros de port de v .
- $out(v)$ est la variable dans laquelle le processus écrira sa décision.
- $MEM(v)$ est le reste de la mémoire allouée au processus.

Remarque 6.21. Nous noterons $\text{CONTACTS}(v)^i$ et $out(v)^i$, le contenu des variables $\text{CONTACTS}(v)$ et $out(v)$ au début d'une étape i d'une exécution.

Remarque 6.22. Initialement, pour tout processus $v \in V(\mathbf{G})$, $out^0(v) = \perp$ et $\text{CONTACTS}(v)^0 = \{\delta_v(w) \mid \forall vw \in A(\mathbf{G})\}$.

1. Le Lemme 6.8.3 de [GR01] est prouvé pour des graphes non dirigés mais peut être étendu aux graphes dirigés comme illustré dans la figure 6.2.

6.3 Problème de l'élection

Le problème de l'élection étudié dans cette partie est le même que celui introduit dans la section 4.1 du chapitre 4. La spécification est rappelée ci-dessous.

Rappel (Spécification $\mathcal{EL}\mathcal{EC}$). Pour tout graphe $\mathbf{G}$, pour tout graphe $\mathbf{G}^{out}$ obtenu suite à une exécution d'un algorithme $\mathcal{A}$ sur $\mathbf{G}$, il existe un unique processus $v \in V(\mathbf{G})$ tel que $out_{\mathbf{G}^{out}}(v) = \text{élu}$ et pour tout processus $w \neq v$, $out_{\mathbf{G}^{out}}(w) = \text{non-élu}$.

6.4 Premiers résultats d'impossibilité et conditions nécessaires

Dans cette section, nous montrons qu'il n'existe pas d'algorithme universel d'élection pour ce modèle. Notre preuve utilise les techniques standards de relèvement des exécutions introduites dans [Ang80]. Par la suite, un graphe étiqueté $\mathbf{H} = (H, \lambda_{\mathbf{H}})$ doté d'un ensemble $B_{\mathbf{H}}$ de messages en transits sera noté $\underline{\mathbf{H}} = (H, \lambda_H, B_H)$. Premièrement, nous étendons la relation de revêtement dirigé et de clôture par successeurs (introduite dans la définition 1.6 du chapitre 1) aux graphes dotés de boîtes de messages en transit.

Définition 6.23. Soit $\underline{\mathbf{G}} = (G, \lambda_G, B_G)$ et $\underline{\mathbf{H}} = (H, \lambda_H, B_H)$, deux graphes étiquetés dotés de messages en transit. Nous noterons $\underline{\mathbf{H}} \subseteq \downarrow \underline{\mathbf{G}}$ si et seulement si $\mathbf{H} \subseteq \downarrow \mathbf{G}$ et si pour tout arc $a \in A(H)$, $B_{\mathbf{H}}(a) = B_{\mathbf{G}}(a)$.

Définition 6.24. Soit $\underline{\mathbf{G}} = (G, \lambda_G, B_G)$ et $\underline{\mathbf{H}} = (H, \lambda_H, B_H)$, deux graphes étiquetés dotés de boîtes de messages en transit. Un morphisme $\varphi : \mathbf{G} \rightarrow \mathbf{H}$ est un revêtement pour (G, λ_G, B_G) et (H, λ_H, B_H) si et seulement si $\varphi : \mathbf{G} \rightarrow \mathbf{H}$ est un revêtement et si pour tout arc $a \in A(\mathbf{G})$, $B_{\mathbf{G}}(a) = B_{\mathbf{H}}(\varphi(a))$.

6.4.1 Lemme de relèvement

Définition 6.25. Pour tout graphe $\mathbf{G}$, et tout processus $v \in V(G)$, notons $Br_{\mathbf{G}}(v) = \{B_{\mathbf{G}}(a) \mid a \in A(\mathbf{G}) \wedge t(a) = v\}$ les tampons permettant à v de recevoir des messages et notons $Be_{\mathbf{G}}(v) = \{B_{\mathbf{G}}(a) \mid a \in A(\mathbf{G}) \wedge s(a) = v\}$ les tampons permettant à v d'en envoyer.

Lemme 6.26 (Une étape). *Étant donné un algorithme $\mathcal{A}$, deux digraphes $\mathbf{G}, \mathbf{H}$, et une étape j d'une exécution de $\mathcal{A}$ sur $\mathbf{H}$, pour toute étape i d'une exécution de $\mathcal{A}$ sur $\mathbf{G}$ tel que $\varphi : (G, \lambda_G, B_G)^i \rightarrow (H, \lambda_H, B_H)^j$ est un revêtement à n -feuilles, alors il existe une succession de n étapes sur $\mathbf{G}$ telle que $\varphi : (G, \lambda_G, B_G)^{i+n} \rightarrow (H, \lambda_H, B_H)^{j+1}$ est un revêtement.*

Démonstration. Considérons un algorithme $\mathcal{A}$ exécuté sur deux digraphes $\mathbf{G}$ et $\mathbf{H}$. Soit i une étape de l'exécution sur $\mathbf{G}$ et j une étape de l'exécution sur $\mathbf{H}$ telle que $\varphi : (G, \lambda_G, B_G)^i \rightarrow (H, \lambda_H, B_H)^j$ est un revêtement à n -feuilles. Soit $\tilde{v} \in V(\mathbf{H})$ le processus actif lors de l'étape j de l'exécution dans $\mathbf{H}$.

Observons que par la proposition 6.10, il existe un entier $n \in \mathbb{N}$ tel que $|\varphi^{-1}(\tilde{v})| = n$. Notons $\varphi^{-1}(\tilde{v}) = \{v_1, \dots, v_n\}$. Sans perte de généralité dans un modèle asynchrone, il existe une continuation de l'exécution sur $\mathbf{G}$ telle que pour tout $1 \leq h \leq n$, le processus $v_h \in \varphi^{-1}(\tilde{v})$ est actif lors de l'étape $i + h - 1$.

Tout d'abord, nous montrons que $(\lambda(v_1), \text{CONTACTS}(v_1))_{\mathbf{G}}^{i+1} = (\lambda(\tilde{v}), \text{CONTACTS}(\tilde{v}))_{\mathbf{H}}^{j+1}$ et que $(Be(v_1), Br(v_1))_{\mathbf{G}}^{i+1} = (Be(\tilde{v}), Br(\tilde{v}))_{\mathbf{H}}^{j+1}$.

Puis nous montrerons que pour tout $1 \leq h \leq n$, si $\tilde{v}$ envoie (resp. reçoit) un message lors de l'étape j alors $Be_{\mathbf{G}}^{i+h-1}(v_h) = Be_{\mathbf{H}}^j(\tilde{v})$ (resp. $Br_{\mathbf{G}}^{i+h-1}(v_h) = Br_{\mathbf{H}}^j(\tilde{v})$).

Initialement, par définition des revêtements, $(\lambda(v_1), \text{CONTACTS}(v_1))_{\mathbf{G}}^i = (\lambda(\tilde{v}), \text{CONTACTS}(\tilde{v}))_{\mathbf{H}}^j$ et $(Be(v_1), Br(v_1))_{\mathbf{G}}^i = (Be(\tilde{v}), Br(\tilde{v}))_{\mathbf{H}}^j$.

Lors de l'étape j , si $\tilde{v}$ change son état interne alors comme $\lambda_{\mathbf{G}}^i(v_1) = \lambda_{\mathbf{H}}^j(\tilde{v})$ et comme v_1 et $\tilde{v}$ exécutent le même algorithme, v_1 change aussi son état interne à partir de mêmes données que $\tilde{v}$ et donc $\lambda_{\mathbf{G}}^{i+1}(v_1) = \lambda_{\mathbf{H}}^{j+1}(\tilde{v})$.

Si $\lambda_{\mathbf{H}}^j(\tilde{v}) \neq \lambda_{\mathbf{H}}^{j+1}(\tilde{v})$ alors, le processus $\tilde{v}$ envoie son nouvel état à tous ses voisins. Comme nous venons de montrer que $\lambda_{\mathbf{G}}^{i+1}(v_1) = \lambda_{\mathbf{H}}^{j+1}(\tilde{v})$ alors v_1 a aussi changé son état interne et envoie donc son nouvel état à tous ses voisins. Comme $\text{CONTACTS}_{\mathbf{G}}^i(v_1) = \text{CONTACTS}_{\mathbf{H}}^j(\tilde{v})$, pour tout processus $\tilde{w} \in V(H)$ tel que $\delta_{\tilde{v}}(\tilde{w}) \in \text{CONTACTS}_{\mathbf{H}}^j(\tilde{v})$, il existe un processus $w \in V(\mathbf{G})$ et un arc $v_1 w \in A(\mathbf{G}^i)$ tel que $\varphi(v_1 w) = \tilde{v} \tilde{w} \in A(\mathbf{H}^j)$ et $\delta_{v_1}(w) \in \text{CONTACTS}_{\mathbf{G}}^i(v_1)$. Ainsi, pour tout message m envoyé par $\tilde{v}$ à un voisin $\tilde{w}$ dans $\mathbf{H}$ via le port $\delta_{\tilde{v}}(\tilde{w})$, v_1 envoie le même message m à w via le port $\delta_{v_1}(w)$. Nous obtenons que si $\tilde{v}$ envoie un message alors $(\lambda(v_1), \text{CONTACTS}(v_1))_{\mathbf{G}}^{i+1} = (\lambda(\tilde{v}), \text{CONTACTS}(\tilde{v}))_{\mathbf{H}}^{j+1}$ et $Be_{\mathbf{G}}^{i+1}(v_1) = Be_{\mathbf{H}}^{j+1}(\tilde{v})$.

Si, lors de l'étape j , $\tilde{v}$ reçoit un message m de la part d'un voisin $\tilde{u}$ alors par définition des revêtements, il existe un processus $u \in V(\mathbf{G})$ tel que $uv_1 \in A(\mathbf{G})$, $\varphi(u) = \tilde{u}$ et donc $\delta_u(v_1) = \delta_{\tilde{u}}(\tilde{v})$.

Si $\tilde{u}$ est "connu" (resp. "inconnu") par $\tilde{v}$ alors $\delta_{\tilde{v}}(\tilde{u}) \in \text{CONTACTS}_{\mathbf{H}}^j(\tilde{v})$ (resp. $\delta_{\tilde{v}}(\tilde{u}) \notin \text{CONTACTS}_{\mathbf{H}}^j(\tilde{v})$). Comme $\text{CONTACTS}_{\mathbf{G}}^i(v_1) = \text{CONTACTS}_{\mathbf{H}}^j(\tilde{v})$, il existe un processus $u \in V(\mathbf{G}^i)$ et un arc $uv_1 \in A(\mathbf{G}^i)$ tel que $\delta_{v_1}(u) \in \text{CONTACTS}_{\mathbf{G}}^i(v_1)$ (resp. $\delta_{v_1}(u) \notin \text{CONTACTS}_{\mathbf{G}}^i(v_1)$). Le processus u est donc "connu" (resp. "inconnu") de v_1 . Par conséquent, $\text{CONTACTS}_{\mathbf{G}}^{i+1}(v_1) = \text{CONTACTS}_{\mathbf{H}}^{j+1}(\tilde{v})$.

Comme $Br_{\mathbf{G}}^i(v_1) = Br_{\mathbf{H}}^j(\tilde{v})$, v_1 reçoit aussi le message $m \in B_{\mathbf{G}}^i(u, v_1)$ de la part du processus u lors de l'étape i et le message m est retiré de la boîte. Par conséquent, $Br_{\mathbf{G}}^{i+1}(v_1) = Br_{\mathbf{H}}^{j+1}(\tilde{v})$ et donc, si $\tilde{v}$ reçoit un message lors de l'étape i , $(\lambda(v_1), \text{CONTACTS}(v_1))_{\mathbf{G}}^{i+1} = (\lambda(\tilde{v}), \text{CONTACTS}(\tilde{v}))_{\mathbf{H}}^{j+1}$ et $Br_{\mathbf{G}}^{i+1}(v_1) = Br_{\mathbf{H}}^{j+1}(\tilde{v})$.

Pour appliquer la preuve pour le processus v_1 aux autres processus, nous montrons que les transitions effectuées lors des étapes précédentes n'influent pas sur les états des processus n'ayant pas encore été actifs. En d'autres mots, la preuve sera valide pour tout processus $v_h \in \varphi^{-1}(\tilde{v})$ $(\lambda(v_1), \text{CONTACTS}(v_1))_{\mathbf{G}}^{i+h-1} = (\lambda(\tilde{v}), \text{CONTACTS}(\tilde{v}))_{\mathbf{H}}^j$ et si $\tilde{v}$ envoie exécute une transition d'envoi de message, alors $Be_{\mathbf{G}}^{i+h-1}(v_h) = Be_{\mathbf{H}}^j(\tilde{v})$ et si $\tilde{v}$ exécute une transition de réception de message, $Br_{\mathbf{G}}^{i+h-1}(v_h) = Br_{\mathbf{H}}^j(\tilde{v})$.

Observons que pour tout processus $v_h \in \varphi^{-1}(\tilde{v})$, seul v_h peut ajouter des messages aux tampons de $Be_{\mathbf{G}}(v_h)$ et seul v_h peut retirer des messages aux tampons de $Br_{\mathbf{G}}(v_h)$.

Si $\tilde{v}$ réceptionne un message alors $Br_{\mathbf{H}}^j(\tilde{v}) \neq Br_{\mathbf{H}}^{j+1}(\tilde{v})$ et comme les canaux sont FIFO, il réceptionnera le premier message m du tampon. Comme par définition, m appartient à un tampon contenu dans $Br_{\mathbf{H}}^j(\tilde{v})$ il existe un message m contenu $Br_{\mathbf{G}}^i(v_h)$. Comme seul v_h peut retirer des messages contenus dans les tampons de $Br_{\mathbf{G}}^i(v_h)$ et comme v_h n'est pas actif entre l'étape i et $h-1$, nous obtenons que $Br_{\mathbf{G}}^i(v_h) = Br_{\mathbf{G}}^{i+h-1}(v_h) = Br_{\mathbf{H}}^j(\tilde{v})$. Par le même raisonnement nous obtenons que si $\tilde{v}$ envoie un message alors $Be_{\mathbf{G}}^i(v_h) = Be_{\mathbf{G}}^{i+h-1}(v_h) = Be_{\mathbf{H}}^j(\tilde{v})$. Par conséquent, nous obtenons que pour tout $1 \leq h \leq n$, $(\lambda(v_h), \text{CONTACTS}(v_h))_{\mathbf{G}}^{i+h-1} = (\lambda(\tilde{v}), \text{CONTACTS}(\tilde{v}))_{\mathbf{H}}^j$ et $(Be(v_h), Br(v_h))_{\mathbf{G}}^{i+h-1} = (Be(\tilde{v}), Br(\tilde{v}))_{\mathbf{H}}^j$. En appliquant la preuve pour le processus v_1 à tous les processus contenus dans $\varphi^{-1}(\tilde{v})$, pour tout $1 \leq h \leq n$, $(\lambda(v_h), \text{CONTACTS}(v_h))_{\mathbf{G}}^{i+h} = (\lambda(\tilde{v}), \text{CONTACTS}(\tilde{v}))_{\mathbf{H}}^{j+1}$ et $(Be(v_h), Br(v_h))_{\mathbf{G}}^{i+h} = (Be(\tilde{v}), Br(\tilde{v}))_{\mathbf{H}}^{j+1}$.

Par conséquent, lors de l'étape $i+n$ de cette exécution sur $\mathbf{G}$, $\varphi : (G, \lambda_G, B_G)^{i+n} \rightarrow (H, \lambda_H, B_H)^{j+1}$ est un revêtement dirigé et ce lemme est prouvé. $\square$

Remarque 6.27. Remarquons que pour tout revêtement $\varphi : \mathbf{G} \rightarrow \mathbf{H}$ et pour tout étiquetage $\lambda_{\mathbf{H}}$ de $\mathbf{H}$, nous pouvons construire un étiquetage $\lambda_{\mathbf{G}}$ pour $\mathbf{G}$ tel que $\varphi : (\mathbf{G}, \lambda_{\mathbf{G}}) \rightarrow (\mathbf{H}, \lambda_{\mathbf{H}})$ est un revêtement.

Lemme 6.28 (Initialisation). *Étant donnés deux digraphes H, G et un revêtement $\varphi : G \rightarrow H$, tout étiquetage initial $\lambda_{\mathbf{H}}^0$ de H induit un étiquetage $\lambda_{\mathbf{G}}^0$ pour G et $\varphi : (G, \lambda_{\mathbf{G}}^0) \rightarrow (H, \lambda_{\mathbf{H}}^0)$ est un revêtement.*

Initialement, il n'y a pas de message en transit dans le réseau et donc, $B_{\mathbf{G}}^0 = B_{\mathbf{H}}^0 = \emptyset$. Ainsi, par le lemme 6.28, pour tout digraphe étiqueté $\mathbf{H} = (H, \lambda_H^0)$ et pour tout graphe G , s'il existe un revêtement $\varphi : G \rightarrow H$, alors il existe un étiquetage λ_G^0 tel que $\varphi : (G, \lambda_G^0, B_G^0) \rightarrow (H, \lambda_H^0, B_H^0)$ est un revêtement. En itérant le lemme 6.26, nous obtenons le lemme suivant.

Lemme 6.29 (Lemme de relèvement dirigé). *Pour tout digraphe $\mathbf{G}, \mathbf{H}$, pour tout algorithme $\mathcal{A}$, pour toute exécution $\rho_{\mathbf{H}}$ de $\mathcal{A}$ sur $\mathbf{H}$, s'il existe un revêtement dirigé $\varphi : \mathbf{G} \rightarrow \mathbf{H}$ à n feuillets, alors il existe une exécution $\rho_{\mathbf{G}}$ de $\mathcal{A}$ sur $\mathbf{G}$ telle que pour toute étape i de $\rho_{\mathbf{H}}$, $\varphi : \mathbf{G}^{i \times n} \rightarrow \mathbf{H}^i$ est un revêtement dirigé à n feuillets.*

Comme le modèle autorise les processus à décider une et une seule fois lors d'une exécution, nous reformulons le lemme de relèvement dirigé ci-dessus pour obtenir le lemme de relèvement des décisions présenté ci-dessous.

Lemme 6.30 (Lemme de relèvement des décisions). *Prenons deux digraphes $\mathbf{H}, \mathbf{G}$, un revêtement dirigé $\varphi : \mathbf{G} \rightarrow \mathbf{H}$, un algorithme $\mathcal{A}$ et une exécution de $\mathcal{A}$ sur $\mathbf{H}$. Il existe une exécution de $\mathcal{A}$ sur $\mathbf{G}$ telle que, pour tout processus $v \in V(G)$, $\text{out}_{\mathbf{G}}(v) = \text{out}_{\mathbf{H}}(\varphi(v))$.*

6.4.2 Lemme d'isolement

Le lemme d'isolement met en avant un certain type d'exécution, appelée exécution isolée, propre au modèle des participants inconnus. Introduites dans la section 3.2 du chapitre 1, les exécutions isolées sont constituées par un sous-ensemble de processus exécutant leur algorithme sans avoir connaissance des autres processus du réseau. Nous avons montré que ces exécutions apparaissent uniquement dans des sous parties $\mathbf{H}$ du réseau $\mathbf{G}$ closes par successeurs lorsque tous les messages à destination du sous-graphe $\mathbf{H}$ et en provenance de $\mathbf{G} \setminus \mathbf{H}$ sont bloqués suffisamment longtemps (communication asynchrone malgré des canaux fiables). Dans une telle exécution, les processus de $\mathbf{H}$ sont dit *isolés* du reste du réseau $\mathbf{G}$. En effet, ils exécuteront leur algorithme comme s'ils étaient dans $\mathbf{H}$ et non dans $\mathbf{G}$.

Nous montrons dans cette partie un lemme d'isolement adapté au modèle des participants inconnus et anonymes. Ce lemme est présenté comme un lemme de relèvement.

Remarque 6.31. Un étiquetage du graphe $\mathbf{G}$ induit un étiquetage de tous les sous-graphes de $\mathbf{G}$. De plus, pour tout digraphe G et H tel que $H \subseteq \downarrow G$, pour tout étiquetage initial λ_G^0 de G , il existe un étiquetage initial λ_H^0 de H tel que $(H, \lambda_H^0, \emptyset) \subseteq \downarrow (G, \lambda_G^0, \emptyset)$.

Nous montrons maintenant que si un graphe $\mathbf{H}$ est clos par successeurs dans un graphe $\mathbf{G}$ alors toute exécution dans $\mathbf{H}$ est une exécution pouvant apparaître dans $\mathbf{G}$. Premièrement, nous montrons ce résultat pour une étape, puis nous étendrons le résultat à toutes les étapes d'une exécution.

Lemme 6.32. *Pour tout digraphe $\mathbf{H} = (H, \lambda_H, B_H)$ et $\mathbf{G} = (G, \lambda_G, B_G)$, si $\mathbf{H} \subseteq \downarrow \mathbf{G}$ alors toute transition exécutée sur $\mathbf{H}$ peut être exécutée sur $\mathbf{G}$. Les graphes $\mathbf{H}' = (H', \lambda_{H'}, B_{H'})$ et $\mathbf{G}' = (G', \lambda_{G'}, B_{G'})$ obtenus après la transition vérifient $\mathbf{H}' \subseteq \downarrow \mathbf{G}'$.*

Démonstration. Nous prouvons ce lemme en construisant une exécution similaire dans deux digraphes $\mathbf{H}, \mathbf{G}$ vérifiant $\mathbf{H} \subseteq \downarrow \mathbf{G}$. Pour simplifier les notations, un sommet de $V(\mathbf{H})$ sera noté $\tilde{v}$ et un sommet de $\mathbf{G}$ sera simplement noté v .

Une étape d'une exécution correspond à une transition locale d'un processus. Supposons qu'un processus $\tilde{v}$ envoie un message m' à un voisin dans $\mathbf{H}$. Par hypothèse, $\text{CONTACTS}_{\mathbf{G}}(v) = \text{CONTACTS}_{\mathbf{H}}(\tilde{v})$ et donc v enverra son message m' via les mêmes numéros de port que $\tilde{v}$, i.e., si $\tilde{v}$ envoie un message m' via un port p alors le sommet $v \in V(G)$ enverra aussi un message m' dans G via le port p (mêmes destinataires, car mêmes numéros de port). Comme $\mathbf{H} \subseteq \downarrow \mathbf{G}$, si un processus $\tilde{w} \in V(\mathbf{H})$ reçoit le message m' dans $\mathbf{H}$ par un port $\delta_{\tilde{w}}(\tilde{v})$ alors le processus $w \in V(G)$ recevra le message m' dans G via le port $\delta_w(\tilde{v})$. Ainsi, pour tout $\tilde{v}\tilde{v}' \in A(\mathbf{H})$, $B'_{\mathbf{G}}(v, v') = B'_{\mathbf{H}}(\tilde{v}, \tilde{v}')$ à la fin de l'étape. Comme le processus v finit l'étape dans G dans le même état $\lambda'(v)$ que $\tilde{v}$ dans $\mathbf{H}$, $(H, \lambda'_{\mathbf{H}}, B'_{\mathbf{H}}) \subseteq \downarrow (G, \lambda'_{\mathbf{G}}, B'_{\mathbf{G}})$. Si un message $m \in B_{\mathbf{H}}(\tilde{v}, \tilde{v}')$ est reçu par un processus $\tilde{v}' \in V(\mathbf{H})$ via un port in , alors v peut aussi recevoir un message m de v' dans G . Comme m est supprimé de $B_{\mathbf{H}}(\tilde{v}, \tilde{v}')$ et $B_{\mathbf{G}}(v, v')$ à la fin de l'étape, on obtient que $B'_{\mathbf{H}}(\tilde{v}, \tilde{v}') = B'_{\mathbf{G}}(v, v')$. Remarquons que $\tilde{v}$ finit l'étape dans le même état que v . En particulier, si $\tilde{v}$ ne connaît pas $\tilde{v}'$ alors v ne connaît pas v' . Dans ce cas, $\delta_{\tilde{v}}(\tilde{v}')$ est ajouté à $\text{CONTACTS}'_{\mathbf{H}}(\tilde{v})$ et à $\text{CONTACTS}'_{\mathbf{G}}(v)$. Comme dorénavant, v (resp. $\tilde{v}$) peut communiquer avec v' (resp. $\tilde{v}'$), nous obtenons que $(H', \lambda_{H'}, B_{H'}) \subseteq \downarrow (G', \lambda_{G'}, B_{G'})$. $\square$

Ainsi, pour tout graphe $\mathbf{G}, \mathbf{H}$ tel que $\mathbf{H} \subseteq \downarrow \mathbf{G}$, pour tout algorithme $\mathcal{A}$ et pour toute exécution $\rho_{\mathbf{H}}$ sur $\mathbf{H}$, nous pouvons appliquer le lemme 6.32 ci-dessus de façon itérative pour obtenir une exécution $\rho_{\mathbf{G}}$ sur G où seuls les sommets de $\mathbf{H}$ sont actifs dans $\rho_{\mathbf{G}}$ et où ces processus effectuent exactement les mêmes transitions que dans l'exécution $\rho_{\mathbf{H}}$. En considérant des exécutions qui se terminent, nous obtenons le lemme suivant,

Lemme 6.33 (Lemme d'isolement des exécutions). *Pour tout graphe $\mathbf{G}, \mathbf{H}$ tel que $\mathbf{H} \subseteq \downarrow \mathbf{G}$, pour toute exécution $\rho_{\mathbf{H}}$ d'un algorithme $\mathcal{A}$ sur $\mathbf{H}$ se terminant, il y a une exécution $\rho_{\mathbf{G}}$ de $\mathcal{A}$ sur $\mathbf{G}$ telle que pour tout sommet $v \in V(G)$, $\text{out}_{\rho_{\mathbf{G}}}(v) = \text{out}_{\rho_{\mathbf{H}}}(v)$.*

Nous proposons une condition $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}-\mathcal{AN}\mathcal{O}}$ satisfaite par toute famille de graphes anonymes $\mathcal{F}$ où le problème de l'élection peut être résolu par un algorithme $\mathcal{F}$ -restreint.

Définition 6.34. Une famille $\mathcal{F}$ satisfait $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}-\mathcal{AN}\mathcal{O}}$ si et seulement si pour tout graphe $\mathbf{G} \in \mathcal{F}$,

- (1) $\mathbf{G}$ est minimal pour les revêtements dirigés,
- (2) pour tout digraphe $\mathbf{H}, \mathbf{H}' \in \mathcal{F}$, si $\mathbf{H} \subseteq \downarrow \mathbf{G}$ et $\mathbf{H}' \subseteq \downarrow \mathbf{G}$ alors $\mathbf{H} \cap \mathbf{H}' \neq \emptyset$,
- (3) pour tout digraphe $\mathbf{H} \in \mathcal{F}$, pour tout digraphe $\mathbf{K} \subseteq \downarrow \mathbf{G}$, si $\mathbf{K}$ est un revêtement dirigé de $\mathbf{H}$ alors $\mathbf{H} \simeq \mathbf{K}$.

Lemme 6.35. *Étant donnée une famille de graphes $\mathcal{F}$, s'il existe un algorithme d'élection $\mathcal{F}$ -restreint, alors $\mathcal{F}$ satisfait $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}-\mathcal{AN}\mathcal{O}}$.*

Démonstration. Par contradiction, prenons une famille $\mathcal{F}$ qui ne satisfait pas $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}-\mathcal{AN}\mathcal{O}}$ et soit $\mathcal{A}$, un algorithme $\mathcal{F}$ -restreint pour l'élection.

Pour tout graphe $\mathbf{G}, \mathbf{H} \in \mathcal{F}$, la condition (1) est nécessaire ([YK96c, CGM12, Cha06]). À partir de [CGN14] et de nos résultats présentés dans le chapitre 4, nous savons que la condition (2) est nécessaire dans ce modèle. Nous montrons maintenant la condition (3). Soit deux digraphes $\mathbf{G}, \mathbf{H} \in \mathcal{F}$ et un digraphe $\mathbf{K}$ tels que $\mathbf{K} \subseteq \downarrow \mathbf{G}$, $\mathbf{K} \not\simeq \mathbf{H}$ et soit $\varphi : \mathbf{K} \rightarrow \mathbf{H}$ un revêtement.

Comme $\mathbf{K}$ n'est pas isomorphe à $\mathbf{H}$ et que $\mathbf{K}$ est un revêtement dirigé de $\mathbf{H}$ via φ , $|V(\mathbf{H})| < |V(\mathbf{K})|$ et φ est un revêtement à au moins deux feuillets. Ainsi, pour tout sommet $v \in V(\mathbf{H})$, $|\{v' \mid v' \in V(\mathbf{K}) \wedge \varphi(v') = v\}| \geq 2$.

Comme $\mathcal{A}$ est un algorithme d'élection pour $\mathcal{F}$, et $\mathbf{H} \in \mathcal{F}$, pour toute exécution ρ' de $\mathcal{A}$ dans $\mathbf{H}$ qui termine, il existe un unique processus $v' \in V(\mathbf{H})$ tel que $out_{\rho'}(v') = \text{élu}$. Comme $\mathbf{K}$ est un revêtement de $\mathbf{H}$ par φ , par le lemme 6.30, il existe une exécution ρ de $\mathcal{A}$ sur $\mathbf{K}$ telle que pour tout sommet $v \in V(\mathbf{K})$ tel que $\varphi(v) = v'$, $out_{\rho}(v) = out_{\rho'}(v') = \text{élu}$. Ainsi, il existe au moins deux processus v_1 et v_2 appartenant à $V(\mathbf{K})$ tels que $\varphi(v_1) = \varphi(v_2) = v'$ décident élu lors de l'exécution ρ , i.e., $out_{\rho}(v_1) = out_{\rho}(v_2) = \text{élu}$.

Comme $\mathbf{K} \sqsubseteq \mathbf{G}$, par le lemme 6.33 d'isolement, il existe une exécution ρ'' de $\mathcal{A}$ sur $\mathbf{G}$ telle que $out_{\rho''}(v') = out_{\rho}(v')$, pour tout processus $v' \in V(\mathbf{G}[\mathbf{K}])$. Comme les processus décident une et une seule fois lors d'une exécution, il existe une exécution ρ'' de $\mathcal{A}$ sur $\mathbf{G}$ avec au moins deux processus v_1 et v_2 qui décident élu , i.e., $out_{\rho''}(v_1) = out_{\rho''}(v_2) = \text{élu}$. Une contradiction avec l'universalité de l'algorithme $\mathcal{A}$. $\square$

6.5 Algorithme d'élection pour les familles de graphes de taille bornée

Prenons un entier B . Nous proposons un algorithme $\mathcal{F}$ -restreint résolvant l'élection dans toute famille $\mathcal{F}$ satisfaisant $\mathfrak{C}_{\mathcal{ELC}-\mathcal{ANCO}}$ contenant des graphes possédant moins de B sommets.

6.5.1 Présentation de l'algorithme d'élection

L'algorithme $\mathcal{A}$ implémente l'algorithme d'énumération gSSP présenté dans [CGM12] le modèle des participants inconnus anonymes. L'algorithme gSSP se compose deux algorithmes fondamentaux : l'algorithme d'énumération de Mazurkiewicz ([Maz97]) et l'algorithme de Szymanski, Shy and Prywes [SSP85], nommé SSP.

L'idée est d'attribuer une identité à chaque processus grâce à l'algorithme d'énumération de Mazurkiewicz. Le problème de l'énumération dans un réseau $\mathbf{G}$ consiste à numéroter les processus avec un entier unique pris dans l'ensemble $[1 \dots |V(\mathbf{G})|]$. Une fois l'énumération terminée, il suffira d'élire le processus ayant reçu l'identité minimale. Comme l'algorithme de Mazurkiewicz dans un modèle anonyme ne permet pas aux processus de détecter localement la fin de l'énumération, nous utiliserons l'algorithme de Szymanski, Shy et Prywes (nécessitant une connaissance globale telle une borne sur la taille ou le diamètre du réseau) pour détecter une telle étape. La connaissance globale considérée par la suite sera un entier B bornant la taille du réseau.

6.5.2 États internes des processus

L'état interne de chaque processus est représenté par le tuple suivant,

$$(n(v), N(v), l(v), M(v), a(v), A(v), out(v))$$

avec,

- la composante $n(v)$ est un entier donné par l'algorithme au processus v . Cet entier correspond à l'identité acquise par le processus durant l'exécution de l'algorithme de Mazurkiewicz. Cette valeur pourra changer pendant l'exécution.

- la composante $N(v) \subset \mathbb{N}^3$ correspond à la liste `CONTACTS` présentée précédemment. $N(v)$ représente donc un ensemble fini de triplets d'entiers contenant les informations les plus récentes que v a reçu de ses voisins. Les triplets contenus dans $N(v)$ sont de la forme (n, p, q) où n est l'identité d'un processus w (acquise via l'algorithme) et p et q sont les numéros de port étiquetant un arc $vw \in A(\mathbf{G})$. Initialement $N^0(v) = \{(-1, \delta_v(w), -1) \mid w \in \text{CONTACTS}^0(v)\}$.
- la composante $l(v) = (in(v), \text{CONTACTS}^0(v)) \subseteq \Lambda \cup 2^{\mathbb{N}}$ est un couple composé de l'étiquette initiale du processus $in(v) \in \Lambda$ présentée dans la définition 6.20. La deuxième composante est la liste initiale des contacts initiaux $\text{CONTACTS}^0(v)$ du processus v . La structure $\text{CONTACTS}^0(v)$ est stockée dans la variable $l(v)$ pour permettre aux processus de reconstruire une carte du graphe de communication initial. Il est important de noter que cette étiquette est invariante lors d'une exécution.
- la composante $out(v) \in \mathbf{out}$ contient la valeur de décision du processus v . Une fois écrite, cette valeur ne pourra plus être modifiée par la suite. L'ensemble `out` contient toutes les décisions possibles des processus et est défini dans la spécification du problème que l'algorithme résout.
- la composante $M(v) \subset \mathbb{N} \times \Lambda \cup 2^{\mathbb{N}} \times \mathbb{N}^3 \times \mathbf{out}$ est la boîte aux lettres du processus v . La boîte aux lettres de v lui sert à stocker les informations apprises (grâce aux messages) tout au long de l'exécution. Une boîte aux lettres contient un ensemble fini d'entrées de la forme (m, l, N, out) encodant l'état interne $(m(w), l(w), N(w), out(w))$ d'un processus w du réseau. Initialement, chaque processus v possède une boîte aux lettres contenant uniquement son état initial. Comme initialement, un processus v n'a pris aucune décision, possède l'identité 0 et n'a ni reçu, ni envoyé de message, $M^0(v) = \emptyset$.
- la composante $a(v)$ est un entier représentant le rayon de confiance du processus. Ce rayon de confiance représente la distance à laquelle les processus possèdent (ou ont possédé) une même boîte aux lettres. Cette variable est réinitialisée à chaque fois que v modifie sa boîte aux lettres.
- la composante $A(v)$ est un ensemble fini de couples d'entiers de la forme (p, a) encodant les rayons de confiance des voisins de v . La première composante $p = \delta_v(w)$ correspond au numéro de port identifiant un processus w voisin de v . La deuxième composante a est le plus récent rayon de confiance de w connu de v .

Remarque 6.36. Nous noterons $n^i(v)$, $M^i(v)$, $a^i(v)$, $A^i(v)$ pour indiquer les contenus des structures $n(v)$, $M(v)$, $a(v)$, $A(v)$ au début du tour i .

6.5.3 Les connaissances globales

Comme pour le modèle des participants inconnus avec identités, les processus sont dotés d'une fonction caractéristique d'une famille de graphes $\mathcal{F}$, notée $\chi_{\mathcal{F}}$. Cette fonction prend en entrée un graphe et plus particulièrement un graphe reconstruit à partir d'une boîte aux lettres d'un processus. Cette fonction retourne vrai ou faux selon l'appartenance du graphe à la famille $\mathcal{F}$. De plus, les processus connaîtront une borne supérieure, notée B , sur la taille du graphe. Nous verrons plus tard que cette hypothèse est suffisante pour obtenir la détection locale de la terminaison.

6.5.4 Les messages

Les messages envoyés par les processus sont de la forme $\langle n, n_{old}, M, a \rangle, p$. Si un processus v envoie un message à un processus w , alors v enverra, à w , le message suivant

$$\langle n(v), n_{old}(v), M(v), a(v), \delta_v(w) \rangle$$

avec :

- $n(v)$, la dernière identité attribuée au processus v par l'algorithme.
- $n_{old}(v)$, l'identité que le processus v a envoyée aux autres processus dans son dernier message. Les processus pourront donc détecter si un processus v a mis à jour son identité ou non.
- $M(v)$, la boîte aux lettres de v .
- $a(v)$, le rayon de confiance de v .
- $p = \delta_v(w)$, le numéro de port qui, à partir de v , mène au destinataire w .

6.5.5 Un ordre sur les vues locales des processus

L'algorithme de Mazurkiewicz se base sur une propriété de monotonie des identités calculées obtenue en établissant un ordre sur les vues locales des processus et leur boîte aux lettres.

Comme une vue locale d'un processus est composée par des triplets d'entiers, nous considérerons l'ordre lexicographique usuel pour les comparer deux à deux, i.e.,

$$(n, p, q) < (n', p', q') \text{ si } n < n' \text{ ou si } n = n' \text{ et } p < p' \text{ ou si } n = n', p = p' \text{ et } q < q'$$

En considérant deux ensembles N_1 et N_2 , la différence symétrique entre ces deux ensembles, notée $N_1 \Delta N_2$, est définie par l'expression suivante : $(N_1 \setminus N_2) \cup (N_2 \setminus N_1)$. Pour comparer deux ensembles N_1 et N_2 , nous considérons que l'ensemble N_1 est inférieur à l'ensemble N_2 , noté $N_1 < N_2$, si le maximum de la différence symétrique $N_1 \Delta N_2$ appartient à N_2 (en suivant l'ordre lexicographique). Notons que les structures N et CONTACTS étant similaires, nous comparerons les listes CONTACTS en suivant le même ordre $<$ que pour la structure N .

Soit $<_l$, un ordre sur les étiquettes de processus (l'ordre lexicographique par exemple). Pour comparer les états internes des processus, nous définissons un ordre, nommé $\triangleleft$, tel que pour tout état local $l = (in, \text{CONTACTS})$ et $l' = (in', \text{CONTACTS}')$, si $l \triangleleft l'$ alors $in <_l in'$ ou alors $in = in'$ et $\text{CONTACTS} < \text{CONTACTS}'$. Par la suite, nous noterons $\preceq$ la clôture réflexive de $\triangleleft$.

Pour finir, nous définissons un ordre $\prec$ permettant de comparer les entrées des boîtes aux lettres entre elles. Pour toutes entrées $(n, l, N), (n', l', N') \in M$, nous noterons $(n, l, N) \prec (n', l', N')$ si $n < n'$; ou alors $n = n'$ et $l \triangleleft l'$; ou alors $n = n', l = l'$ et $N < N'$. Pour comparer les structures A contenant les rayons de confiances des voisins, nous considérerons l'ordre $\prec$.

6.5.6 Les primitives et fonctions

Nous définissons une primitive **SendAll** utilisée par les processus pour envoyer des messages à l'ensemble de leurs voisins connus.

Définition 6.37 (SendAll). Soit v un processus dans un graphe $\mathbf{G}$ et soit i une étape d'une exécution. **SendAll** est une primitive qui, exécutée par un processus v , enverra un message m via l'ensemble de numéros de port $\{p \mid \exists(_, p, _) \in N^i(v)\}$.

Nous définissons maintenant une fonction $\mathbb{T}$ permettant d'extraire le graphe dirigé induit par une boîte aux lettres.

Définition 6.38. Soit $\mathbf{G}$ un graphe, soit $v \in V(\mathbf{G})$ un processus et soit M la boîte aux lettres de v à un moment de l'exécution. Le graphe dirigé $\mathbb{T}(M)$ est défini comme suit :

- $V(\mathbb{T}(M)) = \{n \mid (n, l, N, out) \in M\}$
- $A(\mathbb{T}(M)) = \{a_{n,n',p,q} \mid (n, l, N, out) \in M \text{ et } (n', p, q) \in N\}$
- Pour tout arc $a_{n,n',p,q} \in A(\mathbb{T}(M))$, $\lambda(a_{n,n',p,q}) = (p, q)$.
- Pour tout arc $a_{n,n',p,q} \in A(\mathbb{T}(M))$, $\sigma(a_{n,n',p,q}) = a_{(n', n, q, p)}$ s'il existe, sinon $\sigma(a_{n,n',p,q}) = \perp$.

Définition 6.39. Soit $\mathbf{G}$ un graphe, soit $v \in V(\mathbf{G})$ un processus et soit M la boîte aux lettres de v à un moment de l'exécution. Le graphe dirigé $\mathbb{T}^0(M)$ se définit comme suit :

- $V(\mathbb{T}^0(M)) = V(\mathbb{T}(M))$
- $A(\mathbb{T}^0(M)) = A(\mathbb{T}(M)) \setminus \{a_{n,n',p,q} \text{ tel que } (n, l, N, out) \in M, l = (in, \text{CONTACTS}^0) \text{ et } p \notin \text{CONTACTS}^0\}$
- Pour tout arc $a_{n,n',p,q} \in A(\mathbb{T}^0(M))$, $\lambda(a_{n,n',p,q}) = (p, q)$.
- Pour tout arc $a_{n,n',p,q} \in A(\mathbb{T}^0(M))$, $\sigma(a_{n,n',p,q}) = a_{(n', n, q, p)}$ s'il existe, sinon $\sigma(a_{n,n',p,q}) = \perp$.

Remarque 6.40. Pour tout message $m = \langle (n, n_{old}, M, a), \delta_v(w) \rangle$ et pour toute entrée $(n, l, N, out) \in M$, la liste initiale de contacts CONTACTS^0 du processus identifié par n est stockée dans la variable l . Cette liste permettra aux processus de calculer la fonction $\mathbb{T}^0$ localement.

6.5.7 Présentation de l'algorithme

L'idée principale de l'algorithme est d'effectuer une énumération grâce à l'algorithme de Mazurkiewicz puis d'élire le processus ayant l'identité minimale. Pour détecter un état stable de l'exécution où l'énumération est terminée, nous utiliserons les mécanismes de l'algorithme SSP couplés à une connaissance globale donnée aux processus (ici, une borne sur la taille ou le diamètre des graphes de la famille).

L'algorithme $\mathcal{A}$ est composé de deux procédures. Initialement, un processus qui n'a reçu aucun message exécute la procédure I . En exécutant cette procédure, un processus v initialisera son identité à 1 et enverra un message $m = \langle (1, 0, l_v, M^0(v), a^0(v)), p \rangle$ contenant son état initial à tous ses voisins initialement connus (via la primitive **SendAll**). La deuxième procédure, appelée R , est exécutée par tout processus v recevant un message. Si un processus reçoit le message $m = \langle (n, n_{old}, l, M, a), q \rangle$ en provenance d'un processus w voisin via un port $p = \delta_w(v)$ alors $q = \delta_v(w)$ représente le numéro de port liant le processus w au processus v . À la réception du message, le processus v met à jour sa liste de voisins $N(v)$ si nécessaire (Ligne 14). Pour cela, si le processus émetteur du message w est inconnu, i.e., s'il n'existe pas d'entrée $(_, p, _) \in N(v)$ telle que $\delta_v(w) = p$, ou s'il existe une entrée $(n_{old}, p, _) \in N(v)$ avec $n_{old} \neq n$, alors le triplet (n, p, q) est ajouté à $N(v)$ (et l'ancien est supprimé si besoin).

Après avoir mis à jour $N(v)$, le processus v met à jour sa boîte aux lettres $M(v)$ (si nécessaire) avec toute entrée (n, l, N) présente dans la boîte aux lettres M reçue (Ligne 12). Une fois cette mise à jour faite, le processus v met à jour son identité $n(v)$ s'il possède une identité égale à 0 (Ligne 18) ou s'il existe une entrée $(n(v), l', N') \in M(v)$ inférieure à la sienne suivant l'ordre $\prec$, i.e., si $(l', N') \prec (l(v), N(v))$.

Remarque 6.41. Notons qu'il peut exister deux entrées $(n, l, N, out), (n, l', N', out')$ dans la boîte aux lettres (Ligne 19) d'un processus. Si un tel cas se présente, nous considérerons uniquement l'entrée (n, l, N, out) maximale pour l'ordre $\preceq$.

Le processus continue la procédure en mettant à jour les rayons de confiance de ses voisins ainsi que le sien. Tout d'abord, si le processus a modifié sa boîte aux lettres $M(v)$ lors de cette étape, le processus réinitialise son rayon de confiance à 0 et réinitialise celui de ses voisins stockés dans $A(v)$ à -1 (Ligne 20).

Si le processus a reçu une boîte aux lettres identique à la sienne (Ligne 23), l'agent met à jour sa liste $A(v)$ avec le dernier rayon de confiance connu de son voisin émetteur du message (composante a du message m). Puis, à la ligne 25, si tous les rayons de confiance stockés dans $A(v)$ sont supérieurs ou égaux au rayon de confiance $a(v)$ de v , alors le processus v incrémente son rayon de confiance $a(v)$ (Ligne 26).

En fin de phase, si l'agent a modifié sa boîte aux lettres $M(v)$ ou son rayon de confiance $a(v)$, alors il envoie son nouvel état interne à tous ses voisins "connus" (via la primitive **SendAll** à la ligne 33). Sinon, il n'envoie aucun message.

Un processus décidera une valeur (**élu** ou **non-élu**) s'il n'a jamais décidé par le passé, si son rayon de confiance a atteint la borne B et si le graphe reconstruit à partir de sa carte appartient à la famille $\mathcal{F}$ considérée (Ligne 27). Une fois ces conditions réunies, un processus décidera **élu** s'il n'existe pas d'autre étiquette **élu** dans sa boîte aux lettres $M(v)$ et s'il possède l'identité 1 (Ligne 28). Sinon, le processus décidera **non-élu**.

Remarque 6.42. À partir du moment où un processus décide, il écrit dans sa variable *out* mais il ne termine pas pour autant son exécution. En effet, comme nous allons le voir, les processus ne peuvent pas détecter globalement la détection de la terminaison due aux exécutions isolées. Il est donc nécessaire que les processus continuent leurs exécutions pour toujours afin de relayer les messages et continuer le calcul en cas de participants se manifestant tard dans l'exécution.

6.6 Correction de l'algorithme

Les lemmes 6.43 à 6.46 montrent la convergence de l'énumération (algorithme de Mazurkiewicz). Les lemmes 6.48 à 6.54 montrent la convergence de l'algorithme de stabilisation (SSP). Les preuves respectives ont été adaptées pour le modèle des participants inconnus à partir des preuves proposées dans [GM02, CM07, CGM12]. Les autres lemmes terminent la preuve de correction de l'algorithme 5 et sont propres au modèle des participants inconnus.

Les lemmes suivants montrent les propriétés satisfaites tout au long d'une exécution par la boîte aux lettres d'un processus. Elles sont obtenues directement à partir de l'algorithme. Notons que tout processus n'ayant exécuté aucune transition, possède l'identité 0.

Lemme 6.43. *Étant donné un digraphe $\mathbf{G}$, un processus $v \in V(\mathbf{G})$ et une étape i d'une exécution de $\mathcal{A}$ sur $\mathbf{G}$,*

- (1) *Si $n^i(v) > 0$, alors il existe $(n^i(v), l(v), N^i(v)) \in M^i(v)$,*
- (2) *pour tout $(n, p, q) \in N^i(v)$, il existe $(n, l', N') \in M^i(v)$,*
- (3) *pour tout $(n, p, q) \in N^i(v)$, il existe $(p, a) \in A^i(v)$,*
- (4) *pour tout $(p, a) \in A^i(v)$, il existe $(n, p, q) \in N^i(v)$,*
- (5) *pour tout $(n, p, q), (n', p', q') \in N^i(v)$, $p \neq p'$,*
- (6) *pour tout $(n^i(v), l, N) \in M^i(v)$, $(l, N) \preceq (l^i(v), N^i(v))$.*

Le lemme suivant montre que l'état interne des processus est monotone. En d'autres termes, le contenu des structures des processus converge vers un état stable durant toute exécution.

Algorithme 5 : Élection dans le modèle des participants inconnus anonymes

```

Procédure I():          /*  $n(v_0) = 0$  et aucun message n'a été reçu par  $v_0$  */
1  |  $n \leftarrow 1$ ;
2  |  $N \leftarrow \{(0, p, -1) \mid p \in \text{CONTACTS}_{\mathbf{G}}(v_0)\}$ ;
3  |  $M \leftarrow \{(1, l(v_0), N, \perp)\}$ ;
4  |  $a \leftarrow 0$ ;
5  |  $A \leftarrow \{(p, -1) \mid p \in \text{CONTACTS}_{\mathbf{G}}(v_0)\}$ ;
6  |  $out \leftarrow \perp$ ;
7  | SendAll $\langle 1, 0, M, 0 \rangle$ ;

Procédure R( $\chi_{\mathcal{F}}, B$ ):      /* Message  $\langle (n', n'_{old}, M', a'), q \rangle$  reçu par le port  $p$  */
/* Paramètres:                                     */
/*  $\chi_{\mathcal{F}}$ , fonction caractéristique de la famille  $\mathcal{F}$                                */
/*  $B$ , borne supérieure sur la taille des graphes de  $\mathcal{F}$                                */

8  |  $M_{old} = M$ ;
9  |  $n_{old} = n$ ;
10 |  $a_{old} = a$ ;
11 |  $out_{old} = out$ ;
12 |  $M \leftarrow M \cup M'$ ;

    /* Mise à jour des nouveaux voisins et des identités */
13 | si  $(\_, p, \_) \notin N$  alors
14 |   |  $N \leftarrow N \cup \{n', p, q\}$ ;          /* Nouveau voisin */
15 |   |  $A \leftarrow A \cup \{p, -1\}$ ;
16 |   |  $N \leftarrow N \setminus \{(n'_{old}, p, q)\} \cup \{(n', p, q)\}$ ;
17 |   | si  $\exists (n, l', N', out') \in M$  t.q.  $(l, N) \prec (l', N')$  alors
18 |   |   |  $n \leftarrow 1 + \max\{n' \mid \exists (n', l', N', out') \in M\}$ 
19 |   |  $M \leftarrow M \cup \{(n, l, N, out)\}$ ;          /* Mise à jour boîte aux lettres */
20 |   | si  $M \neq M_{old}$  alors
21 |   |   |  $a \leftarrow 0$ ;          /* Réinitialisation des rayons de confiance */
22 |   |   |  $A \leftarrow \{(p', -1) \mid (\_, p', \_) \in N\}$ 
23 |   | si  $M = M'$  alors
24 |   |   |  $A \leftarrow A \setminus \{(p, a'') \mid (p, a'') \in A\} \cup \{(p, a')\}$ 
25 |   | si  $\forall (p, a') \in A, a \leq a'$  alors
26 |   |   |  $a \leftarrow 1 + \min\{a' \mid \exists (p, a') \in A\}$ ; /* Mise à jour des rayons de confiance */
27 |   | si  $a > B \wedge out = \perp \wedge \chi_{\mathcal{F}}(\mathbb{T}^0(M)) = \text{YES}$  alors
28 |   |   | si  $n = 1$  and  $\nexists (\_, \_, \_, élu) \in M$  alors
29 |   |   |   |  $out \leftarrow élu$ ;          /* Décision du processus */
30 |   |   | sinon
31 |   |   |   |  $out \leftarrow \text{non-élu}$ ;
32 |   |   |  $M \leftarrow M \cup \{(n, l, N, out)\}$ ;
33 |   | si  $a < B \wedge (M \neq M_{old} \vee a \neq a_{old})$  alors
34 |   |   | SendAll $\langle n, n_{old}, M, a \rangle$ ;          /* Propagation du nouvel état */

```

Lemme 6.44. *Étant donné un graphe $\mathbf{G}$, un processus $v \in V(\mathbf{G})$ et une étape i d'une exécution de l'algorithme sur $\mathbf{G}$, les propriétés suivantes sont satisfaites,*

- (1) $n^i(v) \leq n^{i+1}(v)$,
- (2) $N^i(v) < N^{i+1}(v)$,
- (3) $M^i(v) \subseteq M^{i+1}(v)$.

Démonstration. Soit v un processus recevant un message $m = \langle (n, n_{old}, M, a), q \rangle$ d'un processus u par un port p lors d'une étape i d'une exécution de l'algorithme sur un graphe $\mathbf{G}$. Notons que pour tout processus $w \neq v$, les propriétés sont satisfaites, car w n'exécute aucune transition durant cette étape.

- (1) Si $n^i(v) \neq n^{i+1}(v)$ alors par l'algorithme, $n^{i+1}(v) = 1 + \max\{n' \mid (n', l', N') \in M^i(v)\}$. Si $n^i(v) = 1$ alors $n^{i+1}(v) > 0$. Sinon, $n^i(v) > 1$ et alors, par le point 1 du lemme 6.43, il existe une entrée $(n^i(v), l, N) \in M^i(v)$. Comme $n^{i+1}(v) = 1 + \max\{n' \mid (n', l', N') \in M^i(v)\}$, $n^{i+1}(v) \geq n^i(v)$.
- (2) Supposons que le processus v reçoit un message $m = \langle (n, n_{old}, M, a), q \rangle$ en provenance de u lors de l'étape i . Notons que $\delta_u(v) = q$ et $\delta_v(u) = p$.
Si $N^i(v) \neq N^{i+1}(v)$, à partir de l'algorithme, trois cas existent : soit il n'existe pas d'entrée $(_, p, _) \in N^i(v)$ (u est un nouveau voisin) ; soit il existe une entrée $(0, p, -1) \in N^i(v)$ (i.e. v n'a jamais reçu de message en provenance de u) ; soit il existe $(n_{old}, p, q) \in N^i(v)$ et $n_{old} \neq n$ (i.e. v a déjà reçu un message en provenance de u).
Si $(_, p, _) \notin N^i(v)$, alors $N^{i+1}(v) = N^i(v) \cup \{(n, p, q)\}$. Comme $\max\{N^i(v) \triangle N^{i+1}(v)\} = (n, p, q) \in N^{i+1}(v)$ alors $N^i(v) < N^{i+1}(v)$.
Si $(0, p, -1) \in N^i(v)$ alors m est le premier message reçu par v en provenance de u . Le processus v met donc à jour $N^{i+1}(v)$ en y insérant (n, p, q) . Comme par définition, $(0, p, -1) < (n, p, q)$ et que $N^{i+1}(v) = N^i(v) \setminus \{(0, p, -1)\} \cup \{(n, p, q)\}$, nous obtenons que $N^i(v) < N^{i+1}(v)$.
Sinon, soit $m' = \langle (n', n'_{old}, M', a'), q \rangle$, le dernier message reçu par v en provenance de u lors d'une étape $h < i$. Soit $h' < h$ l'étape d'envoi du message par u . Comme par l'algorithme, $n^{h'}(u) = n_{old}$ et $n^h(u) = n$, par le cas (1), $n_{old} < n$ (et $(n_{old}, p, q) \in N^i(v)$). Comme $N^{i+1}(v) = N^i(v) \setminus (n_{old}, p, q) \cup (n, p, q)$, nous obtenons que $\max\{N^i(v) \triangle N^{i+1}(v)\} = (n, p, q) \in N^{i+1}(v)$ et donc $N^i(v) \leq N^{i+1}(v)$.
- (3) Comme nous avons prouvé dans les deux cas ci-dessus que pour toute étape i et pour tout processus v , $n^i(v) \leq n^{i+1}(v)$, $N^i(v) < N^{i+1}(v)$ et $l^i(v) = l^{i+1}(v)$ et comme par l'algorithme, le contenu d'une boîte aux lettres est croissant, nous obtenons que $M^i(v) \subseteq M^{i+1}(v)$.

□

Nous montrons maintenant que, pour toute étape i , les identités présentes dans la boîte aux lettres d'un processus sont toutes attribuées à des processus du réseau et que l'ensemble des identités prises par les processus ne fait que croître lors d'une exécution.

Lemme 6.45. *Étant donné un digraphe $\mathbf{G}$, un processus $v \in V(\mathbf{G})$ et une étape i d'une exécution de $\mathcal{A}$ sur $\mathbf{G}$, pour tout triplet $(m, l, N) \in M^i(v)$, il existe un processus $w \in V(\mathbf{G})$ et une étape $h \leq i$ tels que $n^h(w) = m$, $N^h(w) = N$ et $l_w = l$.*

Démonstration. Soit i une étape d'une exécution de l'algorithme sur un graphe $\mathbf{G}$ et soit v un processus.

Nous montrons que toute identité n présente dans $M^i(v)$ est attribuée à au moins un processus du réseau lors de cette étape i .

Premièrement, remarquons que pour tout triplet (m, l, N) ajouté à l'ensemble $\bigcup_{v \in V(\mathbf{G})} M^i(v)$ lors d'une étape i , il existe un processus $v \in V(\mathbf{G})$ tel que $(n^i(v), l(v), N^i(v)) = (m, l, N)$. Soit $U = \{(u, j) \in V(\mathbf{G}) \times \mathbb{N} \mid n^j(u) = m \wedge j \leq i\}$ et soit $U' = \{(u, j) \in V(\mathbf{G}) \times \mathbb{N} \mid (u, j) \in U, \forall (u', j') \in U, (l(u'), N^{j'}(u')) \prec (l(u), N^j(u)) \text{ ou } (l(u'), N^{j'}(u')) = (l(u), N^j(u)) \text{ et } j' \leq j\}$. Notons qu'il existe une étape $i_0 \in \mathbb{N}$ telle que pour tout $(u, j) \in U'$, $j = i_0$.

Prenons un couple $(u, i_0) \in U'$. Si $i_0 < i$ alors implicitement, $(u, i_0 + 1) \notin U$ et donc, $n^{i_0+1}(u) \neq n^i(u)$. À l'inverse, remarquons que $(u, i_0 + 1) \in U$ et donc, que $(u, i_0) \notin U'$. Ainsi, comme $n^{i_0+1}(u) \neq n^i(u)$, nous obtenons que u met à jour son nombre n lors de l'étape i . Mais comme (u, i_0) est maximale dans U' , u ne peut pas effectuer un tel changement lors de l'étape i (par l'algorithme). Nous obtenons une contradiction. $\square$

Nous montrons maintenant que l'énumération converge vers un état stable. Pour cela, nous montrons que l'ensemble de entiers compris dans l'intervalle $[1 \dots n]$ est attribué aux processus par l'algorithme, pour toute identité n présente dans une boîte aux lettres d'un processus. Cette propriété est essentielle, car elle permet aux processus d'obtenir une borne inférieure sur le nombre de processus du réseau.

Lemme 6.46. *Étant donné un digraphe $\mathbf{G}$, un processus $v \in V(\mathbf{G})$ et une étape i d'une exécution de $\mathcal{A}$ sur $\mathbf{G}$, pour tout triplet $(m, l, N) \in M^i(v)$, pour tout $m' \in [1 \dots m]$, il existe un triplet $(m', l', N') \in M^i(v)$.*

Démonstration. Nous prouvons ce lemme par une induction sur les étapes d'une exécution de l'algorithme sur $\mathbf{G}$. Initialement, le lemme est prouvé, car aucun numéro n'a été attribué. Supposons que le lemme est vrai pour une étape $h < i$ et supposons, sans perte de généralité, que le processus v est actif à l'étape i . Comme $M^i(v)$ indique le contenu de la boîte aux lettres en début d'étape i , nous montrons que pour tout triplet $(m, l, N) \in M^{i+1}(v)$, pour tout $m' \in [1 \dots m]$, il existe un triplet $(m', l', N') \in M^{i+1}(v)$.

Si v exécute la procédure I , alors $M^{i+1}(v) = \{(1, l(v), N(v), out(v))\}$ et le lemme est prouvé. Si v exécute la procédure R , alors soit $m = \langle n, n', M, a \rangle, q$ le message reçu par v provenant d'un voisin w par le port p lors de l'étape i . Soit $j < i$ l'étape d'émission du message m par w . Notons que $M^j(w) = M$. Si $n^{i+1}(v) = n^i(v)$ (v ne change pas son identité lors de cette phase) alors comme $M^{i+1}(v) = M^i(v) \cup M^j(w)$, par induction, le lemme est prouvé. Sinon, $n^{i+1}(v) \neq n^i(v)$ et donc $n^{i+1}(v) = 1 + \max\{n' \mid (n', l', N') \in M^{i+1}(v) \cup M\}$. Par conséquent, $M^{i+1}(v) = M^i(v) \cup M \cup \{(n^{i+1}(v), l(v), N^{i+1}(v))\}$. Ce lemme est prouvé. $\square$

Si un processus change son identité m pour une nouvelle identité $m' > 0$ lors d'une étape i , alors $m' > m$ (Lemme 6.44) ; il existe un processus du réseau ayant le numéro m et un deuxième processus du réseau ayant le numéro m' lors de l'étape i (Lemme 6.45) ; pour tout entier de l'intervalle $[1 \dots m']$, il existe un processus possédant cette identité lors de l'étape i (Lemme 6.46). Pour toute identité m attribuée au processus, il existe m processus du réseau avec des identités de 1 à m . Nous obtenons que les identités attribuées par l'algorithme aux processus sont bornées par $|V(\mathbf{G})|$, le nombre total de processus dans le réseau.

Nous prouvons que l'énumération s'achève en prouvant qu'elle converge vers un état stable. En d'autres mots, il existe une étape i_0 telle que pour tout processus v et pour toute étape $i > i_0$, $n^{i_0}(v) = n^i(v)$, $N^{i_0}(v) = N^i(v)$ et $M^{i_0}(v) = M^i(v)$.

Pour cela, nous commençons par montrer que si la boîte aux lettres d'un processus reste inchangée suite à la réception d'un message alors le rayon de confiance du processus ne diminue pas.

Lemme 6.47. *Étant donné un digraphe G , un processus $v \in V(G)$ et une étape i d'une exécution de $\mathcal{A}$ sur G , si $M^i(v) = M^{i+1}(v)$ alors $a^i(v) \leq a^{i+1}(v)$ et $A^i(v) \prec A^{i+1}(v)$.*

Démonstration. Supposons par l'absurde que lors d'une étape i , il existe un processus v , actif lors de l'étape i , tel que $M^i(v) = M^{i+1}(v)$ et $a^i(v) > a^{i+1}(v)$. Il existe uniquement deux lignes de l'algorithme permettant de modifier le rayon de confiance : les lignes 20 et 25. Comme $M^i(v) = M^{i+1}(v)$, le test à la ligne 20 échoue et $A^i(v)$ n'est pas réinitialisée. Comme $a^i(v) \neq a^{i+1}(v)$, le test à la ligne 25 ne peut que réussir (sinon nous aurions $a^i(v) = a^{i+1}(v)$). Nous en déduisons que pour tout $(p, a) \in A^{i+1}(v)$, $a^i(v) \leq a$ et donc $a^i(v) \leq \min\{a \mid (p, a) \in A^{i+1}(v)\}$. Comme par l'algorithme, $a^{i+1}(v) = \min\{a \mid (p, a) \in A^{i+1}(v)\} + 1 > a^i(v)$, nous obtenons une contradiction et donc $a^i(v) \leq a^{i+1}(v)$.

Supposons par l'absurde que $M^i(v) = M^{i+1}(v)$ et $A^i(v) \succ A^{i+1}(v)$. Comme $A^i(v) \succ A^{i+1}(v)$ et que par le cas précédent, $A^i(v)$ n'est pas réinitialisé lors de cette étape, v a reçu un message $m = \langle (n, n_{old}, M, a), q \rangle$ via un port p . Comme $A^i(v) \neq A^{i+1}(v)$, $A^{i+1}(v) = A^i(v) \setminus \{(p, a') \in A^i(v)\} \cap \{(p, a)\}$, et donc, par la ligne 23, $M = M^i(v)$ et $a < a'$ (car $A^i(v) \succ A^{i+1}(v)$). Soit $h < i$ l'étape où un processus w envoie le message m à v via le port q et notons j, j' les deux étapes telles que le processus w envoie un message $m' = \langle (n', n'_{old}, M', a'), q \rangle$ à v lors de l'étape j et v reçoit le message m' lors de l'étape j' , i.e., $(p, a') \notin A^{j'}(v)$, $(p, a') \in A^{j'+1}(v)$, $a^j(w) = a'$ et $a^h(w) = a$. Notons que $j < j'$. Comme les canaux de communication préservent l'ordre des messages, $j < h$ et $j' < i$. Comme $(p, a') \notin A^{j'}(v)$, $(p, a') \in A^{j'+1}(v)$ et $(p, a') \in A^i(v)$, v n'a pas réinitialisé son rayon de confiance depuis l'étape j' et donc nous obtenons que $M = M^j(w) = M^{j'}(v) = M^i(v)$. Par le même raisonnement, comme $(p, a) \notin A^i(v)$ et $(p, a) \notin A^{i+1}(v)$, $M' = M^h(w) = M^{i+1}(v)$. Par conséquent, comme $M^i(v) = M^{i+1}(v)$ alors $M^j(w) = M^h(w)$ et par conséquent, $a^j(w) \leq a^h(w)$ (premier point du lemme appliqué à w). Comme $a^j(w) = a' < a = a^h(w)$, $A^{i+1}(v) \succ A^i(v)$ et nous obtenons une contradiction. $\square$

Nous montrons maintenant que la distance (non dirigé) entre deux processus du réseau borne supérieurement la différence entre leurs rayons de confiance respectifs. Premièrement, nous montrons cette propriété pour deux processus voisins.

Lemme 6.48. *Étant donné un digraphe G , un processus $v \in V(G)$ et une étape i d'une exécution de $\mathcal{A}$ sur G , si $M^i(v) = M^{i+1}(v)$ alors pour tout $(p, a) \in A^i(v)$, il existe une entrée $(p, a') \in A^{i+1}(v)$ telle que $a' > a^{i+1}(v) - 1$.*

Démonstration. Nous prouvons ce lemme par une récurrence sur les étapes d'une exécution. Initialement, pour tout processus exécutant la procédure I lors d'une étape i , $M^i(v) = M^{i+1}(v)$, $a^i(v) = 0$ et pour tout $(p, a) \in A^i(v)$, $a = -1$. Ainsi, $a^i(v) = a^{i+1}(v)$ pour tout processus exécutant la procédure I lors d'une étape i .

Supposons ce lemme prouvé pour toute étape $h < i$ de l'exécution et pour tout processus. Prenons un processus v actif lors d'une étape i et supposons que $M^i(v) = M^{i+1}(v)$. Soit $m = \langle (n, n_{old}, M, a_1), q \rangle$ le message reçu par v via le port p en provenance d'un processus w . Notons que $\delta_v(w) = p$ et $\delta_w(v) = q$. Soit $h < i$ l'étape telle que $A^{h-1}(v) \neq A^h(v) = A^i(v)$ et $M^h(v) = M^i(v)$ (et $a^h(w) = a^i(v)$).

Si $A^i(v) = A^{i+1}(v)$, alors $a^i(v) = a^{i+1}(v)$ et par hypothèse de récurrence, pour tout $(p, a) \in A^h(v) = A^{i+1}(v)$, $a > a^{i+1}(v) - 1$.

Sinon, $A^i(v) \prec A^{i+1}(v)$ (Lemme 6.47) et donc une entrée $(p, a_1) \in A^{i+1}(v)$ est insérée lors de cette étape i et pour tout $(p, a) \in A^i(v)$, $a < a_1$. Notons que v ne réinitialise pas, $A^i(v)$ car $M^i(v) = M^{i+1}(v)$. Soit $\min_A = \min\{a \mid (p, a) \in A\}$ le rayon de confiance minimal inclus dans une structure A .

Si $a_1 > \min_{A^{i+1}(v)}$, alors $\min_{A^i(v)} = \min_{A^{i+1}(v)}$ et comme $A^h(v) = A^i(v)$, de par notre hypothèse de récurrence, ce lemme est prouvé.

Si $a_1 = \min_{A^{i+1}(v)}$, alors par l'algorithme, $a^{i+1}(v) = \min_{A^{i+1}(v)} + 1 = a_1 + 1$ et donc $a_1 > a^{i+1}(v) - 1$. Ce lemme est prouvé. $\square$

Nous montrons les propriétés d'une boîte aux lettres vérifiées lorsqu'un processus incrémente son rayon de confiance.

Lemme 6.49. *Étant donné un digraphe $\mathbf{G}$, un processus $v \in V(\mathbf{G})$ et une étape i d'une exécution de $\mathcal{A}$ sur $\mathbf{G}$,*

- (1) [**Réception**] *si $a^i(v) > 0$ alors, pour tout $(p, a) \in A^i(v)$, $a > -1$.*
- (2) [**Carte localement injective**] *si $a^i(v) > 0$ alors, pour tout $(n, p, q) \in N^i(v)$, v a reçu un message $m = \langle (n, n_{old}, M, a), q \rangle$ via le port p envoyé lors d'une étape h par un processus w avec $M = M^h(v) = M^i(v)$, $(n, p, q) = (n^h(w), \delta_v(w), \delta_w(v))$, $n > 0$, $a = a^h(w)$ et $(p, a) \in A^i(v)$. Ce message est le dernier à avoir été reçu par v en provenance du processus w .*
- (3) [**Carte localement symétrique**] *si $a^i(v) > 1$ alors, pour tout $(n, p, q) \in N^i(v)$, il existe un processus w et une étape $h < i$ telle que $M^h(w) = M^i(v)$, $vw, wv \in E(\mathbf{G}^h)$ et $(n^i(v), q, p) \in N^h(w)$.*

Démonstration. Nous prouvons les conditions séparément.

(1) Par le lemme 6.48.

(2) Prenons un processus v terminant une étape i avec un rayon de confiance $a^{i+1}(v) > 0$ et soit j l'étape où v a réinitialisé son rayon de confiance pour la dernière fois. Par la condition (4) du lemme 6.43, pour tout $(p, a) \in A^i(v)$, il existe $(n, p, q) \in N^i(v)$ et par le point précédent, $a > -1$. Ainsi, pour tout $(p, a) \in A^i(v)$, v a reçu un message $m = \langle (n, n_{old}, M, a), q \rangle$ via le port p lors d'une étape $j < h < i$ en provenance d'un processus w . Soit ℓ l'étape d'envoi du message par w . Comme lors l'étape h , v n'a pas réinitialisé son rayon de confiance en recevant m nous obtenons que $M = M^h(v)$, $N^h(v) = N^{h+1}(v)$ et v insère le couple (p, a) dans $A^{h+1}(v)$ avec $a = a^\ell(w)$. Comme $N^h(v) = N^{h+1}(v)$ alors $(n^\ell(w), \delta_v(w), \delta_w(v)) = (n, p, q)$ et ce lemme est prouvé.

(3) Par le point (2), pour tout $(n, p, q) \in N^i(v)$, il existe deux étapes $h < h' < i$ et un processus w tels que $p = \delta_v(w)$, $q = \delta_w(v)$, w envoie un message $m_1 = \langle (n^h(w), n_{old}, M^h(w), a^h(w)), q \rangle$ à v lors de l'étape h et v reçoit m_1 lors de l'étape h' avec $M^h(w) = M^i(v)$ et $a^h(w) > 0$ (Lemme 6.48). Comme $M^h(w) = M^i(v)$ alors il existe $(n^h(w), l_w, N^h(w)) \in M^i(v)$. Comme w a envoyé un message à v , alors il existe $(m, q, p') \in N^h(w)$. Étudions les différents cas possibles.

Si $p' = -1$, alors w n'a jamais reçu de message via le port $\delta_w(v)$ avant l'étape h (par l'algorithme). Par conséquent, $(-1, q, p') \in N^h(w)$ et w réinitialise son rayon de confiance ($a^h(w) = 0$). Comme par le lemme 6.48, $a = a^h(w) > 0$, nous obtenons une contradiction. Sinon, w a reçu un message via le port $\delta_w(v)$ précédemment. Par le lemme 6.48, $a^h(w) > 0$ et donc w a reçu message $m_2 = \langle (m, m_{old}, M', a'), p \rangle$ en provenance de v lors d'une étape $j' < h$ car seul le processus v peut envoyer un message sur ce canal de communication. Soit j l'étape d'envoi du message par v . De plus, comme $a^h(w) > 0$, w ne réinitialise pas son rayon lors de cette étape. Nous obtenons donc que $M' = M^h(w)$. Par l'algorithme, w met à jour automatiquement $N^{j'}(w)$ lors de la réception de message. En fin d'étape, $(m, q, p) \in N^{j'+1}(w)$. Comme $M^h(w) = M^i(v)$ et comme $M^h(w) = M'$, nous en déduisons que

$M' = M^i(v)$ et donc que $n^j(v) = n^i(v)$. Par conséquent, $\mathbf{m} = n^j(v)$ et donc $(n^i(v), q, p) \in N^{j'+1}(w)$. Ce lemme est prouvé. $\square$

Avec des exécutions isolées, il est possible qu'une stricte sous-partie des processus possèdent une même boîte aux lettres et voient leur rayon de confiance augmenter et dépasser la borne B sans connaître l'ensemble des processus du réseau. Une telle exécution peut mettre en échec une exécution, car des processus peuvent décider prématurément des valeurs en accord avec leur algorithme mais non cohérentes avec la spécification globale du problème.

Pour caractériser ces processus ayant contribué au contenu d'une boîte aux lettres donnée, nous introduisons un sous-graphe particulier du graphe de communication initial $\mathbf{G}$, noté $\mathbf{G}[M]$.

Définition 6.50. Étant donné un graphe de communication initial $\mathbf{G}$ et un processus $v \in V(\mathbf{G})$, pour toute étape i d'une exécution sur $\mathbf{G}$, nous noterons $\mathbf{G}[M^i(v)]$ le sous-graphe dirigé connexe et maximal de $\mathbf{G}$ incluant v et induit par tout sommet $w \in V(\mathbf{G})$ tel que $(n^h(w), l_w, N^h(w)) \in M^i(v)$, $h < i$.

Ce graphe est illustré par la figure 6.3. Notons que par le lemme 6.45, pour toute entrée $(n, l, N) \in M^i(v)$, il existe une étape $h < i$ et un processus $w \in V(\mathbf{G})$ tel que $n^h(w) = m$, $N^h(w) = N$ et $l_w = l$. Par conséquent, toute entrée de la carte correspond bien à (au moins) un sommet $w \in V(\mathbf{G}[M^i(v)])$.

Définition 6.51. Nous noterons $\mathbf{G}^i[M^i(v)] = \mathbf{G}^i|_{V(\mathbf{G}[M^i(v)])}$, la sous-partie du réseau $\mathbf{G}^i$ induite par les sommets de $V(\mathbf{G}[M^i(v)])$.

Nous présentons maintenant une propriété essentielle du graphe $\mathbf{G}[M^i(v)]$. Cette propriété est obtenue directement à partir des définitions.

Proposition 6.52. Pour tout graphe $\mathbf{G}$, pour tout processus v et toute étape i d'une exécution de l'algorithme sur $\mathbf{G}$, $\mathbf{G}[M^i(v)]$ est connexe et $\mathbf{G}[M^i(v)] \subseteq \mathbf{G}$.

Remarque 6.53. Nous considérons un graphe $\mathbf{G}[M]$ connexe, car il est possible que deux processus dans deux exécutions isolées calculent une même boîte aux lettres M dans deux sous-graphes distincts du réseau. Les processus exécutant une des deux exécutions isolées n'ont alors aucune connaissance des processus exécutant la deuxième exécution isolée.

Le prochain lemme montre que la distance séparant deux sommets de la carte $\mathbb{T}(M^i(v))$ borne supérieurement la différence entre les rayons de confiance des processus respectifs.

Lemme 6.54. Étant donné un digraphe $\mathbf{G}$, un processus $v \in V(\mathbf{G})$ et une étape i d'une exécution de $\mathcal{A}$ sur $\mathbf{G}$, pour toute étape $h < i$ et pour tout processus $w \in \mathbf{G}^i[M^i(v)]$ tel que $\bar{d}_{\mathbb{T}(M^i(v))}(n^i(v), n^h(w)) < a^i(v)$, si $M^i(v) = M^{i+1}(v)$ alors $M^i(v) = M^h(w)$ et $a^h(w) > a^i(v) - \bar{d}_{\mathbb{T}(M^i(v))}(n^i(v), n^h(w)) \geq 0$.

Démonstration. Considérons une étape i d'une exécution de l'algorithme sur un graphe $\mathbf{G}$ et un processus $v \in V(\mathbf{G})$ actif lors de cette étape i vérifiant $M^i(v) = M^{i+1}(v)$. Nous prouvons ce lemme par une récurrence sur la distance (non dirigée) $\bar{d}_{\mathbb{T}(M^i(v))}(n^i(v), \mathbf{m})$ reliant tout processus $\mathbf{m}$ de $V(\mathbb{T}(M^i(v)))$ à $n^i(v)$.

Si $\bar{d}_{\mathbb{T}(M^i(v))}(n^i(v), \mathbf{m}) = 1$, alors il existe $a_{n^i(v), \mathbf{m}, p, q} \in A(\mathbb{T}(M^i(v)))$ et donc il existe $(\mathbf{m}, p, q) \in N^i(v)$ par définition de $\mathbb{T}(M^i(v))$. Supposons que $a^i(v) > 1$. Par la condition (3) du lemme 6.49, il existe un processus w et une étape $h < i$ telle que $n^h(w) = \mathbf{m}$, $M^h(w) = M^i(v)$ et

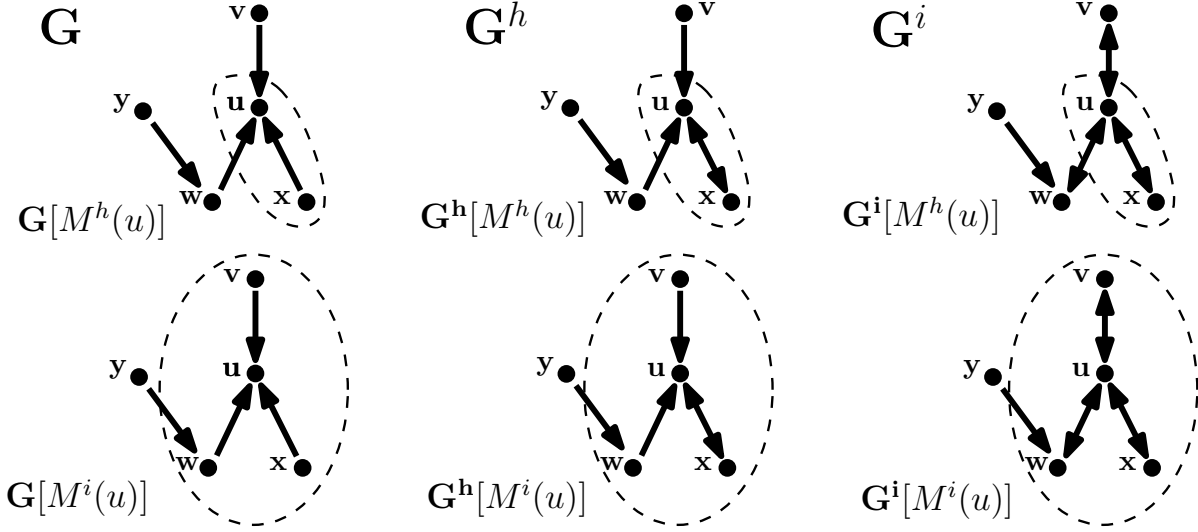


FIGURE 6.3 – Illustration des graphes $\mathbf{G}[M^h(u)]$ et $\mathbf{G}[M^i(u)]$ lors d'une exécution sur un graphe $\mathbf{G}$ où $h < i$, $M^i(u) \neq M^h(u)$ et $a^h(u) = a^i(u) > 1$. Les graphes $\mathbf{G}[M^h(u)]$, $\mathbf{G}^h[M^h(u)]$ et $\mathbf{G}^i[M^h(u)]$ (resp. $\mathbf{G}[M^i(u)]$, $\mathbf{G}^h[M^i(u)]$ et $\mathbf{G}^i[M^i(u)]$) sont représentés en haut (resp. en bas) par une ligne en pointillés. Ici, le processus y n'a toujours pas reçu de message en provenance de w lors des étapes h et i et donc, y n'appartient ni à $\mathbf{G}^h[M^h(u)]$, ni à $\mathbf{G}^i[M^i(v)]$. Pour les mêmes raisons, w n'appartient pas à $\mathbf{G}^h[M^h(u)]$ mais appartient à $\mathbf{G}^i[M^i(v)]$.

$(n^i(v), q, p) \in N^h(w)$. Par la condition (2) du lemme 6.49, $(p, a^h(w)) \in A^i(v)$ et par le lemme 6.48, $a^h(w) > a^i(v) - 1 \geq 1$. Par conséquent, ce cas est prouvé.

Supposons l'hypothèse de récurrence $HR(k)$ suivante. Pour tout $w \in V(\mathbf{G}^i[M^i(v)])$ et pour toute étape $h < i$ telle que $\bar{d}_{\mathbb{T}(M^i(v))}(n^i(v), n^h(w)) \leq k$, si $k < a^i(v)$ alors $M^i(v) = M^h(w)$ et $a^h(w) > a^{i+1}(v) - \bar{d}_{\mathbb{T}(M^i(v))}(n^i(v), n^h(w)) \geq 0$. Nous montrons maintenant que pour tout sommet $w \in V(\mathbf{G}^i[M^i(v)])$ et pour toute étape $h < i$ telle que $\bar{d}_{\mathbb{T}(M^i(v))}(n^i(v), n^h(w)) = k + 1$, si $a^i(v) > k + 1$, alors $M^i(v) = M^h(w)$ et $a^h(w) > a^{i+1}(v) - \bar{d}_{\mathbb{T}(M^i(v))}(n^i(v), n^h(w))$.

Supposons donc que $a^i(v) > k + 1$. Comme la carte $\mathbb{T}(M^i(v))$ est connexe, pour tout processus $\mathbf{m} \in \mathbb{T}(M^i(v))$, si $\bar{d}_{\mathbb{T}(M^i(v))}(n^i(v), \mathbf{m}) = k + 1$ alors il existe un processus $\mathbf{m}' \in V(\mathbb{T}(M^i(v)))$ tel que $\bar{d}_{\mathbb{T}(M^i(v))}(n^i(v), \mathbf{m}') = k$ et $\bar{d}_{\mathbb{T}(M^i(v))}(\mathbf{m}', \mathbf{m}) = 1$. Par la condition (3) du lemme 6.49, il existe un processus w et une étape $h < i$ telle que $n^h(w) = \mathbf{m}$ et il existe un processus w' et une étape $j < i$ telle que $n^j(w') = \mathbf{m}'$. De plus, $(n^h(w), p, q) \in N^j(w')$ et $(n^j(w'), q, p) \in N^h(w)$.

Comme $\bar{d}_{\mathbb{T}(M^i(v))}(n^i(v), n^j(w')) < a^i(v) = k + 1$, nous obtenons que $M^h(w) = M^i(v)$ et $a^j(w') > a^i(v) - k$ par hypothèse de récurrence. Par définition de $\mathbf{G}^i[M^j(w')]$, $n^h(w) \in V(\mathbf{G}^i[M^j(w')])$. Par conséquent, comme $a^j(w') > a^i(v) - k > 1$ et comme $\bar{d}_{\mathbb{T}(M^i(v))}(n^j(w'), n^h(w)) = 1$, en appliquant le premier cas entre $n^j(w')$ et $n^h(w)$, nous obtenons que $\bar{d}_{\mathbb{T}(M^i(v))}(n^j(w'), n^h(w)) < a^j(w')$, $M^j(w') = M^i(v)$ et $a^h(w) > a^j(w') - 1$.

Par conséquent, comme $a^j(w') > a^i(v) - k > 1$, nous obtenons que $a^h(w) > (a^i(v) - k) - 1 = a^i(v) - (k + 1) \geq 0$. Comme $M^h(w) = M^j(w') = M^i(v)$, ce lemme est prouvé. $\square$

Nous listons maintenant les propriétés satisfaites par tout processus qui a atteint un état stable, i.e., son rayon de confiance a atteint la borne B .

Proposition 6.55. *Pour tout entier B , pour tout graphe $\mathbf{G}$ tel que $|V(\mathbf{G})| < B$, pour tout processus v , si lors d'une étape i , $a^i(v) > B$ alors pour tout $(n, l, N) \in M^i(v)$, les propriétés*

suivantes sont satisfaites :

- (i) **Cohérence.** Il existe un processus $w \in V(\mathbf{G}^i[M^i(v)])$ et une étape $h < i$ telle que $M^h(w) = M^i(v)$, $N^h(w) = N$, $n^h(w) = n$, $l(w) = l$ et $a^h(w) > 0$.
- (ii) **Symétrie.** Pour tout $(m, p, q) \in N$, il existe $(m, l', N') \in M^i(v)$ et $(n, q, p) \in N'$.

Démonstration. Nous prouvons les propriétés une par une.

- (i) Par les lemmes 6.45 et 6.54.
- (ii) Par le lemme 6.45, pour tout $(n, l, N) \in M^i(v)$, il existe un processus $w \in V(\mathbf{G}^i[M^i(v)])$ et une étape $h < i$ telle que $n^h(w) = n$, $l_w = l$ et $N^h(w) = N$. Comme $a^i(v) > B$ (Lemme 6.54), $a^h(w) > 1$ et $M^h(w) = M^i(v)$. Par le point (2) de la proposition 6.49 appliqué à w , pour tout $(m, p, q) \in N^h(w)$, il existe un processus u et une étape $j < h$ telle que $M^j(u) = M^h(w)$, $m = n^j(u)$ et $(n^j(u), l_u, N^j(u)) \in M^h(w)$. Comme $(n^j(u), l_u, N^j(u)) \in M^h(w) = M^i(v)$ alors $u \in V(\mathbf{G}^i[M^i(v)])$ et par le lemme 6.54, $a^j(u) > 1$. Par le point (3) de la proposition 6.49 appliqué à u , il existe $(n^h(w), q, p) \in N^j(u)$.

□

Nous prouvons maintenant que si un processus v possède un rayon de confiance strictement supérieur à notre borne B lors d'une étape i , le sous-graphe $\mathbf{G}^i[M^i(v)]$ est clos par successeurs dans $\mathbf{G}^i$.

Lemme 6.56. *Pour toute exécution de l'algorithme sur un graphe $\mathbf{G}$ tel que $|V(\mathbf{G})| \leq B$, pour toute étape i et tout processus v , si $a^i(v) > B$, alors $\mathbf{G}^i[M^i(v)] \subseteq \downarrow \mathbf{G}^i$.*

Démonstration. Soit $B \in \mathbb{N}$, soit i une étape d'une exécution sur un graphe G tel que $|V(G)| < B$ et soit v un processus tel que $a^i(v) > B$. Raisonnons par l'absurde et supposons qu'il existe un arc $a_{u,w,p,q} \in A(\mathbf{G}^i)$ tel que $u \in V(\mathbf{G}^i[M^i(v)])$ et $w \notin V(\mathbf{G}^i[M^i(v)])$. Notons que par définition, il existe $(n, p, q) \in N^i(u)$.

Par la condition (i) de la proposition 6.55, il existe une étape $j < i$ telle que $M^j(u) = M^i(v)$ et $(n^j(u), l_u, N^j(u)) \in M^i(v)$. Pour les mêmes raisons, il existe une étape $h < i$ telle que $M^h(w) = M^i(v)$ et $(n^h(w), l_w, N^h(w)) \in M^i(v)$. Comme $a^i(v) > B$, $a^j(u) > 1$ (Lemme 6.54) et donc, par la condition (ii) de la proposition 6.55, $(n^h(w), \delta_u(w), \delta_w(u)) \in N^j(u)$ et $(n^j(u), \delta_w(u), \delta_u(w)) \in N^h(w)$.

Comme $(n^j(u), l_u, N^j(u)) \in M^i(v)$ et comme $(n^h(w), \delta_u(w), \delta_w(u)) \in N^j(u)$, nous en déduisons que $(n^h(w), l_w, N^h(w)) \in M^i(v)$. Comme $M^h(w) = M^i(v)$, $w \in (\mathbf{G}^i[M^i(v)])$ et nous obtenons une contradiction. □

Par la proposition 1.50, $\mathbf{G}[M^i(v)] \subseteq \mathbf{G}$, $\mathbf{G}^i[M^i(v)] \subseteq \mathbf{G}^i$ et $\mathbf{G} \subseteq \mathbf{G}^i$. Par la définition de $\subseteq \downarrow$ et par la proposition 1.21, nous obtenons le corollaire suivant.

Corollaire 6.57. *Pour toute exécution de l'algorithme sur un graphe $\mathbf{G}$ tel que $|V(\mathbf{G})| \leq B$, pour toute étape i et tout processus v , si $a^i(v) > B$ alors $\mathbf{G}[M^i(v)] \subseteq \downarrow \mathbf{G}$.*

Nous montrons maintenant que si le rayon de confiance d'un processus v dépasse la borne B donnée lors d'une étape i , alors le graphe $\mathbf{G}^i[M^i(v)]$ est un revêtement du graphe induit par la boîte aux lettres du processus v restreint aux voisins initiaux.

Lemme 6.58. *Pour tout processus v dans un graphe $\mathbf{G}$ tel que $|V(\mathbf{G})| \leq B$, si lors d'une étape i d'une exécution de l'algorithme sur $\mathbf{G}$, $a^i(v) > B$ alors $\mathbf{G}^i[M^i(v)]$ est un revêtement dirigé de $\mathbb{T}(M^i(v))$.*

Démonstration. Soit i , une étape d'une exécution de l'algorithme sur un digraphe $\mathbf{G}$ tel que $|V(\mathbf{G})| \leq B$. Soit v un processus de $\mathbf{G}$ possédant un rayon de confiance supérieur à B lors de cette étape, i.e., $a^i(v) > B$.

Pour montrer ce lemme, nous définissons en premier un morphisme $\varphi : \mathbf{G}^i[M^i(v)] \rightarrow \mathbb{T}(M^i(v))$ puis nous montrons que φ est un revêtement dirigé de $\mathbf{G}^i[M^i(v)]$ vers $\mathbb{T}(M^i(v))$.

Soit $\varphi : \mathbf{G}^i[M^i(v)] \rightarrow \mathbb{T}(M^i(v))$ la fonction qui, pour tout sommet $w \in V(\mathbf{G}^i[M^i(v)])$, associe le sommet $n^h(w) \in \mathbb{T}(M^i(v))$ avec $h < i$ l'étape telle que $M^h(w) = M^i(v)$ et $(n^h(w), l_w, N^h(w)) \in M^i(v)$. De plus, pour tout arc $a_{u,w,p,q} \in A(\mathbf{G}^i[M^i(v)])$, $\varphi(a_{u,w,p,q}) = a_{n^j(u), n^h(w), p, q} \in \mathbb{T}(M^i(v))$ avec $n^j(u) = \varphi(u)$ et $n^h(w) = \varphi(w)$.

Par définition de $\mathbb{T}(M^i(v))$ et par le lemme 6.54, tout sommet $w \in V(\mathbf{G}^i[M^i(v)])$, $\varphi(w)$ est bien défini. Remarquons que le graphe $\mathbb{T}(M^i(v))$ peut posséder des boucles et arcs multiples si le graphe $\mathbf{G}$ n'est pas minimal pour les revêtements.

Pour montrer que φ est un morphisme, nous montrons que pour tout arc $a_{u,w,p,q} \in A(\mathbf{G}^i[M^i(v)])$, il existe un arc $a_{\varphi(u), \varphi(w), p, q} = \varphi(a_{u,w,p,q}) \in A(\mathbb{T}(M^i(v)))$.

Par définition de $\mathbf{G}^i[M^i(v)]$, pour tout arc $a_{u,w,p,q} \in A(\mathbf{G}^i[M^i(v)])$, il existe une étape $j \leq h \leq i$ telle que $M^h(w) = M^j(u) = M^i(v)$ et $(n^h(w), l_w, N^h(w)), (n^j(u), l_u, N^j(u)) \in M^i(v)$. Comme $a^i(v) > B > |\mathbf{G}| \geq |\mathbf{G}^i[M^i(v)]|$, par le lemme 6.54, $a^j(u) > 1$ et $a^h(w) > 1$. Par définition de φ , $\varphi(u) = n^j(u)$ et $\varphi(w) = n^h(w)$.

Par le point (i) de la proposition 6.55, $(n^h(w), p, q) \in N^j(u)$ et comme $(n^j(u), l_u, N^j(u)) \in M^i(v)$, il existe $a_{n^j(u), n^h(w), p, q} \in A(\mathbb{T}(M^i(v)))$. Par conséquent, $\varphi(a_{u,w,p,q}) = a_{n^j(u), n^h(w), p, q}$ est bien défini pour tout arc $a_{u,w,p,q} \in A(\mathbf{G}^i[M^i(v)])$. La fonction φ est donc un morphisme qui, de plus, préserve l'étiquetage des ports.

Pour montrer que $\varphi : \mathbf{G}^i[M^i(v)] \rightarrow \mathbb{T}(M^i(v))$ est un revêtement dirigé, nous montrons en premier que φ est localement injectif pour les arcs sortant uniquement. La preuve s'appliquera de façon symétrique aux arcs entrants. Puis nous montrerons que φ est localement surjectif.

L'étiquetage des ports de $\mathbf{G}$ étant injectif, pour tout sommet $u \in V(\mathbf{G}^i[M^i(v)])$ et pour tout arc $a_{u,x,p,q}, a_{u,w,p',q'} \in A(\mathbf{G}^i[M^i(v)])$, $p \neq p'$. Comme nous venons de prouver que φ est un morphisme préservant l'étiquetage des ports, il existe donc deux arcs $a_{\mathbf{n}, \mathbf{m}, p, q}$ et $a_{\mathbf{n}, \mathbf{m}', p', q'}$ dans $A(\mathbb{T}(M^i(v)))$ et $p \neq p'$. Notons que possiblement $\mathbf{m} = \mathbf{m}'$ en cas d'arêtes multiples. Comme par définition de φ , $\varphi(a_{u,x,p,q}) = a_{\mathbf{n}, \mathbf{m}, p, q} \neq \varphi(a_{u,w,p',q'}) = a_{\mathbf{n}, \mathbf{m}', p', q'}$, le morphisme φ est bien localement injectif pour les arcs entrants. Avec le même raisonnement, le morphisme φ est localement injectif pour les arcs sortants.

Pour montrer que φ est localement surjectif, nous montrons que pour tout sommet $\mathbf{n} \in A(\mathbb{T}(M^i(v)))$ et pour tout sommet $u \in \varphi^{-1}(\mathbf{n}) \subseteq V(\mathbf{G}^i)$,

- pour tout arc $a_{\mathbf{n}, \mathbf{m}, p, q} \in A(\mathbb{T}(M^i(v)))$, il existe un unique sommet $w \in V(B_{\mathbf{G}^i[M^i(v)]}(u, 1))$ et un unique arc $a_{u,w,p,q} \in A(\mathbf{G}^i[M^i(v)])$ tel que $\varphi(a_{u,w,p,q}) = a_{\mathbf{n}, \mathbf{m}, p, q}$.
- pour tout arc $a_{\mathbf{m}, \mathbf{n}, q, p} \in A(\mathbb{T}(M^i(v)))$ il existe un unique sommet $w \in V(B_{\mathbf{G}^i[M^i(v)]}(u, 1))$ et un unique arc $a_{w,u,q,p} \in A(\mathbf{G}^i[M^i(v)])$ tel que $\varphi(a_{w,u,q,p}) = a_{\mathbf{m}, \mathbf{n}, q, p}$.

Considérons un sommet $\mathbf{n} \in A(\mathbb{T}(M^i(v)))$, un sommet $u \in V(\mathbf{G}^i[M^i(v)])$ tel que $\mathbf{n} = \varphi(u)$ et un arc $a_{\mathbf{n}, \mathbf{m}, p, q} \in A(\mathbb{T}(M^i(v)))$. Par définition de $\mathbb{T}(M^i(v))$, il existe $(\mathbf{n}, l, N), (\mathbf{m}, l', N') \in M^i(v)$. Par le lemme 6.45, il existe un processus $w \in V(\mathbf{G}^i[M^i(v)])$ et un arc $a_{u,w,p,q} \in A(\mathbf{G}^i[M^i(v)])$ tel que $\varphi(w) = \mathbf{m}$. Comme $\mathbf{G}^i$ possède un étiquetage injectif des ports, pour tout arc $a_{u,x,p',q'} \in A(\mathbf{G}^i[M^i(v)])$, $p' \neq p$. Ainsi, pour tout processus $u \in V(\mathbf{G}^i[M^i(v)])$ tel que $\varphi(u) = \mathbf{n}$, pour tout arc $a_{\mathbf{n}, \mathbf{m}, p, q} \in A(\mathbb{T}(M^i(v)))$, il existe un unique arc $a_{u,w,p,q} \in A(\mathbf{G}^i[M^i(v)])$ tel que $\varphi(a_{u,w,p,q}) = a_{\mathbf{n}, \mathbf{m}, p, q} \in A(\mathbb{T}(M^i(v)))$. Par conséquent, φ est un morphisme localement surjectif pour les arcs sortants. Par le même raisonnement, nous pouvons montrer que φ est un morphisme localement

surjectif pour les arcs entrants. Nous obtenons donc que $\varphi : \mathbf{G}^i[M^i(v)] \rightarrow \mathbb{T}(M^i(v))$ est un morphisme localement injectif et localement surjectif et donc φ est un revêtement dirigé. $\square$

Lemme 6.59. *Pour tout processus v dans un graphe $\mathbf{G}$ tel que $|V(\mathbf{G})| \leq B$, si lors d'une étape i d'une exécution de l'algorithme sur $\mathbf{G}$, $a^i(v) > B$ alors $\mathbf{G}^i[M^i(v)]$ est un graphe symétrique et $\varphi : \mathbf{G}^i[M^i(v)] \rightarrow \mathbb{T}(M^i(v))$ un revêtement dirigé symétrique.*

Démonstration. Ce lemme est obtenu par définition des revêtements et par la condition (ii) du lemme 6.55. En effet, pour toute exécution sur un graphe $\mathbf{G}$ tel que $|V(\mathbf{G})| < B$, si un processus v possède un rayon de confiance $a^i(v) > B$ alors pour tout sommet $u \in V(\mathbf{G}^i[M^i(v)])$, il existe une étape $h < i$ telle que $M^h(u) = M^i(v)$ et $a^h(u) > 1$. Donc, pour tout arc $a_{u,w,p,q} \in A(\mathbf{G}^i[M^i(v)])$, il existe un arc $a_{w,u,q,p} \in A(\mathbf{G}^i[M^i(v)])$.

Nous montrons maintenant que $\varphi : \mathbf{G}^i[M^i(v)] \rightarrow \mathbb{T}(M^i(v))$ est un revêtement symétrique. Par définition des graphes symétriques, pour tout $a_{u,w,p,q} \in A(\mathbf{G}^i[M^i(v)])$, $\sigma(a_{u,w,p,q}) = a_{w,u,p,q}$. Ainsi, nous obtenons par définition de φ que $\varphi(a_{u,w,p,q}) = a_{\varphi(u),\varphi(w),q,p} \in A(\mathbb{T}(M^i(v)))$. Nous pouvons donc définir sans ambiguïté la fonction σ de la façon suivante. Pour tout $a_{u,w,p,q} \in A(\mathbf{G}^i[M^i(v)])$, $\sigma(a_{\varphi(u),\varphi(w),p,q}) = a_{\varphi(w),\varphi(u),q,p} \in A(\mathbb{T}(M^i(v)))$. Ce lemme est prouvé. $\square$

Nous montrons maintenant que φ reste un revêtement si nous restreignons les graphes $\mathbf{G}^i[M^i(v)]$ et $\mathbb{T}(M^i(v))$ aux arcs présents dans le graphe de communication initial $\mathbf{G}$.

Lemme 6.60. *Pour tout processus v dans un graphe $\mathbf{G}$ tel que $|V(\mathbf{G})| \leq B$, si lors d'une étape i d'une exécution de l'algorithme sur $\mathbf{G}$, $a^i(v) > B$ alors $\varphi : \mathbf{G}[M^i(v)] \rightarrow \mathbb{T}^0(M^i(v))$ un revêtement dirigé.*

Démonstration. Considérons un graphe $\mathbf{G}$ tel que $|V(\mathbf{G})| \leq B$, un processus $v \in V(\mathbf{G})$ tel que $a^i(v) > B$ et un revêtement dirigé $\varphi : \mathbf{G}^i[M^i(v)] \rightarrow \mathbb{T}(M^i(v))$. Nous montrons que pour tout arc $a_{u,w,p,q} \in A(\mathbf{G}^i[M^i(v)] \setminus \mathbf{G}[M^i(v)])$, $\varphi(a_{u,w,p,q}) \in A(\mathbb{T}(M^i(v)) \setminus \mathbb{T}^0(M^i(v)))$.

Pour tout arc $a_{u,w,p,q} \in A(\mathbf{G}^i[M^i(v)] \setminus \mathbf{G}[M^i(v)])$, $a_{u,w,p,q} \notin A(\mathbf{G}[M^i(v)])$, et donc $p \notin \text{CONTACTS}^0(u)$. Notons $\mathbf{n} = \varphi(u) \in V(\mathbb{T}^0(M^i(v)))$ l'image de u par φ . Comme φ préserve les étiquetages et comme $(\mathbf{n}, l, N) \in M^i(v)$ nous obtenons que $l(u) = l$ et $\text{CONTACTS}^0(u) \in l$. Comme $p \notin \text{CONTACTS}^0(u)$, nous en déduisons que $a_{\mathbf{n},\mathbf{m},p,q} \notin A(\mathbb{T}^0(M^i(v)))$ par définition de $\mathbb{T}^0(M^i(v))$ et donc pour tout arc $a_{u,w,p,q} \in A(\mathbf{G}^i[M^i(v)] \setminus \mathbf{G}[M^i(v)])$, $\varphi(a_{u,w,p,q}) \in A(\mathbb{T}(M^i(v)) \setminus \mathbb{T}^0(M^i(v)))$. Par conséquent, $\varphi : \mathbf{G}[M^i(v)] \rightarrow \mathbb{T}(M^i(v))$ est un revêtement dirigé. $\square$

Nous montrons maintenant que toute exécution se stabilise et que tout processus décide une valeur finale.

Lemme 6.61. *Soit $\mathcal{F}$ une famille de graphes satisfaisant $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}-\mathcal{AN}\mathcal{O}}$. Pour toute exécution de l'algorithme sur un digraphe $\mathbf{G} \in \mathcal{F}$, pour tout processus $v \in V(\mathbf{G})$, il existe une étape i telle que $\mathbb{T}^0(M^i(v)) \simeq \mathbf{G}$ et $a^i(v) > B$.*

Démonstration. Dans toute exécution, le contenu des structures $N^i(v)$, $M^i(v)$ et $n^i(v)$ d'un processus v lors d'une étape i est croissant et ne peut prendre d'un nombre fini de valeurs (Lemme 6.44). Il existe donc une étape i_0 telle que pour toute étape $i \geq i_0$ et pour tout processus $v, w \in V(\mathbf{G})$, $M^{i_0}(v) = M^i(v) = M^i(w)$. De plus, pour toute étape $i > i_0$, v ne réinitialisera ni son rayon de confiance, ni les rayons de confiance de ses voisins (structure $A^i(v)$) jusqu'à la fin de l'exécution (par l'algorithme). Par conséquent, à partir de l'étape i_0 , le rayon de confiance du processus v ne peut qu'augmenter jusqu'à atteindre la borne B lors d'une étape $i > i_0$. Lors d'une telle étape i , comme $\mathbf{G}^i[M^i(v)]$ est un revêtement dirigé de $\mathbb{T}(M^i(v))$ (Lemme 6.58). Par le lemme 6.60, $\mathbf{G}[M^i(v)]$ est un revêtement dirigé de $\mathbb{T}^0(M^i(v))$. Comme lors d'une telle étape

i , pour tout $w \neq v \in V(\mathbf{G})$, $M^i(v) = M^i(w)$, nous obtenons que $|V(\mathbf{G}^i[M^i(v)])| = |V(\mathbf{G})|$. Comme le graphe $\mathbf{G} \in \mathcal{F}$ et comme $\mathcal{F}$ satisfait $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}-\mathcal{AN}\mathcal{O}}$, $\mathbf{G}$ est minimal pour les revêtements et donc $\varphi : \mathbf{G}[M^i(v)] \rightarrow \mathbb{T}^0(M^i(v))$ est un isomorphisme et $a^i(v) > B$. Ce lemme est prouvé. $\square$

Grâce aux lemmes précédents, nous prouvons qu'au moins un processus est élu dans toute exécution de l'algorithme.

Lemme 6.62. *Pour toute famille de graphes $\mathcal{F}$ satisfaisant $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}-\mathcal{AN}\mathcal{O}}$, pour toute exécution ρ de l'algorithme sur un digraphe $\mathbf{G} \in \mathcal{F}$, il existe une étape i de ρ telle que pour tout processus v de $\mathbf{G}$ et pour toute étape $j \geq i$, $out^j(v) \neq \perp$ et il existe au moins un processus tel que $out^j(v) = \text{élu}$.*

Démonstration. Soit $\mathcal{F}$ une famille de graphes satisfaisant $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}-\mathcal{AN}\mathcal{O}}$ et soit $\mathbf{G} \in \mathcal{F}$ un graphe. Premièrement, par le lemme 6.61, pour toute exécution de l'algorithme sur $\mathbf{G}$, il existe une étape i telle que pour tout processus $v, w \in V(\mathbf{G})$, $M^i(v) = M^i(w)$, $a^i(v) > B$, $a^i(w) > B$ et $\mathbb{T}^0(M^i(v)) \simeq \mathbb{T}^0(M^i(w)) \simeq \mathbf{G}$ car $\mathcal{F}$ satisfait $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}-\mathcal{AN}\mathcal{O}}$ et $\mathbf{G} \in \mathcal{F}$. Sans perte de généralité, supposons qu'aucun processus ne soit élu lors de cette étape (sinon le lemme est prouvé). Comme $\mathbb{T}^0(M^i(w)) \simeq \mathbf{G}$, par le lemme 6.46, il existe un unique processus w ayant l'identité 1. Soit $j > i$, une étape où w est actif, $a^j(w) > B$ et $M^j(w) = M^i(w) = M^i(v)$. Comme $\mathbb{T}^0(M^j(w)) \simeq \mathbf{G}$ et $M^j(w) = M^i(w) = M^i(v)$, $n^j(w) = 1$ et comme $\mathbb{T}^0(M^i(v)) \simeq \mathbf{G} \in \mathcal{F}$, $\mathbb{T}^0(M^j(w)) \in \mathcal{F}$. Par conséquent, lors de l'étape j , comme $out^j(w) = \perp$ (aucun processus n'a décidé par hypothèse), w satisfait toutes les conditions de décisions présentes à la ligne 27. Ainsi, w décidera élu et ce lemme est prouvé. $\square$

Remarque 6.63. Un processus peut voir son rayon de confiance dépasser la taille du graphe et néanmoins avoir une boîte aux lettres ne contenant pas l'état de tous les processus.

Nous prouvons que pour toute famille vérifiant $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}-\mathcal{AN}\mathcal{O}}$ et pour toute exécution dans un graphe de cette famille, il y aura toujours au plus un processus élu.

Lemme 6.64. *Pour toute famille de graphes $\mathcal{F}$ satisfaisant $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}-\mathcal{AN}\mathcal{O}}$, pour toute exécution de l'algorithme 5 sur un digraphe $\mathbf{G} \in \mathcal{F}$, il existe au plus un processus $v \in V(\mathbf{G})$ tel que $out_v = \text{élu}$.*

Démonstration. Soit $\mathcal{F}$, une famille de graphes satisfaisant $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}-\mathcal{AN}\mathcal{O}}$. Par l'absurde, supposons que lors d'une exécution de l'algorithme sur un graphe $\mathbf{G} \in \mathcal{F}$, deux processus u et v soient élus respectivement lors d'une étape i et j , w.l.o.g. $i < j$. Par l'algorithme, les propriétés suivantes sont satisfaites :

1. $a^i(u) > B$ et $a^j(v) > B$
2. $\mathbb{T}^0(M^i(u)) \in \mathcal{F}$ et $\mathbb{T}^0(M^j(v)) \in \mathcal{F}$
3. il n'existe pas d'entrée $(n, l, M, \text{élu})$ ni dans $M^i(v)$, ni dans $M^j(v)$.
4. $n^i(u)$ est minimal parmi les identités présentes dans $M^i(v)$ et $n^j(v)$ est minimal parmi les identités présentes dans $M^j(v)$.
5. $\mathbf{G}[M^i(u)] \subseteq \downarrow \mathbf{G}$ et $\mathbf{G}[M^j(v)] \subseteq \downarrow \mathbf{G}$ (Lemme 6.57)

Par le lemme 6.60, $\mathbf{G}[M^i(u)]$ est un revêtement de $\mathbb{T}^0(M^i(u))$ et $\mathbf{G}[M^j(v)]$ est un revêtement de $\mathbb{T}^0(M^j(v))$. Comme u et v ont décidé, $\mathbb{T}^0(M^i(u))$ et $\mathbb{T}^0(M^j(v))$ appartiennent à $\mathcal{F}$.

Ainsi, par la condition 3 de $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}-\mathcal{AN}\mathcal{O}}$ ($\mathcal{F}$ satisfait $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}-\mathcal{AN}\mathcal{O}}$), $\mathbb{T}^0(M^j(v))$ et $\mathbf{G}[M^j(v)]$ (resp. $\mathbb{T}^0(M^i(u))$ et $\mathbf{G}[M^i(u)]$) sont isomorphes et donc $|V(\mathbb{T}^0(M^j(v)))| = |V(\mathbf{G}[M^j(v)])|$ (resp. $|V(\mathbb{T}^0(M^i(u)))| = |V(\mathbf{G}[M^i(u)])|$). Nous obtenons que tout processus x de $V(\mathbf{G}[M^j(v)])$

(resp. $x \in V(\mathbf{G}[M^i(u)])$) possède une unique identité n_x au sein de $V(\mathbb{T}(M^j(v)))$ (resp. $n_x \in V(\mathbb{T}(M^i(u)))$).

Observons que par la condition 2 de $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}-\mathcal{AN}\mathcal{O}}$, $\mathbf{G}[M^i(u)] \cap \mathbf{G}[M^j(v)] \neq \emptyset$. Par conséquent, il existe un processus $w \in V(\mathbf{G}[M^i(u)] \cap \mathbf{G}[M^j(v)])$.

Soit $h \leq i$ une étape telle que $M^h(w) = M^i(u)$, soit $h' \leq j$ une étape telle que $M^{h'}(w) = M^j(v)$ et soit ℓ une étape où $M^\ell(u) = M^j(v)$.

Si $M^i(u) = M^j(v)$ alors il existe $(n^i(u), l_u, N^i(u), out) \in M^j(v)$. Comme nous avons montré que chaque processus possède une identité unique dans sa boîte aux lettres, nous obtenons que $n^i(u) \neq n^j(v)$ (Lemme 6.43). Comme un processus élu possède l'identité 1, u et v ne peuvent donc pas être élus tous les deux dans cette exécution.

Si $M^i(u) \neq M^j(v)$, alors deux cas existent.

Si $u \notin V(\mathbf{G}[M^j(v)])$ alors comme $M^{h'}(w) = M^j(v)$, $u \notin V(\mathbf{G}[M^{h'}(w)])$ et comme $M^h(w) = M^i(u)$, $u \in V(\mathbf{G}[M^h(w)])$. Comme le contenu des boîtes aux lettres est croissant, nous en déduisons que $M^{h'}(w) \subseteq M^h(w)$ et donc que $h' < h$. Or, comme $i < j$ et comme $M^h(w) = M^i(u) \neq M^j(v) = M^{h'}(w)$, nous en déduisons que $M^h(w) \subseteq M^{h'}(w)$ et par conséquent, nous obtenons que $M^h(w) = M^i(u) = M^j(v) = M^{h'}(w)$, une contradiction.

Si $u \in V(\mathbf{G}[M^j(v)])$ alors $M^i(u) = M^h(w) \subseteq M^{h'}(w) = M^j(v)$ (contenu des boîtes aux lettres croissant) et donc $\mathbf{G}[M^i(u)] \subseteq \mathbf{G}[M^j(v)]$ (via le revêtement). Comme $a^j(v) > B$, il existe une étape $\ell < j$ et une entrée $(n^\ell(u), l_u, M^\ell(u)) \in M^j(v)$ telle que $M^\ell(u) = M^j(v)$. Comme u et v sont élus respectivement lors des étapes i et j , il existe $(n^i(u), l_u, M^i(u), \text{élu}) \in M^{i+1}(u)$ et $(n^j(v), l_v, M^j(v), \text{élu}) \in M^{j+1}(v)$. Nous obtenons une contradiction car il existe une étiquette élu dans $M^j(v)$, impliquant que v ne pourra être élu. Ainsi, dans toute exécution de l'algorithme, un seul et unique processus décidera élu. $\square$

Nous venons de prouver que pour un entier B donné, pour toute famille $\mathcal{F}$ contenant des graphes de taille inférieure à B et satisfaisant $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}-\mathcal{AN}\mathcal{O}}$, pour toute exécution de notre algorithme sur un graphe $\mathbf{G} \in \mathcal{F}$, il existe un unique processus qui décide élu et tous les autres décident non-élu. Nous obtenons le théorème suivant,

Théorème 6.1. *Pour toute famille $\mathcal{F}$ contenant des graphes de taille bornée par un entier B , il existe un algorithme $\mathcal{F}$ -restreint pour $\mathcal{EL}\mathcal{EC} - \mathcal{AN}\mathcal{O}$ si la famille $\mathcal{F}$ satisfait $\mathfrak{C}_{\mathcal{EL}\mathcal{EC}-\mathcal{AN}\mathcal{O}}$.*

Remarque 6.65. Notons que notre algorithme est silencieux au sens où il existe une étape, dans toute exécution de notre algorithme, où aucun message ne circulera et ne sera émis dans le réseau par la suite.

Conclusion

Dans ce chapitre, nous avons étendu le modèle des participants inconnus aux réseaux anonymes. Comme pour les chapitres précédents, nous avons caractérisé les connaissances partielles nécessaires pour résoudre le problème de l'élection dans des familles de graphes de taille bornée. En d'autres termes, nous avons caractérisé les familles de graphes $\mathcal{F}$ de taille bornée admettant un algorithme $\mathcal{F}$ -restreint d'élection dans ce modèle. La condition nécessaire proposée découle naturellement des résultats d'impossibilité due aux réseaux anonymes [Ang80, CGM12] ainsi que des résultats d'impossibilité spécifiques au modèle des participants inconnus. Pour résoudre le problème de l'élection, nous avons adapté l'algorithme $gSSP$ ([CGM12]) au modèle des participants inconnus anonymes. Nous proposons ainsi un algorithme $\mathcal{F}$ -restreint d'élection pour toute famille de graphes $\mathcal{F}$ de taille bornée et satisfaisant notre condition. Pour cela, les processus sont dotés d'une connaissance et d'une borne supérieure sur la taille du réseau.

Cette étude de cas nous a permis de mieux comprendre le modèle des participants inconnus dans les réseaux anonymes. Cette étude permettra, à l'avenir, d'étudier d'autres problèmes fondamentaux dans ce modèle telle la cartographie complète du réseau ou la k -élection. Des questions restent ouvertes pour ce modèle. Existe-t-il d'autres connaissances permettant de résoudre l'élection sur des familles de graphes de taille arbitrairement large ?

Il serait aussi intéressant de généraliser notre résultat à toute tâche distribuée arbitraire. Pour cela, nous pourrions proposer une condition C qui, pour une famille de graphes $\mathcal{F}$ et une spécification d'un problème $\mathcal{S}$ données, caractérise l'existence d'un algorithme $\mathcal{F}$ -restreint pour $\mathcal{S}$. Une telle caractérisation n'est pas triviale, car de multiples facteurs jouent sur la calculabilité des tâches distribuées.

Deuxième partie

Exploration de graphes anonymes

Chapitre 7

Exploration de graphes anonymes par un agent mobile

Introduction

Un agent mobile est une entité de calcul autonome se déplaçant dans un environnement. Un agent peut modéliser un robot se déplaçant sur un terrain inconnu ou un logiciel parcourant le réseau internet. Les agents mobiles sont largement utilisés de nos jours dans l'accomplissement de nombreuses tâches : exploration ou surveillance d'un environnement, collection d'informations, maintenance d'infrastructures ou négociations financières. Le paradigme en algorithmique distribuée des agents mobiles permet de déplacer l'entité de calcul directement aux données au lieu de déplacer de gros volumes de données aux entités de calcul correspondantes.

Nous considérerons un agent mobile évoluant dans un réseau anonyme. Ce réseau sera représenté par un graphe. Un agent évoluant dans un graphe sera positionné sur un sommet (ou noeud) du graphe et se déplacera via les arêtes. Les agents mobiles ont été introduits formellement par Fukuda et al. [FBDM97]. Comme soulignée dans [LO99], l'utilisation des agents mobiles est avantageuse pour de nombreuses raisons comme la robustesse contre l'instabilité des réseaux ; l'amélioration de la latence et de la charge des réseaux ; un gain d'autonomie et de complexité pour les algorithmes. Pour de nombreux problèmes spécifiques aux agents mobiles, l'exploration, i.e., visiter tous les noeuds d'un réseau, est un important prérequis. Le problème de l'exploration d'un environnement inconnu a été introduit par Shannon en 1952 [Sha52] dans le cas d'un automate à états fini, vu comme un agent mobile, explorant un labyrinthe. Dans [FRS03, CGMO06, CDK10a, CDW13] (et bien d'autres), l'approche algébrique est utilisée pour étudier la calculabilité des systèmes à agents mobiles. Dans [Das13], un état de l'art détaillé des systèmes à agents mobiles est présenté.

Nous nous focaliserons sur le problème de l'exploration de graphes anonymes sans connaissance globale. Explorer les graphes anonymes est fondamental en algorithmique distribuée pour plusieurs raisons. Un agent mobile peut avoir un capteur défaillant ou être incapable de lire les étiquettes attachées aux noeuds du réseau. Pour des raisons de sécurité, les noeuds peuvent ne pas fournir leur identité aux agents mobiles voire même mentir dans les pires scénarios. Une autre raison d'explorer les graphes anonymes est le coût de déploiement d'identités dans le réseau. Contrairement au modèle à passage de messages où les identités peuvent être déployées, en amont de l'exécution et en même temps que l'algorithme sur l'ensemble des noeuds du réseau, il n'y a pas d'initialisation du réseau en amont de l'exécution d'un agent mobile. Propager une identité dans le réseau peut donc être significativement plus coûteux dans un modèle à agent

mobile que dans un modèle à passage de messages.

Le problème de l'exploration a été très étudié, et ce, sous différents aspects. Dans [BS94, BA06], les auteurs montrent que l'exploration avec arrêt de tous les graphes dirigés et anonymes est possible en marquant les noeuds du réseau avec un jeton pouvant être déposé/récupéré sur les noeuds du réseau. D'autres approches ont été étudiées comme minimiser la mémoire requise pour résoudre le problème de l'exploration sous diverses hypothèses [DFKP04, FIP⁺05, CFI⁺05, AGP⁺11a]. Dans [AGP⁺11a], les auteurs étudient la mémoire requise pour pouvoir explorer tous les arbres anonymes sans connaissance globale. Ils montrent que $\Theta(\log(n))$ bits de mémoire nécessaires et suffisants pour explorer tout arbre de taille au plus n . Pour cela, ils proposent un algorithme qui décide si l'arbre exploré possède au plus n sommets et si oui, l'explore et s'arrête avec seulement $O(\log(n))$ bits de mémoire. Notons que l'agent retourne sur sa base en fin d'exploration. La principale difficulté de l'exploration des graphes anonymes est de pouvoir décider si un noeud sur lequel arrive l'agent a déjà été visité ou non dans le passé. Dans [CDK10a], les auteurs montrent qu'il est possible d'explorer avec arrêt tous les graphes anonymes en donnant une borne sur le nombre de sommets du graphe ainsi que leur degré maximum.

Malheureusement, explorer tous les graphes anonymes avec arrêt est impossible [Ang80] car certains graphes (de tailles arbitrairement grandes) sont indistinguables pour un agent. Un moyen d'explorer un réseau de façon déterministe dans ce contexte est donc d'effectuer une exploration infinie, c'est à dire, une exploration qui ne finira jamais mais explorera tous les noeuds du réseau à un moment donné. Plusieurs algorithmes existent réalisant ce type d'exploration : le parcours en largeur de type BFS ou bien en utilisant les séquences d'exploration universelles [AKL⁺79, Kou02, Rei08] avec des paramètres croissants. Ce processus infini est appelé *exploration perpétuelle* si l'agent visite infiniment souvent tous les noeuds du réseau. L'exploration perpétuelle est particulièrement utilisée pour la sécurité et la maintenance, i.e., où l'agent doit régulièrement vérifier que son environnement est sûr.

Une spécificité du modèle est qu'avec un unique agent mobile, le calcul est centralisé sur l'agent. La difficulté arrive donc de la répartition des données dans le réseau et de comment diriger l'agent dans le réseau. À la différence d'autres modèles où plusieurs entités de calcul sont présentes, il est évident que, avec un unique agent mobile, la détection locale et globale de la terminaison sont équivalentes.

Un algorithme distribué peut être exécuté dans n'importe quel réseau possédant les primitives nécessaires, et ce, pour toute connaissance donnée à l'agent et hypothèse émise lors de la conception de l'algorithme. Si l'on conçoit un algorithme distribué explorant tous les arbres, rien ne nous empêche d'exécuter cet algorithme dans un graphe qui n'est pas un arbre. Dans un tel cas, l'algorithme peut avoir un comportement arbitraire : il peut ne pas terminer son exécution ou s'arrêter prématurément sans avoir exploré tous les sommets du graphe. Par contre, si cet algorithme est exécuté sur un arbre, alors il visitera tous ces sommets avant de s'arrêter.

Lorsque ce que l'on explore des graphes inconnus sans connaissance, il peut être intéressant d'exécuter plusieurs algorithmes d'exploration spécifiques à certaines familles de graphes pour espérer que l'un d'entre eux soit adapté au réseau et permette à l'agent d'explorer intégralement les sommets du graphe. Dans ce contexte, il est donc important que la terminaison d'un algorithme soit synonyme de correction de l'exécution pour pouvoir composer différents algorithmes d'exploration entre eux.

Dans cette partie, nous nous intéresserons à une exploration *sûre* où la terminaison d'une exécution est synonyme de correction. Dans le chapitre 7, nous présenterons formellement le modèle à base d'agents mobiles et proposerons une caractérisation des graphes explorables par un agent mobile sans connaissance. Pour cela, nous appliquons l'approche algébrique aux agents

mobiles pour caractériser les similarités entre certains. Grâce à cette approche, nous proposons un lemme de relèvement [Ang80, CGM08, YK96c, BV99] illustrant les exécutions problématiques qui en découle et montrons qu'il existe une unique famille explorable maximale dans ce contexte : les arbres. Ces résultats appartiennent au folklore des agents mobile et nous permettent d'obtenir de montrer la similitude avec les résultats suivants.

Dans le chapitre 8, nous verrons comment des informations locales peuvent être utilisées pour étendre la calculabilité de l'exploration avec un agent mobile sans forcément avoir recours aux connaissances globales. Pour cela, nous introduisons un nouveau modèle à agent mobile, appelé agent mobile avec jumelles. Un agent doté de jumelles permet à l'agent de percevoir le graphe induit par les sommets (voisins) à distance 1 de sa position dans le réseau. Grâce à l'approche algébrique, nous caractérisons la famille maximale des graphes explorables avec jumelles, notée $\mathcal{FC}$, et proposerons un algorithme pour explorer cette famille. Nous prouverons aussi une borne inférieure sur la complexité des exécutions de tout algorithme d'exploration pour cette famille $\mathcal{FC}$. Ces résultats ont été publiés lors de la conférence DISC en 2015 [CGN15a].

Dans le chapitre 9, nous exploiterons les jumelles pour montrer que, contrairement au modèle sans jumelles, un agent mobile équipé de jumelles est capable d'explorer efficacement de larges familles de graphes de taille non bornée. Pour cela, nous proposons un algorithme explorant la famille des graphes de *Weetman* en un nombre de mouvements linéaire en la taille du graphe exploré. Cette famille contient notamment les graphes triangulés, les graphes de Johnson et certaines triangulations de la sphère. Ces résultats ont été présentés sous une forme abrégée lors de la conférence francophone ALGOTEL en 2016 [CGN16b] et a reçu le prix de la meilleure communication étudiante. Une version conférence est en cours de rédaction et une version préliminaire est accessible en ligne [CGN16a].

7.1 Définitions et Notations

Dans cette partie, nous considérerons uniquement des graphes non dirigés. Comme de nombreux outils ont déjà été introduits dans le cadre des graphes étiquetés, nous présentons, dans cette section, uniquement les définitions qui diffèrent entre graphes dirigés et non dirigés.

7.1.1 Graphes

Un graphe $G = (V, E)$ est composé d'un ensemble V de sommets et d'un ensemble E d'arêtes reliant les sommets deux à deux.

Soit G un graphe, nous noterons $V(G)$ (resp. $E(G)$) l'ensemble des sommets (resp. arêtes) de G . Si deux sommets $u, v \in V(G)$ sont reliés par une arête (i.e. adjacents ou voisins) dans G , l'arête reliant u et v sera notée uv .

7.1.2 Graphes dirigés

Un graphe étiqueté est un graphe G possédant un étiquetage de ses sommets ou de ses arêtes. L'ensemble des étiquettes possibles est noté Λ . Un tel graphe sera noté $\mathbf{G} = (G, \lambda)$, avec $\lambda : V(G) \rightarrow \Lambda$, une fonction associant à tout sommet v de $V(G)$, une étiquette $\lambda(v)$ contenue dans Λ .

La relation de sous-graphe aux graphes étiquetés comme suit.

Définition 7.1 (Sous-graphe étiqueté). Pour tout graphe étiqueté $\mathbf{H}, \mathbf{G}$, le graphe $\mathbf{H}$ est un sous-graphe de $\mathbf{G}$ (noté $\mathbf{H} \subseteq \mathbf{G}$) si $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$, et si pour tout sommet $v \in V(H)$, $\lambda_H(v) = \lambda_G(v)$.

Définition 7.2 (Sous-graphe étiqueté induit). Pour tout graphe étiqueté $\mathbf{H}$ et $\mathbf{G}$, le graphe $\mathbf{H}$ est un sous-graphe induit de $\mathbf{G}$ si $\mathbf{H} \subseteq \mathbf{G}$ et pour tout $u, v \in V(H)$, si $(u, v) \in E(G)$ alors $(u, v) \in E(H)$.

Définition 7.3 (Graphe restreint). Pour un graphe G donné et un sous-ensemble de sommets $V' \subseteq V(G)$, le graphe $G[V']$ désignera le sous-graphe de G induit par l'ensemble de sommets V' .

Définition 7.4 (Clique). Une clique de taille n est un graphe complet à n sommets, i.e., pour tout couple de sommets u et v dans un graphe complet G , l'arête uv appartient à $E(G)$.

7.1.3 Chemins, Cycles et Boucles

Définition 7.5 (Chemin). Un chemin p de longueur k dans un graphe G est une séquence de sommets $(v_0, \dots, v_k)$ telle que $v_i v_{i+1} \in E(G)$ pour tout $0 \leq i < k$. Un chemin est *simple* si pour tout $i \neq j$, $v_i \neq v_j$.

Définition 7.6. Un plus court chemin entre deux sommets u et v dans un graphe G est un chemin reliant u à v minimisant sa longueur.

Définition 7.7 (Chemin redondant). Un chemin $p = (v_1 \dots v_{i-1} v_i v_{i+1} \dots v_k)$ est redondant s'il existe un entier $1 < i < k$ tel que $v_{i-1} = v_{i+1}$.

Notons qu'un chemin qui n'est pas simple n'est pas forcément redondant.

Définition 7.8 (Cycle). Un *cycle* c de longueur k est un chemin $(v_0, \dots, v_k)$ tel que $v_0 = v_k$. Un cycle $(v_0, \dots, v_k)$ est simple s'il est vide (i.e., $k = 0$) ou si le chemin $(v_0, \dots, v_{k-1})$ est simple.

Définition 7.9 (Boucle). Une *boucle* c dans un graphe G est une séquence de sommets $(v_0, \dots, v_k) \subseteq V(G)$ telle que $v_0 = v_k$ et $v_i = v_{i+1}$ ou $v_i v_{i+1} \in E(G)$, pour tout $0 \leq i < k$.

Définition 7.10. La longueur d'une boucle $c = (v_0, \dots, v_k)$ dans un graphe G , notée $|c|$, est le nombre k de sommets qui la compose. Notons que les sommets dupliqués comptent doubles.

Définition 7.11 (Étiquetage d'un chemin). Étant donné un graphe doté d'un étiquetage des ports δ , un chemin $p = (v_0, \dots, v_k)$ sera étiqueté par $\lambda(p) = (\delta_{v_0}(v_1), \delta_{v_1}(v_2), \dots, \delta_{v_{k-1}}(v_k))$.

Définition 7.12 (Distance). La distance entre deux sommets v et v' dans un graphe G , notée $d_G(v, v')$, est la longueur du plus court chemin reliant v et v' dans G .

Définition 7.13 (Voisinage d'un sommet). Étant donné un graphe G et un sommet $v \in V(G)$, l'ensemble $N_G(v, k) \subseteq V(G)$ est l'ensemble de sommets de G à distance au plus k du sommet $v \in V(G)$.

Remarque 7.14. Nous noterons $N_G(v)$ pour identifier l'ensemble des sommets à distance 1 de v (les voisins).

Définition 7.15 (Boule). Étant donné un graphe G et un sommet $v \in V(G)$, la boule de rayon k autour de v , notée $B_G(v, k) \subseteq G$, est le sous-graphe de G induit par les sommets contenus dans $N_G(v, k)$.

Définition 7.16 (Connexe). Un graphe est *connexe* si, pour tout couple de sommets $u, v \in V(G)$, il existe un chemin dans G reliant le sommet u au sommet v .

Définition 7.17 (Arbre). Un *arbre* est un graphe connexe sans cycle.

Dans ce document, nous supposons toujours des graphes simples et connexes. Une définition alternative pour obtenir un graphe simple est d'interdire les arêtes doubles et les boucles simples (Figure 6.1).

7.1.4 Étiquetage des ports

Un étiquetage des ports est une fonction injective d'étiquetage des sommets assignant à chaque arête adjacente à un noeud donné une identité unique locale. Soit v un sommet d'un graphe G muni d'un étiquetage des ports δ et soit vw une arête de G . Nous noterons $\delta_v(w)$ le numéro de port qui, à partir du noeud v , mène au noeud w , i.e., correspondant à l'extrémité de l'arête vw attachée à v (Figure 7.1).

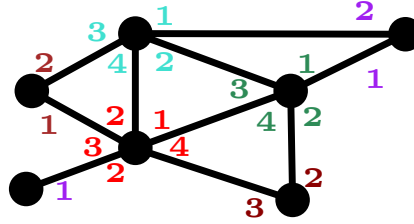


FIGURE 7.1 – Exemple de graphe anonyme doté d'un étiquetage des ports

Par définition, notons que pour toute arête $vu, vw \in E(G)$, $\delta_v(u) \neq \delta_v(w) \in \mathbb{N}$. Nous noterons par (G, δ) le graphe G doté d'un étiquetage des ports $\delta = \{\delta_v\}_{v \in V(G)}$.

Remarque 7.18. Un tel étiquetage permet aux processus de distinguer les différentes arêtes adjacentes à sa position dans le réseau. Cette information permet par exemple de calculer le degré des sommets visités.

Nous noterons $\mathcal{G}^\delta$, l'ensemble des graphes dotés d'un étiquetage des ports quelconque. Pour être le plus général possible, nous voulons que nos algorithmes soient indépendants de l'étiquetage des ports du réseau et donc, pour un graphe G donné, les algorithmes considérés devront être corrects pour tout couple $(G, \delta) \in \mathcal{G}^\delta$. Notons que lors d'une exécution donnée, les numéros de port sont fixes. Par la suite, nous considérerons uniquement des graphes dotés d'un étiquetage des ports $(G, \delta) \in \mathcal{G}^\delta$ et les noterons simplement par G .

7.1.5 Graphes anonymes et avec homonymes

Définition 7.19 (Graphes avec homonymes). Un graphe avec homonymes est un graphe étiqueté (G, ℓ) tel que chaque étiquette $\ell(v)$ n'est forcément unique dans le réseau.

Définition 7.20 (Graphes anonymes). Un graphe anonyme est un graphe ne possédant aucun étiquetage des processus. Notons qu'un graphe anonyme peut être doté d'un étiquetage des ports.

7.1.6 Homomorphisme, Revêtement et Quasi-revêtement

Pour présenter les revêtements de graphes, nous introduisons les fonctions simpliciales et les morphismes de graphes. Notons que les définitions présentées ici sont les versions non dirigées de celles proposées dans la section 1.1 du chapitre 1.

Définition 7.21 (Fonction Simpliciale). Une fonction $\varphi : V(G) \rightarrow V(H)$ entre deux graphes G et H est une *fonction simpliciale* $\varphi : G \rightarrow H$ si pour tout sommet $v \in V(G)$, $\varphi(v) \in V(H)$ et pour toute arête $uv \in E(G)$, $\varphi(u) = \varphi(v) \in V(H)$ ou $\varphi(uv) \in E(H)$.

Nous définissons maintenant les morphismes, des fonctions simpliciales qui préservent la structure des graphes pris en comptes.

Définition 7.22 (Morphisme de graphes). Une fonction simpliciale $\varphi : G \rightarrow H$ entre deux graphes G et H est un *morphisme* si pour toute arête $uv \in E(G)$, $\varphi(uv) = \varphi(u)\varphi(v)$ est une arête de $E(H)$.

Les revêtements de graphes est un outil combinatoire capturant la relation de graphes qui sont localement isomorphes.

Définition 7.23 (Revêtement). Un morphisme $\varphi : G \rightarrow H$ entre deux graphes G et H est un *revêtement de graphe* si pour tout $v \in V(G)$, $\varphi|_{N_G(v)}$ est une bijection entre $N_G(v)$ et $N_H(\varphi(v))$.

Définition 7.24 (Revêtement non trivial). Un revêtement $\varphi : G \rightarrow H$ entre deux graphes G et H est un revêtement de graphe *non trivial* si G et H ne sont pas isomorphes.

Les revêtements possèdent des propriétés intrinsèques très intéressantes présentées dans la définition et la proposition suivantes.

Théorème-Définition 7.25 ([Hat02]). Pour tout graphe G et H , un revêtement $\varphi : G \rightarrow H$ est un revêtement à n -feuilles si pour tout élément $x \in V(H)$ ou $x \in E(H)$, $\varphi^{-1}(x) = n$.

Proposition 7.26. Pour tout graphe G et H tel qu'il existe un revêtement $\varphi : G \rightarrow H$, il existe un entier n tel que φ est un revêtement à n -feuilles.

La relation de revêtement induit un élément maximal (possiblement infini) appelé revêtement universel.

Définition 7.27 (Revêtement universel [Hat02]). Le revêtement universel $\widehat{G}$ d'un graphe G est l'unique graphe (à isomorphisme près) tel que pour tout revêtement G' de G , $\widehat{G}$ est revêtement de G' .

Notons que la construction de Reidmeister présentée dans la section 1.1 du chapitre 1 nous permet d'obtenir une même caractérisation générale des arbres en fonction de leur revêtement universel.

Proposition 7.28 (Corollaire du Lemme 6.8.3 de [GR01]). Un graphe G est isomorphe à son revêtement universel $\widehat{G}$ si et seulement si G est un arbre.

En observant que la construction de Reidmeister permet d'obtenir des revêtements arbitrairement grands dès que le graphe possède au moins un triangle (ou tout cycle de longueur supérieure à 3), nous obtenons les corollaires suivants.

Corollaire 7.29. Un graphe G possède un revêtement universel $\widehat{G}$ infini si et seulement si G n'est pas un arbre.

Corollaire 7.30. Pour tout graphe G , le revêtement universel $\widehat{G}$ de G est un arbre (possiblement infini).

Nous présentons maintenant une généralisation des revêtements : les quasi-revêtements. Cette relation caractérise les graphes localement isomorphes jusqu'à une certaine distance. La définition algébrique des quasi-revêtements présentée ci-dessous a été introduite dans [MMW97]

Définition 7.31 (Quasi-revêtement). Soit G et G' deux graphes, une fonction simpliciale $\gamma : G' \rightarrow G$ est un *quasi-revêtement* de centre $u \in V(G')$ et de rayon r s'il existe un graphe H (possiblement infini) et un revêtement $\varphi : H \rightarrow G$ tel que

- il existe $v \in V(H)$ et $\psi : B_{G'}(u, r) \rightarrow B_H(v, r)$ est un isomorphisme

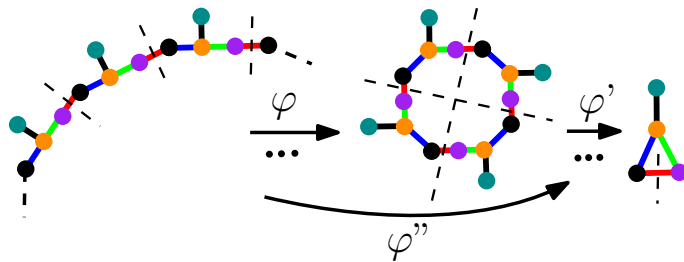


FIGURE 7.2 – Revêtements de graphes

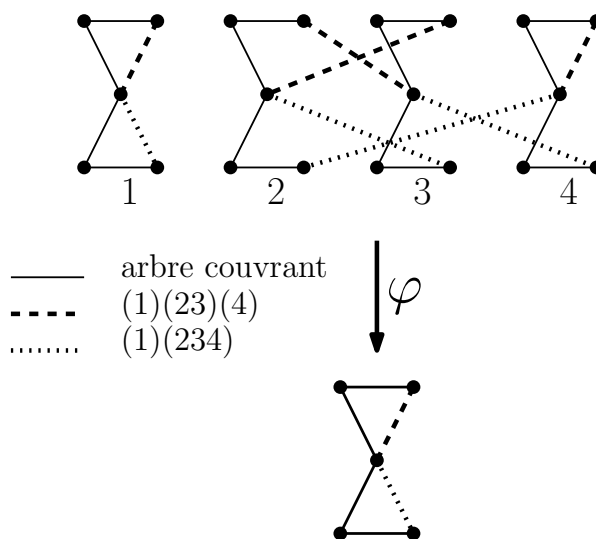


FIGURE 7.3 – Construction de Reidmeister pour les revêtements de graphes

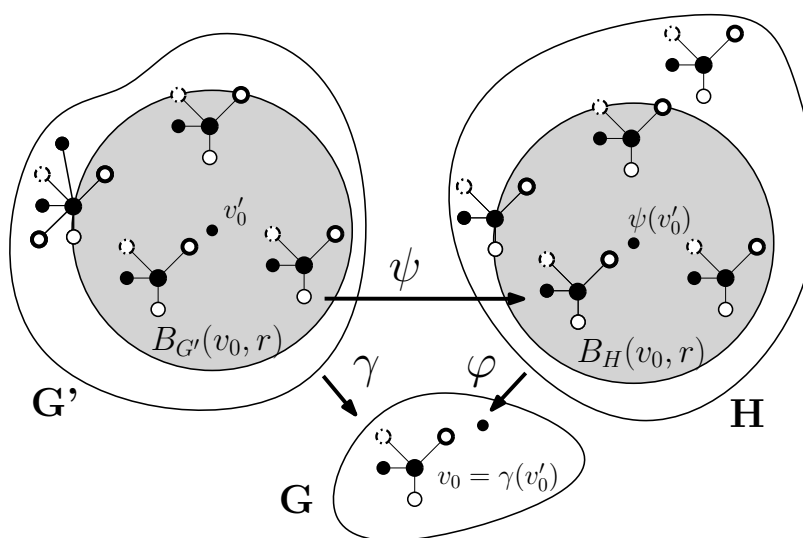


FIGURE 7.4 – Illustration des quasi-revêtements.

— $\gamma = \varphi \circ \psi$.

Définition 7.32 (Quasi-revêtement trivial). Soit G, G' deux graphes et $\gamma : G' \rightarrow G$ un quasi-revêtement de rayon $r \in \mathbb{N}$ et centré en $v \in V(G)$. Le quasi revêtement γ est trivial si G' est un revêtement de G . Sinon, γ sera un quasi revêtement non trivial.

Les définitions présentées ci-dessus s'étendent naturellement aux graphes étiquetés.

Définition 7.33 (Morphismes pour les graphes étiquetés). Pour tout étiquetage des sommets λ de $V(G)$ et λ' de $V(H)$, pour tout étiquetage des ports δ de G et δ' de H , $\varphi : V(G) \rightarrow V(H)$ est un morphisme (resp. un revêtement de graphe) de (G, δ, λ) vers (H, δ', λ') si $\varphi : G \rightarrow H$ est un morphisme (resp. un revêtement de graphe) tel que $\lambda'(\varphi(u)) = \lambda(u)$ pour tout $u \in V(G)$ et tel que $\delta_u(v) = \delta'_{\varphi(u)}(\varphi(v))$ pour toute arête $uv \in E(G)$.

7.2 Modèle standard d'agent mobile

Nous nous basons sur le modèle standard d'agents mobiles, défini formellement ci-dessous. Un agent mobile est une entité de calcul autonome se déplaçant dans un environnement. Dans ce document, l'environnement sera un graphe non dirigé $\mathbf{G} = (G, \lambda, \delta) \in \mathcal{G}^\delta$ avec δ , un étiquetage des ports et λ , un étiquetage des sommets et des arêtes. Par la suite, nous ne considérerons que des systèmes avec un unique agent.

Nous considérons des graphes sans boucle (une arête ayant ses deux extrémités reliées au même sommet) et sans double arête (deux arêtes partageant les mêmes extrémités).

L'agent évolue dans ce graphe de sommet en sommet en suivant les arêtes. Un agent mobile est une machine de Turing (avec une mémoire non bornée). De plus, comme nous considérons une complexité en nombre de mouvements effectués par l'agent, nous supposons qu'un agent accède à sa mémoire et effectue des calculs en temps constant.

Afin de pouvoir se déplacer dans le graphe, un agent positionné sur un sommet aura accès aux numéros de port adjacents à sa position comme illustré sur la figure 7.5.

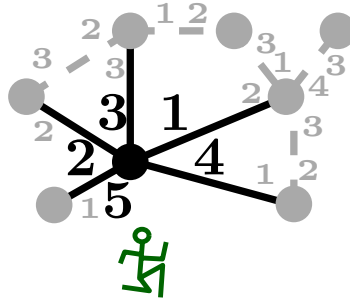


FIGURE 7.5 – Illustration de la vision locale d'un agent positionné sur un noeud du réseau. En noir, les numéros de port accessibles à l'agent.

Le comportement d'un agent est cyclique : positionné sur un noeud, l'agent regarde autour de lui et accède aux informations disponibles localement (qui dépendent du modèle, par exemple, il peut lire toutes les étiquettes attachées au noeud visité). Puis, il exécute son algorithme : il effectue des calculs, change son état interne et se déplace sur un sommet voisin en franchissant une arête. Cette arête est choisie par un numéro de port calculé par l'agent pendant la phase de calcul. Notons que dans le cadre de l'exploration, il est inutile de ne pas se déplacer lors d'une étape de l'exécution. Dans des modèles où plusieurs agents doivent résoudre une tâche

distribuée comme le rendez-vous alors il peut être intéressant de ne pas se déplacer si besoin. Nous supposons que l'agent ne peut pas marquer les noeuds du réseau (les étiquettes attachées aux noeuds sont en lecture seule). Nous supposons que le sommet de départ (appelé aussi, le sommet base de l'agent) n'est pas marqué. Ainsi, tous les noeuds sont a priori indistinguables pour l'agent (mise à part leur degré et leur étiquette).

Dans ce modèle, un agent peut "retourner en arrière" : si un agent quitte un sommet u pour un sommet v en traversant une arête $uv \in E(G)$ étiquetée (p, q) , il saura que, à partir de v , suivre le numéro de port q lui permettra de retourner sur le sommet u .

7.2.1 Problèmes et Algorithmes à base d'agent mobile

Un algorithme pour les agents mobiles peut être vu comme un système de transitions d'état fonction de l'état courant du réseau. À chaque transition d'état (appelée étape d'une exécution), un agent observe son environnement et obtient des informations à partir de sa position courante dans le réseau. Ces informations vont différer en fonction des modèles étudiés. Une fois son environnement observé, il calcule son nouvel état, sa prochaine destination et se déplace sur le sommet voisin correspondant (relié par une arête au sommet courant).

Comme un agent mobile peut être encodé dans les étiquettes λ d'un réseau, la spécification d'un problème dans ce modèle sera, de la même façon que pour le modèle à passage de messages (Définition 1.33 de la partie I), définie par une relation de réétiquetage de graphes formulée par un ensemble de règles de réécriture.

Initialement, nous avons un graphe étiqueté $\mathbf{G}$ avec une étiquette spéciale encodant l'agent. En fin d'exécution, l'étiquetage de $\mathbf{G}$ aura changé et l'étiquetage final du réseau $\mathbf{G}^{out}$ obtenu sera valide s'il satisfait la spécification $\mathcal{S}$ du problème (i.e. $\mathbf{GSG}^{out}$).

Remarque 7.34. Pour caractériser les sommets visités dans l'étiquetage du graphe, les sommets du réseau pourront être dotés d'une étiquette spéciale indiquant le passage (ou non) de l'agent. Cette étiquette servant à écrire des spécifications ne devra pas pouvoir être lue par l'agent pour respecter le modèle (les noeuds du réseau ne peuvent pas être marqués et sont "a priori" indistinguables).

7.2.2 Propriétés des exécutions et détection de la terminaison

Contrairement aux modèles acceptant plusieurs agents mobiles, un agent mobile seul sait quand la configuration globale du réseau est finale car il est seul dans le réseau. Dans le cas où plusieurs agents sont présents, détecter une étape où tous les agents ont fini leur exécution est un problème en soit.

Définition 7.35 (Terminaison). Soit $\mathcal{A}$ un algorithme exécuté par un agent mobile dans un graphe G . Une exécution ρ de $\mathcal{A}$ *termine* sur G si il existe une étape i telle que l'agent arrête de se déplacer pour toujours.

Définition 7.36 (Correction). Soit $\mathcal{A}$ un algorithme résolvant un problème $\mathcal{S}$. Une exécution ρ de $\mathcal{A}$ dans un graphe $\mathbf{G}$ est *correcte* pour $\mathcal{S}$ si ρ termine et que l'étiquetage final $\mathbf{G}^{out}$ du réseau vérifie $\mathcal{S}$, i.e., $\mathbf{GSG}^{out}$.

7.2.3 État de l'art des agents mobiles

Dans le modèle classique "look-compute-move" [KKM10, Chap. 5.6], la vision est principalement globale et les problèmes étudiés sont principalement des problèmes de coordination,

comme par exemple le rendez-vous ou le rassemblement de robots. Notons que le modèle "look-compute-move" avec une vision limitée au voisinage immédiat a été étudié pour ces problèmes (e.g. in [GP13]).

Si l'agent peut uniquement voir les étiquettes et les degrés des noeuds visités, il est connu que tout algorithme universel d'exploration s'arrête si et seulement si le graphe exploré est un arbre.

De plus, un simple parcours en profondeur (DFS) permet d'explorer tous les arbres en un temps linéaire en la taille de l'arbre, i.e., en $O(|T|)$ mouvements où T est l'arbre exploré. Gasiencic et al. [AGP⁺11b] montrent qu'un agent peut explorer tout arbre et revenir au sommet base en utilisant seulement $O(\log n)$ bits de mémoire, atteignant ainsi la borne inférieure connue [DFKP04] pour ce problème.

Dans le cas général des graphes anonymes, l'exploration avec arrêt a été beaucoup étudiée sous diverses hypothèses. Par exemple en donnant une carte du réseau [PP00] ou une borne sur la taille [CDK10a]. Notons que le parcours en profondeur (DFS) est un algorithme d'exploration avec arrêt à partir du moment où l'agent peut borner la taille des chemins du graphe. Dans ce même papier, les auteurs proposent un autre algorithme résolvant l'exploration avec arrêt pour tout graphe en supposant que le sommet base de l'agent est marqué (reconnaissable par tout agent le visitant).

Les différentes hypothèses permettent d'étudier le compromis "algébrique" entre la connaissance globale donnée à l'agent et la difficulté de l'exploration. Par exemple, un graphe anonyme peut être exploré en $O(\Delta^n)$ mouvements par un parcours en profondeur tout en connaissant la taille n du graphe quand le degré maximum est Δ . On peut réduire cette borne à $O(n^3 \Delta^2 \log n)$ en utilisant les séquences universelles d'exploration [AKL⁺79, Kou02]. Ces séquences sont des séquences de numéros de port que l'agent peut suivre. Ces séquences permettent à un agent d'explorer tout graphe de taille au plus n ayant un degré maximum Δ .

Reingold [Rei08] montre que les séquences universelles d'exploration peuvent être construites avec un espace logarithmique. Dans [CFI⁺05, GR08, Ilc08], les auteurs utilisent des connaissances globales pour construire des étiquetages des ports spécifiques (ou des étiquetages des sommets spécifiques) permettant une exploration "facile" et efficace de graphes anonymes. Boldi et Vigna ont présenté des résultats de calculabilité générale pour un modèle à passage de messages anonyme et sûr. Dans [Das13], on trouvera plus de détails concernant l'exploration dans d'autres modèles à agents mobiles comme, un agent doté de jetons qu'il peut déposer/ramasser sur les sommets du graphe.

7.3 Calculabilité de l'exploration sans connaissance initiale

Dans cette partie, nous présentons le problème de l'exploration de graphe par un agent mobile et rappelons les principaux résultats concernant l'exploration sans connaissance globale. Les résultats présentés dans ce chapitre sont connus dans le folklore mais, à notre connaissance, n'ont jamais été rédigés de cette manière.

7.3.1 Le problème de l'exploration sans connaissance

Comme nous allons le montrer, explorer tous les graphes avec arrêt est impossible. Nous considérerons donc des algorithmes d'exploration, dit $\mathcal{F}$ -universels, définis ci-dessous.

Définition 7.37 (Exploration d'une famille de graphe $\mathcal{F}$). Pour toute famille de graphes $\mathcal{F}$, un algorithme $\mathcal{A}$ est un algorithme d'exploration $\mathcal{F}$ -universel si,

- pour tout graphe $G \in \mathcal{F}$, $\mathcal{A}$ visite tous les sommets de G et s'arrête.
- pour tout graphe G , si $\mathcal{A}$ s'arrête, alors G est exploré².

Ce type d'exploration *sûre* permet de composer différents algorithmes entre eux. Sans information sur le graphe exploré, il pourrait être avantageux d'exécuter plusieurs algorithmes d'exploration spécifiques à différentes familles de graphes pour espérer que l'un d'entre eux termine son exploration avec succès. Concevoir des algorithmes d'exploration spécifiques pour certaines familles implique que si les hypothèses émises lors de la conception de l'algorithme ne coïncident pas avec le réseau dans lequel il est exécuté, alors l'exécution peut devenir chaotique et avoir un comportement arbitraire.

Pour résumer, en travaillant avec un agent mobile sans connaissance, il est primordial que la terminaison soit équivalente à la correction pour composer ou exécuter à la suite plusieurs algorithmes distribués. Il est donc important de considérer des algorithmes *universels*, i.e., spécifique à une famille mais ayant un comportement défini sur l'ensemble des réseaux. Par la suite nous considérerons uniquement de tels algorithmes. Grâce à ce type d'exploration, nous obtenons une exploration monotone.

Lemme 7.38. *Soit $\mathcal{A}_1$ et $\mathcal{A}_2$, deux algorithmes d'exploration pour les familles $\mathcal{F}_1$ et $\mathcal{F}_2$ respectivement. L'algorithme composé $\mathcal{A}_1 || \mathcal{A}_2$ est un algorithme d'exploration pour la famille $\mathcal{F}_1 \cup \mathcal{F}_2$*

Nous montrons maintenant pourquoi nous nous intéressons à ce type d'exploration en introduisant la composition d'algorithme d'exploration. Premièrement, nous définissons un type d'algorithmes, dit $\mathcal{F}$ -restreint, conçu pour explorer uniquement une famille de graphes $\mathcal{F}$.

Définition 7.39. Un algorithme d'exploration $\mathcal{F}$ -restreint est un algorithme d'exploration tel que pour tout graphe $G \in \mathcal{F}$, $\mathcal{A}$ visite tous les sommets de G et s'arrête.

Notons que de tels algorithmes sont uniquement spécifiés pour les graphes appartenant à $\mathcal{F}$.

Définition 7.40. Soit $\mathcal{F}_1$ et $\mathcal{F}_2$ deux familles de graphes et soit $\mathcal{A}_1$ un algorithme d'exploration pour $\mathcal{F}_1$ et $\mathcal{A}_2$ un algorithme d'exploration pour $\mathcal{F}_2$.

Une exécution composée de ces deux algorithmes, notée $\mathcal{A}_1 || \mathcal{A}_2$ est une exécution où l'agent exécute les deux algorithmes de façon indépendante. Lors d'une exécution composée, l'agent exécute une première étape de $\mathcal{A}_1$ puis revient sur sa base, sauvegarde son état et le restaure pour exécuter la première étape de l'algorithme $\mathcal{A}_2$ indépendamment de $\mathcal{A}_1$. L'agent effectue ensuite la première étape de $\mathcal{A}_2$, puis revient sur sa base, sauvegarde son état et réinitialise son état pour revenir sur l'état obtenu à la fin de la première étape de $\mathcal{A}_1$. L'agent continue ainsi à exécuter les deux algorithmes de façon alternée jusqu'à ce que l'une des deux exécutions se termine, mettant fin à l'exécution composée.

Pour montrer qu'il est impossible de composer deux algorithmes d'exploration $\mathcal{F}$ -restreint en l'absence de connaissance globale, nous remarquons que la composition peut mener à des exécutions incorrectes, i.e., où l'agent s'arrête sans avoir exploré tous les sommets du graphe.

Remarque 7.41. Considérons une famille de graphes $\mathcal{F}$ et un algorithme $\mathcal{A}$ d'exploration $\mathcal{F}$ -restreint et considérons un graphe G n'appartenant pas à $\mathcal{F}$. Sans information globale, un agent exécutant $\mathcal{A}$ ne connaît pas le graphe G sur lequel il évolue. Par la définition 7.39, il est

2. Une définition similaire mais plus forte serait de demander que l'agent exécute une exploration perpétuelle dans ce cas là. Il est évident que les deux définitions sont équivalentes en termes de calculabilité car il est toujours possible de composer en parallèle (voire Définition 7.40) une exploration perpétuelle de type BFS avec tout algorithme ne s'arrêtant jamais.

Algorithme 6 : Exploration d'une union dénombrable de familles de graphes explorables.

Input : Algorithmes $\mathcal{A}_1 \dots \mathcal{A}_n$ d'exploration pour des familles $\mathcal{F}_1 \dots \mathcal{F}_n$ de graphes explorables
Input : Sommet de départ $v \in V(G)$

- 1 $i \leftarrow 0$;
- 2 **Répéter**
- 3 $i \leftarrow i + 1$;
- 4 **pour** j allant de 1 à i **faire**
- 5 exécuter i étapes de l'algorithme $\mathcal{A}_j$;
- 6 revenir au sommet base de l'agent;
- 7 **Jusqu'à ce que** un algorithme s'arrête;

possible que l'exécution de $\mathcal{A}$ sur G s'arrête sans avoir explorée tous les sommets de G . Prenons maintenant un algorithme $\mathcal{A}'$ d'exploration G – *restreint* et montrons que $\mathcal{A}||\mathcal{A}'$ n'est pas un algorithme d'exploration $\mathcal{F} \cup \{G\}$ -restreint.

Sans perte de généralité, pour toute exécution de l'algorithme $\mathcal{A}||\mathcal{A}'$ sur G , $\mathcal{A}$ peut s'arrêter avant $\mathcal{A}'$ car $G \notin \mathcal{F}$. Par définition, si l'exécution de $\mathcal{A}$ s'arrête alors l'exécution composée $\mathcal{A}||\mathcal{A}'$ s'arrête et donc l'exécution de $\mathcal{A}'$ s'arrête. Nous obtenons une exécution de $\mathcal{A}||\mathcal{A}'$ qui s'arrête sans avoir explorée tous les sommets de G , ce qui, par définition, est une contradiction avec un algorithme d'exploration $\mathcal{F} \cup \{G\}$ -restreint.

Ainsi, sans information globale, la composition d'algorithmes d'exploration $\mathcal{F}$ -restreint n'est pas "monotone" car la composition de tels algorithmes n'est pas sûre, i.e., si l'exécution s'arrête, l'agent mobile ne pourra pas détecter si tous les sommets ont été visités ou non. À l'inverse, la définition d'exploration, présentée dans la définition 7.37, nous permet d'obtenir une relation de "familles explorables" monotone, i.e., si deux familles de graphes $\mathcal{F}_1$ et $\mathcal{F}_2$ sont deux familles explorables, alors l'union des deux familles $\mathcal{F}_1 \cup \mathcal{F}_2$ est une famille de graphes toujours explorable. Il n'est pas évident à première vue qu'il existe une famille maximum de graphes explorables. En effet, il pourrait être possible que deux graphes soient explorables avec des algorithmes spécifiques mais ne soient pas explorables avec un même algorithme.

Pour illustrer la monotonie de cette relation, il suffit de prendre deux algorithmes d'exploration $\mathcal{A}_1$ pour la famille $\mathcal{F}_1$ et $\mathcal{A}_2$ pour la famille $\mathcal{F}_2$. Comme par définition, l'arrêt d'une exécution de $\mathcal{A}_1$ ou $\mathcal{A}_2$ sur un graphe appartenant à $\mathcal{F}_1$ ou $\mathcal{F}_2$ implique que tous les sommets du graphe sont visités, peu importe quel algorithme s'arrête en premier et sur quel graphe est exécuté l'algorithme, toute exécution de l'algorithme composé $\mathcal{A}_1||\mathcal{A}_2$ qui termine, explore bien tous les sommets du graphe. L'algorithme $\mathcal{A}_1||\mathcal{A}_2$ est donc un algorithme $\mathcal{F}_1 \cup \mathcal{F}_2$ -universel d'exploration.

Les arguments présentés ci-dessus permettent de montrer qu'il existe un algorithme d'exploration pour toute union dénombrable de familles de graphes explorables. Pour montrer qu'il existe une famille de graphes explorables maximum nous présentons l'algorithme 6 explorant toute union dénombrable de familles de graphes explorables (possiblement infinies).

Soit $\mathcal{F}_1, \dots, \mathcal{F}_n$ une union dénombrable de familles de graphes explorables et soit $\mathcal{A}_1 \dots \mathcal{A}_n$ les algorithmes d'explorations correspondants. Comme l'ensemble des graphes explorables est un ensemble dénombrable, grâce à l'algorithme 6, nous obtenons le corollaire suivant.

Proposition 7.42. *Il existe une famille de graphes maximum pour l'exploration.*

Par la suite, nous caractériserons la famille de graphes explorables maximum.

7.3.2 Calculabilité de l'exploration

Nous avons montré dans la proposition 7.42 qu'il existe une famille maximum de graphes explorables. Dans cette section, nous caractérisons la famille maximum en réutilisant la même approche algébrique que dans [Ang80, YK96c, CGMO06, Cha06, DFSY07] et que dans la première partie. Pour cela, grâce aux revêtements de graphes, qui sont les outils algébriques adaptés aux graphes anonymes, nous prouverons un lemme de relèvement illustrant les impossibilités liées à la perception locale du réseau d'un agent mobile.

L'état d'un système à agent mobile à un moment donné d'une exécution peut se décrire par deux fonctions présentées ci-dessous.

Définition 7.43. Pour toute étape $i \in \mathbb{N}$ d'une exécution d'un algorithme $\mathcal{A}$ sur un graphe G ,

- $\text{pos}_i(\mathcal{A}, G, v)$ désignera le sommet v du graphe G correspondant à la position de l'agent à la fin d'une étape i d'une exécution de $\mathcal{A}$ sur G .
- $\text{mem}_i(\mathcal{A}, G, v)$ désignera l'état de l'agent (le contenu de sa mémoire interne) à la fin de l'étape i d'une exécution de $\mathcal{A}$ sur G .

Lemme de relèvement

L'outil principal utilisé pour étudier la calculabilité distribuée dans les graphes anonymes est le revêtement de graphes (Définition 7.23). Grâce à cet outil combinatoire, nous sommes en mesure de caractériser, de manière algébrique, les impossibilités liées au modèle. Grâce à un lemme de relèvement présenté ci-dessous, nous exprimons le fait que si deux graphes sont localement isomorphes, alors un agent ne pourra jamais détecter localement une différence entre les deux graphes. Ainsi, pour un algorithme donné, un agent (déterministe) exécutera les mêmes transitions dans les deux graphes. Dans ce cas-là, nous noterons qu'un agent n'est pas en mesure de casser la symétrie entre les deux réseaux.

Nous présentons maintenant un lemme de relèvement des exécutions appliqué à l'exploration.

Lemme 7.44. *Considérons deux graphes G, G' , un revêtement $\varphi : G' \rightarrow G$ et deux sommets $v \in V(G)$ et $v' \in V(G')$. Soit un algorithme $\mathcal{A}$ d'exploration pour G . Pour toute étape i d'une exécution de $\mathcal{A}$ sur G , si $\text{mem}_i(\mathcal{A}, G', v') = \text{mem}_i(\mathcal{A}, G, v)$ et $\varphi(\text{pos}_i(\mathcal{A}, G', v')) = \text{pos}_i(\mathcal{A}, G, v)$, alors $\text{mem}_{i+1}(\mathcal{A}, G', v') = \text{mem}_{i+1}(\mathcal{A}, G, v)$ et $\varphi(\text{pos}_{i+1}(\mathcal{A}, G', v')) = \text{pos}_{i+1}(\mathcal{A}, G, v)$.*

Démonstration. Soit deux graphes G et G' un revêtement $\varphi : G' \rightarrow G$. Soit v le sommet base de G et v' le sommet base de G' . Considérons un algorithme $\mathcal{A}$ et une étape i d'une exécution de $\mathcal{A}$ dans G vérifiant l'énoncé du lemme, i.e, $\text{mem}_i(\mathcal{A}, G', v') = \text{mem}_i(\mathcal{A}, G, v)$ et $\varphi(\text{pos}_i(\mathcal{A}, G', v')) = \text{pos}_i(\mathcal{A}, G, v)$. C'est-à-dire que dans les deux exécutions, l'agent se retrouve exactement dans le même état : il a suivi les mêmes numéros de port et a observé les mêmes étiquettes sur les sommets visités.

Soit $v_i = \text{pos}_i(\mathcal{A}, G, v)$, $v_{i+1} = \text{pos}_{i+1}(\mathcal{A}, G, v)$, la position de l'agent dans G au tour i et $i + 1$. Soit $v'_i = \text{pos}_i(\mathcal{A}, G', v')$, et $v'_{i+1} = \text{pos}_{i+1}(\mathcal{A}, G', v')$, la position de l'agent dans G' au tour i et $i + 1$. Soit $m_i = \text{mem}_i(\mathcal{A}, G, v)$, $m_{i+1} = \text{mem}_{i+1}(\mathcal{A}, G, v)$, $m'_i = \text{mem}_i(\mathcal{A}, G', v')$, et $m'_{i+1} = \text{mem}_{i+1}(\mathcal{A}, G', v')$ l'état de l'agent au tour i et $i + 1$.

Comme φ est un revêtement de G' dans G et comme $\varphi(v'_i) = v_i$, pour toute arête $v'_i w' \in E(G')$, $\varphi(v'_i w') = v_i w \in E(G)$ et $\lambda(v'_i w) = \lambda(v_i w)$. Comme $m_i = m'_i$ et comme l'agent exécute le

même algorithme et reçoit les mêmes entrées dans les deux graphes, il calculera les mêmes valeurs et obtiendra un état identique dans G et G' à l'étape $i + 1$. Nous obtenons que $\text{mem}_{i+1} = \text{mem}'_{i+1}$.

Comme $\mathcal{A}$ calcul les mêmes valeurs dans les deux graphes, le choix de la destination (numéro de port par lequel l'agent quittera le sommet courant) sera le même dans les deux exécutions. Nous obtenons que $\delta_{v_i}(v_{i+1}) = \delta_{v'_i}(v'_{i+1})$ et donc que $\varphi(v'_{i+1}) = v_{i+1}$ car φ préserve les étiquetages des ports. $\square$

En itérant le lemme précédent, nous obtenons le lemme de relèvement.

Lemme 7.45 (Lemme de relèvement). *Considérons deux graphes G et G' tels qu'il existe un revêtement $\varphi : G' \rightarrow G$. Pour tout algorithme $\mathcal{A}$, pour tout sommet $v \in V(G)$ et $v' \in V(G')$, si $\varphi(v) = v'$ alors pour toute étape $i \in \mathbb{N}$ d'une exécution de $\mathcal{A}$ sur G , $\text{mem}_i(\mathcal{A}, G', v') = \text{mem}_i(\mathcal{A}, G, v)$ et $\varphi(\text{POS}_i(\mathcal{A}, G', v')) = \text{POS}_i(\mathcal{A}, G, v)$.*

Pour obtenir une caractérisation plus précise des graphes impossibles à explorer avec un même algorithme, nous utilisons la relation de quasi-revêtements et présentons un lemme de quasi-relèvement ci-dessous. Ce lemme se prouve aisément par les mêmes techniques de preuve que pour le lemme 7.45.

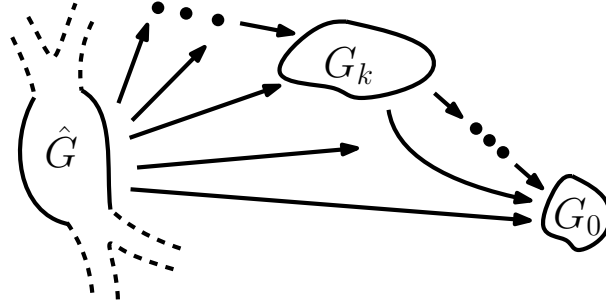


FIGURE 7.6 – Illustration de la relation transitive induite par les revêtements de graphe. Nous illustrons dans cet exemple une chaîne infinie de graphes $G_0, G_1, \dots, G_k, \dots$ telle que $\forall i \in \mathbb{N}$, $\hat{G}$ est revêtement de G_i et $\forall i > 0, G_i$ est revêtement de G_{i-1} . Cette chaîne infinie n'est possible que si le graphe G^0 sur l'exemple n'est pas un arbre.

Lemme 7.46 (Lemme de quasi-relèvement). *Considérons deux graphes G et G' tels qu'il existe un quasi-revêtement $\mu : G' \rightarrow G$ de rayon r et centré en $v_0 \in V(G)$. Pour tout algorithme $\mathcal{A}$, pour toute étape $i \leq r$ d'une exécution de $\mathcal{A}$ sur G basée en v_0 , pour tout sommet $v' \in V(G')$, $\text{mem}_i(\mathcal{A}, G', v') = \text{mem}_i(\mathcal{A}, G, \gamma(v'))$ et $\mu(\text{POS}_i(\mathcal{A}, G', v')) = \text{POS}_i(\mathcal{A}, G, \gamma(v'))$.*

Les quasi-revêtements permettent de caractériser *exactement* les familles de graphes impossibles à explorer avec un même algorithme. Pour montrer qu'il n'existe pas d'algorithme d'exploration universel dans ce modèle, nous montrons d'abord que la famille des *arbres*, i.e., contenant tous les arbres, est la seule famille ayant un rayon de quasi-revêtement borné.

Proposition 7.47. *Pour tout arbre T , il existe un entier k tel que pour tout sommet $v \in V(T)$ et pour tout graphe G , s'il existe un quasi-revêtement non trivial $\gamma : G \rightarrow T$ de rayon r centré en $v \in V(G)$ alors $r \leq k$.*

Démonstration. Soit T un arbre et soit $\gamma : G \rightarrow T$ un quasi-revêtement non trivial de rayon r centré en $u \in V(G)$. Par définition, soit G' un graphe et soit $\varphi : G' \rightarrow T$ un revêtement et soit $\psi : B_G(u, r) \rightarrow B_{G'}(v, r)$ un isomorphisme. Par construction, G' est isomorphe à T (chaque arbre

est isomorphe à son revêtement universel). Nous obtenons que $r \leq |B_G(u, r)| = |B_{G'}(v, r)| \leq |T|$. Mais comme γ est non trivial, $r > |T|$, une contradiction. Par conséquent, pour tout arbre T et tout quasi-revêtement $\gamma : G \rightarrow T$, le rayon de γ sera toujours inférieur ou égal à $|T|$. $\square$

Nous montrons que sans information globale, il n'existe pas d'algorithme d'exploration pour l'ensemble des graphes, i.e., il n'existe pas d'algorithme universel pour ce problème. Ce résultat découle logiquement des résultats présentés dans [Ang80, YK96b].

Proposition 7.48. *Il n'existe pas d'algorithme d'exploration pour toute famille $\mathcal{F}$ contenant strictement les arbres.*

Démonstration. Raisonnons par l'absurde et prenons une famille $\mathcal{F}$ contenant strictement les arbres, i.e., $\text{Arbres} \subsetneq \mathcal{F}$. Soit $\mathcal{A}$ un algorithme d'exploration pour la famille $\mathcal{F}$. Soit un graphe $G \in \mathcal{F}$ non arbre et un sommet base $v_0 \in V(G)$.

Par définition, toute exécution ρ de $\mathcal{A}$ sur G se termine lors d'une étape i après avoir explorée tous les sommets de G . Notons $|\rho|$, le nombre de mouvements nécessaires pour que $\mathcal{A}$ explore G . Nous montrons qu'un tel algorithme pour $\mathcal{F}$ ne peut exister. Comme G n'est pas un arbre, son revêtement universel $\hat{G}$ est un arbre infini (Corollaires 7.29 et 7.30). Soit $\varphi : \hat{G} \rightarrow G$ un revêtement de graphe et soit $\hat{v}$, un sommet de $\hat{G}$ tel que $\varphi(\hat{v}) = v$.

Comme $\hat{G}$ est un arbre infini, prenons $B = B_{\hat{G}}(\hat{v}_0, |\rho| + 1)$ la boule centrée en $\hat{v}_0$ de rayon $|\rho| + 1$ dans $\hat{G}$. Comme B est un sous-graphe induit de $\hat{G}$, B est un arbre appartenant à la famille $\mathcal{F}$ et de taille supérieure à $|\rho|$. De plus, soit $\mu : B \rightarrow G$, le quasi-revêtement centré en $\hat{v}_0$ et de rayon supérieur strictement à $|\rho|$ (Définition 7.31).

À partir du lemme de quasi-relèvement (Lemme 7.46), pour toute exécution ρ de $\mathcal{A}$ sur G avec un sommet base v_0 , l'algorithme $\mathcal{A}$ exécuté dans B avec un sommet base $\tilde{v}_0 \in \mu^{-1}(v_0)$ produira une exécution ρ' tel que pour toute étape $i \leq |\rho|$, $\text{mem}_i(\mathcal{A}, B, \tilde{v}_0) = \text{mem}_i(\mathcal{A}, G, v_0)$ et $\mu(\text{POS}_i(\mathcal{A}, B, \tilde{v}_0)) = \text{POS}_i(\mathcal{A}, G, v)$. De plus, l'algorithme $\mathcal{A}$ étant déterministe, l'agent arrêtera son exécution ρ' après $|\rho|$ mouvements.

Comme $|V(B)| > |\rho| \geq |V(G)|$, par définition, ρ' ne visite donc pas tous les sommets de B . Nous obtenons une contradiction car $B \in \mathcal{F}$ et $\mathcal{A}$ est un algorithme d'exploration pour $\mathcal{F}$. $\square$

Pour montrer que la famille des arbres est explorable, nous proposons un algorithme d'exploration pour la famille des arbres. Dans cet algorithme, l'agent utilise une pile composée de couple (p, flag) avec p un numéro de port et $\text{flag} \in \{\text{vrai}, \text{faux}\}$ un booléen. L'agent possède une primitive `getPorts`, qui utilisée sur un sommet v du réseau, retourne l'étiquetage de port des arêtes adjacentes au sommet courant, i.e., l'ensemble $\{(p, q) \mid vw \in E(\mathbf{G}) \wedge \delta_v(w) = p, \delta_w(v) = q\}$. Notons que l'agent doté de sa capacité de look-back peut facilement faire des allers et retours sur les arêtes adjacentes à sa position pour obtenir les étiquetages de port complets autour de sa position.

Initialement, l'agent, positionné sur sa base, empile les étiquettes des arêtes adjacentes au sommet base (grâce à la primitive `getPorts`) et initialise leur drapeau à `faux` car les arêtes n'ont jamais été franchies par l'agent. À chaque étape, l'agent dépile un couple (p, flag) et se déplace sur l'arête correspondante au numéro p dépilé. Si $\text{flag} = \text{faux}$ alors l'agent effectue deux opérations de pile. Il empile le numéro de port lui permettant de revenir sur le sommet où il était situé précédemment (grâce au "look back") et empile tous les autres numéros de ports adjacents à sa position dans le réseau. Si $\text{flag} = \text{vrai}$, alors l'agent est revenu sur un sommet déjà visité, il n'effectue aucune opération de pile. L'agent répète cette procédure jusqu'à obtenir une pile vide.

Algorithme 7 : Parcours en profondeur

```

1  $\text{pile} \leftarrow \{p \mid (p, q) \in \text{getPorts}\};$ 
2 Répéter
3   Dépiler  $(p, \text{flag})$  de  $\text{pile}$ 
4   Traverser l'arrête étiquetée  $(p, q)$ 
5   si  $\text{flag} = \text{faux}$  alors
6      $\text{pile} \leftarrow \{(q, \text{vrai})\};$ 
7      $\text{pile} \leftarrow \{(p', \text{faux}) \mid (p', q') \in \text{getPorts}\};$ 
8 Jusqu'à ce que  $\text{pile} \neq \emptyset;$ 

```

Notons qu'un symbole `vrai` est inséré dans la pile pour prévenir l'agent qu'il revient sur un sommet déjà visité et qu'il ne doit pas empiler les numéros de ports rencontrés. De plus, le symbole `vrai` est empilé uniquement lors de la première visite d'un sommet. L'agent ayant dépilé le symbole `vrai` dès la deuxième visite du sommet, il n'empilera plus de numéro de port liés correspondant à ce sommet par la suite. Comme un parcours en profondeur s'arrête si et seulement si le graphe ne possède aucun cycle, nous obtenons le théorème suivant.

Théorème 7.1. *L'algorithme 7 est un algorithme d'exploration pour la famille des arbres.*

Dans la proposition 7.48, nous avons montré que toute famille contenant strictement les arbres n'était pas explorable. Dans le théorème 7.2, nous avons montré l'existence d'un algorithme d'exploration pour cette famille. Nous obtenons donc le théorème suivant.

Théorème 7.2. *La famille des arbres est l'ensemble maximum des graphes explorables pour l'exploration.*

Le théorème 7.2 montre le lien entre la calculabilité distribuée et la structure algébrique des revêtements associés à un modèle donné. Pour illustrer ce phénomène et expliquer l'efficacité de cette approche, nous proposons dans le lemme suivant une série d'équivalences obtenues à partir des théorèmes 7.2 et des définitions des revêtements et quasi-revêtements.

Lemme 7.49. *Soit G un graphe. Les propositions suivantes sont équivalentes,*

- G est explorable
- G est un arbre
- G est isomorphe à $\widehat{G}$, son revêtement universel
- le revêtement universel $\widehat{G}$ de G est fini
- le rayon de quasi revêtement de G est borné

Un point intéressant est que la famille des arbres peut être explorée de manière optimale par un algorithme effectuant un simple parcours en profondeur du graphe [CPR12].

Conclusion

Nous venons de voir que dans un modèle anonyme, il est impossible d'assurer la terminaison des exécutions sans compromettre l'*universalité* des algorithmes d'exploration (Théorème 7.2). Cette détection locale de la terminaison peut être obtenue en dotant l'agent d'informations

globales à propos du réseau ou de la famille à explorer. Par exemple, en donnant à l'agent une borne sur la taille ou le diamètre du graphe exploré, il est possible d'explorer tous les chemins de longueur k . Cet ajout augmente donc la calculabilité du modèle au détriment de l'universalité des algorithmes, i.e., les algorithmes deviennent encore plus spécifiques aux réseaux et si le nombre de processus change ou si les identités des processus sont corrompues, une réinitialisation de l'ensemble du système est nécessaire pour relancer les processus avec des données cohérentes. De plus, de telles informations peuvent être difficiles (voire impossibles) à calculer et/ou à maintenir en pratique, dans de grands réseaux par exemple.

Dans ce chapitre, nous introduisons un type d'exploration doté d'une terminaison plus contrainte que l'exploration "classique" avec arrêt. Un algorithme $\mathcal{A}$ résout le problème de l'exploration sur une famille $\mathcal{F}$ si pour tout graphe arbitraire G , si $\mathcal{A}$ s'arrête, alors G est exploré et pour tout graphe $G \in \mathcal{F}$, $\mathcal{A}$ explore G et s'arrête. Ce nouveau type d'exploration permet d'obtenir des algorithmes plus intéressants, i.e., pouvant se composer de façon sûre sans information globale. Dans ce chapitre, nous montrons des résultats déjà connus autour de l'exploration de graphes. Ce problème n'est pas nouveau, mais, à notre connaissance, n'a jamais été caractérisé de façon algébrique. Grâce à cette approche, nous montrons que la famille des arbres est la famille maximale pour l'exploration, i.e., tout autre graphe ne peut être exploré par un algorithme d'exploration, et ce, même en concevant un algorithme d'exploration spécifique pour ce graphe [Ang80]. Ces résultats nous permettent aussi d'introduire un outil mathématique primordial pour étudier la calculabilité distribuée des réseaux anonymes : les (quasi-)revêtements des graphes. En effet, les revêtements caractérisent exactement la relation de "graphes indistinguables" pour un agent mobile naviguant dans un réseau anonyme. L'approche algébrique utilisée est classique et nous a permis d'obtenir une caractérisation élégante des graphes explorables sans connaissance : les arbres sont les seuls graphes isomorphes à leur revêtement universel et sont les seuls graphes explorables (avec arrêt) sans connaissance.

Dans le prochain chapitre, nous introduirons un nouveau modèle d'agent mobile appelé *agents mobiles avec jumelles*. Avec cette extension du modèle, nous élargissons les connaissances locales données à l'agent pour étudier leur impact sur la calculabilité générale de l'exploration. En utilisant la même méthodologie que dans ce chapitre, nous caractériserons les graphes explorables par un agent mobile doté de jumelles.

Chapitre 8

Exploration de graphes anonymes par un agent mobile avec jumelles

Dans ce chapitre, nous étudions le problème de l'exploration de graphes par un agent mobile avec jumelles. Ces résultats ont été publiés lors de la conférence DISC en 2015 [CGN15a]. Ce travail a aussi été présenté en français lors de la conférence francophone ALGOTEL en 2015 [CGN15b].

Introduction

Dans ce chapitre, nous élargissons les informations locales données à l'agent et étudions l'impact sur la calculabilité de l'exploration.

Pour ce faire, l'agent sera doté de jumelles : un capteur permettant à l'agent, positionné sur un noeud du réseau, d'obtenir les relations d'adjacences des noeuds voisins à sa position dans le réseau. Les jumelles sont une extension naturelle pour le modèle des agents mobiles. Tout en ne nécessitant aucune information globale, elles augmentent considérablement la calculabilité distribuée des agents mobiles. En effet, contrairement aux informations locales pouvant se calculer de façon autonome par un agent mobile, les informations globales sont difficiles (voire impossibles) à maintenir efficacement dans de très grands réseaux.

Les informations locales sont significativement moins coûteuses à maintenir et plus faciles à calculer dans un système distribué que des connaissances globales, il est intéressant de les étudier pour comprendre les limites de ce type de connaissances sur la calculabilité d'un système distribué. En suivant la même méthodologie algébrique, nous étendrons la caractérisation et les outils présentés dans le chapitre précédent pour obtenir une caractérisation complète des graphes explorables sans connaissance par un agent mobile doté de jumelles.

Dans ce chapitre, nous montrons qu'un agent équipé de jumelles peut explorer des familles de graphes bien plus vastes que les arbres. Par exemple, un agent équipé de jumelles peut explorer tous les graphes triangulés ainsi que certaines triangulations planaires. D'un autre côté, nous montrons que cette augmentation de la calculabilité du problème est obtenue au détriment de la complexité des exécutions. A notre connaissance, explorer des graphes avec un agent mobile équipé de jumelles n'a jamais été étudié.

Nous donnons une caractérisation complète des graphes explorables par un agent équipé de jumelles en adaptant les revêtements à notre modèle avec jumelles. Ces revêtements sont une spécialisation des revêtements de graphes (vu comme des complexes simpliciaux, et en particulier, des complexes de cliques). Les revêtements simpliciaux et les complexes de cliques

sont des notions standards en topologie discrète (voir la section 8.2.3 pour plus de détails).

Dans la section 8.1, nous présentons le modèle dit "à agent mobile doté de jumelles". Dans la section 8.2, nous présentons les définitions et notations utilisées par la suite. Dans cette partie, nous présentons une extension des "vues des sommets", introduites par Yamashita et Kameda, pour notre modèle. Dans la section 8.3, nous proposons une caractérisation des graphes explorables dans ce modèle. Nous montrons l'impossibilité d'explorer certains graphes. Notre preuve, à la Angluin, se base sur les revêtements simpliciaux. De plus, nous montrons une première borne inférieure pour l'exploration avec jumelles. Dans la section 8.4, nous montrons que notre condition est suffisante en proposant un algorithme explorant tous les graphes explorables dans ce modèle et en prouvant sa correction. Dans la section 8.5, nous montrons que, malgré le gain de calculabilité apportée par les jumelles, la complexité de tout algorithme d'exploration universel croît plus vite que toute fonction calculable.

8.1 Agents mobiles avec jumelles

Dans cette section, nous introduisons formellement les agents mobiles avec jumelles.

8.1.1 Modèle

Nous considérerons des agents mobiles identiques à ceux présentés dans le chapitre précédent (chapitre 7). Les jumelles sont un capteur permettant à l'agent de "voir" le (ou d'accéder au) sous-graphe induit par l'ensemble des sommets du réseau adjacents à sa position (incluant les étiquetages de port ainsi que les étiquettes associées aux sommets). Nous considérerons que la portée des jumelles est constante tout au long d'une exécution.

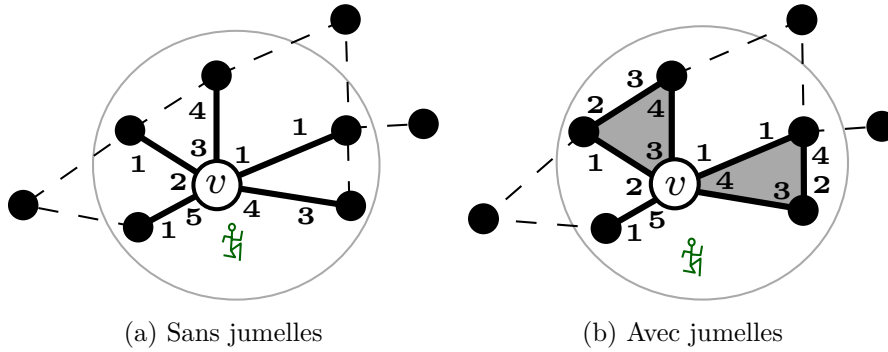


FIGURE 8.1 – Vision d'un agent sans et avec jumelles.

8.1.2 Étiquetage de jumelles

Pour réutiliser les techniques et algorithmes définis sur des graphes étiquetés, nous considérons que chaque nœud v est étiqueté par la boule de rayon 1 autour de v . Nous appellerons un tel étiquetage, un *étiquetage de jumelles*. Il est assez évident qu'un agent explorant des graphes dotés d'un étiquetage de jumelles aura la même puissance de calcul qu'un agent équipé de jumelles explorant des graphes non pourvus d'un tel étiquetage (la primitive "jumelles" permet uniquement d'obtenir plus d'information sur son voisinage et n'autorise pas plus de mouvements). La différence avec le modèle standard se situe donc dans la structure des étiquettes. Désormais, nous supposons que tout sommet v d'un graphe G sera doté d'une étiquette $\nu(v)$ encodant

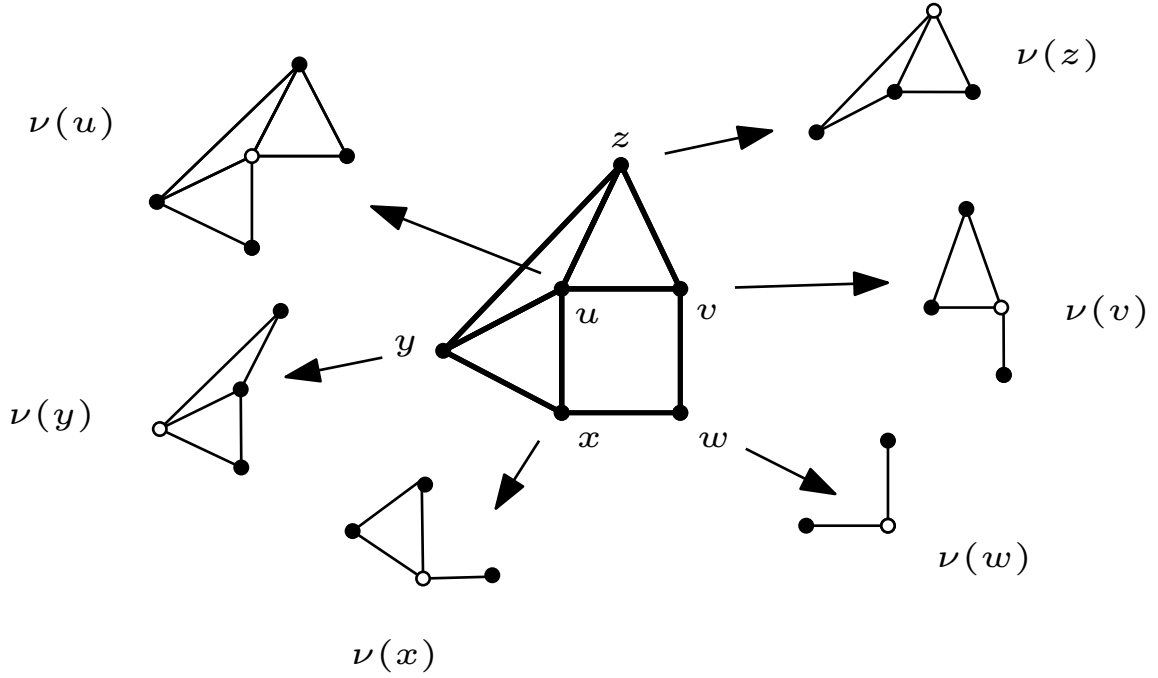


FIGURE 8.2 – Illustration des étiquettes de jumelles. Le sommet blanc dans une étiquette de jumelles indique le sommet à qui appartient l'étiquette. Dans cet exemple, nous avons omis les numéros de ports par souci de lisibilité.

son étiquetage de jumelles du sommet, i.e., $\nu(v)$ encode un graphe isomorphe à $B_G(v, 1)$ avec les numéros de ports correspondants. Un exemple est présenté dans la figure 8.2.

8.2 Définitions

8.2.1 Vues des sommets

La "vue d'un sommet" a été introduite par Yamashita et Kameda dans [YK96c] pour étudier la calculabilité distribuée dans les réseaux anonymes. La vue d'un sommet caractérise l'information maximale (dans le pire des cas) qu'une entité distribuée peut calculer dans un réseau anonyme. La *vue* d'un sommet v dans un graphe étiqueté $(G, label)$ est un arbre possiblement infini composé de l'ensemble des chemins non redondants de G basé en v (Définition 7.7). Nous présentons maintenant une définition constructive.

Définition 8.1. [YK96c] La vue $\mathcal{T}_G(v)$ d'un sommet v dans un graphe étiqueté $\mathbf{G}$ est un arbre enraciné et étiqueté construit récursivement comme suit. La racine de $\mathcal{T}_G(v)$, notée x_0 , est étiquetée $LABEL(x_0) = \lambda(v)$. Pour tout sommet v_i voisin de v , un noeud x_i est inséré dans $V(\mathcal{T}_G(v))$ avec une étiquette $LABEL(x_i) = \lambda(v_i)$ et une arête x_0x_i est insérée dans $E(\mathcal{T}_G(v))$ étiquetée par $\delta_{x_0}(x_i) = \delta_v(v_i)$ et $\delta_{x_i}(x_0) = \delta_{v_i}(v)$. Pour finir, tout noeud x_i voisin de x_0 est identifié avec la racine de l'arbre $\mathcal{T}_G(v_i)$ dans lequel on a retiré tous les chemins redondants, i.e., tout chemin $p = y_1 \dots y_k \subset V(\mathcal{T}_G(v_i))$ tel que $x_i = y_1$ et $\delta_{y_1}(y_2) = \delta_{x_i}(x_0)$.

Étant donné un entier $k \in \mathbb{N}$, la vue $\mathcal{T}_G(v)$ tronquée à la profondeur k sera notée $\mathcal{T}_G(v, k)$. Si le contexte nous le permet, nous la noterons $\mathcal{T}(v, k)$.

Par les informations (partielles) sur la topologie du réseau qu'elles contiennent, les vues induisent une relation d'équivalence, notée $\sim$, sur les sommets.

Définition 8.2. Étant donnés deux graphes G et H et deux sommets $v \in V(G)$ et $w \in V(H)$, si $\mathcal{T}_G(v, k) = \mathcal{T}_H(w, k)$ alors $v \sim_k w$.

Pour illustrer l'importance d'utiliser cet outil, il a été prouvé [YK96c] que si les vues des sommets sont similaires, alors il sera impossible pour une entité distribuée de faire une distinction entre les différentes entités du réseau. Globalement, cela implique certaines impossibilités pour des problèmes fondamentaux (tel l'élection) [Ang80, YK96b]. La cohérence des vues en algorithmique distribuée explique l'efficacité de cette approche algébrique car les revêtements de graphe préservent les vues des sommets et ne préservent que cela. En effet, étant donné un revêtement $\varphi : G \rightarrow H$ donné, deux sommets appartiendront à une même classe d'équivalence $\sim$ s'ils possèdent une même image dans G via le revêtement φ . La proposition suivante montre que les vues sont préservées par la relation de revêtement.

Proposition 8.3. Soit $\varphi : G \rightarrow H$ un revêtement entre un graphe G et un graphe H , pour tout $v \in V(G)$, $\mathcal{T}_G(v) = \mathcal{T}_H(\varphi(v))$.

Comme tous les sommets possédant une même image par un revêtement partageront une même vue, la relation est donc transitive. Il est possible de créer des revêtements tels que deux sommets possèdent une même vue sans avoir une même image par le revêtement, comme par exemple le graphe présenté dans la figure 8.4 où tous les sommets possèdent une même vue sans pour autant avoir une même image.

8.2.2 Vues avec étiquetage de jumelles

Nous étendons la définition des vues (introduites dans [YK96c]) pour prendre en compte les étiquettes de jumelles attachées à chaque sommet d'un graphe.

Dorénavant, tout noeud x_u présent dans la vue $\mathcal{T}_G(v)$ d'un sommet $v \in V(G)$ sera doté de l'étiquette de jumelles $\nu(u)$ du sommet correspondant à x_u dans G . Les vues avec jumelles sont illustrées Figure 8.5.

8.2.3 Complexes Simpliciaux

Les définitions de cette section sont des notions standards de topologie discrète (voir [LS77]).

Définition 8.4 (Simplexe). Étant donné un ensemble V , un *simplexe* s de *dimension* $n \in \mathbb{N}$ est un sous-ensemble de V de cardinal $n + 1$.

Définition 8.5 (Complexe simplicial). Un *complexe simplicial* K est une collection de simplexes telle que pour tous simplexes $s, s' \subset V$, si $s' \subseteq s$ alors $s' \in K$.

Définition 8.6 (Dimension d'un complexe simplicial). Un complexe simplicial K est k -dimensionnel si la plus grande dimension d'un simplexe de K est k .

Un graphe G peut être vu comme un complexe 1-dimensionnel où l'ensemble des sommets $V(G)$ sont les simplexes de dimension 0 et l'ensemble des arêtes $E(G)$ correspondent aux simplexes de dimension 1.

Réciproquement, étant donné un complexe simplicial K , les simplexes de dimension 0 de K sont appelés les *sommets* de K et les simplexes de dimension 1 de K sont appelés les *arêtes*. Les simplexes de dimension 2 seront appelés *triangles*.

Nous noterons $V(K)$ (resp. $E(K)$) pour indiquer l'ensemble des sommets (resp. des arêtes) de K .

Définition 8.7 (1-squelette). Le 1-squelette d'un complexe K est le graphe $G(K) = (V(K), E(K))$.

Un complexe est dit connexe si son 1-squelette est connexe. Nous considérerons toujours des complexes connexes.

Définition 8.8 (Étoile d'un sommet). L'étoile $St(v, K)$ d'un sommet v dans un complexe simplicial K est le sous-complexe défini par l'ensemble de simplexes de K contenant v et leurs sous-simplexes.

Nous étendons les notions de fonctions simpliciales et de revêtements aux complexes simpliciaux comme suit.

Rappel (Fonction simpliciale). Étant donnés deux complexes simpliciaux K et K' , une fonction $\varphi : V(K) \rightarrow V(K')$ est une *fonction simpliciale* $\varphi : K \rightarrow K'$ si pour tout simplexe $s = \{v_1, \dots, v_k\} \in K$, $\varphi(s) = \{\varphi(v_1), \dots, \varphi(v_k)\}$ est un simplexe de K' .

Notons que la notion de fonction simpliciale ne change pas de la précédente définition pour les graphes. Ici, tout élément du complexe K possède une image dans le complexe K' et la dimension des éléments (des simplexes) n'est pas forcément préservée. Une fonction simpliciale préservant la dimension des simplexes est appelée un morphisme simplicial.

Définition 8.9 (Morphisme simplicial). Étant donnés deux complexes simpliciaux K et K' , une fonction simpliciale $\varphi : K \rightarrow K'$ est un *morphisme simplicial* si pour tout simplexe s de K , $\varphi(s) \in K'$ est un simplexe de K' de même dimension.

Définition 8.10 (Revêtement simplicial). Étant donnés deux complexes simpliciaux K et K' , un morphisme simplicial $\varphi : K \rightarrow K'$ est un *revêtement simplicial* si pour tout sommet $v \in V(K)$, $\varphi|_{St(v, K)}$ est une bijection entre $St(v, K)$ et $St(\varphi(v), K')$.

Des exemples de revêtements simpliciaux sont présentés dans la figure 8.7. Nous présentons maintenant le *revêtement universel* d'un complexe K .

Théorème-Définition 8.11 (Revêtement universel). Pour tout complexe simplicial K , il existe un complexe possiblement infini et unique (à isomorphisme prêt), noté $\widehat{K}$, ainsi qu'un revêtement simplicial $\mu : \widehat{K} \rightarrow K$ tel que pour tout complexe K' , pour tout revêtement simplicial $\varphi : K' \rightarrow K$, il existe un revêtement simplicial $\gamma : \widehat{K} \rightarrow K'$ tel que $\varphi \circ \gamma = \mu$.

8.2.4 Complexe de cliques

Définition 8.12 (Complexe de cliques). Étant donné un graphe $G = (V, E)$, le *complexe de cliques* de G , noté $\mathcal{K}(G)$, est le complexe simplicial constitué par les cliques de G .

Formellement, chaque clique de taille n de G induit un simplexe de dimension $n - 1$ dans $\mathcal{K}(G)$. Notons que pour tout graphe G , le 1-squelette de $\mathcal{K}(G)$ est G . Des exemples de complexes de cliques sont présentés dans la figure 8.9.

Notons que si $\varphi : \mathcal{K}(G) \rightarrow \mathcal{K}(G')$ est un revêtement simplicial, alors φ est aussi un revêtement de graphe de G vers G' . Notons aussi que la réciproque ne tient pas. Par exemple, dans la figure 8.8, il n'y a pas de revêtement simplicial entre les graphes de gauche (possédant plus de 4 sommets) et celui de droite (en comportant 4) car celui de droite comporte un triangle alors que les deux complexes de gauche en sont dépourvus.

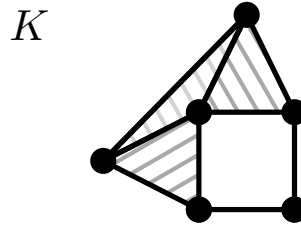


FIGURE 8.6 – Illustration d'un complexe simplicial composé de 0-simplexes (les sommets), de 1-simplexes (les arêtes) et de trois 2-simplexes (les triangles).

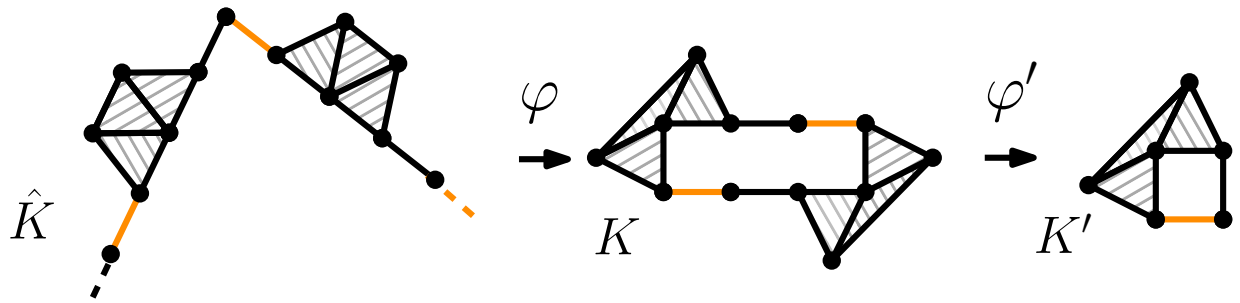


FIGURE 8.7 – Exemples de revêtements simpliciaux. Ici, $\widehat{K}$ est un graphe infini et est un revêtement simplicial (et universel) de K et K' . K est un revêtement (simplicial) de K' .

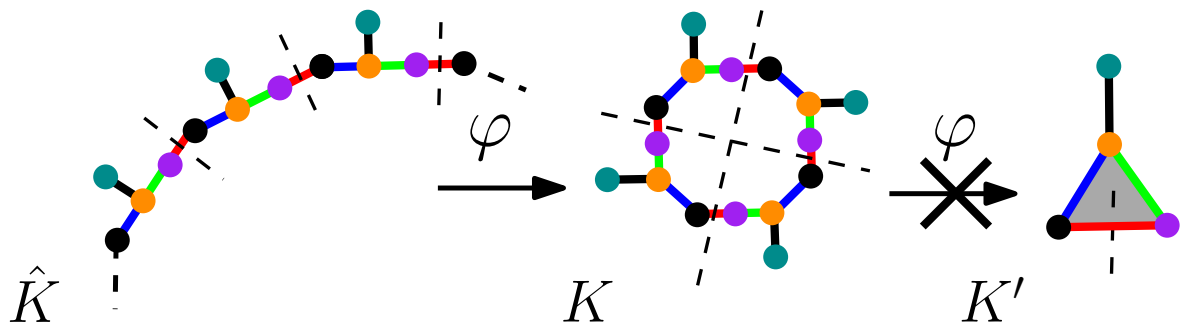


FIGURE 8.8 – Illustration des limites des revêtements simpliciaux. À droite, la relation de revêtement simplicial ne s'applique pas car K' possède un triangle alors que K n'en a aucun.

8.2.5 Homotopie et Contractibilité

Une homotopie élémentaire caractérise les opérations élémentaires permettant de contracter ou d'étendre une boucle le long des triangles d'un complexe donné. Ces notions sont introduites car un agent doté de jumelles est en mesure de distinguer les triangles d'un complexe et donc, sera en mesure de calculer des relations élémentaires sur ces triangles.

Définition 8.13 (Homotopie élémentaire). Soit $c = (v_0, v_1, \dots, v_{i-1}, v_i, v_{i+1}, \dots, v_k)$ et $c' = (v_0, v_1, \dots, v_{i-1}, v_{i+1}, \dots, v_k)$ deux boucles dans un complexe K . Les boucles c et c' sont reliées par une *homotopie élémentaire* si une des conditions suivantes est satisfaite :

(Contraction) $v_i = v_{i+1}$

(Retour arrière) $v_{i-1} = v_{i+1}$

(Pousser un triangle) $v_{i-1}v_iv_{i+1}$ est un triangle du complexe K . Dans le cadre restreint de complexes de cliques, la condition devient $v_{i-1}v_{i+1}$ est une arête de complexe de clique K .

Notons que la relation d'homotopie élémentaire entre deux boucles est une relation symétrique. Cette relation étant aussi transitive et réflexive, elle induit une relation d'équivalence, appelée *homotopie*. Pour illustrer cette relation, prenons le complexe de cliques d'une sphère triangulée par exemple. Deux boucles seront homotopes si l'une peut être obtenue en déplaçant la deuxième le long des bords de la sphère, i.e., si les deux boucles sont reliées par une séquence d'homotopies élémentaires.

Définition 8.14 (Relation d'homotopie). Deux boucles c et c' dans un complexe K sont *homotopes* s'il existe une séquence de boucles $c_1 \dots, c_\ell \subseteq K$ vérifiant pour tout $1 \leq i < \ell$ que $c_1 = c, c_\ell = c'$ et que c_i est reliée par une homotopie élémentaire à c_{i+1} .

Nous allons nous intéresser à une certaine catégorie de boucles, les boucles contractibles. Une boucle est contractible si elle est homotope à un sommet, la boucle triviale. Nous étendons ci-dessous cette relation de contractibilité en prenant en compte le nombre d'homotopies élémentaires requises pour contracter la boucle.

Définition 8.15 (k -contractibilité). Une boucle est k -contractible, $k \in \mathbb{N}$, si elle est homotope à la boucle triviale par une séquence de k homotopies élémentaires.

Définition 8.16 (Contractibilité). Une boucle est contractible s'il existe un entier k tel que la boucle est k -contractible.

Diagramme de disque.

Définition 8.17. Un graphe est k -connexe si le nombre minimum de sommets à retirer pour que le graphe ne soit plus connexe est k .

Les définitions présentées dans ce paragraphe proviennent de [BH99]. Nous présentons maintenant un *diagramme de disque* : outil permettant de manipuler la structure combinatoire d'un ensemble d'homotopies d'une boucle ou d'un cycle dans un complexe donné. Nous l'utiliserons souvent par la suite.

Définition 8.18 (diagramme de disque). Étant donnée une boucle $c = (v_0, v_1, \dots, v_n = v_0)$ dans un complexe simplicial K , un *diagramme de disque* (D, f) de c est une paire où D est une triangulation planaire 2-connexe et f est une fonction simpliciale de D vers K telle que la face externe de D , notée ∂D , est un cycle simple $(v'_0, v'_1, \dots, v'_n)$ tel que $f(v'_i) = v_i$, pour tout $0 \leq i \leq n$.

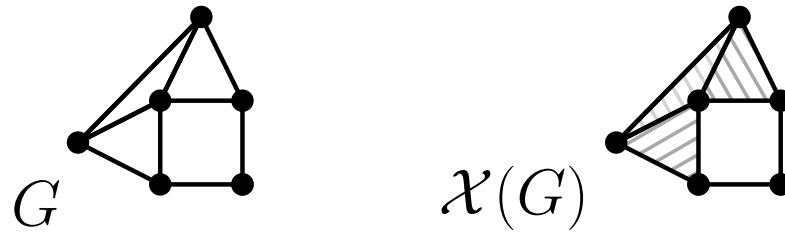


FIGURE 8.9 – Illustration d'un graphe G (à gauche) et de son complexe de cliques $\mathcal{K}(G)$ associé (à droite)

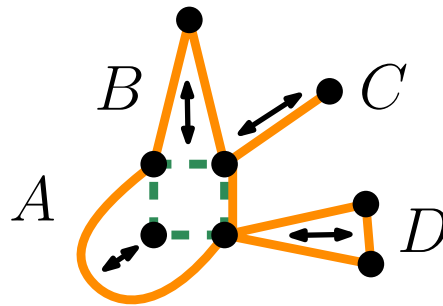


FIGURE 8.10 – Exemple de deux cycles homotopes : Le cycle plein est homotope au cycle en pointillé par la séquence d'homotopies élémentaires A,B,C et D. Notons que les deux cycles ne sont pas contractibles.

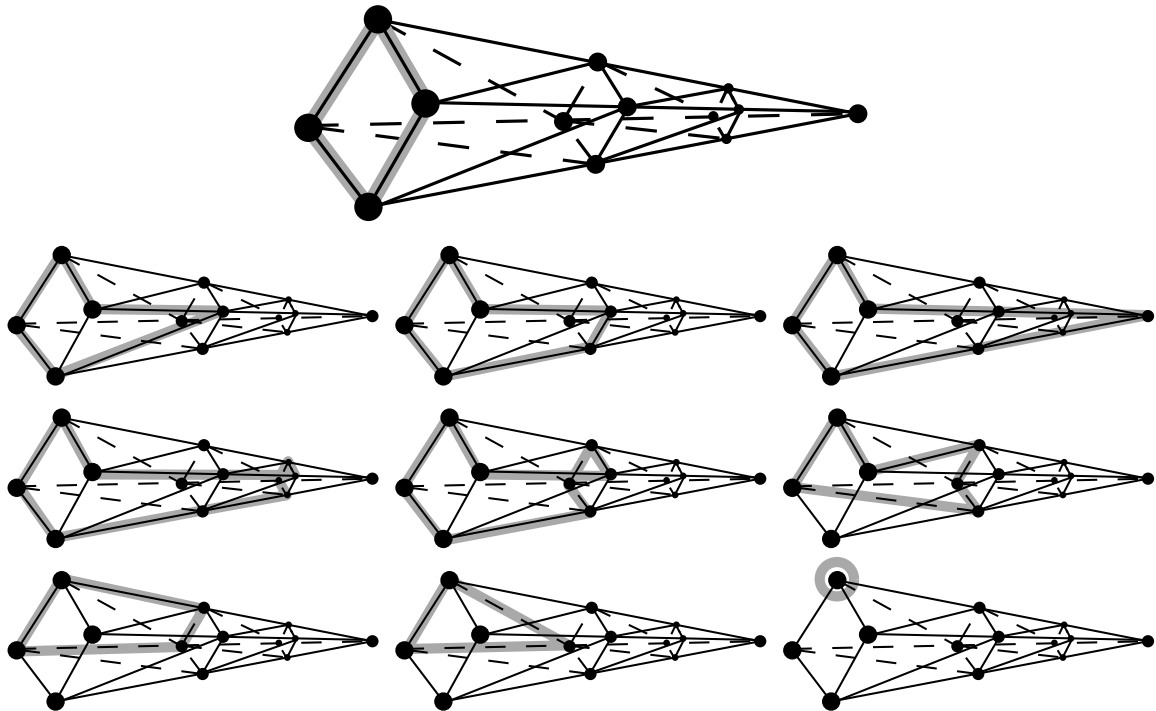


FIGURE 8.11 – Exemple d'une surface (en forme de manche à air) où certains cycles doivent être élargis pour se contracter en un point. Les traits épais représentent les différentes étapes de la contraction du cycle gris sur cette surface.

Un diagramme de disque est illustré sur la figure 8.12.

Définition 8.19 (Diagramme minimal et Aire d'un cycle). L'aire d'un diagramme de disque D , notée $\mathcal{A}(D)$, est égale au nombre de triangles contenus dans D . Un diagramme de disque (D, f) pour un cycle c est minimal s'il minimise son aire $\mathcal{A}(D)$. L'aire d'un cycle c , notée $\mathcal{A}(c)$, est égale à l'aire d'un diagramme de disque minimal pour c .

Les propositions suivantes illustrent les propriétés unissant la contractibilité d'une boucle et son aire. La première proposition est obtenue en suivant la définition de la contractibilité.

Proposition 8.20 ([BH99]). *Pour toute boucle b dans un complexe K , si b est contractible alors il existe un diagramme de disque (D, f) pour b .*

Proposition 8.21 ([BH99]). *Pour toute boucle b dans un complexe K , si b est k -contractible alors $\mathcal{A}(b) \leq k$ et à l'inverse, si $\mathcal{A}(b) \leq k$ alors b est $(k + |b|)$ -contractible.*

La proposition suivante montre que tester la contractibilité des cycles simples dans un complexe donné suffit à montrer la contractibilité du complexe. Notons que le nombre d'homotopies élémentaires requises pour contracter une boucle n'est pas forcément borné par le nombre de sommets du graphe (voir Figure 8.12).

Proposition 8.22. *Si toutes les boucles simples dans un complexe K sont contractibles alors toutes les boucles de K sont contractibles.*

Démonstration. Prenons un complexe K possédant uniquement des boucles contractibles et prenons une boucle $c = (u_0, u_1, \dots, u_k, u_0)$ qui n'est pas simple. Nous prouvons cette proposition par induction sur la longueur $|c| = k$ de la boucle c .

Comme la boucle c est non simple, il existe $1 \leq i, j \leq k$ tel que $u_j = u_i$. Sans perte de généralité, supposons que $u_0 = u_i$ et que $c_1 = (u_0, u_1, \dots, u_{i-1}, u_0)$ et $c_2 = (u_i, \dots, u_k, u_i)$ sont deux boucles. Par induction, comme $|c_1| < k$ et $|c_2| < k$, les deux boucles c_1 et c_2 sont contractibles.

Si $1 \leq i \leq 2$, alors c et c_2 sont reliées par une homotopie élémentaire. Si $k - 1 \leq i \leq k$, alors c et c_1 sont reliées par une homotopie élémentaire. Supposons maintenant que $3 \leq i \leq k - 2$. Par induction, c_1 possède un diagramme de disque (D_1, f_1) tel que $\partial D_1 = (u'_0, \dots, u'_{i-1})$ et $f_1(u'_j) = u_j$ pour tout $0 \leq j \leq i - 1$. De même pour c_2 , il existe un diagramme de disque (D_2, f_2) tel que $\partial D_2 = (u'_i, \dots, u'_k)$ et $f_2(u'_j) = u_j$ pour tout $i \leq j \leq k$.

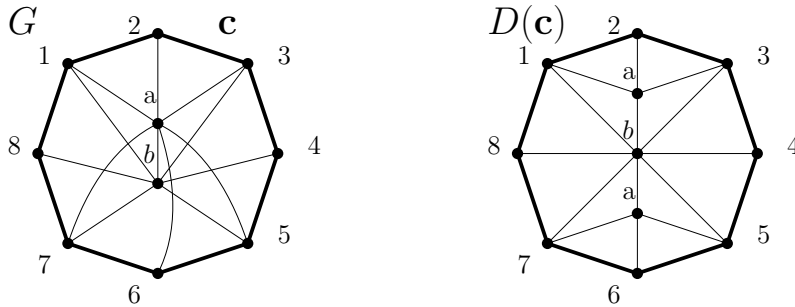


FIGURE 8.12 – Graphe G pour lequel le nombre d'homotopies élémentaires pour contracter un cycle n'est pas borné par le nombre de sommets du graphe. Un diagramme de disque pour le cycle c est représenté à droite pour illustrer les homotopies requises pour contracter c

Prenons le graphe D obtenu comme suit : $V(D) = V(D_1) \cup V(D_2)$, $E(D) = E(D_1) \cup E(D_2) \cup \{u'_{i-1}u'_i, u'_ku'_0, u'_0u'_i\}$. Notons que D est une triangulation planaire telle que $\partial D = (u'_0, \dots, u'_{i-1}, u'_i, \dots, u'_k)$ (voir la figure 8.13). Pour tout $u' \in V(D)$, soit $f(u') = f_1(u')$ si $u' \in V(D_1)$ et $f(u') = f_2(u')$ sinon. Pour toute arête $u'v' \in E(D)$, nous obtenons que soit $u', v' \in V(D_1)$, soit $u', v' \in V(D_2)$, soit $u'v' \in \{u'_{i-1}u'_i, u'_ku'_0, u'_0u'_i\}$.

Dans le premier cas, $f(u') = f_1(u')$ est soit égal ou adjacent à $f(v') = f_1(v')$ car f_1 est une fonction simpliciale. Dans le second cas, pour les mêmes raisons, nous obtenons que soit $f(u') = f(v')$; soit $f(u')f(v') \in E(G)$. Dans le dernier cas, nous savons que $f(u'_i) = f(u'_0) = u_0$ est soit égal, soit adjacent à $u_{i-1} = f(u'_{i-1})$ (resp. à $u_k = f(u'_k)$). Donc, f est une fonction simpliciale de D vers G telle que $f(\partial D) = c$. Ainsi, il existe un diagramme de disque (D, f) pour c , et donc c est contractible.

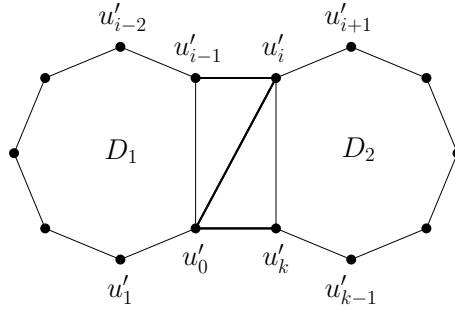


FIGURE 8.13 – Illustration de la preuve de la proposition 8.22

□

8.2.6 Simple Connexité

Définition 8.23 (Simple Connexité). Un complexe *simplement connexe* est un complexe où toute boucle est homotope à un sommet, i.e., peut être réduite à un sommet par une séquence d'homotopies élémentaires.

Ces complexes possèdent de nombreuses propriétés combinatoires et topologiques qui vont nous être utiles par la suite.

Proposition 8.24 ([LS77]). *Soit K un complexe connexe, K est isomorphe à son revêtement universel $\widehat{K}$ si et seulement si K est simplement connexe.*

Il est évident que pour tester la simple connexité d'un complexe K , vérifier la contractilité des boucles simples de K est suffisant. Nous obtenons donc une définition alternative pour la simple connexité en passant par l'existence de diagrammes de disque.

Proposition 8.25. *Un complexe K (fini) est simplement connexe si et seulement si toute boucle simple de K admet un diagramme de disque.*

8.3 Caractérisation des graphes explorables avec jumelles

La proposition suivante illustre le fait que la notion de revêtement de graphes dotés d'étiquetage de jumelles est équivalente à la notion de revêtement simplicial pour les complexes de cliques.

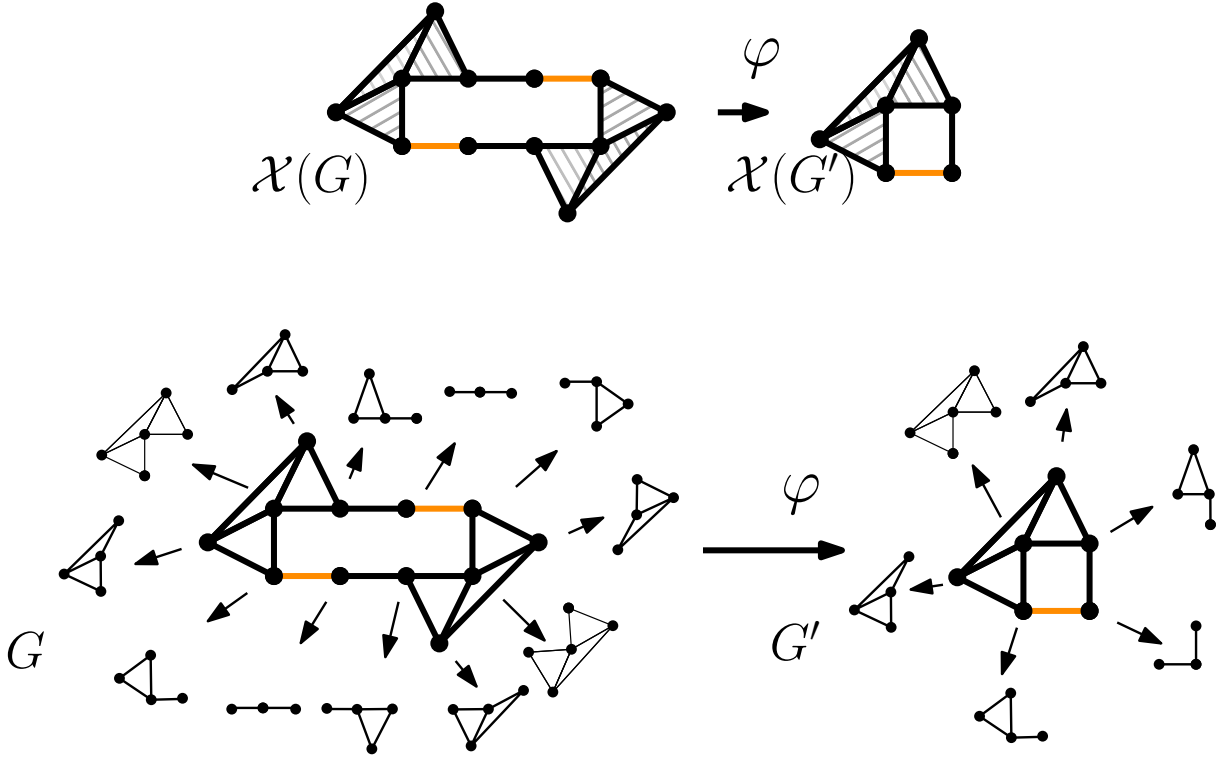


FIGURE 8.14 – Illustration de l'équivalence entre un revêtement simplicial et un revêtement de graphes préservant les étiquetages de jumelles

Proposition 8.26. *Soit deux graphes G et H dotés d'un étiquetage de jumelles et un morphisme $\varphi : G \rightarrow H$. Le morphisme φ est un revêtement de graphes de G vers H si et seulement si φ est un revêtement simplicial de $\mathcal{K}(G)$ vers $\mathcal{K}(H)$.*

La figure 8.14 illustre cette équivalence. Dans la figure 8.8 un contre exemple est illustré. Dans cet exemple, il n'existe pas de revêtement "de graphe" préservant les étiquettes de jumelles entre K et K' car des sommets de K' appartiennent à un triangle tandis que K ne possède aucun triangle. Comme nous allons le montrer, un agent mobile naviguant dans des graphes anonymes sans information globale ne peut pas résoudre tous les problèmes. Pour montrer ces impossibilités, nous réutilisons la méthodologie et les résultats standards de calculabilité distribuée [Ang80, BV01, CGM12, YK96c]. Ces résultats utilisent la structure des revêtements de graphes pour caractériser les graphes indistinguables par une entité de calcul sans information globale comme un agent mobile par exemple.

Pour caractériser la calculabilité du modèle, nous allons donc devoir étudier la relation induite par les revêtements préservant les étiquetages de jumelles (Proposition 8.26). Dorénavant, nous considérerons uniquement des revêtements simpliciaux et pour simplifier les notations, un revêtement simplicial sera simplement noté "revêtement". Nous noterons explicitement "revêtement de graphes" le cas échéant.

Nous distinguons deux familles partitionnant l'ensemble des graphes (finis) en se basant sur les revêtements de leurs complexes de cliques.

- $\mathcal{FC} = \{G \mid G \text{ est un graphe fini et son revêtement universel } \mathcal{K}(G) \text{ est fini}\}$
- $\mathcal{IC} = \{G \mid G \text{ est un graphe fini et son revêtement universel } \mathcal{K}(G) \text{ est infini}\}$

Notons que la famille $\mathcal{FC}$ possède une intéressante sous famille, la famille $\mathcal{SC} = \{G \mid G \text{ est fini et } \mathcal{K}(G) \text{ est simplement connexe}\}$ des graphes ayant un complexe de cliques simplement connexe. Les graphes simplement connexes sont introduits par la définition 8.23 de façon combinatoire.

Nous présentons ci-dessous d'autres caractérisations possibles équivalentes.

Définition 8.27. Un complexe (fini) K est simplement connexe si et seulement si il satisfait une des propositions suivantes

- (i) tout cycle c de K est contractible ;
- (ii) tout cycle c de K admet un diagramme de disque (D, f) .
- (iii) K est isomorphe à son revêtement universel $\widehat{K}$.

Pour montrer que la famille $\mathcal{FC}$ est maximum pour l'exploration avec jumelles, nous réutilisons la même méthodologie, avec des outils adaptés au modèle des jumelles, que dans le chapitre précédent. Nous introduisons une notation liant un graphe et le revêtement universel de son complexe de cliques,

Définition 8.28. Pour tout graphe G , $\widehat{G}$ représente le 1-squelette du revêtement universel du complexe de cliques de $\mathcal{K}(G)$.

Premièrement, dans les lemmes 8.29 et 8.30, nous proposons un lemme de relèvement pour les revêtements de complexes de cliques. Ce lemme montre que toute exécution dans un graphe G peut être "relevée" dans tout graphe G' tel que $\mathcal{K}(G')$ est un revêtement de $\mathcal{K}(G)$. En particulier, dans le 1-squelette $\widehat{G}$ du revêtement universel de $\mathcal{K}(G)$.

Nous montrons en premier le résultat pour une transition de l'algorithme, puis nous généraliserons à toute exécution.

Lemme 8.29. *Considérons deux graphes G et G' tels qu'il existe un revêtement $\varphi : \mathcal{K}(G') \rightarrow \mathcal{K}(G)$. Soit $\mathcal{A}$ un algorithme, soit $v \in V(G)$ et $v' \in V(G')$ deux sommets, et soit $i \in \mathbb{N}$ une étape d'une exécution de $\mathcal{A}$ sur G . Si $\text{mem}_i(\mathcal{A}, G', v') = \text{mem}_i(\mathcal{A}, G, v)$ et $\varphi(\text{POS}_i(\mathcal{A}, G', v')) = \text{POS}_i(\mathcal{A}, G, v)$, alors $\text{mem}_{i+1}(\mathcal{A}, G', v') = \text{mem}_{i+1}(\mathcal{A}, G, v)$ et $\varphi(\text{POS}_{i+1}(\mathcal{A}, G', v')) = \text{POS}_{i+1}(\mathcal{A}, G, v)$.*

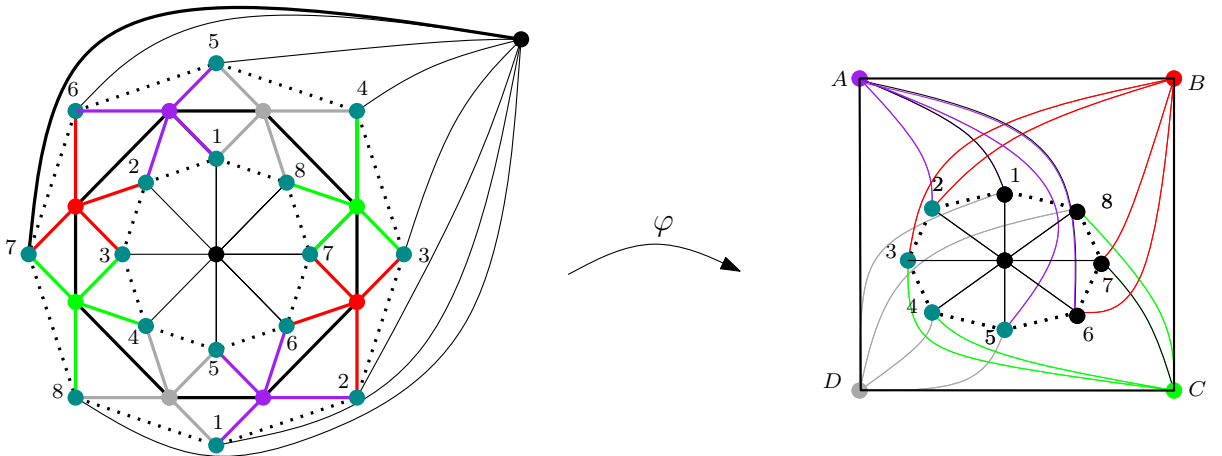


FIGURE 8.15 – Triangulation du plan projectif appartenant à $\mathcal{FC} \setminus \mathcal{SC}$ et son revêtement universel fini. Le graphe de droite (comportant 13 sommets) est une triangulation du plan projectif qui n'est pas simplement connexe (cycle ABCD non homotope à un point). Le graphe de gauche est son revêtement universel fini (à deux feuillets) qui lui est simplement connexe (il comporte 26 sommets). Dans cet exemple, les sommets ont été numérotés pour plus de lisibilité

Démonstration. Soit deux graphes G et G' tels qu'il existe un revêtement $\varphi : \mathcal{K}(G') \rightarrow \mathcal{K}(G)$. Soit deux sommets $v \in V(G)$ et $v' \in V(G')$ tels que $\varphi(v) = \varphi(v')$ et tels que v est le sommet base dans G et v' est le sommet base dans G' . Considérons un algorithme $\mathcal{A}$ et une étape i d'une exécution de $\mathcal{A}$ dans G vérifiant l'énoncé du lemme, i.e, $\text{mem}_i(\mathcal{A}, G', v') = \text{mem}_i(\mathcal{A}, G, v)$ et $\varphi(\text{POS}_i(\mathcal{A}, G', v')) = \text{POS}_i(\mathcal{A}, G, v)$.

Soit $v_i = \text{POS}_i(\mathcal{A}, G, v)$, $v_{i+1} = \text{POS}_{i+1}(\mathcal{A}, G, v)$, $v'_i = \text{POS}_i(\mathcal{A}, G', v')$, et $v'_{i+1} = \text{POS}_{i+1}(\mathcal{A}, G', v')$. Soit $m_i = \text{mem}_i(\mathcal{A}, G, v)$, $m_{i+1} = \text{mem}_{i+1}(\mathcal{A}, G, v)$, $m'_i = \text{mem}_i(\mathcal{A}, G', v')$, et $m'_{i+1} = \text{mem}_{i+1}(\mathcal{A}, G', v')$.

Comme φ est un revêtement de $\mathcal{K}(G')$ vers $\mathcal{K}(G)$ et comme $\varphi(v'_i) = v_i$, les graphes $B_G(v_i, 1)$ et $B_{G'}(v'_i, 1)$ sont isomorphes (et ont les mêmes étiquettes et numéros de port). Donc, comme $m_i = m'_i$, $\mathcal{A}$ possède les mêmes entrées dans les deux graphes et donc, lors de la mise à jour de son état, l'agent (déterministe) calculera les mêmes valeurs et obtiendra un état identique dans G et G' . Ainsi, nous obtenons que $m_{i+1} = m'_{i+1}$. De plus, comme $\mathcal{A}$ calcule les mêmes valeurs dans les deux graphes, le choix de la destination (numéro de port par lequel l'agent quittera le sommet courant) sera le même dans les deux exécutions. Nous obtenons que $\delta_{v_i}(v_{i+1}) = \delta_{v'_i}(v'_{i+1})$. Comme δ est un étiquetage de ports pour G' préservé dans G par φ , nous obtenons que $\varphi(v_{i+1}) = \varphi(v'_{i+1})$. $\square$

En itérant le lemme précédent, nous obtenons le lemme de relèvement suivant.

Lemme 8.30 (Lemme de relèvement). *Considérons deux graphes G et G' tels qu'il existe un revêtement $\varphi : \mathcal{K}(G') \rightarrow \mathcal{K}(G)$. Pour tout algorithme $\mathcal{A}$, pour tout sommet $v \in V(G)$ et $v' \in V(G')$ tel que $\varphi(v') = v$, et pour toute étape $i \in \mathbb{N}$ d'une exécution de $\mathcal{A}$ sur G , $\text{mem}_i(\mathcal{A}, G', v') = \text{mem}_i(\mathcal{A}, G, v)$ et $\varphi(\text{POS}_i(\mathcal{A}, G', v')) = \text{POS}_i(\mathcal{A}, G, v)$.*

En utilisant le lemme de relèvement ci-dessus, nous sommes en mesure de prouver un premier résultat concernant les graphes (in)explorables avec jumelles. Nous en déduisons aussi une borne inférieure concernant la complexité de l'exploration pour les familles explorables.

Proposition 8.31. *Tout graphe G explorable par un agent avec jumelles appartient à $\mathcal{FC}$.*

Démonstration. Supposons que ce n'est pas le cas et prenons un algorithme d'exploration $\mathcal{A}$ explorant un graphe $G \in \mathcal{IC}$ avec un sommet base $v_0 \in V(G)$. Soit r le nombre de mouvements effectués par l'agent pour explorer G à partir de v_0 .

Soit $\hat{G}$ le 1-squelette du revêtement universel de $\mathcal{K}(G)$. Soit $\varphi : \mathcal{K}(\hat{G}) \rightarrow \mathcal{K}(G)$ un revêtement et soit $\hat{v}_0 \in V(\hat{G})$ un sommet de $\hat{G}$ tel que $\varphi(\hat{v}_0) = v_0$.

Par le lemme 8.30, en exécutant $\mathcal{A}$ sur $\hat{G}$, $\mathcal{A}$ s'arrête après r étapes. Prenons le graphe $H = B_{\hat{G}}(\hat{v}_0, r+1)$. Comme $G \in \mathcal{IC}$, $\hat{G}$ est un graphe infini et donc, $|V(H)| \geq r+1$.

En exécutant $\mathcal{A}$ sur H , $\mathcal{A}$ se comporte de façon identique à une exécution dans $\hat{G}$ pendant les r premières étapes (car les r premières étapes dépendent uniquement de la boule $B_{\hat{G}}(\hat{v}_0, r) \simeq B_H(\hat{v}_0, r)$). Donc $\mathcal{A}$, exécuté dans H avec le sommet base $\hat{v}_0$, s'arrêtera après r étapes. Comme $|V(H)| \geq r+1$, $\mathcal{A}$ s'arrête avant d'avoir exploré tous les sommets de H . Nous obtenons une contradiction. $\square$

La borne inférieure sur la complexité de $\mathcal{A}$ est obtenue à partir du lemme de relèvement ci-dessus appliqué à tout revêtement $\varphi : \mathcal{K}(\hat{G}) \rightarrow \mathcal{K}(G)$. Supposons que nous ayons un algorithme $\mathcal{A}$ qui s'arrête sur un graphe G lors d'une phase i . Si $|V(\hat{G})| > i+1$ alors $\mathcal{A}$ s'arrête sur $\hat{G}$ sans avoir visité tous les sommets de $\hat{G}$ (au plus un sommet peut être visité par étape), une contradiction. Nous obtenons donc le lemme suivant.

Corollaire 8.32 (Borne inférieure). *Pour tout graphe G et tout algorithme d'exploration $\mathcal{A}$ pour G , $\mathcal{A}$ nécessite au moins $|V(\widehat{G})| - 1$ déplacements de l'agent.*

Nous prouvons maintenant que comme les arbres pour les revêtements de graphes, la famille $\mathcal{FC}$ est la famille qui contient tous les graphes ayant un rayon de quasi-revêtement (simplicial) borné.

Proposition 8.33. *Pour tout $G \in \mathcal{FC}$, il existe k tel que pour tout sommet $v \in V(G)$ et pour tout graphe G' , s'il existe un quasi-revêtement non trivial $\gamma : \mathcal{K}(G') \rightarrow \mathcal{K}(G)$ de rayon r centré en $v \in V(G)$ alors $r \leq k$.*

Démonstration. Nous montrons ce lemme par l'absurde. Soit G, G' deux graphes (finis) tels que $G \in \mathcal{FC}$ et soit $\gamma : G' \rightarrow G$ un quasi-revêtement non trivial de rayon $r > |V(\widehat{G})|$ centré en $u \in V(G)$. Par définition, il existe un graphe K et un revêtement $\varphi : K \rightarrow G$ et il existe un sommet $\tilde{u} \in V(K)$ tel que $\varphi(\tilde{u}) = u$. Il existe aussi un isomorphisme $\psi : B_{G'}(u', r) \rightarrow B_K(\tilde{u}, r)$. Puisque le revêtement est non trivial, $|V(K)| \geq r$.

Par définition des revêtements universels, $\widehat{G}$ est aussi un revêtement de K . Nous obtenons donc que $r \leq |V(K)| \leq |V(\widehat{G})| < r$. Nous obtenons une contradiction. $\square$

Nous obtenons à partir du lemme ci-dessus les propositions suivantes.

Proposition 8.34. *Pour tout graphe $G \in \mathcal{FC}$, G possède un rayon de quasi-revêtement de rayon borné.*

En remarquant que seule la taille du revêtement universel limite le rayon des quasi-revêtements de graphe, nous obtenons la proposition suivante.

Proposition 8.35. *Pour tout graphe $G \in \mathcal{IC}$, pour tout sommet $v \in V(G)$ et pour tout entier r , il existe G' et il existe un quasi-revêtement $\gamma : G' \rightarrow G$ de rayon r centré en v .*

8.4 Exploration de $\mathcal{FC}$

Nous présentons dans cette section un algorithme d'exploration pour la famille de graphes $\mathcal{FC}$. Le but de l'algorithme 8 est d'explorer tous les chemins d'une longueur suffisamment large pour contenir tout le graphe. Une fois la bonne longueur atteinte, l'agent pourra s'arrêter. La principale difficulté à résoudre est donc de décider quand s'arrêter. Cela est possible car les graphes de $\mathcal{FC}$ ont tous un rayon de quasi-revêtement borné (Proposition 8.34). Intuitivement, il n'est pas possible de trouver un complexe *simplement connexe* ressemblant "localement" à une *stricte sous-partie* d'un autre complexe (Corollaire 8.39).

8.4.1 Présentation de l'algorithme

Soit G un graphe, soit $v_0 \in V(G)$, le sommet base de l'agent, et soit k , un entier initialisé à 1. L'algorithme 8 s'exécute par phase. Au début d'une phase, l'agent suit tous les chemins de longueur au plus $2k$ partant de v_0 tout en construisant la vue $\mathcal{T}(v_0, 2k)$ de v_0 à distance $2k$. Il explorera ainsi tous les sommets contenus dans la boule $B(v_0, 2k)$. Puis, l'algorithme recherche un graphe H simplement connexe (Ligne 8) ; possédant au plus k sommets (Ligne 6) et possédant un sommet ayant la même vue que le sommet base de l'agent à distance $2k$ (Ligne 7).

Si un tel graphe H est trouvé, l'algorithme arrête son exécution. Sinon, l'agent incrémente k , rejoint sa base et commence une nouvelle phase.

Remarque 8.36 (Calculabilité de l'algorithme). Décider la k -contractibilité d'un cycle donné est calculable en considérant toutes les séquences possibles d'homotopies élémentaires de longueur au plus k . Comme le nombre total des cycles simples dans un graphe est fini et énumérable, l'algorithme 8 peut être implémenté sur une machine de Turing.

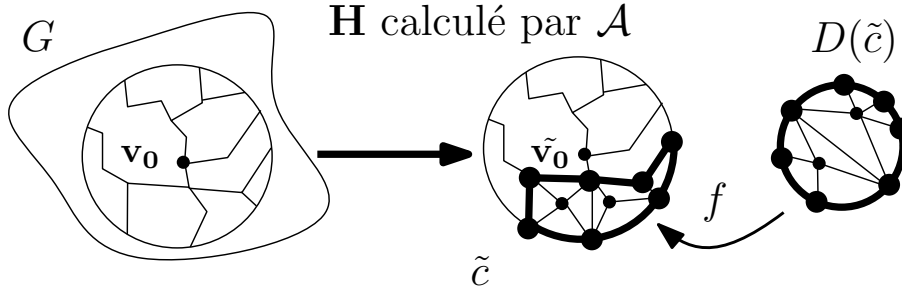


FIGURE 8.16 – Illustration de l'algorithme 8

8.4.2 Correction de l'algorithme

Pour prouver la correction de l'algorithme, nous prouvons que lorsque le premier graphe H satisfaisant les propriétés de l'algorithme est trouvé, alors $\mathcal{K}(H)$ est en fait le revêtement universel de $\mathcal{K}(G)$ où G est le graphe exploré (Corollaire 8.39).

Rappelons qu'étant donné un chemin p dans un graphe G , $\lambda(p)$ est la séquence de numéros de ports sortant suivie par p dans G . Nous noterons par $G[v][\lambda(p)]$, le sommet de G atteint en suivant la séquence de numéros de port $\lambda(p)$ à partir d'un sommet v . Nous montrons (Proposition 8.37) que si nous fixons un sommet $\tilde{v}_0 \in V(H)$ tel que $\tilde{v}_0 \sim_{2k} v_0$ alors il est possible de définir de façon non ambiguë une fonction φ allant des sommets de H vers les sommets de G de la façon suivante.

Algorithme 8 : Exploration de la famille $\mathcal{FC}$

```

1  $k := 0$ ;
2 Répéter
3   Incréments  $k$  ;
4   Calculer  $\mathcal{T}(v_0, 2k)$  en explorant tous les chemins de longueur  $2k$  issue de  $v_0$ ;
5   Énumérer tous les graphes de taille inférieure à  $k$  et chercher un graphe  $H$  tel que :
6   -  $|V(H)| < k$ , et
7   - Il existe un sommet  $\tilde{v}_0 \in V(H)$  tel que  $\tilde{v}_0 \sim_{2k} v_0$ , et
8   - tout cycle simple de  $\mathcal{K}(H)$  est  $k$ -contractible ;
9 Jusqu'à ce que  $H$  soit défini;
10 Arrêter l'exploration;
```

Proposition 8.37. Soit G un graphe tel que l'algorithme 8, exécuté dans G avec un sommet base v_0 , s'arrête. Soit l'entier k et le graphe H calculés par l'algorithme avant de s'arrêter et soit $\tilde{v}_0 \in V(H)$ tel que $v_0 \sim_{2k} \tilde{v}_0$.

Il existe un morphisme $\varphi : H \rightarrow G$ tel que pour tout chemin p de taille k de H allant de $\tilde{v}_0$ à $\tilde{u}$, $\varphi(\tilde{u}) = G[v_0][\lambda(p)] = u \in V(G)$. De plus, pour tout sommet $\tilde{u} \in V(H)$, pour tout couple de chemins $\tilde{q}, \tilde{q}'$ qui, à partir de $\tilde{v}_0$, mènent à $\tilde{u}$ dans H , $G[v_0][\lambda(\tilde{q})] = G[v_0][\lambda(\tilde{q}')] = u$.

Pour montrer cette proposition, il va nous falloir montrer que le morphisme φ est bien défini pour tout couple de chemins entre deux sommets identiques.

La preuve de la proposition 8.37 est technique et nécessite une induction non triviale dans l'espace des homotopies. Premièrement, nous montrons qu'un tel morphisme φ est bien défini pour tout chemin et tout cycle attaché à l'une des extrémités du chemin.

Lemme 8.38. *Soit G un graphe tel que l'algorithme 8 exécuté sur G avec un sommet base $v_0 \in V(G)$ s'arrête. Soit k et H l'entier et le graphe calculés par l'algorithme avant de s'arrêter et soit $\tilde{v}_0 \in V(H)$ tel que $v_0 \sim_{2k} \tilde{v}_0$.*

Pour tout chemin $\tilde{p} = (\tilde{v}_0, \tilde{v}_1, \dots, \tilde{v}_{|\tilde{p}|} = \tilde{u})$ et tout cycle $\tilde{c} = (\tilde{u} = \tilde{u}_0, \dots, \tilde{u}_{|\tilde{c}|} = \tilde{u})$ de H tel que $|\tilde{p}| + |\tilde{c}| + \mathcal{A}(\tilde{c}) \leq 2k$, il existe un unique sommet $u \in V(G)$ tel que $G[v_0][\lambda(\tilde{p})] = u = G[u][\lambda(\tilde{c})]$.

Démonstration. Remarquons que $|V(H)| < k$, que H est simplement connexe et que pour tout cycle simple $\tilde{c} \subset H$, $\mathcal{A}(\tilde{c}) \leq k$. Comme $v_0 \sim_{2k} \tilde{v}_0$ et comme $|\tilde{p}| \leq 2k$, il y a un unique $u \in V(G)$ tel que $u = G[v_0][\lambda(\tilde{p})]$, de plus $\nu(u) = \nu(\tilde{u})$. Soit $m = |\tilde{c}|$; comme $|\tilde{p}| + |\tilde{c}| \leq 2k$, pour tout $0 \leq i \leq m$, il existe un sommet u_i tel que $G[u_i][\lambda(\tilde{u}_0, \tilde{u}_1, \dots, \tilde{u}_i)] = u_i$ et $G[u_i][\lambda(\tilde{u}_i, \tilde{u}_{i-1}, \dots, \tilde{u}_0)] = u_0$.

Nous montrons ce lemme par une induction sur $\mathcal{A}(\tilde{c})$, l'aire du cycle $\tilde{c}$. Si $\mathcal{A}(\tilde{c}) = 0$, alors soit $|\tilde{c}| = (\tilde{u})$ et nous prouvons ce lemme; soit $\tilde{c} = (\tilde{u} = \tilde{u}_0, \tilde{u}_1, \tilde{u}_2 = \tilde{u})$ et alors $u_0 = u_2$ car $\nu(u) = \nu(\tilde{u})$.

Supposons que $\mathcal{A}(\tilde{c}) \geq 1$. Prenons un diagramme de disque (D, f) minimal pour $\tilde{c}$ et soit $\bar{u} = \bar{u}_0, \bar{u}_1, \dots, \bar{u}_m = \bar{u}$ les sommets de ∂D envoyés par f respectivement sur $\tilde{u} = \tilde{u}_0, \tilde{u}_1, \dots, \tilde{u}_m = \tilde{u}$. Comme D est une triangulation planaire, il existe un chemin $(\bar{w}_1 = \bar{w}_1, \bar{w}_2, \dots, \bar{w}_\ell = \bar{u}_{m-1})$ dans le voisinage de $\bar{u} = \bar{u}_0 = \bar{u}_m$.

Nous distinguons deux cas en fonction de si $\bar{w}_j$ est en bordure de D , i.e., sur ∂D , ou pas.

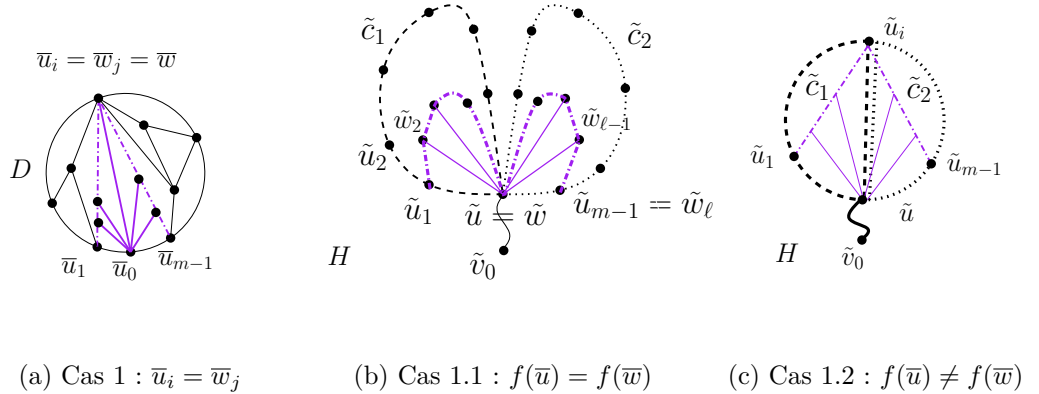


FIGURE 8.17 – Cas 1 de la preuve de la preuve du lemme 8.38

Cas 1. Il y a $2 \leq i \leq m-2$ et $2 \leq j \leq \ell-1$ tel que $\bar{u}_i = \bar{w}_j$.

Soit $\bar{w} = \bar{w}_j = \bar{u}_i$ et soit $\tilde{w} = f(\bar{w}) = \tilde{u}_i$. Comme f est une fonction simpliciale et comme $\bar{u}\bar{w} \in E(D)$, soit $\tilde{u} = \tilde{w}$, soit $\tilde{u}\tilde{w} \in E(H)$.

Supposons premièrement que $\tilde{u} = \tilde{w}$. Soit $\tilde{c}_1 = (\tilde{u} = \tilde{u}_0, \tilde{u}_1, \dots, \tilde{u}_{i-1}, \tilde{w} = \tilde{u})$ et $\tilde{c}_2 = (\tilde{u} = \tilde{w}, \tilde{u}_{i+1}, \dots, \tilde{u}_m = \tilde{u})$.

Concernant le cycle $\tilde{c}_1$, nous pouvons construire un diagramme de disque (D_1, f_1) à partir de (D, f) en retirant tous les sommets qui ne sont pas dans le cycle $(\bar{u}_0, \bar{u}_1, \dots, \bar{u}_{i-1}, \bar{w}, \bar{u})$ et en

contractant l'arête $\overline{u\bar{w}}$. Comme le triangle $\overline{u_0\bar{w}_{\ell-1}\bar{w}_\ell}$ n'apparaît pas dans D_1 , $\mathcal{A}(\tilde{c}_1) \leq \mathcal{A}(D_1) < \mathcal{A}(D) = \mathcal{A}(\tilde{c})$. De façon équivalente, $\mathcal{A}(\tilde{c}_2) < \mathcal{A}(\tilde{c})$. De plus, remarquons que $|\tilde{p}| + |\tilde{c}_1| + \mathcal{A}(\tilde{c}_1) < |\tilde{p}| + |\tilde{c}| + \mathcal{A}(\tilde{c}) \leq 2k$ et $|\tilde{p}| + |\tilde{c}_2| + \mathcal{A}(\tilde{c}_2) \leq 2k$. Par notre hypothèse d'induction, nous obtenons que $G[u][\lambda(\tilde{c}_1)] = u$ et $G[u][\lambda(\tilde{c}_2)] = u$. Ainsi, $G[u][\lambda(\tilde{c})] = G[u][\lambda(\tilde{c}_1) \cdot \lambda(\tilde{c}_2)] = G[u][\lambda(\tilde{c}_2)] = u$.

Supposons maintenant que $\tilde{u}\tilde{w} \in E(H)$ et soit $\tilde{c}_1 = (\tilde{u} = \tilde{u}_0, \tilde{u}_1, \dots, \tilde{u}_{i-1}, \tilde{w}, \tilde{u})$ et $\tilde{c}_2 = (\tilde{u}, \tilde{w}, \tilde{u}_{i+1}, \dots, \tilde{u}_m = \tilde{u})$. Comme dans le premier cas, il est facile de voir qu'à partir du diagramme (D, f) , nous pouvons construire deux diagrammes de disques (D_1, f_1) et (D_2, f_2) pour $\tilde{c}_1$ et $\tilde{c}_2$ tels que $\mathcal{A}(\tilde{c}_1) < \mathcal{A}(\tilde{c})$ et $\mathcal{A}(\tilde{c}_2) < \mathcal{A}(\tilde{c})$. Comme précédemment, $|\tilde{p}| + |\tilde{c}_1| + \mathcal{A}(\tilde{c}_1) \leq 2k$ et $|\tilde{p}| + |\tilde{c}_2| + \mathcal{A}(\tilde{c}_2) \leq 2k$. Par induction, nous avons que $G[u][\lambda(\tilde{c}_1)] = u$ et $G[u][\lambda(\tilde{c}_2)] = u$. Soit $\tilde{c}' = \tilde{c}_1 \cdot \tilde{c}_2 = (\tilde{u} = \tilde{u}_0, \tilde{u}_1, \dots, \tilde{u}_{i-1}, \tilde{w}, \tilde{u}, \tilde{w}, \tilde{u}_{i+1}, \dots, \tilde{u}_m = \tilde{u})$. Notons que $G[u][\lambda(\tilde{c}')] = G[u][\lambda(\tilde{c})]$. Ainsi, $G[u][\lambda(\tilde{c})] = G[u][\lambda(\tilde{c}_1) \cdot \lambda(\tilde{c}_2)] = G[u][\lambda(\tilde{c}_2)] = u$.

Cas 2. Pour tout $1 \leq i \leq m-1$ et pour tout $2 \leq j \leq \ell-1$, $\bar{u}_i \neq \bar{w}_j$. Pour tout $1 \leq i \leq \ell$, soit $\tilde{w}_i = f(\bar{w}_i)$. Premièrement, supposons que $\ell = 2$ et que $\tilde{u}_1 = \tilde{w}_1 = \tilde{w}_2 = \tilde{u}_{m-1}$. Prenons le chemin $\tilde{p}' = \tilde{p} \cdot (u_1)$ et prenons le cycle $\tilde{c}' = (\tilde{u}_1, \tilde{u}_2, \dots, \tilde{u}_{m-2})$. Notons que $|\tilde{p}'| = |\tilde{p}| + 1$ et que $|\tilde{c}'| = |\tilde{c}| - 2$. À partir de (D, f) , nous pouvons construire un diagramme de disque pour $\tilde{c}'$ en supprimant le sommet $\bar{u}_0$ et en contractant une arête $\tilde{u}_1\tilde{u}_{m-1}$. Ainsi, $\mathcal{A}(\tilde{c}') < \mathcal{A}(\tilde{c})$ et $|\tilde{p}'| + |\tilde{c}'| + \mathcal{A}(\tilde{c}') < |\tilde{p}| + |\tilde{c}| + \mathcal{A}(\tilde{c}) \leq 2k$. Par induction, $G[u_1][\lambda(\tilde{c}')] = u_1$, et donc, $G[u][\lambda(\tilde{c})] = G[u_1][\lambda(\tilde{c}') \cdot \lambda(\tilde{u}_{m-1}, \tilde{u})] = G[u_1][\lambda(\tilde{u}_1, \tilde{u})] = u$.

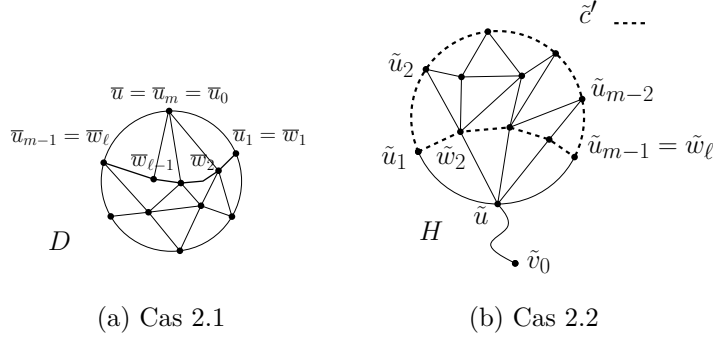


FIGURE 8.18 – Cas 2 de la preuve de la preuve du lemme 8.38

Supposons que $\tilde{w}_1 \neq \tilde{w}_\ell$ où $\ell > 2$. Comme (D, f) est un diagramme de disques minimal, nous obtenons que pour tout $1 \leq i \leq \ell-1$, $\tilde{w}_i \neq \tilde{w}_{i+1}$, $\tilde{w}_i\tilde{w}_{i+1} \in E(H)$ et $\tilde{u}\tilde{w}_i \in E(H)$.

Soit $\tilde{p}' = \tilde{p} \cdot (\tilde{u}_1)$, soit $\tilde{p}_1 = (\tilde{u}_1, \tilde{u}_2, \dots, \tilde{u}_{m-1})$, soit $\tilde{p}_2 = (\tilde{u}_{m-1} = \tilde{w}_\ell, \tilde{w}_{\ell-1}, \dots, \tilde{w}_1 = \tilde{u}_1)$ et soit $\tilde{c}' = \tilde{p}_1 \cdot \tilde{p}_2$. À partir de (D, f) , nous pouvons construire un diagramme de disque pour $\tilde{c}'$ en supprimant un sommet $\bar{u}_0$, et donc, $\mathcal{A}(\tilde{c}') < \mathcal{A}(\tilde{c})$. De plus, notons que $|\tilde{p}'| = |\tilde{p}| + 1$, que $|\tilde{c}'| = |\tilde{c}| + \ell - 3$ et que $\mathcal{A}(\tilde{c}') \leq \mathcal{A}(\tilde{c}) - \ell + 1$. Ainsi, $|\tilde{p}'| + |\tilde{c}'| + \mathcal{A}(\tilde{c}') \leq |\tilde{p}| + |\tilde{c}| + \mathcal{A}(\tilde{c}) - 1 < 2k$. Donc, par induction, $G[u_1][\lambda(\tilde{c}')] = G[u_{m-1}][\lambda(\tilde{p}_2)] = u_1$.

Soit $\tilde{p}_2' = (\tilde{u}_1 = \tilde{w}_1, \tilde{w}_2, \dots, \tilde{w}_\ell = \tilde{u}_{m-1})$. Notons que $G[u_{m-1}][\lambda(\tilde{p}_2) \cdot \lambda(\tilde{p}_2')] = u_{m-1}$. Ainsi $G[u][\lambda(\tilde{c})] = G[u_1][\lambda(\tilde{p}_1) \cdot \lambda(\tilde{u}_{m-1}, \tilde{u})] = G[u_1][\lambda(\tilde{p}_1) \cdot \lambda(\tilde{p}_2) \cdot \lambda(\tilde{p}_2') \cdot \lambda(\tilde{u}_{m-1}, \tilde{u})] = G[u_1][\lambda(\tilde{p}_2') \cdot \lambda(\tilde{u}_{m-1}, \tilde{u})]$. Comme $v_0 \sim_{2k} \tilde{v}_0$ et comme $|\tilde{p}| < 2k$, $\nu(\tilde{u}) = \nu(u)$. Ainsi, comme $\tilde{p}_2' = (\tilde{w}_1, \dots, \tilde{w}_\ell)$ est un chemin situé dans le voisinage de $\tilde{u}$, $G[u_1][\lambda(\tilde{p}_2') \cdot \lambda(u_{m-1}, u)] = G[u][\lambda(\tilde{u}, \tilde{u}_1) \cdot \lambda(\tilde{p}_2') \cdot \lambda(\tilde{u}_{m-1}, \tilde{u})] = u$ et ainsi, $G[u][\lambda(\tilde{c})] = u$. $\square$

Nous sommes maintenant en mesure de prouver la proposition 8.37.

Preuve de la proposition 8.37. Raisonnons par l'absurde et considérons l'ensemble $\mathcal{P}_0$ composé de toutes les paires de chemins $(\tilde{q}, \tilde{q}')$ qui sont des contre-exemples au résultat et tel que $|\tilde{q}| \leq |\tilde{q}'|$.

Parmi tous les couples de chemins de $\mathcal{P}_0$, soit $\mathcal{P}_1$ l'ensemble des couples de chemins tel que $|\tilde{q}| + |\tilde{q}'|$ est de longueur minimum. Parmi tous les couples de chemins de $\mathcal{P}_1$, soit $\mathcal{P}_2$ l'ensemble des couples de chemins qui minimisent $|\tilde{q}| - \ell$ où ℓ est la longueur du plus grand préfixe commun entre $\tilde{q}$ et $\tilde{q}'$.

Sans perte de généralité, soit $(\tilde{q}, \tilde{q}') \in \mathcal{P}_2$ tel que $\tilde{q} = (\tilde{v}_0 = \tilde{u}_0, \tilde{u}_1, \dots, \tilde{u}_m = \tilde{u})$ et $\tilde{q}' = (\tilde{v}_0 = \tilde{u}'_0, \tilde{u}'_1, \dots, \tilde{u}'_{m'} = \tilde{u})$.

Supposons en premier que le chemin $\tilde{q}$ n'est pas simple, i.e., il existe $i < j$ tel que $\tilde{u}_i = \tilde{u}_j$. Choisissons i et j tels que j est minimum. Ainsi, pour tout $0 \leq i_1 < i_2 \leq j - 1$, $\tilde{u}_{i_1} \neq \tilde{u}_{i_2}$. Prenons un chemin $\tilde{p} = (\tilde{u}_0, \tilde{u}_1, \dots, \tilde{u}_i)$ et un cycle $\tilde{c} = (\tilde{u}_i, \tilde{u}_{i+1}, \dots, \tilde{u}_j = \tilde{u}_i)$. Notons que tous les sommets de $\tilde{p}$ et $\tilde{c}$ sont distincts et comme $|V(H)| \leq k$, alors $|\tilde{p}| + |\tilde{c}| \leq k$. Comme tous les cycles de H sont k -contractibles, $\mathcal{A}(\tilde{c}) \leq k$. Ainsi, par le lemme 8.38, $G[v_0][\lambda(\tilde{u}_0, \dots, \tilde{u}_i)] = G[v_0][\lambda(\tilde{u}_0, \dots, \tilde{u}_j)]$. Soit $\tilde{q}_1 = (\tilde{u}_0, \dots, \tilde{u}_{i-1}, \tilde{u}_i = \tilde{u}_j, \tilde{u}_{j+1}, \dots, \tilde{u}_m)$ et notons que nous venons de prouver que $G[v_0][\lambda(\tilde{q})] = G[v_0][\lambda(\tilde{q}_1)]$. Comme $\tilde{q}_1$ est un chemin de $\tilde{v}_0$ à $\tilde{u}$ et comme $|\tilde{q}_1| < |\tilde{q}|$, à partir de la définition de $\mathcal{P}_1$, $G[v_0][\lambda(\tilde{q}_1)] = G[v_0][\lambda(\tilde{q}')]$. Ainsi, $G[v_0][\lambda(\tilde{q})] = G[v_0][\lambda(\tilde{q}')]$ et donc, $\tilde{q}$ est un chemin simple. En utilisant les mêmes arguments, nous montrons que $\tilde{q}'$ est aussi un chemin simple.

Soit ℓ la longueur de plus long préfixe commun de $\tilde{q}$ et $\tilde{q}'$. Notons que si $\ell = |\tilde{q}|$, alors $\tilde{q} = \tilde{q}'$ et il n'y a rien à prouver. Supposons que $\ell < |\tilde{q}|$ et notons que pour tout $i \leq \ell$, $\tilde{u}_i = \tilde{u}'_i$. Prenons le plus petit index $i > \ell$ tel que $\tilde{u}_i$ apparaît dans $\tilde{q}'$. Soit j le plus petit index tel que $\tilde{u}_i = \tilde{u}'_j$. Notons que comme $\tilde{q}$ et $\tilde{q}'$ sont simples, $j > \ell$ et tous les sommets $u_1, \dots, u_\ell, u_{\ell+1}, \dots, u_i$, et $u'_{\ell+1}, \dots, u'_{j-1}$ sont distincts.

Soit $\tilde{q}_1 = (\tilde{u}_0, \tilde{u}_1, \dots, \tilde{u}_i)$ et soit $\tilde{q}'_1 = (\tilde{u}'_0, \tilde{u}'_1, \dots, \tilde{u}'_j)$. Prenons le chemin simple $\tilde{p} = (\tilde{u}_0, \tilde{u}_1, \dots, \tilde{u}_\ell)$ et le cycle simple $\tilde{c} = (\tilde{u}_\ell, \tilde{u}_{\ell+1}, \dots, \tilde{u}_i = \tilde{u}'_j, \tilde{u}'_{j-1}, \dots, \tilde{u}'_\ell)$. Comme tous les sommets de $\tilde{p}$ et $\tilde{c}$ sont distincts, $|\tilde{p}| + |\tilde{c}| \leq k$, et comme tous les cycles simples de H sont k -contractibles, $\mathcal{A}(\tilde{c}) \leq k$. Donc, par le lemme 8.38, il existe un unique sommet $w \in V(G)$ vérifiant que $w = G[v_0][\lambda(\tilde{q}_1)] = G[v_0][\lambda(\tilde{q}'_1)]$.

Soit $\tilde{q}_2 = (\tilde{u}_i, \tilde{u}_{i+1}, \dots, \tilde{u}_m)$ et soit $\tilde{q}'_2 = (\tilde{u}'_j, \tilde{u}'_{j+1}, \dots, \tilde{u}'_{m'})$. Notons que $G[v_0][\lambda(\tilde{q})] = G[w][\lambda(\tilde{q}_2)]$ et que $G[v_0][\lambda(\tilde{q}')] = G[w][\lambda(\tilde{q}'_2)]$. Si $j < i$, soit $\tilde{q}'_3 = \tilde{q}' = \tilde{q}'_1 \cdot \tilde{q}'_2$ et soit $\tilde{q}_3 = \tilde{q}'_1 \cdot \tilde{q}_2$. Si $i \leq j$, soit $\tilde{q}_3 = \tilde{q} = \tilde{q}_1 \cdot \tilde{q}_2$ et soit $\tilde{q}'_3 = \tilde{q}_1 \cdot \tilde{q}'_2$. Notons que si $j < i$ ou si $i < j$, alors $|\tilde{q}_3| + |\tilde{q}'_3| < |\tilde{q}| + |\tilde{q}'|$. Si $i = j$, $|\tilde{q}_3| + |\tilde{q}'_3| = |\tilde{q}| + |\tilde{q}'|$, et la longueur du préfixe commun entre $\tilde{q}_3$ et $\tilde{q}'_3$ est $i > \ell$. Ainsi, dans tous les cas, à partir de notre choix de $\tilde{q}$ et $\tilde{q}'$, nous savons que $G[v_0][\lambda(\tilde{q}_3)] = G[v_0][\lambda(\tilde{q}'_3)]$. De plus, comme $w = G[v_0][\lambda(\tilde{q}_1)] = G[v_0][\lambda(\tilde{q}'_1)]$, nous obtenons que $G[w][\lambda(\tilde{q}_2)] = G[v_0][\lambda(\tilde{q}_3)] = G[v_0][\lambda(\tilde{q}'_3)] = G[w][\lambda(\tilde{q}'_2)]$. Ainsi, $G[v_0][\lambda(\tilde{q})] = G[w][\lambda(\tilde{q}_2)] = G[w][\lambda(\tilde{q}'_2)] = G[v_0][\lambda(\tilde{q}')]$, contredisant notre hypothèse sur le couple $(\tilde{q}, \tilde{q}')$.

□

Corollaire 8.39. Soit G un graphe et soit $v_0 \in V(G)$ un sommet tel que l'algorithme 8, exécuté dans G avec un sommet base v_0 , s'arrête. Soit H le graphe calculé par l'algorithme avant de s'arrêter. Le complexe de cliques $\mathcal{K}(H)$ est le revêtement universel de $\mathcal{K}(G)$.

Démonstration. Par la définition de l'algorithme 8, le complexe $\mathcal{K}(H)$ est simplement connexe. Nous prouverons donc que $\mathcal{K}(H)$ est un revêtement de $\mathcal{K}(G)$ pour obtenir le résultat.

Soit un sommet $\tilde{v}_0 \in V(H)$ tel que $v_0 \sim_{2k} \tilde{v}_0$. Pour tout sommet $\tilde{u} \in V(H)$, Soit $\tilde{p}_{\tilde{u}}$ un chemin de $\tilde{v}_0$ à $\tilde{u}$ et soit $\varphi(\tilde{u}) = G[v_0][\lambda(\tilde{p}_{\tilde{u}})]$.

Par la proposition 8.37, $\varphi(\tilde{u})$ est indépendant du choix de $\tilde{p}_{\tilde{u}}$. Comme $v_0 \sim_{2k} \tilde{v}_0$ et comme $|V(H)| \leq k$, pour tout $\tilde{u} \in V(H)$, $\nu(\varphi(\tilde{u})) = \nu(\tilde{u})$.

Nous obtenons que pour tout $\tilde{u} \in V(H)$ et pour tout sommet voisin $\tilde{w} \in N_H(\tilde{u})$, il existe un unique sommet $w \in N_G(\varphi(\tilde{u}))$ tel que $\lambda(\tilde{u}, \tilde{w}) = \lambda(\varphi(\tilde{u}), w)$. Réciproquement, pour tout $w \in N_G(\varphi(\tilde{u}))$, il existe un unique sommet $\tilde{w} \in N_H(\tilde{u})$ tel que $\lambda(\tilde{u}, \tilde{w}) = \lambda(\varphi(\tilde{u}), w)$.

Dans les deux cas, prenons $\tilde{p}_{\tilde{w}} = \tilde{p}_{\tilde{u}} \cdot (\tilde{u}, \tilde{w})$; $\tilde{p}_{\tilde{w}}$ est un chemin de $\tilde{v}_0$ à $\tilde{w}$. Par la proposition 8.37, $\varphi(\tilde{w}) = G[v_0][\lambda(\tilde{p}_{\tilde{w}})] = G[u][\lambda(\tilde{u}, \tilde{w})] = w$. Donc, φ est un revêtement de graphe de H dans G . Comme les étiquetages des jumelles sont préservés dans les vues des sommets, φ préserve aussi les étiquetages des jumelles et donc le complexe $\mathcal{K}(H)$ est un revêtement du complexe $\mathcal{K}(G)$. $\square$

Avant de prouver que l'algorithme 8 est un algorithme d'exploration pour $\mathcal{FC}$, remarquons que lorsque l'on considère des complexes (ou graphes) connexes, les revêtements sont surjectifs. Par conséquent, l'agent aura exploré le graphe G lorsque l'algorithme s'arrête.

Théorème 8.1. *L'algorithme 8 est un algorithme d'exploration pour $\mathcal{FC}$.*

Démonstration. Soit $G \in \mathcal{FC}$. Exécutons l'algorithme sur G . D'après le corollaire 8.39, nous savons que si l'algorithme 8 s'arrête, alors le complexe de cliques $\mathcal{K}(H)$ du graphe H calculé par l'algorithme est un revêtement de $\mathcal{K}(G)$. De plus, comme $|V(G)| \leq |V(H)| \leq k$ et comme l'agent a construit $\mathcal{T}_G(v, 2k)$, l'agent a visité tous les sommets de G . Nous venons de prouver que si l'algorithme 8 s'arrête, alors le graphe est exploré.

Soit $\hat{G}$, le 1-squelette du revêtement universel de $\mathcal{K}(G)$. Comme $G \in \mathcal{FC}$, $\hat{G}$ est fini et il existe $k' \in \mathbb{N}$ tel que tout cycle simple de $\hat{G}$ est k' -contractible. Soit $k = \max(|V(\hat{G})|, k')$. À la fin de la phase k , comme $\mathcal{K}(\hat{G})$ est le revêtement universel de $\mathcal{K}(G)$, il existe un sommet $\tilde{v}_0 \in V(\hat{G})$ tel que $\mathcal{T}_G(v_0) = \mathcal{T}_{\hat{G}}(\tilde{v}_0)$. Ainsi, $\mathcal{T}_G(v_0, 2k) = \mathcal{T}_{\hat{G}}(\tilde{v}_0, 2k)$, $|V(\hat{G})| \leq k$, et tout cycle simple de $\mathcal{K}(\hat{G})$ est k -contractible. La condition d'arrêt de l'algorithme est donc satisfaite lors de l'itération k . $\square$

À partir de la proposition 8.31 et par le théorème 8.1 ci-dessus, nous obtenons le corollaire suivant.

Corollaire 8.40. *La famille $\mathcal{FC}$ est maximum pour l'exploration avec jumelles.*

8.5 Complexité du problème d'exploration

Dans la section précédente, nous avons fourni une borne inférieure sur l'exploration de la famille $\mathcal{FC}$ (Corollaire 8.32). Dans cette section, nous montrons que la complexité de tout algorithme d'exploration pour la famille $\mathcal{FC}$ ne peut pas être bornée supérieurement par une fonction calculable.

La première raison de cette impossibilité est simple et intuitive. Si le 1-squelette $\hat{G}$ du revêtement universel du complexe de cliques de G est fini, alors par le corollaire 8.32, en exécutant sur G n'importe quel algorithme d'exploration, l'exécution nécessitera au moins $|V(\hat{G})| - 1$ étapes avant de s'arrêter. En d'autres mots, si une telle borne existe alors elle sera bornée par une fonction prenant en paramètre la taille du revêtement universel $\hat{G}$.

Toutefois, nous montrons, dans le théorème 8.2, qu'en considérant seulement les graphes ayant un complexe de cliques simplement connexe (i.e., un graphe isomorphe à son revêtement universel), il n'existe pas d'algorithme d'exploration pour cette classe de graphes ayant une complexité bornée par une fonction calculable. Nos preuves s'appuient sur un résultat de Haken [Hak73] montrant qu'il est indécidable de décider si un complexe simplicial fini est simplement connexe ou non.

Remarque 8.41. Il est important de souligner que le résultat original de Haken s'applique aux complexes simpliciaux en général et non au cas plus restreint des complexes de cliques, qui sont une sous-famille des complexes simpliciaux. Toutefois, comme la subdivision barycentrique K' d'un complexe simplicial arbitraire K est un complexe de clique simplement connexe si et seulement si K est simplement connexe (voir [Hat02]), la réduction est valide.

Nous montrons maintenant, par l'absurde, que s'il existe un algorithme d'exploration $\mathcal{A}$ et une fonction calculable $\mathfrak{t} : \mathbb{N} \rightarrow \mathbb{N}$ tels que pour tout graphe $G \in \mathcal{SC}$, $\mathcal{A}$ visite tous les sommets de G et s'arrête en au plus $\mathfrak{t}(|V(G)|)$ étapes, alors il est possible de décider, de façon algorithmique, si le complexe de cliques d'un graphe G arbitraire est simplement connexe ou non. Cependant, ce problème étant indécidable [Hak73], nous obtenons une contradiction.

8.5.1 L'algorithme

L'algorithme 9 est un algorithme de décision pour la simple connexité. Notons que cet algorithme n'est pas un algorithme distribué. Il prend en entrée un graphe G et décide si le complexe de cliques $\mathcal{K}(G)$ de G est simplement connexe ou non. Pour cela, l'algorithme 8 simule un algorithme d'exploration $\mathcal{A}$ donné sur le graphe G pendant $\mathfrak{t}(|V(G)|)$ étapes. Puis, si $\mathcal{A}$ ne s'est pas arrêté pendant les $\mathfrak{t}(|V(G)|)$ premières étapes, l'algorithme retourne NON car par définition $\mathcal{A}$ est correct pour $\mathcal{FC}$ et comme $\mathcal{A}$ ne s'est pas arrêté, alors $G \in \mathcal{IC}$. Sinon, $\mathcal{A}$ s'est arrêté en moins de $\mathfrak{t}(|V(G)|)$ étapes. Dans ce cas-là, nous énumérons tous les graphes et cherchons un graphe H vérifiant que,

- $|V(G)| < |V(H)| \leq \mathfrak{t}(|V(G)|)$,
- le complexe de cliques $\mathcal{K}(H)$ est un revêtement de $\mathcal{K}(G)$.

Si un graphe H est trouvé respectant les propriétés voulues, alors l'algorithme retourne NON car par définition, $G \notin \mathcal{SC}$. Pour finir, si aucun graphe n'est trouvé, alors l'algorithme retourne OUI car $H \simeq \hat{H}$ et donc $H \in \mathcal{SC}$.

Algorithme 9 : Un algorithme pour tester la simple connexité

```

1 Paramètre d'entrée : Un graphe  $G$ 
2 Simuler  $\mathcal{A}$  pendant  $\mathfrak{t}(|V(G)|)$  étapes à partir d'un sommet base  $v_0$ ;
3 si  $\mathcal{A}$  s'arrête en au plus  $\mathfrak{t}(|V(G)|)$  étapes alors
4   si il existe un graphe  $H$  tel que  $|V(G)| < |V(H)| \leq \mathfrak{t}(|V(G)|)$  et tel que le complexe de
      cliques  $\mathcal{K}(H)$  est un revêtement du complexe de cliques  $\mathcal{K}(G)$  alors
5     décider NON;                                     /*  $\mathcal{K}(G)$  n'est pas simplement connexe */
6   sinon
7     décider OUI;                                       /*  $\mathcal{K}(G)$  est simplement connexe */
8 sinon décider NON;                                     /*  $\mathcal{K}(G)$  n'est pas simplement connexe */;
```

8.5.2 Correction

Notons que si l'algorithme retourne NON à la ligne 8, alors par hypothèse, $\mathcal{K}(G) \notin \mathcal{SC}$ (la fonction $\mathfrak{t}$ borne la complexité de l'exploration pour tout graphe appartenant à $\mathcal{SC}$). Si l'algorithme retourne NON à la ligne 5, alors il existe un graphe H tel que $|V(H)| > |V(G)|$. Par définition, comme il existe un revêtement $\mathcal{K}(H)$ non trivial de $\mathcal{K}(G)$, $\mathcal{K}(G)$ n'est pas simplement connexe.

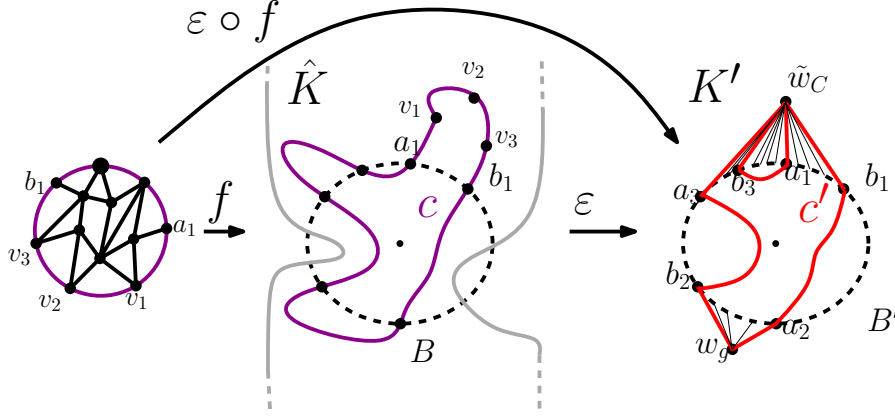


FIGURE 8.19 – Illustration de la preuve du théorème 8.2

Pour montrer que l'algorithme 9 est correct, i.e., qu'il décide de la simple connexité, il est suffisant de montrer que quand l'algorithme retourne OUI pour un graphe G , alors le complexe de clique $\mathcal{K}(G)$ est simplement connexe.

Théorème 8.2. *Soit $\mathcal{A}$ un algorithme qui explore tout graphe fini $G \in \mathcal{SC}$. Pour toute fonction calculable $\mathfrak{t} : \mathbb{N} \rightarrow \mathbb{N}$, il existe un graphe $G \in \mathcal{SC}$ tel que toute exécution de $\mathcal{A}$ sur G mettra strictement plus de $\mathfrak{t}(|V(G)|)$ étapes avant de s'arrêter.*

Pour prouver ce théorème, nous procédons par l'absurde en montrant que la fonction $\mathfrak{t}$ nous permet de construire un quasi-revêtement H de G simplement connexe et de rayon suffisamment grand (supérieure à $\mathfrak{t}(|V(G)|)$) pour que $\mathcal{A}$ nécessite strictement plus de $\mathfrak{t}(|V(G)|)$ étapes pour visiter tous les sommets de H . Ainsi, nous obtiendrons une contradiction car H est simplement connexe et $\mathcal{A}$ explore $\mathcal{SC}$ par définition. Pour conclure, nous utiliserons le lemme 8.30 appliqué au quasi-revêtement entre H et G introduit ci-dessus. Ce quasi-revêtement ayant un rayon plus grand que $\mathfrak{t}(|V(G)|)$, $\mathcal{A}$ ne pourra faire de différence entre G et H et s'arrêtera sans avoir exploré tous les sommets de H .

Démonstration. Soit $G \notin \mathcal{SC}$ et $\hat{G}$ le 1-squelette du revêtement universel du complexe de cliques $\mathcal{K}(G)$. Prenons un revêtement φ de $\mathcal{K}(\hat{G})$ vers $\mathcal{K}(G)$ et soit $\hat{v}_0 \in V(\hat{G})$ un sommet tel que $\varphi(\hat{v}_0) = v_0$. Par le lemme 8.30, en exécutant $\mathcal{A}$ sur $\hat{G}$ avec un sommet base $\hat{v}_0$, $\mathcal{A}$ s'arrête après au plus $\mathfrak{t}(|V(G)|)$ étapes. Si $\hat{G}$ est fini alors $G \in \mathcal{FC}$ et $\hat{G} \in \mathcal{SC}$. À partir de nos hypothèses sur $\mathcal{A}$, en exécutant $\mathcal{A}$ sur $\hat{G}$, $\mathcal{A}$ doit explorer tous les sommets de $\hat{G}$ avant de s'arrêter. Donc, $\mathcal{K}(\hat{G})$ est un revêtement de $\mathcal{K}(G)$ avec au plus $\mathfrak{t}(|V(G)|)$ sommets. Comme $\mathcal{K}(G)$ n'est pas simplement connexe, nous obtenons que $|V(G)| < |V(\hat{G})|$. Dans ce cas, l'algorithme retourne NON et nous obtenons le résultat.

Supposons maintenant que $\hat{G}$ est infini et donc que $G \in \mathcal{IC}$. Soit $r = \mathfrak{t}(|V(G)|)$ et $B = B_{\hat{G}}(\hat{v}_0, r)$. En exécutant $\mathcal{A}$ sur $\hat{G}$ à partir d'un sommet base $\hat{v}_0$, tous les noeuds visités par $\mathcal{A}$ appartiennent à B . Étant donnés deux sommets $\hat{u}$ et $\hat{v}$ de $V(\hat{G})$, nous noterons $\hat{u} \equiv_B \hat{v}$ s'il existe un chemin partant de $\hat{u}$ et allant à $\hat{v}$ dans $\hat{G} \setminus B$. Il est évident que $\equiv_B$ est une relation d'équivalence et que tout sommet de B est seul dans sa classe d'équivalence. Soit $\hat{u} \in V(\hat{G})$ un sommet, nous noterons la classe d'équivalence de $\hat{u}$ par $[\hat{u}]$. Soit H un graphe défini comme suit :

- $V(H) = \{[\hat{u}] \mid \hat{u} \in V(\hat{G})\}$
- $E(H) = \{[\hat{u}][\hat{v}] \mid \exists \hat{u}' \in [\hat{u}], \hat{v}' \in [\hat{v}], \hat{u}'\hat{v}' \in E(\hat{G})\}$

Le graphe H ainsi construit est illustré Figure 8.19. Nous montrons maintenant que le complexe de cliques $\mathcal{K}(H)$ est simplement connexe. Soit $\varphi : V(\hat{G}) \rightarrow V(H)$ une fonction définie par $\varphi(\hat{u}) = [\hat{u}]$. Par construction de H , pour toute arête $\hat{u}\hat{v} \in E(\hat{G})$, soit $[\hat{u}] = [\hat{v}]$, soit $[\hat{u}][\hat{v}] \in E(H)$. Donc φ est un morphisme simplicial. Prenons une boucle $c_0 = (u_1, u_2, \dots, u_p)$ de H . Par construction de H , il existe une boucle $\hat{c}_0 = (\hat{u}_{1,1}, \dots, \hat{u}_{1,\ell_1}, \hat{u}_{2,1}, \dots, \hat{u}_{2,\ell_2}, \dots, \hat{u}_{p,1}, \dots, \hat{u}_{p,\ell_p})$ dans G telle que pour tout $1 \leq i \leq p$ et tout $1 \leq j \leq \ell_i$, $\varphi(\hat{u}_{i,j}) = u_i$. Remarquons que $\varphi(\hat{c}_0) = (\varphi(\hat{u}_{1,1}) = u_1, \dots, \varphi(\hat{u}_{1,\ell_1}) = u_1, \varphi(\hat{u}_{2,1}) = u_2, \dots, \varphi(\hat{u}_{2,\ell_2}) = u_2, \dots, \varphi(\hat{u}_{p,1}) = u_p, \dots, \varphi(\hat{u}_{p,\ell_p}) = u_p)$ est homotope à c_0 .

Comme $\mathcal{K}(\hat{G})$ est simplement connexe, $\hat{c}_0$ est contractible et donc, il existe une séquence $\hat{c}_0, \hat{c}_1, \dots, \hat{c}_p$ telle que $|\hat{c}_p| = 1$ et il y a une homotopie élémentaire entre $\hat{c}_{i-1}$ et $\hat{c}_i$ pour tout $1 \leq i \leq p$. Comme φ est un morphisme simplicial, pour tout $1 \leq i \leq p$, il y a une homotopie élémentaire entre $\varphi(\hat{c}_{i-1})$ et $\varphi(\hat{c}_i)$. Ainsi, $\varphi(\hat{c}_0)$ est contractible, impliquant que c_0 est une boucle contractible de H . Donc, H est simplement connexe car toutes ses boucles sont contractibles.

Comme G est fini, le degré de tout sommet de $\hat{G}$ est borné par $|V(G)|$ est donc, le nombre de classes d'équivalences pour la relation $\equiv_B$ est fini. Impliquant que H est un graphe fini et donc, que H appartient à la classe $\mathcal{SC}$. De plus, comme pour tout sommet $\hat{u} \in B$, $[\hat{u}] = \{\hat{u}\}$, la boule $B_H([\hat{v}_0], r)$ de rayon r est isomorphe à B . Finalement, quand $\mathcal{A}$ est exécuté sur H en partant du sommet base $[\hat{v}_0]$, par le lemme 8.30, $\mathcal{A}$ s'arrête après au plus r étapes sans avoir visité tous les sommets de H , contredisant notre hypothèse sur $\mathcal{A}$. $\square$

Remarque 8.42. L'algorithme d'exploration 8 calcule le revêtement universel du graphe exploré. Ainsi, en exécutant notre algorithme sur une famille de graphes simplement connexes, la carte obtenue en fin d'exécution sera isomorphe au graphe exploré. Par contre, pour tout graphe appartenant à la famille de graphes $\mathcal{FC}$, l'agent n'est pas en mesure de décider si la carte reconstruite est isomorphe au graphe exploré. Contrairement au modèle sans jumelles, les deux problèmes ne sont pas équivalents dans ce cas. Nous présentons, sur la figure 8.20, un exemple de graphe appartenant à $\mathcal{FC} \setminus \mathcal{SC}$. Dans un tel graphe G , l'agent ne pourra pas distinguer s'il évolue dans G (à gauche) ou dans le revêtement universel $\hat{G}$ (à droite). Ce graphe a été présenté plus tôt sur la figure 8.15.

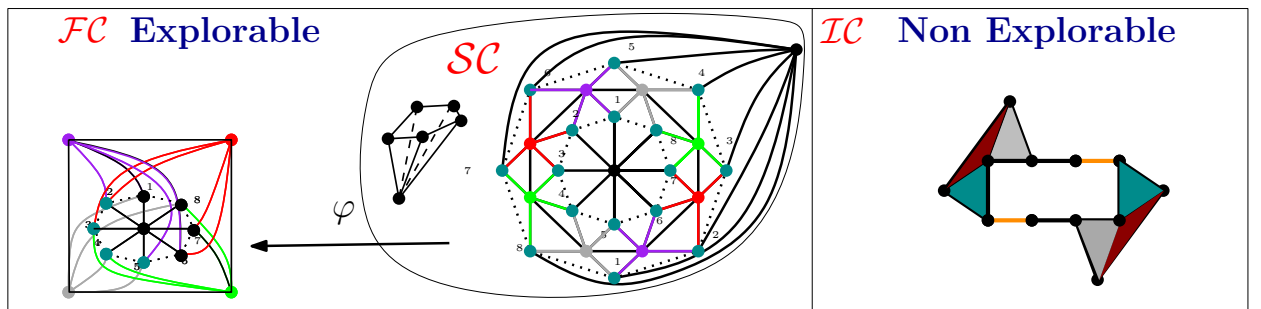


FIGURE 8.20 – Classification des graphes explorables avec jumelles. La famille $\mathcal{FC}$ maximum pour l'exploration est composée des graphes possédant un revêtement universel de leur complexe de cliques fini. La famille $\mathcal{SC}$ incluse dans $\mathcal{FC}$ contient les graphes possédant un complexe de cliques simplement connexe. La famille $\mathcal{IC}$ contient les graphes possédant un revêtement universel infini de leur complexe de cliques. Cette famille n'est pas explorable. Le graphe représenté dans la famille $\mathcal{SC}$ possédant 26 sommets est le revêtement universel (représenté par une flèche) du graphe possédant 13 sommets dans la famille $\mathcal{FC}$. De par nos résultats, un agent ne pourra pas distinguer dans lequel de ces deux graphes il évolue.

Conclusion

Dans ce chapitre, nous avons introduit un nouveau modèle d'agent mobile appelé agent mobile avec jumelles. Les jumelles modélisent un capteur permettant à l'agent, positionné sur un noeud du réseau, de "voir" le graphe induit par les sommets à distance 1 de sa position dans le réseau. Nous montrons que les jumelles permettent d'augmenter la calculabilité de l'exploration dans les réseaux anonymes, et ce, sans information globale. Pour cela, nous réutilisons la méthodologie introduite dans le chapitre précédent pour montrer qu'il existe une famille de graphes, notée $\mathcal{FC}$, qui est maximale pour l'exploration dans ce modèle. La famille de graphes $\mathcal{FC}$ contient tous les graphes possédant un revêtement universel fini de leur complexe de cliques. La famille de graphes $\mathcal{FC}$ généralise la famille de graphes maximale pour l'exploration sans jumelles : les arbres. Ainsi, il n'existe pas d'algorithme d'exploration pour tout graphe n'appartenant pas à la famille $\mathcal{FC}$. Pour obtenir ces résultats, nous avons utilisé des outils de topologie discrète pour caractériser exactement les impossibilités dans ce modèle. Ce modèle est d'autant plus intéressant qu'il enrichit la calculabilité du modèle en différenciant deux problèmes fondamentaux que sont la construction de carte et l'exploration (Remarque 8.42 et Figure 8.15).

Même si les jumelles sont une amélioration naturelle pour les agents mobiles, il apparaît pourtant que, malgré le gain de calculabilité obtenu, le prix à payer concernant la complexité des algorithmes universels pour cette famille est coûteux : le nombre de mouvements, vue comme une fonction en la taille du graphe, croît plus vite que toute fonction calculable.

Pour continuer l'étude de ce modèle, nous souhaiterions élargir le champ de vision des jumelles en augmentant la portée leur à une distance k donnée : un agent doté de jumelles augmentées lui permettant de voir le graphe induit par les sommets à distance k autour de sa position. Nous conjecturons que dans ce modèle, la famille maximale pour l'exploration sera l'ensemble des graphes possédant un revêtement universel fini et simplement connexe de leur complexe k -cellulaire, i.e., le complexe qui, pour tout cycle simple c tel que $|c| \leq 2k$, fait correspondre une cellule (ou face) de même dimension dans le complexe cellulaire. De plus, nous conjecturons que la complexité de l'exploration pour cette famille ne pourra pas être bornée par une fonction calculable pour les mêmes raisons que pour les jumelles à distance 1 : une telle fonction nous permettrait de décider de la simple connexité d'un complexe cellulaire arbitraire.

Dans le chapitre suivant, nous montrons que les jumelles permettent quand même d'explorer de larges familles de graphes efficacement, i.e., en un nombre de mouvements linéaire.

Chapitre 9

Exploration efficace avec jumelles

Dans ce chapitre, nous continuons l'étude des agents mobiles avec jumelles. Ces travaux ont été présentés dans une version courte lors de la conférence francophone ALGOTEL en 2016. [CGN16b] et a reçu le prix de la meilleure communication étudiante. Un article est en cours de rédaction. Une première version est accessible sur Arxiv [CGN16a].

Introduction

Dans ce chapitre, nous nous intéressons à l'exploration efficace de graphes anonymes par un agent mobile doté de jumelles. Nous avons montré dans les chapitres 7 et 8 qu'il existe une famille de graphes maximale pour l'exploration avec jumelles et nous avons proposé un algorithme d'exploration universel pour cette famille. Contrairement à l'exploration sans jumelles, nous avons montré que le nombre de mouvements d'un tel algorithme, vue comme une fonction en la taille du graphe, est très important et croît plus vite que toute fonction calculable. Dans ce chapitre, nous nous intéressons à l'exploration efficace et nous montrons qu'il est possible, avec jumelles et sans connaissance globale, d'explorer efficacement (i.e. en un nombre linéaire de mouvements) de larges familles de graphes avec arrêt. Pour cela, nous proposons un algorithme d'exploration efficace pour une famille de graphes particulière, appelée graphes de *Weetman*. Cette famille de graphes a été introduite dans [Wee94] et contient des familles de graphes connues telles la famille des graphes triangulés ou encore les graphes de Johnson. Les résultats obtenus dans ce chapitre améliorent les résultats obtenus jusqu'alors concernant l'exploration sans connaissance. Des exemples de graphes de Weetman sont présentés dans la figure 9.1. Pour explorer les graphes de Weetman, nous utiliserons les jumelles pour construire une carte du graphe exploré. La particularité ici est que grâce aux jumelles (et aux propriétés métriques des graphes de Weetman), nous sommes en mesure de distinguer les sommets visités et non visités du graphe. Distinguer les noeuds visités et non visités du réseau est une des problématiques majeures de l'exploration à résoudre si l'on veut que l'agent arrête son exécution. Pour résoudre ce problème, plusieurs solutions ont été proposées dans la littérature comme par exemple, utiliser des informations globales telle une borne sur la taille du réseau [CDK10a] ou utiliser un ou plusieurs jetons à déposer sur les noeuds du réseau [BS94, BA06]. Une autre solution consiste à développer des étiquetages spécifiques pour les sommets du graphe [DP99, AH00]. Cette étiquette peut contenir une identité ou toute autre information pouvant servir à distinguer les noeuds entre eux et permettant d'encoder de façon unique les chemins du réseau.

Dans ce chapitre, nous considérerons uniquement des graphes non dirigés et anonymes, i.e., où les sommets ne possèdent aucun étiquetage pouvant être lu par l'agent et seuls les étiquetages

de port locaux pourront être utilisés pour différencier les noeuds du réseau entre eux.

Dans la section 9.1, nous rappelons le problème de l'exploration de graphes, nous introduirons formellement les graphes de Weetman puis nous présenterons les notations utilisées par la suite. Dans la section 9.2, nous présenterons l'algorithme d'exploration pour les graphes de Weetman. Dans la section 9.3, nous prouverons la correction de l'algorithme d'exploration. Pour finir, dans la section 9.4, nous montrerons que notre algorithme explore bien tous les graphes de Weetman en un nombre de mouvements linéaire en la taille du graphe exploré.

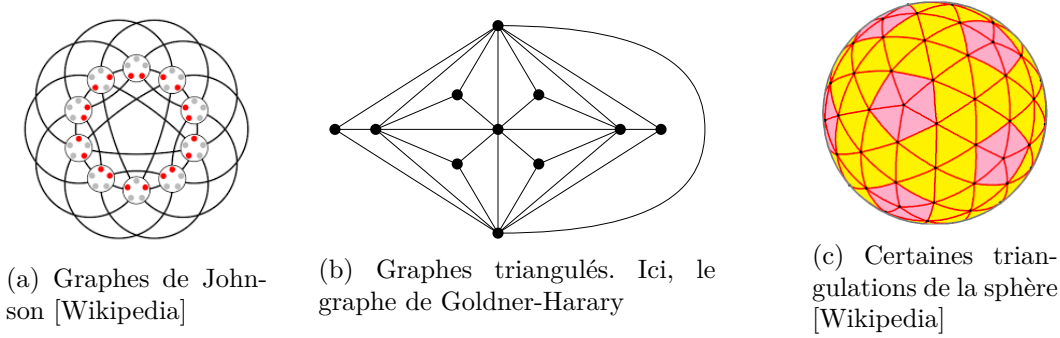


FIGURE 9.1 – Illustration des graphes de Weetman

9.1 Préliminaires

9.1.1 Exploration à base d'agent mobile avec jumelles

Nous considérons le modèle à agent mobile doté de jumelles introduit dans le chapitre 8. Nous rappelons la définition du problème de l'exploration.

Rappel (Définition 7.37). Un algorithme $\mathcal{A}$ est un algorithme d'exploration pour une famille de graphes $\mathcal{F} \subseteq \mathcal{G}^\delta$, si pour tout graphe $G = (V, E) \in \mathcal{F}$ et pour tout sommet base $v_0 \in V$,

- si $G \in \mathcal{F}$ alors l'agent explore tous les sommets de G et s'arrête,
- si $G \notin \mathcal{F}$ et que l'agent s'arrête alors tous les sommets de G sont explorés.

9.1.2 Définitions et Notations

Nous introduisons maintenant quelques définitions et notations propres au chapitre. En premier nous définissons quelques familles de graphes que nous explorerons par la suite. Ces familles sont illustrées sur la figure 9.1.

Définition 9.1 (Graphe de Johnson $J(n, k)$ [BCN89]). Le graphe de Johnson $J(n, k)$ possède un sommet par sous-ensemble de k éléments de l'ensemble $\{1, \dots, n\}$ et possède une arête entre deux sommets si l'intersection des deux sous-ensembles correspondants est de taille $k - 1$.

Définition 9.2 (Chemin isométrique). Un chemin p reliant deux sommets u et v dans un graphe G est isométrique si p est un plus court chemin reliant u à v dans G .

Définition 9.3 (Pont et corde dans un graphe). Étant donné un cycle C dans un graphe G , un pont du cycle C dans G est un plus court chemin reliant deux sommets non consécutifs de C qui est plus court que la distance séparant les sommets dans C . Un pont dans un cycle C sera appelé corde s'il est constitué d'une unique arête.

Définition 9.4 (Graphes triangulés). Un graphe est triangulé si tout cycle de taille au moins 4 possède une corde.

Définition 9.5 (Graphes bridged). Un graphe appartient à la famille des graphes bridged si pour tout cycle de taille au moins 4 possède un pont. Une définition alternative est que tous les cycles isométriques sont de longueur 3.

Toute notation non définie ici a été définie dans un chapitre précédent.

Définition 9.6 (Sphère). Étant donné un graphe $G = (V, E)$, un sommet $v \in V(G)$ et un entier i , la sphère de rayon i autour du sommet v , notée $S^i(v)$, est le sous graphe de G induit par les sommets à distance i de v , i.e., $S^i(v) \simeq G[\{w \in V(G) \mid d(v, w) = i\}]$.

Définition 9.7 (Cluster). Étant donné un graphe G , un sommet $v \in V(G)$, un entier i , un cluster $C^i(v)$ du graphe G est une composante connexe de la sphère $S^i(v)$.

Notons que les sommets d'un même cluster sont inclus dans une même sphère $S^k(v_0)$ et sont donc à une même distance de v_0 .

Définition 9.8. Étant donné un graphe G , un sommet base $v_0 \in V(G)$ et un cluster C de G , nous noterons $d_G(v_0, C)$ la distance séparant les sommets du cluster C et v_0 .

Par la suite, nous manipulerons uniquement des graphes dotés d'un sommet base $v_0 \in V(G)$ et nous considérerons des sphères et de clusters centrés sur le sommet base v_0 de l'agent. Comme le sommet base est invariant lors d'une exécution, une sphère $S^i(v_0)$ sera notée S^i .

Nous introduisons maintenant une partition des arêtes d'un graphe en fonction des clusters du graphe. Cette partition est illustrée dans la figure 9.2.

Définition 9.9. Soit G un graphe et $v \in V(G)$ un sommet, une arête est **verticale** si les deux sommets qu'elle relie appartiennent à des clusters de G distincts. Une arête est **horizontale** si les deux sommets qu'elle relie appartiennent à un même cluster de G .

9.1.3 Graphes de Weetman

Les graphes de Weetman se définissent par deux conditions métriques "simples".

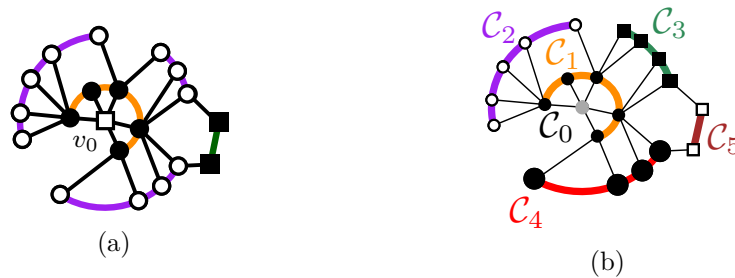


FIGURE 9.2 – Illustration des sphères (a) et des clusters (b). En (a), les quatre sphères sont représentées par des formes de sommets différents (carré blanc, disque noir, disque blanc et carré noir). En (b), six clusters se distinguent. Le cluster C_0 contient le sommet base de l'agent (sommet gris). Les disques noirs (resp. disques blancs, carrés noirs, gros disques noirs et carrés blancs) représentent le cluster C_1 (resp. C_2 , C_3 , C_4 , C_5). Les arêtes verticales sont représentées par des traits fins et les arêtes horizontales par des traits larges.

Définition 9.10 (Graphes de Weetman [Wee94]). Un graphe G est un graphe de Weetman s'il satisfait les conditions suivantes pour tout sommet $v_0 \in V(G)$:

Condition du triangle $\text{TRI}(v_0)$. Pour toute arête $vw \in E(S_G^k(v_0))$, il existe un sommet $u \in V(S_G^{k-1}(v_0))$ tel que uvw est un triangle.

Condition de l'intervalle $\text{INT}(v_0)$. Pour tout sommet $v \in V(S_G^k(v_0))$, l'ensemble de sommets $\{u \in S_G^{k-1}(v_0) \mid uv \in E(G)\}$ induit un sous-graphe *connexe* de G .

Nous présentons maintenant une propriété fondamentale vérifiée par tout graphe de cette famille : le graphe d'adjacence des clusters d'un graphe de Weetman est un arbre. Plus généralement, cette propriété est satisfaite par tout graphe possédant un complexe de cliques simplement connexe (au sens de la définition 8.23). Néanmoins, comme la preuve d'une telle proposition nécessite des outils topologiques complexes tels les diagrammes de disques, non requis dans la suite de ce chapitre, nous prouveront cette propriété uniquement pour les graphes de Weetman.

Lemme 9.11. *Le graphe d'adjacence des clusters d'un graphe de Weetman est un arbre.*

Démonstration. Étant donné un graphe de Weetman G et un sommet base $v_0 \in V(G)$, supposons par l'absurde qu'il existe une séquence de clusters $\mathcal{C} = C^1 C^2 \dots C^\ell C^1$ disjoints et adjacents, i.e., pour tout $1 \leq i \leq \ell$, il existe $uv \in E(G)$ avec $u \in V(C^i)$ et $v \in V(C^{(i+1)\% \ell})$ et pour tout $1 \leq i, j \leq \ell$, $C^i \neq C^j$ (voir figure 9.4).

Pour une telle séquence $\mathcal{C}$ de clusters, notons $d(\mathcal{C}) = \max\{d_G(v_0, C^i) \mid C^i \in \mathcal{C}\}$ la distance maximale entre v_0 et les clusters de la séquence. Pour tout cycle c dans G , notons $n_{\max}(c) = |\{v \in V(c) \mid \forall v' \in V(c), d_G(v_0, v) \geq d_G(v_0, v')\}|$, le nombre de sommets du cycle maximisant la distance par rapport à v_0 .

Parmi tous les cycles de clusters dans G , prenons une séquence $\mathcal{C}$ minimisant $d(\mathcal{C})$. Parmi tous les cycles de G traversant la séquence de $\mathcal{C}$, prenons un cycle c minimisant $n_{\max}(c)$.

Sans perte de généralité, l'ensemble de sommets $\{v \in V(c) \mid d_G(v_0, v) = d(\mathcal{C})\}$ appartient à un unique cluster $C^i \in \mathcal{C}$ (et $d(v_0, C^i) = d(\mathcal{C})$). De plus, les deux clusters C^{i-1} et C^{i+1} contiennent uniquement des sommets à distance $d(\mathcal{C}) - 1$ de v_0 .

Nous prouvons qu'il existe soit un cycle de clusters $\mathcal{C}'$ telle que $d(\mathcal{C}') < d(\mathcal{C})$, soit un cycle c' tel que $n_{\max}(c') = n_{\max}(c) - 1$ pour le cycle de clusters $\mathcal{C}$.

Si $n_{\max}(c) = 1$, i.e., $|c \cap C^i| = 1$, alors il existe $uv, vw \in E(c)$ tels que $u \in V(C^{i-1})$, $v \in V(C^i)$ et $w \in V(C^{i+1})$. Comme $d(v_0, u) = d(v_0, w) = d(c) - 1$ et que $C^{i-1} \neq C^{i+1}$, il n'existe aucune arête $uw \in E(G)$. À partir de la condition $\text{INT}(v_0)$ appliquée aux sommets u, v et w , il existe un chemin $p = (s_1, \dots, s_n) \subset S^{k-1}$ tel que, pour tout $1 \leq \ell < n$, $s_1 = u, s_n = w$ et $s_\ell s_{\ell+1} v$ forme un triangle dans G (illustré dans la figure 9.4). Comme $u, w \in V(S^{k-1})$ et que le chemin $p \subset S^{k-1}$ relie u à w , nous obtenons par définition que $C^{i-1} = C^{i+1}$. Par conséquent, il existe un cycle de clusters $\mathcal{C}' = \mathcal{C} \setminus C^i$ tel que $d(\mathcal{C}') < d(\mathcal{C})$.

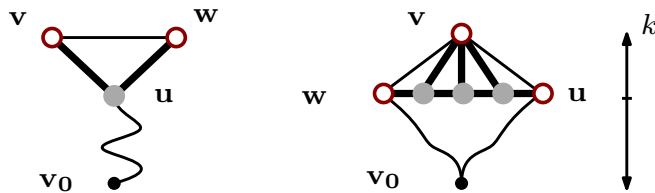


FIGURE 9.3 – Illustration de la propriété $\text{TRI}(v_0)$ à gauche et $\text{INT}(v_0)$ à droite.

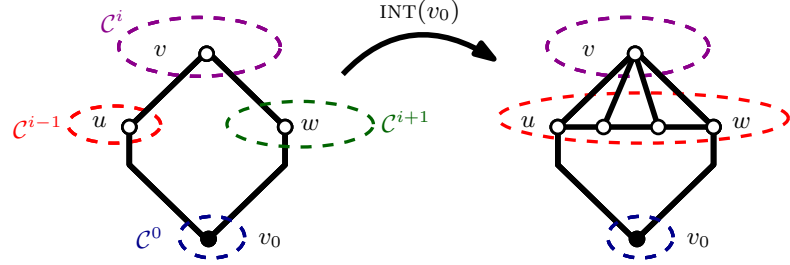


FIGURE 9.4 – Illustration du cas 1 de la preuve du lemme 9.11

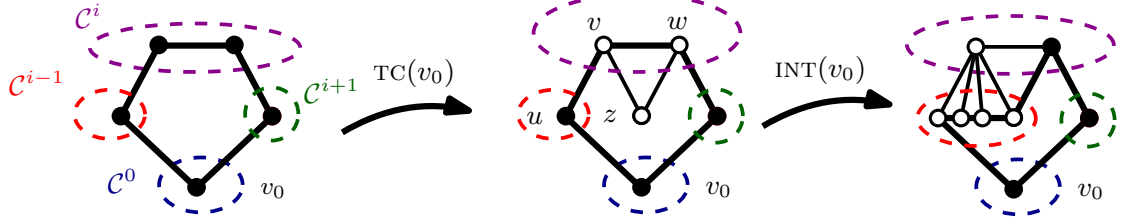


FIGURE 9.5 – Illustration du cas 2 de la preuve du lemme 9.11

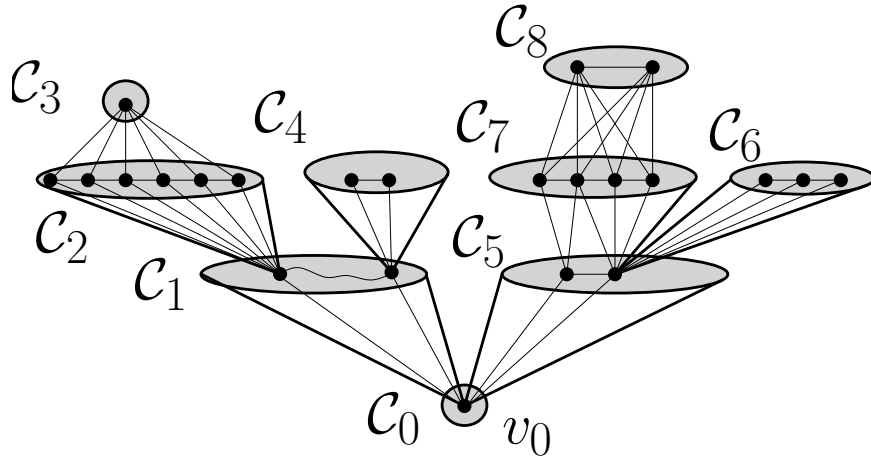


FIGURE 9.6 – Illustration du graphe d'adjacence des clusters dans un graphe de Weetman. Les clusters sont numérotés en fonction d'un parcours en profondeur donné pour cet arbre

Si $|c \cap C^i| > 1$, alors prenons $uvw \in V(c)$ tels que $u \in V(C^{i-1})$, $v \in V(C^i)$ et $w \in V(C^i)$. Observons que $d(v_0, v) = d(v_0, w) = d(v_0, u) + 1$.

À partir de la condition $\text{TRI}(v_0)$ appliquée à l'arête $vw \in E(c)$, il existe un sommet $z \in V(S^{k-1})$ tel que $zvw \in E(G)$ est un triangle. En appliquant la condition $\text{INT}(v_0)$ sur les trois sommets uvw , il existe un chemin $q = (s_0, \dots, s_m) \subset S^{k-1}$ tel que, pour tout $0 \leq i \leq m$ $s_0 = u$, $s_m = z$ et $vs_i \in E(G)$. Comme, par définition, z appartient à C^{i-1} , il existe un cycle $c' = c \setminus \{uv\} \cup \{q\}$ traversant la séquence de clusters $\mathcal{C}$. Comme $n_{\max}(c') < n_{\max}(c)$, nous obtenons une contradiction sur le choix de c . $\square$

Notons que dans certains cas, l'arbre formé par les clusters d'un graphe de Weetman est un chemin comme par exemple pour les graphes de Johnson (définition 9.1).

Remarque 9.12. Par le même raisonnement que pour la preuve ci-dessus, i.e., utiliser les conditions INT et TRI pour "remplir" les cycles avec des triangles, nous pouvons prouver que les graphes de Weetman sont simplement connexes et donc (en accord avec les résultats présentés dans le chapitre 8) appartiennent à la famille maximale des graphes explorables avec jumelles, notée $\mathcal{FC}$.

La topologie du graphe d'adjacence des clusters en "arbre" nous permettra d'explorer efficacement tout graphe de Weetman.

Corollaire 9.13. *Pour tout graphe de Weetman G , tout sommet base $v_0 \in V(G)$ et tout cluster C à distance k de v_0 , tous les chemins simples menant du sommet base v_0 aux sommets du cluster C traversent un seul et unique cluster C' à distance $k - 1$ de v_0 .*

9.2 Algorithme d'exploration efficace pour les graphes de Weetman

Le but de l'algorithme d'exploration présenté dans ce chapitre est d'explorer tous les clusters du graphe en effectuant un parcours en profondeur de l'arbre des clusters. Pour cela, l'agent construit une carte du réseau tout au long de l'exécution. La particularité de ce type de graphe est qu'un agent peut faire une distinction entre les sommets visités et les sommets non visités de sa carte. Il sera donc capable de calculer les clusters de sa carte qui n'ont pas encore été visités et de les ordonner pour les visiter de façon efficace.

L'algorithme 10 s'exécute par phase. Au début d'une phase, l'agent détermine dans sa carte, notée $\mathcal{M}$, le cluster suivant à explorer (contenant des sommets qui n'ont pas encore été visités) suivant un parcours en profondeur (Ligne 4). Puis, il calcule un plus court chemin qui, depuis sa position actuelle, mène au cluster et visite tous ses sommets (Ligne 5). L'agent poursuit sa phase en suivant ce chemin et en visitant ainsi (pour la première fois) tous les sommets du cluster (Ligne 6). Pour chaque sommet du cluster visité, l'agent accède à l'étiquetage de jumelles du sommet exploré de $V(G)$ grâce à la primitive `getBino` (Ligne 8). Cet étiquetage de jumelles est stocké dans une structure adaptée, notée $\mathcal{B}$ (décrite dans la suite de ce chapitre). Une fois tous les sommets du cluster visités, l'agent met à jour sa carte avec les informations recueillies précédemment (Lignes 16 à 24), puis vérifie l'intégrité de sa carte (Ligne 25). Si une erreur est détectée dans la carte durant cette vérification alors l'agent décide de continuer son exploration pour toujours en rentrant dans une *boucle infinie*. On peut supposer que l'agent fera des allers/retours sur une arête par exemple. Si aucune erreur n'est détectée et que tous les clusters présents dans la carte ont été visités, alors, l'agent arrête son exécution. Sinon, l'agent entame une nouvelle phase d'exploration.

Les erreurs détectées par l'agent peuvent être de différentes natures : détecter une incohérence entre la carte construite et les étiquetages de jumelles rencontrées ou alors une double arête ou une boucle simple présente dans la carte. Ce mécanisme de détection d'erreur permet de respecter la terminaison de l'exploration. Nous en discuterons plus en détail dans la section 9.4.1.

9.2.1 La carte $\mathcal{M}$

Tout au long d'une exécution, l'agent construit une carte du réseau, notée $\mathcal{M}$. Tout sommet de la carte sera représenté par un entier unique, noté $\mathbf{n}$ et une arête sera représentée par un couple d'entiers $\mathbf{nm}$. La carte $\mathcal{M}$ est dotée d'un étiquetage des ports induit par l'étiquetage des ports δ du graphe G . Pour distinguer les sommets non visités de ceux déjà visités de sa carte, l'agent maintient à jour une liste nommée **AVisiter** contenant les identités des sommets non visités (Lignes 1, 7 et 27). Étant donnée une étape i d'une exécution de l'algorithme sur un graphe G , nous noterons $\mathcal{M}^i$ et **AVisiter** ^{i} , pour désigner leur contenu en fin d'étape i . Initialement, $V(\mathcal{M}^0)$ et **AVisiter**⁰ contiennent uniquement un sommet numéroté 0, noté $\mathbf{n}_0$, représentant le sommet base de l'agent (et aussi le premier cluster à explorer). Comme un agent ne retire ni ne modifie d'élément dans sa carte (Lignes 16 à 24), pour toute phase i , $\mathcal{M}^i \subseteq \mathcal{M}^{i+1}$.

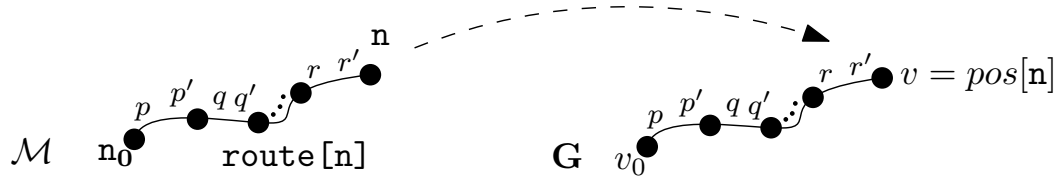


FIGURE 9.7 – Illustration de la fonction POS, qui pour un sommet $\mathbf{n} \in V(\mathcal{M}) \setminus \mathbf{AVisiter}^i$, fait correspondre le sommet $v \in V(G)$ qui, à partir de la base de l'agent v_0 , est accessible par la séquence de numéros de port **route**[$\mathbf{n}$].

Nous noterons $\mathcal{M}[\mathbf{n}][p_0 \dots p_k]$ pour distinguer un sommet de la carte en fonction d'un sommet initial $\mathbf{n}$ et d'une séquence de numéros de port $p_0 \dots p_k \in \mathbb{N}^k$,

Définition 9.14. Pour toute étape i d'une exécution de l'algorithme, pour toute séquence de numéros de port $p_0 \dots p_k \in \mathbb{N}^k$, s'il existe un chemin $\mathbf{p}$ reliant un sommet $\mathbf{n}$ à un sommet $\mathbf{m}$ dans $\mathcal{M}^i$ étiqueté $\lambda(\mathbf{p}) = p_0 \dots p_k$, alors $\mathcal{M}^i[\mathbf{n}][p_0 \dots p_k] = \mathcal{M}^i[\mathbf{n}][\lambda(\mathbf{p})] = \mathbf{m}$. Sinon, $\mathcal{M}^i[\mathbf{n}][p_0 \dots p_k] = \perp$.

De la même façon que pour la carte, nous introduisons une notation spécifique aux graphes.

Définition 9.15. Pour tout graphe G , pour tous sommets $v, w \in V(G)$ et pour toute séquence de numéros de port $p_0 \dots p_k$, s'il existe un chemin $\mathbf{p}$ reliant v à w étiqueté $\lambda(\mathbf{p}) = p_0 \dots p_k$ alors $v.\lambda(\mathbf{p}) = w$ et si un tel chemin n'existe pas dans G , alors $v.\lambda(\mathbf{p}) = \perp$.

Nous introduisons une notation nous permettant d'exprimer l'ensemble des numéros de port traversés par l'agent durant une exécution.

Définition 9.16. Pour toute exécution de l'algorithme sur un graphe G et pour tout entier i , nous noterons **route** ^{i} $\subseteq \mathbb{N}^{\mathbb{N}}$ la séquence de numéros de port suivie par l'agent à partir du début de l'exécution et jusqu'à la fin de l'étape i .

Définition 9.17. Pour tout sommet $\mathbf{n} \in \mathcal{M}$ visité par l'agent, nous noterons **route**[$\mathbf{n}$] le préfixe de **route** correspondant à la séquence de numéros de port suivie par l'agent entre le début de l'exécution et sa première visite de $\mathbf{n}$. Si ce sommet $\mathbf{n} \in \mathcal{M}$ n'a jamais été visité par l'agent, nous noterons **route**[$\mathbf{n}$] = $\perp$.

Notons que le sommet $\mathcal{M}[\mathbf{n}_0][\mathbf{route}]$ correspond à la position de l'agent dans sa carte et $\mathcal{M}[\mathbf{n}_0][\mathbf{route}[\mathbf{n}]]$ au sommet $\mathbf{n}$ dans $\mathcal{M}$. De plus, par définition, pour tout sommet de la carte $n \notin \mathbf{AVisiter}$, $\mathbf{route}[\mathbf{n}]$ est constant tout au long d'une exécution.

Définition 9.18. Pour toute exécution dans un graphe G , pour toute étape i et pour toute séquence de numéros de port $\mathbf{route} = p_0 \dots p_k$ traversant une séquence de sommets $v_0 \dots v_{k+1}$ (i.e., pour tout $p_i \in \mathbf{route}$, $p_i = \delta_{v_i}(v_{i+1})$), la séquence de numéros de port inverse à $\mathbf{route}$ sera notée $\mathbf{route}^{-1} = \{p'_k \dots p'_0 \mid p'_i = \delta_{v_{i+1}}(v_i), \forall 0 \leq i < k\}$

Remarque 9.19. Par hypothèse, l'agent connaît le numéro de port par lequel il accède à un noeud du réseau (capacité de "look-back") et peut maintenir la structure $\mathbf{route}^{-1}$ à jour tout au long d'une exécution.

Définition 9.20. En fin d'étape i d'une exécution de $\mathcal{A}$ dans un graphe G , la fonction $\text{POS}^i : V(\mathcal{M}^i \setminus \mathbf{AVisiter}^{i+1}) \rightarrow V(G)$ assigne, à tout sommet $\mathbf{n} \in V(\mathcal{M}^i \setminus \mathbf{AVisiter}^{i+1})$, le sommet $\text{POS}[\mathbf{n}] = v_0.\mathbf{route}[\mathbf{n}]$.

Si une étape i se termine sans boucle infinie, l'agent a calculé $\mathcal{M}^{i+1}$ en parcourant une séquence de numéros de port prise dans la carte $\mathcal{M}^i$. Notons qu'un agent, positionné sur un sommet $v \in V(G)$ correspondant à un noeud $\mathbf{n} \in V(\mathcal{M}^i)$ dans sa carte, voulant traverser une arête $\mathbf{nm} \in E(\mathcal{M})$ étiquetée (p, q) détectera si l'arête en question n'existe pas dans le graphe G , i.e., s'il n'existe pas d'arête $vw \in E(G)$ possédant un tel étiquetage. Ainsi, comme $\mathcal{M}^{i+1} \setminus \mathbf{AVisiter}^{i+1}$ contient uniquement des sommets déjà visités, la fonction $\text{POS}^i : V(\mathcal{M}) \rightarrow V(G)$ est définie sans ambiguïté pour tout sommet $\mathbf{n} \in V(\mathcal{M}^{i+1} \setminus \mathbf{AVisiter}^{i+1})$. Notons que l'agent ne dispose pas d'une telle fonction et seule sa carte lui permettra de s'orienter dans le réseau (voir algorithme).

Remarque 9.21. Comme pour tout $\mathbf{n} \in V(\mathcal{M}^i \setminus \mathbf{AVisiter}^i)$, $\text{POS}^i[\mathbf{n}] = \text{POS}^{i+1}[\mathbf{n}]$ (à partir des définitions de $\mathbf{route}^i$ et POS^i), la fonction POS^i sera simplement notée POS dans la suite de ce document.

9.2.2 Étiquetage de jumelles

Supposons un agent mobile explorant un sommet $u = \text{POS}[\mathbf{n}]$ dans un graphe G . Pour accéder à l'étiquetage de jumelles d'un sommet u , l'agent utilise la procédure `getBino()` qui lui retourne un encodage du graphe $B_G(u, 1)$ (Ligne 8). L'étiquetage de jumelles obtenu à partir du sommet $u = \text{POS}[\mathbf{n}]$ sera noté $\mathcal{B}[\mathbf{n}]$. Un exemple d'un tel étiquetage est présenté sur la figure 9.8.

Définition 9.22. L'étiquetage $\mathcal{B}[\mathbf{n}]$ retourné par la procédure `getBino()` est le suivant,

- toute arête $uv \in V(B_G(u, 1))$ est encodée par le couple $(p, p') = \lambda(uv)$,
- tout triangle $uvw \in B_G(u, 1)$ incluant u est encodé par le couple $(pqr, r'q'p')$ tel que $u.p = v$, $u.r' = w$ et $u.pqr = u.r'q'p' = u$.

Remarque 9.23. Notons que l'étiquetage de jumelles est invariant et peut être encodé dans la mémoire interne des noeuds du réseau. Pour respecter le modèle, nous supposons que cette étiquette ne pourra pas être modifiée par l'agent.

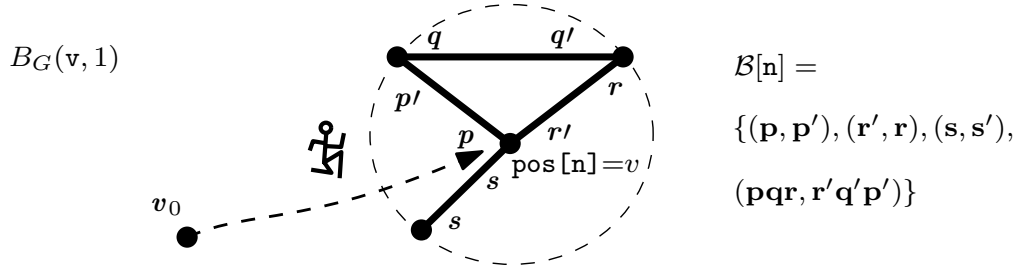


FIGURE 9.8 – Encodage de l'étiquette de jumelles d'un sommet du graphe. L'agent, positionné sur le sommet $v = \text{pos}[n] \in V(G)$, accède à l'étiquette $\nu(v)$ et la stocke dans la structure $\mathcal{B}[n]$.

9.2.3 Structures utilisées

La première structure utilisée, notée $\mathcal{B}$ (introduite précédemment) est un tableau contenant tous les étiquetages de jumelles obtenus durant la phase en cours. Nous noterons $\mathcal{B}[n]$ l'étiquetage de jumelles obtenu à partir du sommet $\text{pos}[n] \in V(G)$.

La structure **PRE-VERT** encode les nouvelles arêtes verticales, i.e., les arêtes reliant deux clusters différents de la carte (un cluster est donc le successeur du second dans l'arbre des clusters). Chaque élément de **PRE-VERT** est appelé un pré-sommet. Lorsqu'un agent explore un sommet $u = \text{pos}[n] \in V(G)$ lors d'une étape i , pour tout couple $(p, p') \in \mathcal{B}[n]$, si $\mathcal{M}^i[n][p]$ n'est pas un sommet de $V(\mathcal{M})$ alors le pré-sommet (n, p, p') est ajouté à la structure **PRE-VERT**. Notons que p' est stocké avec le pré-sommet (n, p) seulement pour simplifier la mise à jour de la carte $\mathcal{M}$ en fin de phase. De par notre stratégie d'exploration, tous les sommets nouvellement découverts n'appartiennent pas au cluster visité et sont donc liés par une arête (verticale) à, au moins, un sommet du cluster visité. Cette propriété nous assure que l'agent sera en mesure de calculer, grâce à l'ensemble des pré-sommets produits durant l'exploration d'un cluster, tous les sommets et arêtes "verticales" de G vus lors de cette phase.

Pour identifier le nombre exact de sommets nouvellement découverts, nous devons identifier quels sont les pré-sommets "équivalents", i.e., correspondant à un même sommet du graphe. Cette "équivalence" entre pré-sommets est formalisée par la relation élémentaire, notée $\equiv$, définie ci-dessous et illustrée dans la figure 9.9.

Définition 9.24 (Relation $\equiv$). Soit i une étape d'une exécution de $\mathcal{A}$ dans un graphe G . Deux pré-sommets (n, p, p') et $(m, q', q) \in \text{PRE-VERT}^i$ vérifient la relation $\equiv$ s'il existe une entrée $(pqr, r'q'p') \in \mathcal{B}[n]$ telle que $\mathcal{M}^i[n][r'] = m$ et $\mathcal{M}^i[n][p] = \perp$ ou son symétrique ($\mathcal{M}^i[n][p] = m$ et $\mathcal{M}^i[n][r'] = \perp$).

En d'autres termes, deux pré-sommets (n, p, p') et $(m, q', q) \in \text{PRE-VERT}$ satisfont la relation $\equiv$ s'il existe une arête $uv \in E(G)$ telle que $u = \text{pos}[n]$, $v = \text{pos}[m]$, $\lambda(uv) = (r', r)$ et $u.p = v.q' = w \in V(G)$ (voir Figure 9.9).

Pour calculer cette relation, nous stockons dans la structure $\mathcal{R}_{\equiv}$ tous les couples de pré-sommets vérifiant la relation $\equiv$. Une entrée de cette structure $\mathcal{R}_{\equiv}$ est un couple $((n, p, p'), (m, q, q'))$ contenant deux pré-sommets (inclus dans **PRE-VERT**) qui sont équivalents par la relation $\equiv$.

La clôture réflexive et transitive de la relation $\equiv$, notée par $\equiv^*$, est une relation d'équivalence. Cette relation regroupe les pré-sommets qui encodent des arêtes verticales pointant un même sommet du graphe.

La dernière structure utilisée dans l'algorithme, notée $\mathcal{HOR}$, est utilisée pour identifier les arêtes "horizontales" nouvellement découvertes, i.e., les arêtes qui relient deux sommets nou-

vement découverts n'appartenant pas à la carte $\mathcal{M}^i$. Ces arêtes sont appelées "horizontales" car elles relient deux sommets d'un même cluster. Comme il n'est pas possible d'identifier de telles arêtes avant la fin de l'exploration du cluster, la structure $\mathcal{HOR}$ contient seulement les pré-sommets encodant les extrémités de ces arêtes ainsi que leur étiquetage des ports respectif. Un agent visitant un sommet $\mathbf{n}$ de sa carte ajoutera une entrée de la forme $(\mathbf{n}, p, r', (q, q'))$ à la structure $\mathcal{HOR}$ pour toute entrée $(pqr, r'q'p') \in \mathcal{B}[\mathbf{n}]$ telle que $\mathbf{n}.p = \mathbf{n}.r' = \perp$ et $(\mathbf{n}, p, p'), (\mathbf{n}, r', r) \in \text{PRE-VERT}$.

9.2.4 Description d'une phase

Supposons qu'un agent soit situé sur un sommet $u \in V(G)$ et démarre une nouvelle phase i . Soit $\mathbf{n} \in V(\mathcal{M}^i)$ le sommet de la carte encodant la position actuelle de l'agent, i.e., $\text{POS}[\mathbf{n}] = u$.

Au début de la phase, l'agent calcule un plus court chemin (une séquence de numéros de port) dans sa carte qui, à partir de sa position actuelle, relie et visite un cluster $\mathcal{C}$ inexploré (suivant l'ordre du parcours en profondeur) "efficacement". Nous verrons plus tard comment visiter un cluster de façon efficace. Notons qu'en suivant ce chemin, l'agent explorera, dans G , l'ensemble de sommets $\text{POS}[V(\mathcal{C})]$.

L'agent continue sa phase en suivant le chemin c calculé pour visiter les différents sommets du cluster $\mathcal{C}$. Pour tout noeud $\mathbf{n} \in \mathcal{C}$, l'agent se positionne sur le sommet $\text{POS}[\mathbf{n}]$, et stocke l'étiquetage de jumelles du sommet $\text{POS}[\mathbf{n}]$ dans la structure dédiée $\mathcal{B}[\mathbf{n}]$ grâce à la primitive `getBino()`.

Une fois le cluster visité, l'agent met à jour ses structures de données en fonction des étiquetages de jumelles récupérés et stockés dans la structure $\mathcal{B}$: La structure **PRE-VERT** est mise à jour à la ligne 9, la structure $R_{\equiv}$ est mise à jour à la ligne 12 et la structure $\mathcal{HOR}$ est mise à jour à la ligne 14.

Une fois les structures mises à jour, l'agent met à jour sa carte (procédure détaillée dans le prochain paragraphe) et vérifie la cohérence de sa nouvelle carte, i.e., il vérifie que sa carte est localement bijective en comparant ce qu'il a vu du graphe G et sa carte : pour tout sommet $\mathbf{n} \in V(\mathcal{C})$, $\mathcal{B}[\mathbf{n}]$ doit être isomorphe à $B_{\mathcal{M}}(\mathbf{n}, 1)$ (Ligne 25). S'il détecte une erreur dans sa carte, l'agent entre dans une boucle infinie. Sinon, il commence une nouvelle phase en calculant le prochain cluster à visiter en suivant un ordre DFS. Vérifier la cohérence de la carte avec le

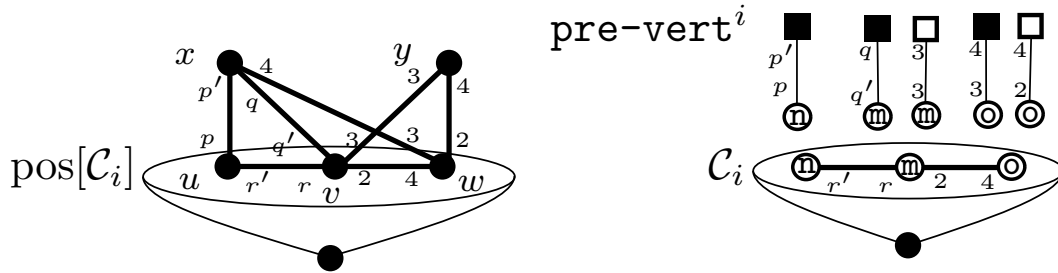


FIGURE 9.9 – Illustration de la relation $\equiv$ et de sa clôture transitive $\equiv^*$. À gauche, les sommets du graphe exploré durant cette phase (i.e. $\text{POS}[C_i]$). À droite, la carte en début de phase représentant le cluster C_i et les pré-sommets. Ici, les couples de pré-sommets $(\mathbf{n}, p, p')$ et $(\mathbf{m}, q', q)$; $(\mathbf{m}, q', q)$ et $(\mathbf{o}, 3, 4)$; $(\mathbf{m}, 3, 3)$ et $(\mathbf{o}, 4, 2)$ vérifient la relation $\equiv$. La relation d'équivalence $\equiv^*$ regroupe les pré-sommets en deux classes. La première contient $(\mathbf{n}, p, p')$, $(\mathbf{m}, q', q)$ et $(\mathbf{o}, 3, 4)$ (carré noir). La deuxième contient $(\mathbf{o}, 3, 4)$ et $(\mathbf{m}, 3, 3)$ (carré blanc).

graphe exploré permet de contrôler les exécutions de l'algorithme sur des graphes qui ne sont pas des graphes de Weetman. Différentes erreurs pouvant être rencontrées sont présentées et illustrées dans la sous-section 9.4.1 (Figures 9.17 et 9.16).

9.2.5 Construction et mise à jour de la carte $\mathcal{M}$

L'idée principale pour construire une carte cohérente avec le graphe exploré est de correctement faire correspondre les sommets nouvellement découverts avec les pré-sommets contenus dans la structure **PRE-VERT**.

Pour correctement faire correspondre les sommets nouvellement découverts avec les pré-sommets calculés lors de l'étape, nous calculons la clôture transitive $\equiv^*$ de la relation $\equiv$ avec les pré-sommets recueillis dans $\mathcal{R}_{\equiv}$, puis nous calculons le quotient $\text{PRE-VERT}/\equiv^*$ de la relation $\equiv^*$ avec l'ensemble de pré-sommets PRE-VERT^i produits lors de la phase i (Ligne 16). Par définition de $\equiv^*$, chaque classe d'équivalence obtenue identifie bien un sommet nouvellement découvert (un ensemble d'arêtes nouvellement découvertes pointant un même sommet non présent dans la carte). De plus, cette relation identifie aussi toutes les arêtes verticales nouvellement découvertes, i.e., les relations d'adjacences entre les sommets nouvellement découverts et les sommets de la carte.

Nous décrivons maintenant la mise à jour de la carte. Pour toute classe d'équivalence $[\mathbf{n}, p, p'] \in \text{PRE-VERT}^i/\equiv^*$, un nouveau sommet est ajouté à l'ensemble $V(\mathcal{M}^i)$ (Ligne 20) numéroté par la valeur d'un compteur, noté **compteur**. Puis, pour tout pré-sommet $(\mathbf{n}', q, q') \in \text{PRE-VERT}^i$ appartenant à la classe d'équivalence $[\mathbf{n}, p, p']$, une arête étiquetée (q, q') est ajoutée à l'ensemble $E(\mathcal{M}^i)$ reliant le sommet $\mathbf{n}'$ et le sommet ajouté $\mathcal{M}[\mathbf{n}] [p]$ (Ligne 21). Finalement, pour toute entrée $(\mathbf{n}', p, q, (r, r')) \in \mathcal{HOR}$, une arête étiquetée (r, r') est ajoutée à l'ensemble $E(\mathcal{M})$ reliant les deux sommets $\mathcal{M}[\mathbf{n}'] [p]$ et $\mathcal{M}[\mathbf{n}'] [q]$ (Ligne 24). Entre chaque classe d'équivalence, le compteur est incrémenté pour garantir l'unicité des identités des sommets de la carte.

Définition 9.25. Nous noterons C^i le cluster visité par l'agent lors d'une phase i d'une exécution sur un graphe G .

Nous présentons maintenant quelques propriétés satisfaites par une carte et sa bordure tout au long d'une exécution.

Proposition 9.26. Pour tout sommet base $v_0 \in V(G)$, pour toute phase i d'une exécution de l'algorithme 10 sur un graphe G , les conditions suivantes sont vérifiées :

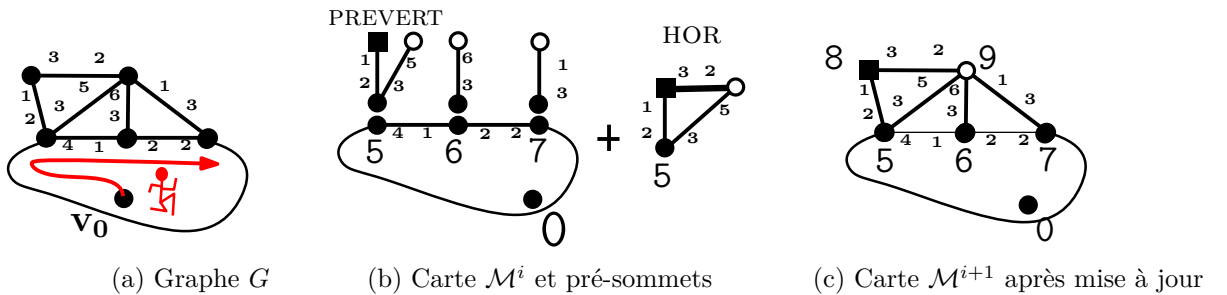


FIGURE 9.10 – Mise à jour de la carte. Au centre, la carte $\mathcal{M}^i$ et le contenu de la structure $\mathcal{HOR}$. Les arêtes pendantes de $\mathcal{M}^i$ sont les pré-sommets calculés lors de cette phase. Les différents types de pré-sommets encodent les classes d'équivalence (par la relation $\text{PRE-VERT}/\equiv^*$). Ici, deux classes d'équivalence sont calculées : carrés noirs et cercles blancs. À droite, la carte $\mathcal{M}^{i+1}$ obtenue en fin de mise à jour.

Algorithme 10 : Exploration des graphes de Weetman

```

1  AVisiter  $\leftarrow \{0\}$  ; /* Initialisation AVisiter */
2   $V(\mathcal{M}) \leftarrow \{0\}$  ; /* Initialisation  $\mathcal{M}^0$  */
3  Répéter
4  | Calculer le prochain cluster  $\mathcal{C}$  non visité dans la carte suivant l'ordre d'un parcours en
   | profondeur;
5  | Calculer le plus court chemin  $c$  qui, de la position actuelle de l'agent dans la carte,
   | mène au cluster  $\mathcal{C}$  et visite tous ses sommets;
   | /* exploration du cluster  $\mathcal{C}$  */
6  | pour tout sommet  $n \in \mathcal{C}$  visité lors du parcours du chemin  $c$  faire
7  | | AVisiter  $\leftarrow \text{AVisiter} \setminus \{n\}$ ;
8  | |  $\mathcal{B}[n] \leftarrow \text{getBino}()$ ;
9  | | pour tout couple  $(p, q) \in E(\mathcal{B}[n])$ , tel que  $\mathcal{M}[n][p]$  n'est pas défini faire
10 | | |  $\text{PRE-VERT} \stackrel{\cup}{\leftarrow} (n, p, q)$  ; /* Traitement des pré-sommets */
11 | | pour tout couple  $(pqr, r'q'p') \subseteq \mathcal{B}[n]$  faire
12 | | | si  $\mathcal{M}[n][r'] = m$ ,  $\mathcal{M}[n][p] \notin V(\mathcal{M})$  et  $\mathcal{M}[m][q'] \notin V(\mathcal{M})$  alors
13 | | | |  $\mathcal{R} \stackrel{\cup}{\leftarrow} ((n, p, p'), (m, q', q))$ ; /* Traitement de arêtes verticales */
14 | | | si  $\mathcal{M}[n][r'] \notin V(\mathcal{M})$  et  $\mathcal{M}[n][p] \notin V(\mathcal{M})$  alors
15 | | | |  $\mathcal{HOR} \stackrel{\cup}{\leftarrow} (n, p, r', (q, q'))$ ; /* Traitement des arêtes horizontales */
   | /* Mise à jour de la carte */
16 | pour chaque classe d'équivalence  $e \in (\text{PRE-VERT}/\equiv^*)$  faire
17 | | compteur ++ ;
18 | | AVisiter  $\leftarrow \text{AVisiter} \cup \{\text{compteur}\}$  ; /* Ajout d'un nouveau sommet à visiter */
19 | | pour tout  $(n, p, q) \in e$  faire
20 | | |  $\mathcal{M}[n][p] = \text{compteur}$  ; /* Ajout d'une nouvelle arête verticale */
21 | | |  $\mathcal{M}[\text{compteur}][q] = n$ ;
22 | pour toute entrée  $(n, p, r', (q, q')) \in \mathcal{HOR}$  faire
23 | |  $\mathcal{M}[\mathcal{M}[n][p]][q] = \mathcal{M}[n][r']$ ; /* Ajout d'une nouvelle arête horizontale */
24 | |  $\mathcal{M}[\mathcal{M}[n][r']][q'] = \mathcal{M}[n][p]$ ;
25 | si il existe  $n \in \mathcal{C}$  tel que  $\mathcal{B}[n]$  n'est pas isomorphe à  $B_{\mathcal{M}}(n, 1)$  alors
26 | | Continuer pour toujours l'exploration sur une arête; /* Détection d'erreur */
27 Jusqu'à ce que  $\text{AVisiter} \neq \emptyset$ ;

```

- (i) **Visite de la bordure** : $V(C^i) \subseteq AVisiter^i$
- (ii) **Visite unique** : $V(C^i) \cap AVisiter^{i+1} = \emptyset$
- (iii) **Cohérence des pré-sommets** Pour tous sommets $\mathbf{n}, \mathbf{n}' \in V(C^i)$, si $[\mathbf{n}, p, p'] \neq [\mathbf{n}', q', q] \in PRE-VERT/\equiv^*$ alors $\mathcal{M}^{i+1}[\mathbf{n}][p] \neq \mathcal{M}^{i+1}[\mathbf{n}'][q']$.
- (iv) **Carte croissante** : $\mathcal{M}^i \subseteq \mathcal{M}^{i+1}$
- (v) **Cohérence de la carte** : Pour tout sommet $\mathbf{n} \in V(C^i)$, il existe $u = v_0.route^i[\mathbf{n}] \in V(G)$
- (vi) **Complétude des pré-sommets** Pour tout sommet $\mathbf{n} \in V(C^i)$, si $(p, p') \in \mathcal{B}[\mathbf{n}]$ alors il existe $m = \mathcal{M}^{i+1}[\mathbf{n}][p]$ et $\mathcal{M}^{i+1}[m][p'] = \mathbf{n}$
- (vii) **Complétude des triangles** Pour tout sommet $\mathbf{n} \in V(C^i)$, si $(pqr, p'q'r') \in \mathcal{B}^i[\mathbf{n}]$ alors $\mathcal{M}^{i+1}[\mathbf{n}][pq] = \mathcal{M}[\mathbf{n}][r']$ et $\mathcal{M}^{i+1}[\mathbf{n}][r'q'] = \mathcal{M}^{i+1}[\mathbf{n}][p]$.
- (viii) **Complétude de la carte** Pour tous sommets $\mathbf{n}, \mathbf{n}' \in V(C^i)$, si $(\mathbf{n}, p, p'), (\mathbf{n}', q', q) \in PRE-VERT^i$ et $(\mathbf{n}, p, p') \equiv (\mathbf{n}', q', q)$ alors $(pqr, p'q'r') \in \mathcal{B}^i[\mathbf{n}]$ et $(p'q'r', pqr) \in \mathcal{B}^i[\mathbf{n}']$.
- (ix) **Connexité de la bordure** Pour tout $\mathbf{n} \in AVisiter^{i+1}$, il existe un sommet $m \in V(\mathcal{M}^i) \setminus AVisiter^{i+1}$ tel que $mn \in E(\mathcal{M}^i)$.

Démonstration. Pour toute phase i d'une exécution de l'algorithme 10 sur un graphe G se terminant sans erreur,

- (i) La propriété est obtenue par définition
- (ii) La propriété est obtenue par définition
- (iii) La valeur de **compteur** n'étant jamais décrémentée, celle-ci ne peut qu'augmenter lors d'une exécution. La valeur du compteur sera donc différente lors de toute phase et pour chaque nouveau sommet inséré dans la carte.
- (iv) La propriété est obtenue à partir du point (iii) et par l'algorithme car aucun élément de la carte n'est retiré, supprimé ou modifié lors d'une exécution.
- (v) Si un agent finit une phase i sans boucle infinie, alors pour tout sommet $\mathbf{n}$ où se situe l'agent en début de phase et tout chemin c calculé qui à partir de $\mathbf{n}$ rejoint et explore le cluster C^i , il existe un sommet $POS(\mathbf{n})$ et un chemin $POS(c)$ dans G tel que $\lambda(c) = \lambda(POS(c))$ (sinon, l'agent aurait détecté une erreur lors du parcours du chemin c et aurait commencé une boucle infinie). Donc, si un agent finit une étape sans boucle infinie, pour tout sommet $\mathbf{n} \in V(C^i)$, il existe $u = v_0.route^i[\mathbf{n}] \in V(G)$.
- (vi) À partir des lignes 9,16 et 20.
- (vii) Un triangle est créé dans la carte à partir des lignes 9,16 et 20.
- (viii) À partir de la condition (vi) et des lignes 16, 21 et 24.
- (ix) Par l'algorithme.

□

9.3 Correction de l'algorithme

Nous prouvons maintenant que notre algorithme est correct. Pour cela, nous montrons les propriétés suivantes.

Correction : Pour tout graphe, si l'algorithme s'arrête, alors le graphe est exploré.

Terminaison : Pour tout graphe de Weetman, l'algorithme s'arrête.

Par la suite nous considérerons des étapes d'une exécution se terminant sans boucle infinie.

Nous montrons que, pour tout graphe, si l'agent arrête son exécution, alors la carte construite par l'agent est un revêtement du graphe exploré. Pour montrer cela, nous prouverons le théorème suivant en plusieurs étapes.

Théorème 9.1. *Pour toute phase i d'une exécution de l'algorithme sur un graphe G , il existe un morphisme $\varphi^i : \mathcal{M}^i \rightarrow G$ tel que*

- φ^i est localement injectif sur $\mathcal{M}^i[\text{AVisiter}^i]$.
- φ^i est localement bijectif sur $\mathcal{M}^i \setminus \mathcal{M}^i[\text{AVisiter}^i]$ et $B_{\mathcal{M}}(\mathbf{n}, 1) \simeq B_G(\varphi^i(\mathbf{n}), 1)$, pour tout $\mathbf{n} \in \mathcal{M}^i \setminus \mathcal{M}^i[\text{AVisiter}^i]$.

Notons que $\mathcal{M}^{i+1}$ est calculée en fin de phase i et que lors de l'exploration du cluster C^i , l'agent utilise la carte $\mathcal{M}^i$. Pour prouver ce théorème, nous procédons par récurrence sur les phases d'une exécution.

Initialement, le premier cluster $\mathcal{C}^0$ exploré contient uniquement le sommet base v_0 de l'agent et la carte contient initialement un unique sommet $\mathbf{n}_0$ (par l'algorithme). Par conséquent, $\mathcal{M}^0$ et $\mathcal{C}^0$ sont isomorphes. La phase 0 consiste donc à calculer $\mathcal{M}^1$ en cartographiant le voisinage de v_0 . Notons que l'agent ne se déplacera pas durant cette phase.

Lemme 9.27 (Cas initial). *Pour toute exécution de l'algorithme 10 sur un graphe G , $\mathcal{M}^1$ est isomorphe à $B_G(v_0, 1)$.*

Démonstration. Initialement, $\mathbf{n}_0 = 0$ et $\mathcal{M}^0 = (\{0\}, \emptyset)$ (Ligne 2). L'agent, positionné sur sa base, ne se déplace pas durant cette phase, accède à l'étiquetage de jumelles de v_0 et le stocke dans $\mathcal{B}[0]$ (Ligne 8). Comme G possède un étiquetage des ports injectif, pour tout couple $(p, p'), (q, q') \in \mathcal{B}[0]$, $p \neq q$. Par conséquent, pour tout couple $(p, p'), (q, q') \in \mathcal{B}[0]$, $[\mathbf{n}, p] \neq [\mathbf{n}, q]$ (Condition (iii) de la proposition 9.26) et donc pour toute arête $v_0v \in E(G)$, il existe une unique arête $\mathbf{n}_0\mathbf{m} \in E(\mathcal{M}^1)$ étiquetée $\lambda(v_0v)$ (Condition (vi) de la proposition 9.26). Par l'algorithme, pour toute arête $\mathbf{nm} \in E(\mathcal{M}^1)$, il existe un couple $(p, p') \in \mathcal{B}[\mathbf{n}]$. Par le cas précédent, pour toute arête $\mathbf{nm} \in E(\mathcal{M}^1)$, il existe une arête $uv \in B_G(v_0, 1)$.

Pour tout triangle v_0vw de $E(B(v_0, 1))$, il existe une entrée $(pqr, r'q'p') \in \mathcal{B}[\mathbf{n}]$ telle que $\mathcal{M}^0[0][p] = \mathcal{M}^0[0][r'] = \perp$. À partir des cas précédents, il existe deux sommets $\mathcal{M}^1[\mathbf{n}_0][pq] = \mathbf{m}'$ et $\mathcal{M}^1[\mathbf{n}_0][r'q'] = \mathbf{m}$ dans $\mathcal{M}^1$. Par la condition (vii) de la proposition 9.26, il existe donc un unique triangle $\mathbf{n}_0\mathbf{mm}'$ étiqueté $(pqr, r'q'p')$ dans $\mathcal{M}^1$, pour tout $(pqr, r'q'p') \in \mathcal{B}[0]$. □

Nous considérons un morphisme $\varphi^h : \mathcal{M}^h \rightarrow G$ complétant la fonction POS pour considérer les sommets non visités de la carte au début d'une étape h (sommets contenus dans AVisiter^h). Supposons qu'un agent termine une phase $h < i$, pour tout sommet $\mathbf{n} \in V(\mathcal{M}^h)$,

- si $\mathbf{n} \notin \text{AVisiter}^h$ alors $\varphi^h(\mathbf{n}) = v_0.\text{route}[\mathbf{n}]$,
- si $\mathbf{n} \in \text{AVisiter}^h$ alors $\varphi^h(\mathbf{n}) = \varphi^h(\mathbf{m}).p$ où $\mathbf{m} \notin \text{AVisiter}^h$ et $\mathbf{m}.p = \mathbf{n}$.

Supposons que pour toute étape $h \leq i$, le morphisme $\varphi^h : \mathcal{M}^h \rightarrow G$ vérifie le théorème 9.1. Nous montrons qu'en fin d'étape i , le morphisme $\varphi^{i+1} : \mathcal{M}^{i+1} \rightarrow G$ vérifie le théorème 9.1.

Premièrement, comme tout sommet $\mathbf{n}$ visité lors de l'étape i appartient à la carte $M^i(v)$ calculée en fin d'étape $i-1$. De par nos hypothèses de récurrence appliquées à $M^i(v)$, la proposition 9.28 est vérifiée.

Proposition 9.28. *Pour toute exécution de l'algorithme dans un graphe $\mathbf{G}$ (avec un sommet base v_0), pour toute étape i qui termine sans boucle infinie, pour tout sommet $\mathbf{n} \in V(\mathcal{M}^i)$, si $\mathbf{n} \notin \mathbf{AVisiter}^{i+1}$, alors il existe un sommet $v \in V(G)$ et un chemin c reliant v_0 à v dans G tel que $\text{route}[p] = \lambda(c)$.*

Corollaire 9.29. *Pour tout cluster C^h visité lors d'une étape $h < i$, pour tout sommet $\mathbf{n} \in V(C^h)$, $B_G(\varphi^i(\mathbf{n}), 1) \simeq \text{getBino}(\mathbf{n})$.*

Nous montrons maintenant qu'en fin d'étape i , le morphisme φ^{i+1} vérifie le théorème 9.1. Pour cela, nous étendons le morphisme φ^i en un morphisme φ^{i+1} défini sur l'ensemble des sommets de $\mathcal{M}^{i+1}(v)$.

Définition 9.30. Soit $\varphi^{i+1} : V(\mathcal{M}^{i+1}) \rightarrow V(G)$ la fonction telle que, pour tout sommet $\mathbf{n} \in V(\mathcal{M}^{i+1})$,

- si $\mathbf{n} \notin \mathbf{AVisiter}^{i+1}$ alors $\varphi^{i+1}(\mathbf{n}) = v_0.\text{route}[\mathbf{n}]$
- si $\mathbf{n} \in \mathbf{AVisiter}^{i+1}$ alors $\varphi^{i+1}(\mathbf{n}) = \varphi^i(\mathbf{m}).p$ avec $\mathbf{m} \notin \mathbf{AVisiter}^{i+1}$ et $\mathbf{m}.p = \mathbf{n}$.

Premièrement, nous montrons que la fonction φ^{i+1} est bien définie pour tout sommet appartenant à $V(\mathcal{M}^i)$ et que l'image d'un sommet de $V(\mathcal{M}^i)$ est préservée par φ^{i+1} .

Proposition 9.31. *Pour toute phase i d'une exécution de l'algorithme 10 sur un graphe G , pour tout sommet $\mathbf{n} \in V(\mathcal{M}^i)$. $\varphi^i(\mathbf{n}) = \varphi^{i+1}(\mathbf{n})$.*

Démonstration. Notons que par la proposition 9.28, la fonction φ^i est bien définie sur l'ensemble des sommets de $V(\mathcal{M}^i)$. Si $\mathbf{n} \notin \mathbf{AVisiter}^{i+1}$ alors comme $\varphi^{i+1}(\mathbf{n}) = v_0.\text{route}^i[\mathbf{n}]$ et que $\text{route}^i[\mathbf{n}] = \text{route}^h[\mathbf{n}]$, pour tout $h < i$ (Proposition 9.28), ce cas est prouvé. Sinon $\mathbf{n} \in V(\mathcal{M}^i) \cap \mathbf{AVisiter}^{i+1}$ et alors, par définition, $\varphi^i(\mathbf{n}) = \varphi^{i-1}(\mathbf{m}).p$ avec $\mathbf{m} \notin \mathbf{AVisiter}^{i+1}$ et $\mathbf{m}.p = \mathbf{n}$. Comme par le cas précédent, $\varphi^{i-1}(\mathbf{m}) = \varphi^i(\mathbf{m}) = \varphi^{i+1}(\mathbf{m})$, par la proposition 9.26, $\mathcal{M}^i \subseteq \mathcal{M}^{i+1}$ et donc, par définition, $\varphi^{i+1}(\mathbf{n}) = \varphi^i(\mathbf{m}).p = \varphi^i(\mathbf{n}) \in V(G)$, ce lemme est prouvé. $\square$

À partir de la proposition 9.31, nous prouverons uniquement que φ^{i+1} est bien définie pour les sommets contenus dans l'ensemble $V(\mathcal{M}^{i+1} \setminus \mathcal{M}^i) (\subseteq \mathbf{AVisiter}^{i+1})$, i.e., les sommets nouvellement découverts. Nous montrons maintenant que l'image d'un sommet de $\mathbf{AVisiter}^{i+1}$ par φ^{i+1} est indépendante du choix du sommet adjacent $\mathbf{m} \in V(\mathcal{M}^i)$ (voir Définition 9.30). Pour cela, nous montrons qu'une classe d'équivalence de la relation $\equiv^*$ rassemble tous les pré-sommets de PRE-VERT^i encodant un même sommet de $V(G)$.

Lemme 9.32. *Pour toute phase i d'une exécution de l'algorithme 10 sur un graphe G , pour tous sommets $\mathbf{m}, \mathbf{m}' \in V(C^i)$, pour tout sommet $\mathbf{n} \in V(\mathbf{AVisiter}^{i+1})$, si $\mathbf{m}\mathbf{n}, \mathbf{m}'\mathbf{n} \in E(\mathcal{M}^{i+1})$ alors $\varphi^i(\mathbf{m}).\delta_{\mathbf{m}}(\mathbf{n}) = \varphi^i(\mathbf{m}').\delta_{\mathbf{m}'}(\mathbf{n}) = w \in V(G)$.*

Démonstration. Prenons une étape i se terminant sans boucle infinie et prenons un sommet $\mathbf{m} \in V(C^i)$. Notons que pour tout sommet $w \in V(C^i)$, $w \in \mathbf{AVisiter}^i \subset V(\mathcal{M}^i)$ et donc $\varphi^i(w)$ est bien défini par hypothèse de récurrence. Premièrement, nous montrons que pour tout sommet $\mathbf{m} \in V(C^i)$ et pour tout sommet $\mathbf{n} \in V(\mathbf{AVisiter}^{i+1})$, si $\mathbf{m}\mathbf{n} \in E(\mathcal{M}^{i+1})$ alors $\varphi^i(\mathbf{m}).\delta_{\mathbf{m}}(\mathbf{n}) = w$ est un sommet de G . À partir de la condition (vi) de la proposition 9.26, il existe un couple $(p, p') \in \mathcal{B}^i[\mathbf{m}]$ tel que $\lambda(\mathbf{m}\mathbf{n}) = (p, p')$. À partir du corollaire 9.29 et comme $\mathbf{m} \in V(\mathcal{M}^i)$, il existe un sommet $v = \varphi^i(\mathbf{m}) \in V(G)$ et $B_G(v, 1)$ est isomorphe à $\mathcal{B}^i[\mathbf{m}]$. Par conséquent, il existe un sommet $w \in V(B_G(\varphi^i(\mathbf{m}), 1))$ et une arête $vw \in E(G)$ étiquetée (p, p') . Comme $\varphi^i(\mathbf{m}).\delta_v(w) = w \in V(G)$, nous prouvons ce cas.

Nous prouvons maintenant que pour toutes arêtes $mn \neq m'n \in E(\mathcal{M}^{i+1})$, si m et $m' \in V(C^i)$ et $n \in V(\text{AVisiter}^{i+1})$ alors $\varphi^i(m) \cdot \delta_m(n) = \varphi^i(m') \cdot \delta_{m'}(n) \in V(G)$. Prenons deux arêtes $mn, m'n \in E(\mathcal{M}^{i+1})$ avec $m, m' \in V(C^i)$ et $n \in V(\text{AVisiter}^{i+1})$. À partir de l'algorithme, il existe deux pré-sommets $(m, p, p'), (m', q', q) \in \text{PRE-VERT}^i$ tels que $(m, p, p') \equiv^* (m', q', q)$. À partir de l'algorithme, il existe deux sommets $w, w' \in V(G)$ tels que $\varphi^i(m) \cdot \delta_m(n) = w$ et $\varphi^i(m') \cdot \delta_{m'}(n) = w'$.

Supposons par l'absurde que $w \neq w'$. S'il existe une arête $mm' \in E(\mathcal{M}^{i+1})$, alors $mm'n$ est un triangle et comme $n \in V(\text{AVisiter}^{i+1})$, il existe deux pré-sommets $(m, p, p'), (m', q', q) \in \text{PRE-VERT}^i$ tels que $(m, p) \equiv (m', q')$ (par définition). Par les conditions (viii) et (vii) de la proposition 9.26, il existe $(pqr, r'q'p') \in \mathcal{B}[m]$ (resp. $(p'q'r', rqp) \in \mathcal{B}[m']$) avec $\mathcal{M}[m][pq] = m'$ et $\mathcal{M}[m'][q'p'] = m$. À partir du corollaire 9.29, il existe donc un unique sommet $w \in V(B_G(\varphi^i(m), 1) \cap B_G(\varphi^i(m'), 1))$ tel que $\varphi^i(m) \cdot p = \varphi^i(m') \cdot q' = w$. Nous obtenons une contradiction.

S'il n'existe pas d'arête $mm' \in E(\mathcal{M}^{i+1})$, alors comme $(m, p, p') \equiv^* (m', q', q)$, il existe une séquence de pré-sommets $(m_1, p_1), \dots, (m_k, p_k) \in \text{PRE-VERT}^i$ vérifiant

- $(m, p) = (m_1, p_1)$ et $(m', q) = (m_k, p_k)$
- $\forall 1 \leq h < k, (m_h, p_h) \equiv (m_{h+1}, p_{h+1})$

Supposons ce lemme prouvé pour toute séquence de pré-sommets de longueur $h < k \in \mathbb{N}$, i.e., si $(m, p) = (m_1, p_1) \equiv^* (m_h, p_h)$, $h < k$, alors il existe un unique sommet $w \in V(B_G(\varphi^i(m), 1) \cap B_G(\varphi^i(m_h), 1))$ tel que $w = \varphi^i(m) \cdot p = \varphi^i(m_h) \cdot p_h$. Nous prouvons le cas $h = k$.

Comme $\equiv^*$ est une relation transitive, $(m_1, p_1) \equiv^* (m_{k-1}, p_{k-1})$. Par notre hypothèse de récurrence, il existe un unique sommet $w \in V(B_G(\varphi^i(m_1), 1) \cap B_G(\varphi^i(m_{k-1}), 1))$ tel que $w = \varphi^i(m_1) \cdot p = \varphi^i(m_{k-1}) \cdot p_{k-1}$. Comme $(m_{k-1}, p_{k-1}) \equiv (m_k, p_k)$, nous obtenons par le premier cas qu'il existe un unique sommet $w \in V(B_G(\varphi^i(m_{k-1}), 1) \cap B_G(\varphi^i(m_k), 1))$ tel que $\varphi^i(m_{k-1}) \cdot p_{k-1} = \varphi^i(m_k) \cdot p_k = w \in V(G)$. Comme nous venons de prouver que $\varphi^i(m_1) \cdot p_1 = \varphi^i(m_k) \cdot p_k = w \in V(G)$, on a $w = w'$ et donc, nous obtenons une contradiction. $\square$

Soit $m, m' \in C^h$ deux sommets visités lors d'une phase $h \leq i$ et adjacents à un sommet $n \in V(\text{AVisiter}^{i+1})$. Comme nous venons de prouver que $\varphi^{i+1}(n) = \varphi^i(m) \cdot \delta_m(n) = \varphi^i(m') \cdot \delta_{m'}(n)$ (Lemme 9.32), nous obtenons que $\varphi^{i+1}(n)$ est bien défini dans G et est indépendante du choix du sommet $m \in C^h$ requis par la définition. Nous montrons maintenant que φ^{i+1} est un morphisme.

Lemme 9.33. *Pour toute phase i d'une exécution de l'algorithme 10 sur un graphe G , $\varphi^{i+1} : \mathcal{M}^{i+1} \rightarrow G$ est un morphisme préservant l'étiquetage des ports.*

Démonstration. Nous prouvons ce lemme en montrant que φ^{i+1} est bien défini pour les arêtes "verticales" puis, pour les arêtes "horizontales". Soit $mn \in E(\mathcal{M}^{i+1})$ une arête "verticale" de la carte, i.e., $m \in C^h$, $h < i$ et $n \in \text{AVisiter}^{i+1}$ un sommet non visité en fin d'étape i . Comme $m \in V(\mathcal{M}^i)$, $\varphi^{i+1}(m) = \varphi^i(m)$ et donc, à partir du lemme 9.32, il existe un unique sommet $v \in V(G)$ tel que $v = \varphi^i(m) \cdot \delta_m(n)$, nous obtenons par définition de φ^{i+1} que $\varphi^{i+1}(mn) = \varphi^{i+1}(m) \varphi^{i+1}(n) \in E(G)$ et $\lambda(mn) = \lambda(\varphi^{i+1}(mn))$.

Soit $nn' \in E(\text{AVisiter}^{i+1})$, une arête "horizontale" étiquetée (r, r') . À partir de l'algorithme, il existe un sommet $m \in V(\mathcal{M}^i)$ tel que $mn, mn' \in E(\mathcal{M}^i)$. À partir de la condition (vii) de la proposition 9.26, nous savons qu'il existe $(pqr, r'q'p') \in \mathcal{B}[m]$ avec $\lambda(nm) = (p, p')$ et $\lambda(mn') = (q, q')$ et $\lambda(n'n) = (r, r')$.

À partir de la proposition 9.28 et du cas précédent, il existe donc un triangle $vw w'$ tel que $\lambda(mnn') = \lambda(vww') = (pqr, p'q'r')$, $\varphi^{i+1}(n) = v$ et $\varphi^{i+1}(mn') = \varphi^{i+1}(m) \varphi^{i+1}(n') = ww' \in E(B_G(\varphi^{i+1}(n), 1))$ avec $\lambda(ww') = \lambda(mn')$. Le morphisme φ^{i+1} préserve donc les arêtes horizontales et les triangles de la carte, ce lemme est prouvé. $\square$

Pour finir, nous prouvons que pour tout sommet n visité pendant la phase i (appartenant à C^i), $B_G(\varphi^{i+1}(n), 1)$ est isomorphe à $B_{\mathcal{M}^{i+1}}(n, 1)$.

Lemme 9.34. *Pour tout sommet $\mathbf{n} \in V(C^i)$, $B_G(\varphi^{i+1}(\mathbf{n}), 1)$ est isomorphe à $B_{\mathcal{M}^{i+1}}(\mathbf{n}, 1)$.*

Démonstration. Nous prouvons ce lemme par récurrence sur les étapes d'une exécution. Initialement, ce lemme est prouvé (Lemme 9.27). Supposons ce lemme prouvé pour toute étape $h < i$. Nous montrons ce lemme pour l'étape i . Prenons un sommet $\mathbf{n} \in V(C^i)$ et montrons que $B_G(\varphi^{i+1}(\mathbf{n}), 1)$ est isomorphe à $B_{\mathcal{M}^{i+1}}(\mathbf{n}, 1)$. Comme G et $\mathcal{M}^i$ sont dotés d'un étiquetage des ports injectif et comme φ^{i+1} est un morphisme préservant l'étiquetage des graphes, nous obtenons que pour tout $\mathbf{n} \in V(C^i)$, $\varphi^{i+1} : B_G(\mathbf{n}, 1) \rightarrow G$ est localement injectif. À partir du corollaire 9.29, $B_G(\varphi^{i+1}(\mathbf{n}), 1)$ est isomorphe à $\mathcal{B}^i[\mathbf{n}]$. Comme par la ligne 25, $\mathcal{B}^i[\mathbf{n}] \simeq B_{\mathcal{M}^{i+1}}(\mathbf{n}, 1)$, nous obtenons que pour tout sommet $\mathbf{n} \in V(C^i)$, $B_{\mathcal{M}^{i+1}}(\mathbf{n}, 1)$ est isomorphe à $B_G(\varphi^{i+1}(\mathbf{n}), 1)$. $\square$

Les lemmes 9.27, 9.33 et 9.34 prouvent le théorème 9.1. Nous prouvons maintenant que si l'agent arrête son exécution lors d'une étape i alors φ^{i+1} est localement injective sur les bords de la carte $\mathcal{M}^{i+1}$, i.e., pour les sommets appartenant à AVisiter^{i+1} . Pour cela, remarquons que lorsqu'un agent arrête son exécution lors d'une phase i dans un graphe G , alors aucune erreur n'a été détectée et $\text{AVisiter}^{i+1} = \emptyset$, i.e., tous les sommets de sa carte sont visités. Comme $\text{AVisiter}^{i+1} = \emptyset$, par le théorème 9.1, nous en déduisons que φ^{i+1} est un morphisme localement bijectif de $\mathcal{M}^{i+1}$ vers G . Par définition, φ^{i+1} est un revêtement. Nous obtenons le corollaire suivant.

Corollaire 9.35. *Si l'agent arrête son exécution lors d'une phase i d'une exécution sur un graphe G , alors $\varphi^{i+1} : \mathcal{M}^{i+1} \rightarrow G$ est un revêtement préservant les étiquetages de jumelles.*

Remarquons que comme les revêtements sont des fonctions surjectives, un agent terminant son exécution aura toujours construit une carte plus "grande" ou égale au graphe exploré. Un tel cas peut se produire si le complexe de clique du graphe exploré n'est pas simplement connexe.

Pour montrer que tous les sommets du graphe G sont explorés, prenons une exécution sur G se terminant (sans boucle infinie) lors d'une phase i . Sans perte de généralité, si l'agent arrête son exécution, alors tous les sommets de $\mathcal{M}^i$ ont été visités lors de l'étape i et donc $\mathcal{M}^i \simeq \mathcal{M}^{i+1}$. Par la définition des revêtements, toute séquence de numéros de port **route** visitant tous les sommets de $\mathcal{M}^i$ explorera tout graphe G tel que $\mathcal{M}^i$ est un revêtement de G . Ainsi, par le corollaire 9.35, si l'agent s'arrête lors d'une étape i de l'exécution, alors la séquence **route** ^{i} traverse tous les sommets de G . Nous obtenons la propriété de correction suivante.

Théorème 9.2 (Correction). *Pour toute exécution de l'algorithme 10 sur un graphe G , si l'agent termine son exécution, alors tous les sommets de G ont été explorés.*

Pour finir, nous prouvons que notre algorithme vérifie la propriété de terminaison, i.e., Pour tout graphe de Weetman, notre algorithme s'arrête. Pour cela, nous prouvons en premier que dans un graphe de Weetman, l'agent construit une carte isomorphe au graphe exploré.

Lemme 9.36. *Pour tout graphe de Weetman G , pour toute étape i d'une exécution de l'algorithme sur G , il existe un morphisme $\varphi^{i+1} : \mathcal{M}^{i+1} \rightarrow G$ tel que $\mathcal{M}^{i+1}$ et $G[\varphi^{i+1}(\mathcal{M}^{i+1})]$ sont isomorphes.*

Démonstration. Nous procédons par récurrence sur les étapes d'une exécution. Le cas initial est prouvé grâce au lemme 9.27. Nous considérons le morphisme $\varphi^{i+1} : \mathcal{M}^{i+1} \rightarrow G$ défini dans le théorème 9.1. Supposons ce lemme vérifié pour toute étape $h \leq i$, i.e., $\mathcal{M}^h$ et $G[\varphi^h(\mathcal{M}^h)]$ sont isomorphes.

Nous prouvons qu'en fin d'étape i , pour tout sommet $v \in V(G)$, il existe un unique sommet $\mathbf{n} \in V(\mathcal{M}^{i+1})$, tel que $\varphi^{i+1}(\mathbf{n}) = v$. La figure 9.11 illustre les différents cas de cette preuve.

Soit $C^i \subseteq \mathcal{M}^i$, le cluster exploré par l'agent lors de l'étape i . Par hypothèse de récurrence, $\varphi^i(C^i)$ est un cluster de G isomorphe au cluster C^i et donc les distances entre les sommets de la carte et la base de l'agent $\mathbf{n}_0$ dans la carte sont les mêmes que celles séparant leurs images dans G avec le sommet base v_0 .

Supposons par l'absurde qu'il existe deux sommets $\mathbf{n}, \mathbf{n}' \in V(\text{AVisiter}^{i+1} \setminus \text{AVisiter}^i)$ tels que $\mathbf{n} \neq \mathbf{n}'$ et $\varphi^{i+1}(\mathbf{n}) = \varphi^{i+1}(\mathbf{n}') \in V(G)$. Comme $\mathbf{n}, \mathbf{n}' \in V(\text{AVisiter}^{i+1} \setminus \text{AVisiter}^i)$, $\mathbf{n}, \mathbf{n}'$ ont été ajoutés lors de l'étape i (lors de la visite du cluster C^i) et, à partir de la définition de φ^{i+1} , il existe deux sommets $\mathbf{m}, \mathbf{m}' \in V(C^i)$ tels que $\varphi^{i+1}(\mathbf{n}) = \varphi^i(\mathbf{m}).p$ et $\varphi^{i+1}(\mathbf{n}') = \varphi^i(\mathbf{m}').q$. Comme $\mathbf{m}, \mathbf{m}' \in \mathcal{M}^i$, à partir de nos hypothèses de récurrence, il existe $w, w' \in V(G)$ tels que $\varphi^i(\mathbf{m}) = \varphi^{i+1}(\mathbf{m}) = w$ (resp. $\varphi^i(\mathbf{m}') = \varphi^{i+1}(\mathbf{m}') = w'$). Par le théorème 9.1, $B_G(w, 1) \simeq B_{\mathcal{M}^{i+1}}(\mathbf{m}, 1)$, $B_G(w', 1) \simeq B_{\mathcal{M}^{i+1}}(\mathbf{m}', 1)$, $\varphi^{i+1}(\mathbf{mn}) = ww \in E(B_G(w, 1))$ et $\varphi^{i+1}(\mathbf{m}'\mathbf{n}') = w'v \in E(B_G(w, 1))$. Comme G est un graphe de Weetman et que $d_G(v_0, w) = d_G(v_0, w') = d_G(v_0, v) - 1$, par la condition INT(v_0) appliquée au triplet w, w', v , il existe une séquence de sommets $\{w = w_1 \dots w_\ell = w'\} \subset \varphi^{i+1}(C^i)$ telle que pour tout $1 \leq i < \ell$, $w_i w_{i+1} \in E(G)$ et pour tout $1 \leq i \leq \ell$, $w_i v \in E(G)$ et $\lambda(w_i v) = (p_i, q_i)$. Nous procédons par récurrence sur la longueur de cette séquence. Pour toute séquence de longueur 2, il existe $ww' \in E(G)$. Ainsi, $ww'v$ est un triangle dans G (Figure 9.11a). Comme $B_G(w, 1) \simeq B_{\mathcal{M}^i}(\mathbf{m}, 1)$ et $B_G(w', 1) \simeq B_{\mathcal{M}^i}(\mathbf{m}', 1)$ (Théorème 9.1), il existe un triangle $\mathbf{mm}'\mathbf{n}$ inclus dans $B_{\mathcal{M}^i}(\mathbf{m}, 1) \cap B_{\mathcal{M}^i}(\mathbf{m}', 1)$ tel que $\lambda(\mathbf{mm}'\mathbf{n}) = \lambda(ww'v)$. Par conséquent, $\mathcal{M}[\mathbf{m}][p] = \mathcal{M}[\mathbf{m}'][q] = \mathbf{n} = \mathbf{n}'$, nous obtenons une contradiction.

Supposons ce lemme vérifié pour toute séquence de longueur $\ell < k$. Par conséquent, pour tout sommet $\mathbf{n} \in V(\text{AVisiter}^{i+1})$, pour toute séquence de sommets $\{\mathbf{m}_1, \dots, \mathbf{m}_\ell\} \subseteq V(C^i)$ et pour tout $1 \leq h \leq \ell$, nous obtenons que $\varphi^{i+1}(\mathbf{m}_i) = w_i$, $\mathcal{M}[\mathbf{m}_i][\delta_{w_i}(v)] = \mathbf{n}$ et pour tout $1 \leq h < \ell$, $\mathbf{m}_i \mathbf{m}_{i+1} \in E(\mathcal{M}^i)$.

Nous prouvons maintenant ce lemme pour toute séquence de longueur k (Figure 9.11b). Par l'absurde, supposons qu'il existe une séquence $\{\mathbf{m} = \mathbf{m}_1 \dots \mathbf{m}_k = \mathbf{m}'\}$ telle que $\mathcal{M}[\mathbf{m}][p] = \mathbf{n} \neq \mathbf{n}' = \mathcal{M}[\mathbf{m}'][q]$. Par hypothèse de récurrence $\mathcal{M}[\mathbf{m}][p] = \mathcal{M}[\mathbf{m}_{k-1}][p_{k-1}] = \mathbf{n} = \mathbf{n}'$. Par définition, $w_{k-1}w'v$ forme un triangle dans G et nous avons prouvé précédemment que $\mathcal{M}[\mathbf{m}_{k-1}][p_{k-1}] = \mathcal{M}[\mathbf{m}'][q]$. Comme $\mathbf{n} = \mathcal{M}[\mathbf{m}][p] = \mathcal{M}[\mathbf{m}_{k-1}][p_{k-1}] = \mathcal{M}[\mathbf{m}'][q] = \mathbf{n}'$, $\mathbf{n} = \mathbf{n}'$ et par conséquent, nous obtenons une contradiction. Ainsi, tout sommet adjacent au cluster $\varphi^{i+1}(C^i)$ exploré lors de cette phase est bien identifié dans la carte. De plus, par le théorème 9.1 appliqué au cluster C^i , l'ensemble des arêtes verticales adjacentes au cluster $\varphi^{i+1}(C^i)$ est en bijection avec l'ensemble des arêtes verticales adjacentes au cluster C^i dans $E(\mathcal{M}^{i+1})$.

Nous montrons maintenant que φ^{i+1} préserve les arêtes horizontales de G (Figure 9.11c). Pour cela, prenons deux sommets $\mathbf{n} \neq \mathbf{n}'$ dans $V(\text{AVisiter}^{i+1})$ et une arête $ww' \in E(G)$ et prouvons que si $ww' = \varphi^{i+1}(\mathbf{n})\varphi^{i+1}(\mathbf{n}') \in E(G)$ alors il existe une arête $\mathbf{nn}' \in E(\mathcal{M}^{i+1})$. Par l'absurde, supposons $\mathbf{nn}' \notin E(\mathcal{M}^{i+1})$. Par notre hypothèse de récurrence $d_G(v_0, w) = d_G(v_0, w')$. Comme G est un graphe de Weetman, à partir de la condition TRI(v_0) avec l'arête $ww' = \varphi^{i+1}(\mathbf{n})\varphi^{i+1}(\mathbf{n}') \in E(G)$, il existe un sommet $u \in V(\varphi^{i+1}(C^i))$ tel que $uw, uw' \in E(G)$. Comme $u \in V(\varphi^{i+1}(C^i))$, il existe $\mathbf{m} \in V(C^i)$ tel que $\varphi^{i+1}(\mathbf{m}) = u$. Comme par définition, $\varphi^{i+1}(\mathbf{mn}) = uw$, $\varphi^{i+1}(\mathbf{m}\mathbf{n}') = uw'$ et $B_G(u, 1) \simeq B_{\mathcal{M}^{i+1}}(\mathbf{m}, 1)$ (Théorème 9.1), il existe une arête $\mathbf{nn}' \in E(\mathcal{M}^{i+1})$, contredisant nos hypothèses. Par conséquent, par le théorème 9.1 appliqué au cluster C^i , l'ensemble des arêtes horizontales de $E(G)$ adjacentes au cluster $\varphi^{i+1}(C^i)$ est en bijection avec l'ensemble des arêtes horizontales de $E(\mathcal{M}^{i+1})$ adjacentes au cluster C^i et donc, $\mathcal{M}^{i+1}$ et $G[\varphi^{i+1}(\mathcal{M}^{i+1})]$ sont isomorphes.

□

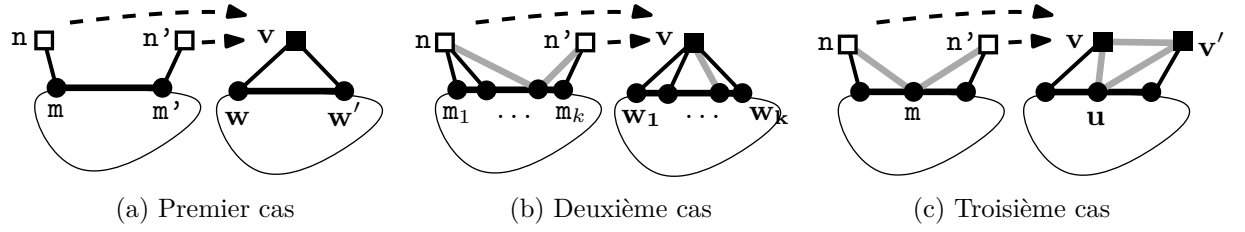


FIGURE 9.11 – Illustration des trois cas de la preuve du lemme 9.36. Les ronds noirs représentent les sommets du cluster C^i (visité lors de l'étape i) et leurs images dans G , i.e., $\varphi^i(C^i)$. Les carrés blancs représentent les sommets de AVisiter^i (les sommets inexplorés de la carte)

Théorème 9.3 (Terminaison). *Pour tout graphe de Weetman G , l'algorithme explore tous les sommets de G et s'arrête.*

Démonstration. Prenons un graphe de Weetman G , une étape i et considérons le morphisme $\varphi^{i+1} : \mathcal{M}^{i+1} \rightarrow G$ défini dans le théorème 9.1. Par la condition (iv) du lemme 9.26, $|\mathcal{M}^0|$ est croissante et par le lemme 9.36, $|\mathcal{M}^i| = |\varphi^i(\mathcal{M}^i)|$, pour toute étape i . Par conséquent (G est un graphe fini), il existe une étape i telle que $|\mathcal{M}^i| = |G|$ et $\mathcal{M}^{i+1}$ est isomorphe à $G[\varphi^{i+1}(\mathcal{M}^{i+1})]$ (Lemme 9.36). Comme lors d'une telle étape, l'agent a exploré tous les sommets du graphe et tous les sommets de sa carte ont été visités. L'agent arrêtera donc son exploration en ayant exploré intégralement le graphe. $\square$

Remarque 9.37. L'algorithme 10 peut être modifié pour explorer tout graphe appartenant à la famille $\mathcal{FC}$ et si le graphe exploré est un graphe de Weetman, alors il l'explorera efficacement. Pour cela, il suffit d'exécuter l'algorithme 8 dès que l'agent détecte une erreur lors d'une exécution de l'algorithme 10.

9.4 Complexité de l'algorithme

Premièrement, nous présentons des propriétés générales sur les arbres qui nous seront utiles pour calculer la complexité de notre algorithme. Ces propriétés sont connues et peuvent être facilement prouvées.

Proposition 9.38. *Les propriétés suivantes sont satisfaites,*

1. *Pour tout arbre T , $|E(T)| = |V(T)| - 1$,*
2. *Pour tout arbre T , le nombre de mouvements requis pour explorer T est au plus $2 \times |E(T)| \leq 2 \times (|V(T)| - 1)$ mouvements (et ce, même si l'agent doit retourner sur sa base en fin d'exploration),*
3. *Explorer un arbre couvrant d'un graphe (arbitraire) permet d'explorer tous les sommets du graphe.*

Soit G un graphe de Weetman et soit v_0 le sommet base de l'agent dans G . Soit $\mathcal{C}$ l'ensemble des clusters de G obtenus à partir du sommet $v_0 \in V(G)$.

Nous déterminons la complexité de notre algorithme dans le pire des cas en comptant séparément le nombre d'arêtes horizontales et verticales traversées par l'agent. La figure 9.13 illustre le comportement de l'agent dans un graphe de Weetman.

Pour déterminer le nombre d'arêtes horizontales traversées par l'agent, observons que pour tout cluster C présent dans sa carte, l'agent, dans le pire des cas, explore deux fois un arbre

couvrant du cluster C . Une première fois pour visiter tous les sommets du cluster C et une seconde fois pour aller explorer les clusters adjacents à C nouvellement découverts. Notons que pour obtenir une exploration efficace, l'agent doit bien faire attention à l'ordre d'exploration des clusters pour que le nombre d'arêtes traversées reste linéaire en la taille du graphe. Cet ordre peut être calculé par l'agent grâce au corollaire 9.13. Ainsi, pour tout cluster $C \in \mathcal{C}$, un agent traversera au maximum $4 \times (|V(C)| - 1)$ arêtes. Notons que si un cluster est une "feuille" dans l'arbre de clusters de G , alors l'agent ne visitera qu'une seule fois l'arbre couvrant du cluster dans le pire des cas.

Les seules arêtes verticales traversées sont les arêtes franchies par l'agent pour quitter un cluster et se rendre à un cluster adjacent. Comme les clusters de la carte forment un arbre et que l'agent parcourt les clusters suivant un parcours en profondeur, un agent traversera, au plus, 2 arêtes verticales par cluster.

Nous obtenons la formule suivante en sommant, pour chaque cluster, les arêtes verticales et horizontales traversées.

$$\sum_{C \in \mathcal{C}} 4 \times (|V(C)| - 1) + 2 \leq \sum_{C \in \mathcal{C}} 4 \times V(C)$$

Comme par définition, $|V(G)| = \sum_{C \in \mathcal{C}} |V(C)|$, nous obtenons que l'agent effectue au plus $4 \times |V(G)|$ mouvements pour explorer un graphe de Weetman G . Notre algorithme d'exploration explore donc les graphes de Weetman avec une complexité linéaire ($O(|V(G)|)$ mouvements). Nous obtenons le théorème suivant,

Théorème 9.4. *Pour tout graphe de Weetman G , l'algorithme 10 explore tous les sommets et calcule une carte isomorphe à G en effectuant au plus $4 \times |V(G)|$ mouvements.*

Remarque 9.39. Pour tout graphe G possédant un revêtement universel de son complexe de cliques $\widehat{G}$ qui est un graphe de Weetman, l'agent explorera G en au plus $4 \times |V(\widehat{G})|$ mouvements.

Nous donnons maintenant une borne inférieure triviale. En effet, un agent devant visiter tous les sommets d'un graphe G , il devra se déplacer au moins $|V(G)| - 1$ fois pour tous les visités. Par exemple, cette complexité est atteinte si le graphe est un chemin.

Théorème 9.5. *Pour tout graphe de Weetman G , l'algorithme 10 explore tous les sommets de G en effectuant au moins $\theta(|V(G)|)$ mouvements.*

9.4.1 Exécution sur des graphes qui ne sont pas des graphes de Weetman

Il est intéressant d'observer que l'algorithme 10 exécuté dans un graphe arbitraire peut avoir différents comportements particuliers. Si le revêtement universel $\widehat{G}$ du complexe de cliques d'un graphe G est infini, alors à partir du corollaire 8.32 présenté dans le chapitre 8, l'agent n'arrêtera jamais son exécution (Figure 9.14). Sinon, plusieurs cas se présentent. Si le graphe G exploré à partir d'un sommet $v_0 \in V(G)$ satisfait les conditions $\text{INT}(v_0)$ et $\text{TRI}(v_0)$, alors l'algorithme explorera G efficacement. Notons que ce graphe peut ne pas être un graphe de Weetman s'il existe un sommet $v \in V(G)$ tel que les deux conditions ne sont pas respectées en v .

Si le graphe G exploré possède un revêtement universel $\widehat{G}$ qui est un graphe de Weetman *fini* alors l'agent ne détectera aucune erreur lors de son exécution mais la complexité de l'exploration dépendra de la taille de $\widehat{G}$ et non de G . Si le revêtement universel $\widehat{G}$ n'est pas un graphe de Weetman alors l'agent détectera une erreur et débutera une boucle infinie. L'idée derrière cette détection d'erreur est qu'un agent évoluant dans un graphe simplement connexe qui n'est pas

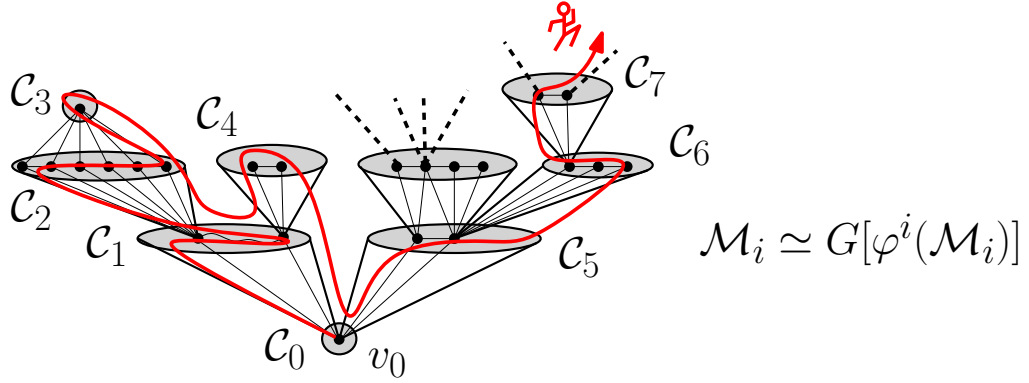


FIGURE 9.12 – Illustration de la preuve du théorème 9.3. Nous représentons les clusters d'un graphe de Weetman G et le parcours de l'agent dans ce graphe. Les clusters sont numérotés en fonction de l'étape où l'agent les a explorés. Comme nous pouvons le voir, les clusters sont explorés suivant un parcours en profondeur.

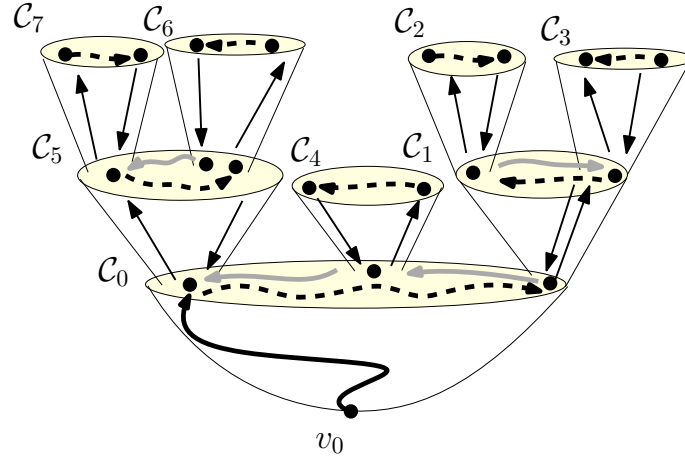


FIGURE 9.13 – Illustration de la preuve de la complexité de l'algorithme 10 exécuté dans un graphe de Weetman G avec un sommet base v_0 . Les clusters sont numérotés en fonction de leur ordre d'exploration. Le premier parcours d'un cluster est représenté par un trait pointillé noir. Un deuxième parcours, représenté par un trait gris plein, peut aussi être effectué si le cluster est adjacent à d'autres clusters non explorés. Les arêtes verticales traversées par l'agent sont représentées par les arcs fins.

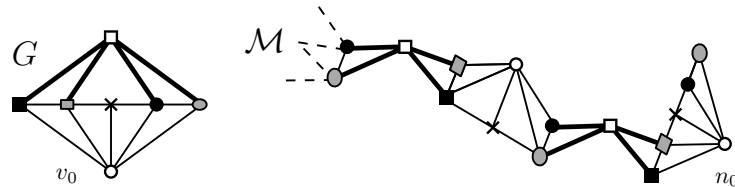


FIGURE 9.14 – Exécution (infinie) sur un graphe $G \in \mathcal{IC}$. Le graphe G ne respecte pas la condition de l'intervalle $\text{INT}(v_0)$. L'agent construisant une carte de $\hat{G}$ (qui est infini), si $\hat{G}$ est un graphe de Weetman, alors l'agent ne détectera aucune erreur mais ne terminera jamais son exécution. Si $\hat{G}$ n'est pas un graphe de Weetman et alors l'agent détectera une erreur à un moment donné de l'exécution.

un graphe de Weetman construira une carte contenant des sommets du graphe dupliqués, i.e., distincts dans la carte mais correspondant à un même sommet du graphe. Or, comme $\widehat{G}$ est simplement connexe, il n'existe aucun revêtement de $\widehat{G}$ non trivial (non isomorphe). Comme l'agent construit une carte qui n'est pas localement bijective avec le graphe exploré, l'agent détectera à un moment donné de l'exécution que sa carte ne correspond pas à ce qu'il observe en explorant le graphe. Les figures 9.15, 9.16 et 9.17 illustrent ce phénomène en montrant des exemples d'exécution où l'agent détecte une erreur.

Conclusion

Dans ce chapitre, nous montrons que des informations locales supplémentaires permettent à un agent d'explorer efficacement de plus grandes familles de graphes que les arbres. Notre principale contribution est d'avoir montré qu'il existe une famille de graphes, les graphes de Weetman, capable d'être explorée en un nombre linéaire de mouvements, et ce, sans information globale telle une borne sur la taille ou le diamètre du réseau. Cette famille contient de larges familles de graphes telles que les graphes triangulés, les graphes "bridged" ([FJ86, SC83]) ou les graphes de Johnson. Pour montrer ce résultat, nous avons proposé un algorithme d'exploration efficace des graphes de Weetman et avons montré sa correction. Cet algorithme construit une carte du réseau qui permet de distinguer les sommets explorés des sommets non explorés. Cette distinction permet à l'agent de ne pas se perdre et de ne pas visiter infiniment souvent les mêmes sommets du graphe. Notre algorithme s'appuie sur les deux conditions métriques simples caractérisant les graphes de Weetman. De plus, il explore tous les graphes possédant un revêtement universel de son complexe de cliques qui est un graphe de Weetman en un nombre de mouvements linéaire en la taille du revêtement universel.

Il n'est pas évident à première vue de concevoir un algorithme d'*exploration* efficace et spécifique à un graphe simplement connexe donné (au sens où l'algorithme est conçu pour un seul graphe mais ne doit pas se tromper s'il est exécuté sur d'autres graphes). En effet, la simple connexité du graphe bornant les rayons de quasi-revêtement, un tel algorithme pourrait exister mais, au vu de nos précédents résultats, cet algorithme ne sera pas forcément efficace.

Une autre question non résolue ici serait de déterminer si la composition d'algorithmes d'exploration efficaces est un algorithme d'exploration efficace pour l'union des familles explorées par les algorithmes composés.

De plus, nous ne savons pas si une caractérisation élégante des graphes explorables en temps linéaire existe. À l'inverse, les graphes appartenant à $\mathcal{FC}$ et qui ne sont pas simplement connexes ne peuvent pas être explorés en temps linéaire car seul leur revêtement universel déterminera la complexité de leur exploration. Peuvent-ils être explorés en temps polynomial ? Une autre famille de graphes qui pourrait, à première vue, être explorée en temps polynomial est la famille des graphes δ -hyperboliques. Notons que pour explorer ces graphes, nous devons sûrement doter l'agent de jumelles dont la portée dépend de l'hyperbolicité du graphe. Ces graphes possèdent des propriétés topologiques bornant les distances inter sommets. Ces dernières pistes ne sont pas triviales dans le sens où un agent pourrait devoir réparer les erreurs détectées dans la carte "durant l'exécution".

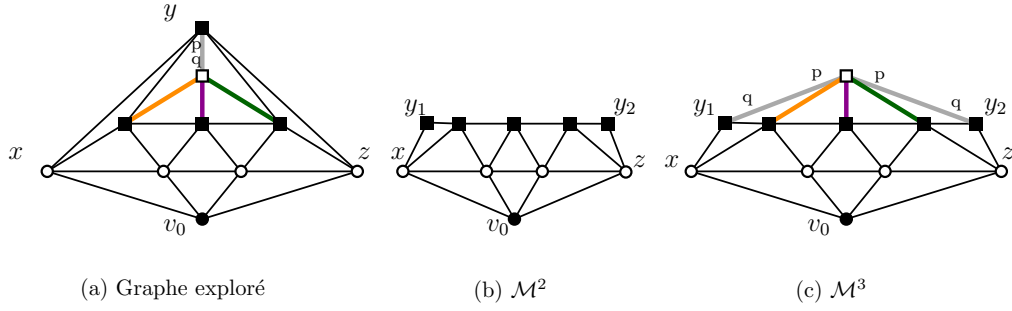


FIGURE 9.15 – Illustration d'un graphe $G \in \mathcal{SC} \setminus \text{Weetman}$ en (a) ne satisfaisant pas INT (v_0) entre x , y et z . Les ronds blancs (resp. carrés noirs et carrés blancs) sont les sommets à distance 1 (resp 2 et 3) de v_0 . En (b) et (c), la carte construite par l'agent lors des étapes 2 et 3 explorant G à partir de v_0 . Lors de la phase 3, l'agent détectera une erreur car il existe deux arêtes reliées à un même sommet et possédant un même étiquetage des ports. De plus, l'agent pourra détecter l'erreur en comparant l'étiquette de jumelles du sommet "carré blanc" à ce qu'il a calculé dans sa carte.

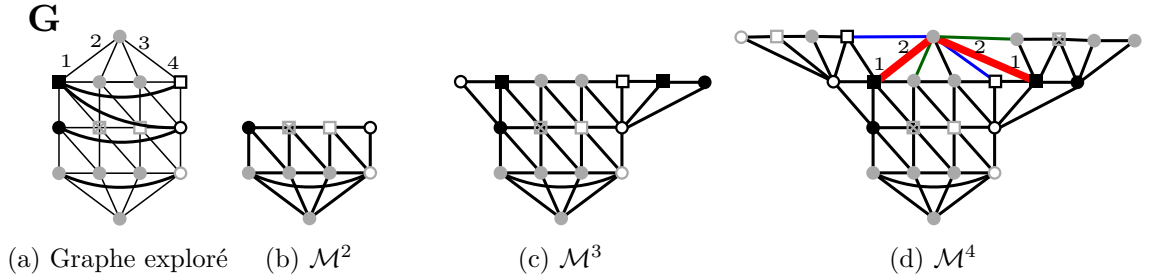


FIGURE 9.16 – Exemple d'exécution sur un graphe $G \in \mathcal{SC} \setminus \text{Weetman}$, i.e., G possède un complexe de cliques simplement connexe et possède une arête (reliant le rond blanc au rond noir) ne respectant pas la condition TRI(v_0) avec v_0 , le sommet du bas. Ici, l'agent détectera un conflit lors de la phase 4 (figure (d)) suite à l'ajout de la deuxième arête en gras. L'erreur est détectée car le sommet gris est relié à deux arêtes étiquetées de la même manière. Notons que les arêtes reliant les carrés blancs (resp. carrés noirs et ronds gris) au sommet gris du haut sont aussi dupliquées dans cet exemple.

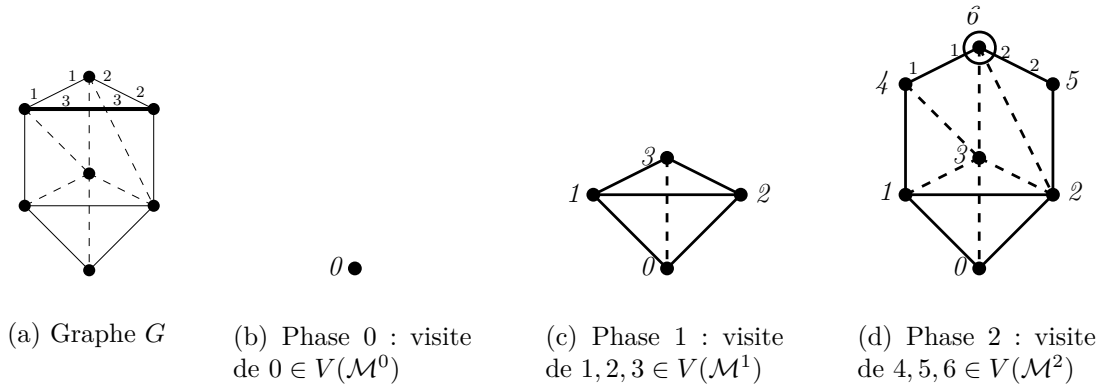


FIGURE 9.17 – Détection d'une erreur (Ligne 25 de l'algorithme) suite à l'exploration d'un graphe ne satisfaisant pas la condition TRI (v_0) (arête en gras dans la figure (a)). Ici, lors de la visite du sommet 6 durant la phase 2 de l'exécution, l'agent détectera une erreur car il existe $(231, 132) \in \mathcal{B}[6]$ avec $\mathcal{M}[6][2] = 5$, $\mathcal{M}[6][1] = 4$, et $45 \notin E(\mathcal{M}^2)$.

Conclusion générale et Perspectives

Dans cette thèse, nous avons illustré l'impact des connaissances sur la calculabilité distribuée de deux modèles distribués : le modèle des participants inconnus et les agents mobiles. L'approche utilisée pour étudier ces modèles est inspirée de l'approche dite algébrique. Cette approche a été notre guide et nous a permis d'obtenir des résultats généraux de calculabilité dans des modèles "exotiques", i.e., éloignés des modèles classiques. À première vue, les outils utilisés classiquement avec cette approche (les morphismes de graphes) n'apparaissent pas adaptés pour étudier ces modèles. Nous avons donc identifié les nouveaux outils algébriques adaptés à chaque modèle. Puis, avec une même méthodologie basée sur cette approche, nous avons obtenu une caractérisation de la calculabilité distribuée de ces modèles.

Dans la première partie, nous avons caractérisé, pour le modèle des participants inconnus, les connaissances nécessaires et suffisantes permettant de résoudre certains problèmes fondamentaux comme le problème de l'élection, de cartographie du réseau ou de la k -élection. Nos preuves sont constructives et pour chacun des problèmes étudiés, un algorithme est donné résolvant le problème en donnant les conditions adéquates aux processus. Nous avons ensuite étendu le modèle aux réseaux anonymes. Dans le modèle des participants inconnus anonymes, nous avons étudié le problème de l'élection et nous avons caractérisé les connaissances nécessaires à sa résolution. Puis nous avons proposé un algorithme pour le problème de l'élection dans des familles de graphes de taille bornée utilisant de telles connaissances et une borne sur la taille des graphes. Ce résultat combine les résultats obtenus dans le modèle avec identité et les résultats obtenus concernant l'élection dans les réseaux anonymes à passage de messages.

Nous avons ensuite étudié l'impact des connaissances locales dans le modèle à agent mobile. Pour cela, nous avons augmenté la perception locale d'un agent mobile en le dotant de jumelles. Ces jumelles permettent à l'agent de distinguer le graphe induit par les sommets à distance 1 de sa position dans le réseau. Nous montrons que cette augmentation du modèle augmente la calculabilité de l'exploration en caractérisant la famille maximale de graphes explorables dans ce modèle. Pour rappel, sans jumelles, seuls les arbres peuvent être explorés avec arrêt. Avec Jumelles, les graphes triangulés sont explorables par exemple et en toute généralité, tout graphe appartenant à $\mathcal{FC}$ est explorable. Nous avons montré que, malgré le gain de calculabilité obtenu grâce aux jumelles, la complexité de tout algorithme universel d'exploration dans ce modèle croît plus vite que toute fonction calculable. Ce résultat est intéressant, car il lie la calculabilité des machines de Turing et la calculabilité distribuée. À notre connaissance, seul [FGKS13] ont présenté des résultats de ce type.

Nous clôturons cette étude en montrant que les jumelles peuvent quand même aider à explorer de larges familles de graphes efficacement. Pour cela, nous avons proposé un algorithme d'exploration des graphes de Weetman qui les explore en un nombre de mouvements linéaire en la taille du graphe exploré. Cette famille contient, entre autres, les graphes triangulés, les graphes de Johnson, ...

Les connaissances apportées aux processus sont donc déterminantes en algorithmique distri-

buée, aussi bien d'un point de vue calculabilité que complexité.

Les perspectives des travaux concernant les participants inconnus sont de généraliser les résultats obtenus à toute tâche distribuée dans ce modèle. Le but serait de proposer une condition, liant les connaissances données aux processus à un problème distribué, nécessaire et suffisante pour résoudre le problème en dotant les processus de la connaissance. Si possible, il serait intéressant de concevoir un méta-algorithme qui, pour tout couple problème et connaissance satisfaisant notre condition, résout le problème si les processus sont dotés de la connaissance. Un tel algorithme devra donc prendre en entrée, une connaissance et la spécification $\mathcal{S}$ du problème à résoudre.

Les perspectives des travaux concernant les agents mobiles sont de résoudre d'autres problèmes grâce aux jumelles : rendez-vous entre plusieurs agents, exploration collaborative, captures de zones, ... De plus il serait intéressant d'étendre la portée des jumelles et, avec la même approche algébrique, caractériser la calculabilité des agents mobiles dotés de jumelles à distance k . Une question reste en suspend concernant les familles maximales de graphes explorables en temps linéaire et/ou polynomial : la famille des graphes de Weetman est-elle maximale pour l'exploration efficace ? Quelles sont les familles de graphes explorables en temps polynomial avec des jumelles ? Il serait aussi intéressant de voir si cette approche peut être étendue aux réseaux dynamiques ou tout autre modèle modélisant certaines pannes transitoires. Par exemple, que se passe t'il pour un agent mobile doté de jumelles évoluant dans des réseaux où les arêtes apparaissent et/ou disparaissent de manière sporadique. Il faudrait alors repenser complètement les outils algébriques considérés pour appliquer la même approche dans ces modèles.

Bibliographie

- [ABFG08] E. A. P. Alchieri, A. N. Bessani, J. Fraga, and F. Greve. Byzantine consensus with unknown participants. In *OPODIS*, page 22–40, 2008.
- [ABGdSF16] E. A. Pelinson Alchieri, A. Bessani, F. Greve, and J. da Silva Fraga. Knowledge connectivity requirements for solving byzantine consensus with unknown participants. *IEEE Transactions on Dependable and Secure Computing*, PP(99) :1–1, 2016.
- [AGP⁺11a] C. Ambühl, L. Gąsieniec, A. Pelc, T. Radzik, and X. Zhang. Tree exploration with logarithmic memory. *ACM Trans. Algorithms*, 7(2) :17 :1–17 :21, March 2011.
- [AGP⁺11b] C. Ambühl, L. Gąsieniec, A. Pelc, T. Radzik, and X. Zhang. Tree exploration with logarithmic memory. *ACM Transactions on Algorithms*, 7(2) :17 :1–17 :21, 2011.
- [AH00] S. Albers and M. R. Henzinger. Exploring unknown environments. *SIAM J. Comput.*, 29(4) :1164–1188, February 2000.
- [AKL⁺79] R. Aleliunas, R. M. Karp, R.J. Lipton, L. Lovász, and C. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *FOCS 1979*, pages 218–223, 1979.
- [Ang80] D. Angluin. Local and global properties in networks of processors. In *Proceedings of the 12th Symposium on Theory of Computing*, pages 82–93, 1980.
- [BA06] Bender and Al. The power of a pebble : Exploring and mapping directed graphs, 2006.
- [BCN89] E. Brouwer, C. Cohen, and A. Neumaier. *Distance-Regular Graphs*. Springer-Verlag Berlin Heidelberg, 1989.
- [BH99] M. Bridson and A. Haefliger. *Metric Spaces of Non-positive Curvature*. Springer-Verlag, 1999.
- [BLMS90] M. Billaud, P. Lafon, Y. Metivier, and E. Sopena. *Graph rewriting systems with priorities*, pages 94–106. Springer Berlin Heidelberg, Berlin, Heidelberg, 1990.
- [BS94] M. A. Bender and D. K. Slonim. The power of team exploration : two robots can learn unlabeled directed graphs. In *Proc. of the 35rd Annual Symposium on Foundations of Computer Science*, pages 75–85. IEEE Computer Society Press, Los Alamitos, CA, 1994.
- [BV99] P. Boldi and S. Vigna. Computing anonymously with arbitrary knowledge. In *PODC*, pages 181–188, 1999.
- [BV01] P. Boldi and S. Vigna. An effective characterization of computability in anonymous networks. In *DISC*, 2001.

- [CDK10a] J. Chalopin, S. Das, and A. Kosowski. *Constructing a Map of an Anonymous Graph : Applications of Universal Sequences*, pages 119–134. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010.
- [CDK10b] J. Chalopin, S. Das, and A. Kosowski. *Constructing a Map of an Anonymous Graph : Applications of Universal Sequences*, pages 119–134. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.
- [CDW13] J. Chalopin, S. Das, and P. Widmayer. *Deterministic Symmetric Rendezvous in Arbitrary Graphs : Overcoming Anonymity, Failures and Uncertainty*, pages 175–195. Springer New York, Jan 2013.
- [CFI⁺05] R. Cohen, P. Fraigniaud, D. Ilcinkas, A. Korman, and D. Peleg. Label-guided graph exploration by a finite automaton. In *ICALP 2005*, volume 3580 of *LNCS*, pages 335–346, 2005.
- [CGM08] J. Chalopin, E. Godard, and Y. Métivier. Local terminations and distributed computability in anonymous networks. In *DISC*, 2008.
- [CGM12] J. Chalopin, E. Godard, and Y. Métivier. Election in partially anonymous networks with arbitrary knowledge in message passing systems. *Distributed Computing*, 25(4) :297–311, 2012.
- [CGMO06] J. Chalopin, E. Godard, Y. Métivier, and R. Ossamy. *Mobile Agent Algorithms Versus Message Passing Algorithms*, pages 187–201. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.
- [CGN14] J. Chalopin, E. Godard, and A. Naudin. What do we need to know to elect in networks with unknown participants ? In *SIROCCO*, volume 8576 of *LNCS*, pages 279–294. Springer, 2014.
- [CGN15a] J. Chalopin, E. Godard, and A. Naudin. Anonymous graph exploration with binoculars. Technical report, LIF, Université Aix-Marseille and CNRS, 2015. submitted.
- [CGN15b] J. Chalopin, E. Godard, and A. Naudin. Exploration de graphes anonymes avec des jumelles. In *ALGOTEL 2015 - 17èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, Beaune, France, June 2015.
- [CGN16a] J. Chalopin, E. Godard, and A. Naudin. Using binoculars for fast exploration and map construction in chordal graphs and extensions, 2016. <http://arxiv.org/abs/1604.05915>.
- [CGN16b] J. Chalopin, E. Godard, and A. Naudin. Utiliser des jumelles pour explorer rapidement les graphes triangulés. In *ALGOTEL 2016 - 18èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, Bayonne, France, May 2016.
- [Cha06] J. Chalopin. *Algorithmique distribuée, calculs locaux et homomorphismes de graphes*. PhD thesis, Université Bordeaux 1, 2006.
- [CM07] J. Chalopin and Y. Métivier. An efficient message passing election algorithm based on mazurkiewicz’s algorithm. *Fundam. Inf.*, 80(1-3) :221–246, January 2007.
- [CPR12] J. Czyzowicz, A. Pelc, and M. Roy. *Tree Exploration by a Swarm of Mobile Agents*, pages 121–134. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
- [CSS77] D. Cavin, Y. Sasson, and A. Schiper. Distributed systems, towards a formal approach. *Information Processing*, 1977.

- [CSS04] D. Cavin, Y. Sasson, and A. Schiper. Consensus with unknown participants or fundamental self-organization. *Proc. of 3rd ADHOC-NOW*, July 2004.
- [Das13] S. Das. Mobile agents in distributed computing : Network exploration. *Bulletin of the EATCS*, 109 :54–69, Aug 2013.
- [DFKP04] K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc. Tree exploration with little memory. *J. Algorithms*, 51(1) :38–63, 2004.
- [DFSY07] Shantanu Das, Paola Flocchini, Nicola Santoro, and Masafumi Yamashita. Fault-tolerant simulation of message-passing algorithms by mobile agents. In *Proceedings of the 14th International Conference on Structural Information and Communication Complexity*, SIROCCO’07, pages 289–303, Berlin, Heidelberg, 2007. Springer-Verlag.
- [DP99] X. Deng and C. H. Papadimitriou. Exploring an unknown graph. *Journal of Graph Theory*, 32(3) :265–297, 1999.
- [FBDM97] M. Fukuda, L. F. Bic, M. B. Dillencourt, and F. Merchant. Messages versus messengers in distributed programming. In *Proceedings of the 17th International Conference on Distributed Computing Systems*, ICDCS ’97, pages 347–, Washington, DC, USA, 1997. IEEE Computer Society.
- [FGKS13] P. Fraigniaud, M. Göös, A. Korman, and J. Suomela. What can be decided locally without identifiers ? In *PODC’13 - ACM Symposium on Principles of Distributed Computing*, pages 157–165, Montreal, Canada, July 2013. ACM. 1 + 15 pages, 3 figures.
- [FIP⁺05] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science*, 345 :331 – 344, 2005. Mathematical Foundations of Computer Science 2004 Mathematical Foundations of Computer Science 2004.
- [FJ86] M. Farber and R. E. Jamison. Convexity in graphs and hypergraphs. *SIAM J. Algebraic Discrete Methods*, 7 :433–444, 1986.
- [FRS03] P. Flocchini, A. Roncato, and N. Santoro. Computing on anonymous networks with sense of direction. *Theoretical Computer Science*, 301(1) :355 – 379, 2003.
- [GM02] E. Godard and Y. Métivier. A characterization of families of graphs in which election is possible. In *Foundations of Software Science and Computation Structures*, volume 2303 of *Lecture notes in computer science*, pages 159–171. Springer-Verlag, 2002.
- [GMM00] E. Godard, Y. Métivier, and A. Muscholl. The power of local computations in graphs with initial knowledge. In *Proc. of Theory and Applications of Graph Transformations’98*, number 1764 in LNCS, pages 71–84. Springer, 2000.
- [God02] E. Godard. *Réécritures de Graphes et Algorithmique Distribuée*. PhD thesis, Université Bordeaux I, 351 cours de la Libération, F-33405 Talence Cedex, France, 2002.
- [God15] E. Godard. *Calculabilité distribuée dans les réseaux de communication*. Habilitation à diriger des recherches, Aix Marseille Université, Laboratoire d’informatique fondamentale, 2015.
- [GP13] S. Guilbault and A. Pelc. Gathering asynchronous oblivious agents with local vision in regular bipartite graphs. *Theor. Comput. Sci.*, 509 :86–96, 2013.

- [GR01] C. Godsil and G. F. Royle. *Algebraic Graph Theory*. Springer-Verlag New York, 2001.
- [GR08] L. Gąsieniec and T. Radzik. Memory efficient anonymous graph exploration. In *Graph-Theoretic Concepts in Computer Science, WG 2008*, volume 5344 of *LNCS*, pages 14–29, 2008.
- [GSAS12] F. Greve, P. Sens, L. Arantes, and V. Simon. Eventually strong failure detector with unknown membership. *The Computer Journal*, 55(12) :1507–1524, 2012.
- [GT07] F. Greve and S. Tixeuil. Knowledge connectivity vs. synchrony requirements for fault-tolerant agreement in unknown networks. In *Dependable Systems and Networks 2007*, pages 82–91, 2007.
- [Hak73] W Haken. Connections between topological and group theoretical decision problems. In *Word Problems Decision Problems and the Burnside Problem in Group Theory*, volume 71 of *Studies in Logic and the Foundations of Mathematics*, pages 427–441. North-Holland, 1973.
- [Hat02] A. Hatcher. *Algebraic topology*. Cambridge University Press, 2002.
- [HKR14] Maurice Herlihy, Dmitry Kozlov, and Sergio Rajsbaum. Distributed computing through combinatorial topology. In *Distributed Computing Through Combinatorial Topology*. Morgan Kaufmann, Boston, 2014.
- [Ilc08] D. Ilcinkas. Setting port numbers for fast graph exploration. *Theor. Comput. Sci.*, 401(1–3) :236–242, 2008.
- [KKM10] E. Kranakis, D. Krizanc, and E. Markou. *The Mobile Agent Rendezvous Problem in the Ring*. Synthesis lectures on distributed computing theory. Morgan & Claypool Publishers, 2010.
- [Kou02] M. Koucký. Universal traversal sequences with backtracking. *J. Comput. Syst. Sci.*, 65(4) :717–726, 2002.
- [LO99] D. B. Lange and M. Oshima. Seven good reasons for mobile agents. *Commun. ACM*, 42(3) :88–89, March 1999.
- [LS77] R.C. Lyndon and P.E. Schupp. *Combinatorial Group Theory*. Ergebnisse der Mathematik und ihrer Grenzgebiete. Springer-Verlag, 1977.
- [Maz97] A. Mazurkiewicz. Distributed enumeration. *Inf. Processing Letters*, 61 :233–239, 1997.
- [MMW97] Y. Métivier, A. Muscholl, and P. A. Wacrenier. About the local detection of termination of local computations in graphs. In *SIROCCO*, 1997.
- [PP00] P. Panaite and A. Pelc. Impact of topographic information on graph exploration efficiency. *Networks*, 36(2) :96–103, 2000.
- [Rei32] K. Reidemeister. Einführung in die kombinatorische topologie. Vieweg, Brunzwick, 1932.
- [Rei08] O. Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4), 2008.
- [Ros00] K. H. Rosen. *Handbook of Discrete and Combinatorial Mathematics*. Chapman & Hall/CRC, 2000.
- [SC83] V.P. Soltan and V.D. Chepoi. Conditions for invariance of set diameters under d-convexification in a graph. *Cybernetics*, 19 :750–756, 1983.

- [Sha52] C. E. Shannon. Presentation of a maze solving machine. *Shannon : Collected Papers*, PC-03319, Mar 1952.
- [SSP85] B. Szymanski, Y. Shy, and N. Prywes. Terminating iterative solutions of simultaneous equations in distributed message passing systems. In *Proc. of the 4th Symposium of Distributed Computing*, pages 287–292, 1985.
- [Tel00] G. Tel. *Introduction to distributed algorithms*. Cambridge UP, 2000.
- [Wee94] G. M. Weetman. A construction of locally homogeneous graphs. *Journal of the London Mathematical Society*, 50(1) :68–86, 1994.
- [YK96a] M. Yamashita and T. Kameda. Computing on anonymous networks. I. characterizing the solvable cases. *IEEE Transactions on Parallel and Distributed Systems*, 7 :69–89, Jan 1996.
- [YK96b] M. Yamashita and T. Kameda. Computing on anonymous networks. II. decision and membership problems. *IEEE Transactions on Parallel and Distributed Systems*, 7 :90–96, Jan 1996.

Résumé

Dans cette thèse, nous étudions l'impact des connaissances sur la calculabilité distribuée de problèmes au sein des réseaux distribués. Dans un système distribué, les entités de calcul ne possèdent que des informations partielles et locales pour résoudre des tâches globales. Néanmoins, ces informations partielles ne permettent pas de résoudre toutes les tâches et, dans ce cas, il sera nécessaire de donner plus de connaissances (locales ou globales) aux entités. Dans une première partie, nous caractérisons les connaissances nécessaires et suffisantes permettant de résoudre des problèmes tels la cartographie, l'élection et la k -élection dans un modèle à passage de messages particulier : le modèle des participants inconnus. Ce modèle est une extension naturelle du modèle à passage de messages permettant de modéliser un certain dynamisme des réseaux. Nos preuves sont constructives et pour chacun des problèmes étudiés, une condition caractérisant les connaissances nécessaires et suffisantes est fournie et un algorithme utilisant toute connaissance satisfaisant notre condition est proposé (et montré correct). Nous étendons ensuite le modèle aux graphes anonymes et, avec la même méthodologie, nous présentons une condition nécessaire sur la connaissance à fournir aux processus pour résoudre le problème de l'élection et nous proposons un algorithme utilisant une telle connaissance et une borne sur la taille du réseau. Dans la seconde partie, nous étudions l'impact des connaissances locales sur la calculabilité du problème de l'exploration de graphes anonymes avec arrêt. Pour cela, nous introduisons un nouveau modèle d'agent mobile doté d'un capteur spécial, nommé jumelles, lui permettant de percevoir le graphe induit par les sommets adjacents à sa position dans le réseau. Dans ce modèle, nous caractérisons exactement les graphes explorables sans connaissance globale et nous proposons un algorithme les explorant. Cette connaissance locale apportée à un coût car la complexité de tout algorithme d'exploration pour ces graphes croît plus vite que toute fonction calculable. Pour finir cette étude, nous montrons que de larges familles de graphes peuvent être explorées en un nombre de mouvements linéaire en la taille du graphe exploré. Nous proposons un algorithme qui, grâce aux jumelles, explore efficacement les graphes de Weetman contenant, entre autres, les graphes triangulés, les graphes de Johnson et certaines triangulations planaires.

Abstract

In this study, we show how knowledge impacts the computability in distributed systems. First, we characterize what we need to know to elect in the unknown participant model. This model is a natural extension for the message passing model that formalises dynamicity that occurs in some networks. We give a necessary and sufficient condition on the knowledge needed to solve the following fundamental problems : map construction, leader election and k -leader election. For each of them, we provide an algorithm solving the problem using any knowledge satisfying our condition. Then, we extend the model to anonymous networks. We characterize, with the same methodology, the knowledge needed to solve an election in this model and we provide an algorithm using such a knowledge and a bound on the size of the network. In the second part, we study the impact of local knowledge on the computability of the anonymous graph exploration problem. We introduce a new model of mobile agents where an agent is endowed with binoculars, a local sensor permitting to perceive the graph induced by the vertices adjacent to its location. In this model, we characterize the graphs that can be explored by a single mobile agent without any global information and we provide an algorithm exploring all of them. Unfortunately, universal algorithm has a cost : The number of moves required by such an algorithm cannot be bounded by a computable function. Finally, we prove that large classes of graphs like chordal graphs, Johnson graphs, ... can be explored in a linear number of moves using binoculars by providing an exploration algorithm for the family of Weetman graphs.