Analyses et preuves formelles d'algorithmes distribués probabilistes

Allyx Fontaine

université de BORDEAUX

Thèse présentée
pour obtenir le grade de

Docteur de l'Université de Bordeaux

École Doctorale de Mathématiques et d'Informatique
Spécialité Informatique

par

Allyx Fontaine

Analyses et Preuves Formelles d'Algorithmes Distribués Probabilistes

Sous la direction de : Akka Zemmari
(co-directeur : Pierre Castéran)

Soutenue le 16/06/2014

Membres du jury :

M. Pierre Castéran	Maître de Conférences (LaBRI)	Co-Directeur
M. Jérémie Chalopin	Chargé de Recherche (LIF)	Examinateur
M. Dominique Méry	Professeur (LORIA)	Rapporteur
M. Mohamed Mosbah	Professeur (LaBRI)	Président
M. Vlady Ravelomanana	Professeur (LIAFA)	Rapporteur
M. Akka Zemmari	Maître de Conférences HDR (LaBRI)	Directeur

Titre Analyses et Preuves Formelles d'Algorithmes Distribués Probabilistes

Résumé L'intérêt porté aux algorithmes probabilistes est, entre autres, dû à leur simplicité. Cependant, leur analyse peut devenir très complexe et ce particulièrement dans le domaine du distribué. Nous mettons en évidence des algorithmes, optimaux en terme de complexité en bits résolvant les problèmes du MIS et du couplage maximal dans les anneaux, qui suivent le même schéma. Nous élaborons une méthode qui unifie les résultats de bornes inférieures pour la complexité en bits pour les problèmes du MIS, du couplage maximal et de la coloration. La complexité de ces analyses pouvant facilement mener à l'erreur et l'existence de nombreux modèles dépendant d'hypothèses implicites nous ont motivés à modéliser de façon formelle les algorithmes distribués probabilistes correspondant à notre modèle (par passage de messages, anonyme et synchrone), en vue de prouver formellement des propriétés relatives à leur analyse. Pour cela, nous développons une bibliothèque, *RDA*, basée sur l'assistant de preuve *Coq*.

Mots-clés Algorithme distribué, Algorithme probabiliste, Analyse, Preuve formelle, Assistant de preuve

Title Analyses and Formal Proofs of Randomised Distributed Algorithms

Abstract Probabilistic algorithms are simple to formulate. However, their analysis can become very complex, especially in the field of distributed computing. We present algorithms - optimal in terms of bit complexity and solving the problems of MIS and maximal matching in rings - that follow the same scheme. We develop a method that unifies the bit complexity lower bound results to solve MIS, maximal matching and coloration problems. The complexity of these analyses, which can easily lead to errors, together with the existence of many models depending on implicit assumptions motivated us to formally model the probabilistic distributed algorithms corresponding to our model (message passing, anonymous and synchronous). Our aim is to formally prove the properties related to their analysis. For this purpose, we develop a library, called *RDA*, based on the *Coq* proof assistant.

Keywords Distributed algorithm, Randomised algorithm, Analysis, Formal method, Proof assistant

Remerciements

Je souhaite tout d'abord exprimer mes remerciements à mon directeur de thèse, Akka Zemmari, tant pour son apport scientifique que pour son soutien moral. J'ai pu apprendre beaucoup auprès de lui et je lui suis très reconnaissante de m'avoir guidée, encouragée et conseillée.

Je remercie mon co-encadrant de thèse, Pierre Castéran, qui a guidé mes travaux grâce à son expertise en Coq. Par ses suggestions, il a contribué à en améliorer la qualité.

Je tiens à remercier les rapporteurs de cette thèse, Messieurs les Professeurs Dominique Méry et Vlady Ravelomanana, pour l'intérêt qu'ils y ont porté et les remarques qui m'ont permis de l'amender. J'adresse aussi mes remerciements à Messieurs Jérémie Chalopin et Mohamed Mosbah qui ont accepté d'examiner mon travail.

Un grand merci à Yves Métivier qui m'a beaucoup apporté tant intellectuellement qu'humainement au travers de nos discussions. Toujours de bon conseil, par ses commentaires et relectures, il a pris une grande part à l'aboutissement de cette thèse.

Je remercie Assia Mahboubi, Christine Paulin-Mohring, ainsi que Mike Robson avec qui j'ai pu travailler. Ils ont su m'apporter les éclaircissements dont j'avais besoin.

J'adresse de chaleureux remerciements à mes enseignants de l'ENSEIRB, qui sont devenus mes collègues et qui m'ont encouragée dans la voie de la recherche. J'ai été intégrée dans l'équipe pédagogique où j'ai pris plaisir à enseigner. Je pense notamment à Toufik Ahmed, Frédéric Herbreteau, Denis Lapoire, Mohamed Mosbah, David Renault, Antoine Rollet et Aymeric Vincent.

Je n'oublie pas les membres du laboratoire : les permanents, le personnel administratif et les doctorants. Je remercie Vincent Filou et Mohamed Tounsi pour les échanges d'idées. Je pense aussi à Razanne Issa, Ève Garnaud, Noémie-Fleur Sandillon-Rezer et Maurin Nadal avec qui j'ai commencé l'aventure en Master, ainsi qu'à Jérôme Charton, Florent Berthaut, Simon Naulin, Ludovic Paulhac, Pascal Desbarats, Marie Beurton-Aimar que j'ai rencontrés par la suite.

J'adresse mes derniers remerciements à toute ma famille, mes frères et mes parents. D'aussi loin que je me souvienne, j'ai voulu m'orienter vers l'enseignement et la recherche. Ma reconnaissance va à mes parents pour m'avoir confortée dans cette voie et manifesté un soutien sans faille. Je leur dédie cette thèse.

Table des matières

Introduction

Intérêts

Motivations

Les algorithmes distribués ont reçu une attention considérable et ont été étudiés intensivement ces dernières années. De nombreux travaux ont été effectués pour étudier leur puissance de calcul et/ou pour concevoir des solutions efficaces. Tel [75] donne différents avantages, de façon non exhaustive, de l'utilisation des systèmes distribués :

- échange d'information : l'émergence des réseaux intranets et extranets, le télétravail, rendent l'échange de données primordial dans les entreprises ou les organisations ;
- partage de ressources : de plus en plus, il est possible d'équiper tous les employés avec des ordinateurs, cela reste moins nécessaire pour les périphériques comme les imprimantes ou les stockages de sauvegarde ;
- augmentation de la fiabilité grâce à la réplication : si certains processus du système tombent en panne, cela peut ne pas avoir d'impact sur d'autres processus qui pourraient prendre en charge la tâche des processus mis en échec ;
- augmentation des performances via la parallélisation : la présence de plusieurs processus permet de faire des tâches en parallèle ;
- simplification de la conception : les algorithmes distribués sont souvent énoncés simplement, cependant la réalisation de tâches en simultané présente des difficultés.

Point de Vue Probabiliste

Les algorithmes distribués probabilistes ont été introduits pour garantir une meilleure efficacité ou pour résoudre des problèmes qui n'ont pas de solutions déterministes.

Si tous les processus sont indistinguables, les problèmes de l'algorithmique distribuée deviennent symétriques et les résoudre dans des réseaux d'un point de vue global devient plus difficile (voire impossible). Angluin [4] a étudié le problème de l'élection dans un réseau anonyme.

Un algorithme distribué résout le problème de l'élection s'il termine toujours et si dans sa configuration finale, exactement un processus est marqué *élu* et tous les autres processus sont *non-élus*. De plus, il est supposé que lorsqu'un sommet prend l'état *élu* ou *non-élu*, il reste dans cet état jusqu'à la fin de l'exécution de l'algorithme. Le problème de l'élection consiste à distinguer un unique sommet dans le graphe, qui est dans l'état *élu* alors que tous les autres sont dans l'état *non-élu*.

Les travaux d'Angluin montrent qu'il n'existe pas d'algorithme distribué qui, exécuté par chacun des processus, permet de résoudre ce problème dans un anneau de taille arbitraire en un temps fini. En effet, il apparaît qu'une symétrie peut être créée et ne peut être brisée sans impliquer un calcul infini ou un résultat erroné. C'est pourquoi on a recours aux algorithmes probabilistes. Même si les processus sont identiques, ils ont un générateur de nombres aléatoires, ce qui permet de casser des symétries dans le réseau.

Modèle Théorique des Algorithmes Distribués

Notre modèle est le modèle synchrone par passage de messages dont une présentation générale est faite par Tel [75] dans le chapitre 9 ou par Lynch [56] dans le chapitre 8.

Système Synchrone

Chaque processus p dans le réseau représente une entité qui est capable d'exécuter des pas de calcul, en envoyant et en recevant des messages par ses ports. Nous considérons des systèmes ***synchrones*** et un réveil synchrone des processus : les processus procèdent par rondes et commencent l'algorithme en même temps.

Remarque 1 *Cette hypothèse est forte mais reste valide et acceptable « dans la seule perspective d'analyser des algorithmes distribués ». (cf. [50]). Elle permet de mener des calculs de complexité qui donnent une borne inférieure pour des systèmes basés sur des hypothèses moins fortes (et donc plus proches de la réalité).*

Une action de chaque processus se réalise en une suite de pas discrets appelée ***ronde***. Dans une ronde, des processus envoient des messages (zéro ou plusieurs) à leurs voisins, reçoivent des messages (zéro ou plusieurs) de leurs voisins et effectuent un ensemble de calculs locaux. Les messages envoyés sont reçus pendant la même ronde, c'est-à-dire, si le processus p envoie un message au processus q à la $i^{\text{ème}}$ ronde, alors le message est reçu par q pendant la $i^{\text{ème}}$ ronde de q.

Système Anonyme

Le réseau est ***anonyme*** : des identités uniques ne sont pas disponibles pour différencier les processus lors de l'exécution. Nous supposons que les processus n'ont pas accès à des connaissances globales du réseau comme la taille ou une borne supérieure de la taille. Les processus n'ont pas besoin d'informations relatives à la position des sommets ou à leur distance dans le graphe. Cependant, un processus sait distinguer ses voisins.

Remarque 2 *L'hypothèse de l'anonymat est souvent considérée lorsque les processus ne peuvent divulguer leur identité pendant l'exécution, et ce pour des raisons liées à leur vie privée ou à des problèmes de sécurité. De plus, chaque processus peut être intégré dans un réseau à grande échelle ce qui rend difficile voire impossible la garantie de l'unicité des identifiants.*

Algorithme Distribué Probabiliste

Un ***algorithme distribué*** est une collection d'***algorithmes locaux*** (un associé à chaque processus). Un algorithme local est décrit par les actions qui se passent lors d'une ronde.

Un ***algorithme probabiliste*** est un algorithme qui fait des choix aléatoires basés sur une certaine distribution de probabilité.

Un ***algorithme distribué probabiliste*** est une collection d'algorithmes probabilistes locaux.

Remarque 3 *Comme notre réseau est anonyme, si deux processus ont le même degré, leurs algorithmes locaux probabilistes sont identiques et ont la même distribution.*

Par nature, les algorithmes probabilistes peuvent ne pas se terminer ou encore calculer un résultat incorrect. En général, on distingue deux catégories d'algorithmes probabilistes :

- un ***algorithme Las Vegas*** est un algorithme qui termine avec une probabilité strictement positive (en général égale à 1) et qui produit toujours, lorsqu'il termine, un résultat correct ;
- un ***algorithme Monte Carlo*** est un algorithme qui termine en un temps borné mais dont le résultat est incorrect avec une certaine probabilité.

Lorsqu'on parlera d'une propriété ***avec forte probabilité***, cela reviendra à dire avec probabilité $1 - o(n^{-1})$ où n est la taille du réseau.

Représentation du Système Distribué par un Graphe

Un système distribué est constitué de processeurs connectés à l'aide d'un réseau de communication. Les systèmes distribués sont caractérisés par le fait qu'il n'existe pas de composante centrale qui gère les autres : chaque composante est responsable de son propre fonctionnement.

Un système distribué est représenté par un graphe fini simple connexe $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ dont les sommets (l'ensemble V) correspondent aux processus et les arêtes (l'ensemble E) aux liens de communication.

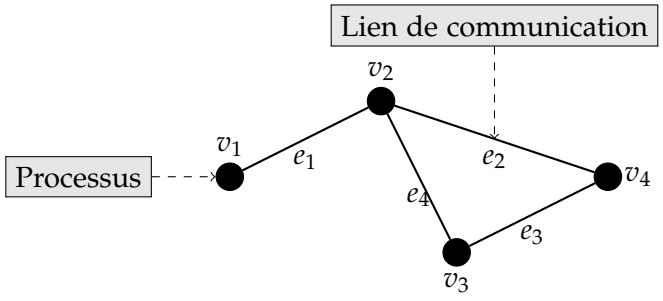


FIGURE 1 – *Représentation d'un réseau par un graphe.*

Le nombre de sommets est appelé taille du graphe et est noté n, et le nombre d'arêtes est noté m. L'arête qui lie deux sommets adjacents v et w est notée $\{v, w\}$. On note $deg(v)$ le degré d'un sommet v dans le graphe G, c'est-à-dire le nombre de sommets adjacents à v.

Chaque processus peut distinguer les différentes arêtes incidentes, *i.e.*, pour chaque sommet $u \in \mathsf{V}$, il existe une bijection entre les voisins de u

dans G et l'ensemble d'entiers compris entre 0 et $deg(u) - 1$. Les nombres associés par chaque sommet aux arêtes incidentes sont appelés ***numéros de ports***. Par la suite, la lettre δ sera utilisée pour désigner une numérotation de ports.

Communication par Passage de Messages

La communication entre les processus se fait par ***échange de messages*** : chaque processus envoie un message à son voisin en le déposant dans le lien correspondant à ce voisin. Les liens de communication sont *FIFO*, *i.e.*, pour chaque lien de communication, les messages sont délivrés dans l'ordre dans lequel ils ont été envoyés. Nous considérons seulement des systèmes fiables : aucune faute n'arrive que ce soit sur les processus ou sur les liens de communication.

Complexités

Complexité en Temps. La ***complexité en temps*** est le nombre maximal de rondes nécessaires pour que chaque processus finisse son calcul (cf. [67]).

Complexité en Bits. Une ronde d'un bit est une ronde telle que chaque processus peut envoyer/recevoir au plus 1 bit à/de chaque voisin. Kothapalli et al. [43] définissent la ***complexité en bits*** d'un algorithme $\mathcal{A}$ comme le nombre de rondes d'un bit nécessaire pour mener à terme l'algorithme.

Quelques Problèmes Classiques de l'Algorithmique Distribuée Probabiliste

Brisure de Symétrie

Lorsque nous constatons des symétries dans le graphe, résoudre de façon déterministe des problèmes devient plus difficile voire impossible. Nous nous intéressons plus particulièrement à la brisure de symétrie dans le graphe complet anonyme de deux sommets. Il apparaît qu'on ne peut distinguer les deux sommets de ce graphe car ils ont tous les deux un voisin et c'est un graphe anonyme. Le problème de la brisure de symétrie consiste à donner un identifiant différent à chacun des sommets. Un exemple est donné dans la Figure 2.

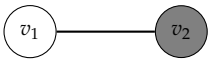


Figure 2 – *Exemple de brisure de symétrie dans le graphe complet de deux sommets. Les sommets se distinguent l'un de l'autre car ils ne sont pas dans le même état : il y en a un blanc et un gris.*

Rendez-vous

Dans un réseau de processeurs communiquant par échange de messages en mode synchrone, l'émetteur et le récepteur doivent être tous les deux prêts à communiquer. Un rendez-vous (ou une poignée de main) permet de réaliser des communications exclusives entre des paires de sommets voisins. Une solution à ce problème est une brique de base pour la mise en œuvre de systèmes de réécriture impliquant des règles sur les arêtes (voir [63]) ou encore, pour la réalisation (après un nombre d'itérations suffisant) d'un couplage maximal du graphe représentant le réseau. La Figure 3 donne l'exemple d'un graphe où se produisent deux rendez-vous simultanément.

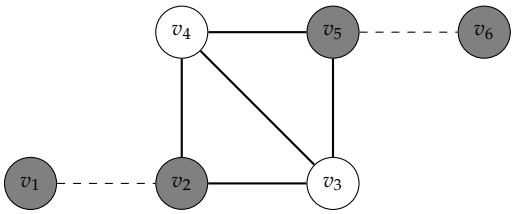


FIGURE 3 – *Exemple de rendez-vous. Nous pouvons observer deux rendez-vous : v_1 est en rendez-vous avec v_2 et v_5 est en rendez-vous avec v_6. Ces deux rendez-vous ont lieu simultanément, il aurait pu y en avoir moins ou plus.*

Couplage Maximal

Un ***couplage*** est un sous-ensemble M de E tel que tous les éléments de M n'ont pas d'extrémité commune deux à deux. Un couplage M est dit ***maximal*** si toute arête de G est soit dans M, soit a une extrémité commune avec une arête de M. La Figure 4 présente un exemple de couplage maximal.

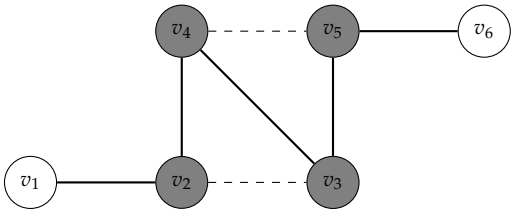


FIGURE 4 – *Exemple de couplage maximal. Le sommet v_2 est en rendez-vous avec le sommet v_3 et le sommet v_4 est en rendez-vous avec le sommet v_5.*

Remarque 4 *Le couplage de la Figure 4 n'est pas un couplage maximum. Un exemple de couplage maximum est : $\{\{v_1, v_2\}, \{v_3, v_4\}, \{v_5, v_6\}\}$. Calculer un couplage maximum est NP-difficile [32, problem GT10].*

Ensemble Indépendant Maximal

Un ***ensemble indépendant*** est un sous-ensemble I de V tel que tous les éléments de I sont deux à deux à une distance supérieure strictement à 1 (*i.e.*, non-adjacents). Un ensemble indépendant I est dit ***maximal*** (par la

suite nous emploierons l'abréviation ***MIS*** pour *Maximal Independent Set*) si tout sommet de G est soit dans I, soit adjacent à un sommet de I. La Figure 5 présente un exemple de MIS.

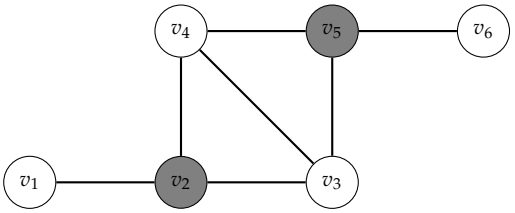


FIGURE 5 – *Exemple d'ensemble indépendant maximal. Les sommets v_2 et v_5 sont dans cet ensemble.*

Remarque 5 *L'ensemble indépendant de la Figure 5 n'est pas un ensemble indépendant maximum qui dans ce graphe est de cardinalité* 3*.*

Vérification des Preuves

L'intérêt porté aux algorithmes probabilistes est dû tant à leur simplicité qu'à leur efficacité ou la possibilité qu'ils offrent de résoudre certains problèmes qui ne peuvent pas l'être de façon déterministe. Cependant, leur analyse peut devenir très complexe et ce particulièrement dans le domaine du distribué. Lynch [55] répertorie de façon non-exhaustive des preuves d'impossibilité concernant les systèmes distribués et fait comme commentaire général que « l'utilisation des modèles formels force les gens à rendre leurs hypothèses explicites ». Ainsi, la complexité des preuves qui peuvent facilement mener à l'erreur et l'existence de nombreux modèles dépendant d'hypothèses implicites nous ont motivés à modéliser de façon formelle les algorithmes distribués probabilistes correspondant à notre modèle, en vue de prouver formellement des propriétés relatives à leur analyse. Pour cela, nous développons une bibliothèque, *RDA* [30], avec l'assistant de preuve *Coq* [57], la bibliothèque *Alea* [6] et l'extension *Ssreflect* [34].

L'Assistant de Preuve *Coq*

Un assistant de preuve est un logiciel permettant l'écriture et la vérification de preuves mathématiques, soit sur des théorèmes au sens usuel des mathématiques, soit sur des assertions relatives à l'exécution de programmes informatiques. L'écriture de preuves entièrement formelles est une activité fastidieuse car même les étapes considérées comme évidentes doivent être entièrement prouvées. De nombreux assistants de preuve existent (Isabelle[39], HOL [36], Rodin [71], PVS [68], etc), celui que nous avons choisi est *Coq*.

Coq [57, 13] est un assistant de preuve interactif. Il fournit un langage formel permettant d'écrire des définitions mathématiques, des algorithmes et des théorèmes. Il permet de manipuler des assertions mathématiques, de vérifier des preuves de ces assertions, d'aider à la recherche de preuves formelles et de déduire un programme certifié des preuves

constructives de spécifications. *Coq* est implémenté en Objective Caml. La version actuellement utilisée est la 8.4. *Coq* a été récompensé par le prix SIGPLAN1 Programming Languages Software Award en janvier 2014.

Coq est fondé sur le calcul des constructions inductives, d'où il tire l'origine de son nom : Coc, Calculus Of Constructions. La logique de *Coq* (théorie des types, induction, polymorphisme, types dépendants) est suffisamment expressive pour nous permettre de définir des algorithmes polymorphes et des classes d'algorithmes. Par exemple, des énoncés tels que « Il n'existe pas d'algorithmes de la classe $\mathcal{C}$ réalisant telle tâche » peuvent être énoncés et prouvés. D'ailleurs, de tels résultats d'impossibilité ont été démontrés en *Coq* dans la bibliothèque *Loco* par Castéran et Filou [18].

Coq permet d'opérer une approche ascendante. On commence par prouver des lemmes basiques pour ensuite les appliquer dans les preuves de lemmes ou de théorèmes plus élaborés. Tout ceci forme des outils pour la preuve. Lors de l'élaboration d'une preuve, *Coq* met en évidence les hypothèses et le (ou les) but(s) à atteindre. En se reposant sur les hypothèses et des outils préalablement prouvés, on traite le but à atteindre en utilisant des tactiques permettant de faire une étude de cas, d'automatiser des preuves, etc.

Ssreflect [34], pour *small scale reflection, ou réflexion à petite échelle*, est une extension de *Coq*. Elle apporte de nombreuses améliorations des fonctionnalités du système *Coq*, notamment concernant les scripts, la gestion des contextes de preuves, la réécriture, l'expansion des définitions et l'évaluation partielle. Elle fournit des tactiques, *i.e.*, des étapes de raisonnement, qui permettent la reformulation (ou *réflexion*) des problèmes sous une forme plus concrète. Par exemple, les prédicats peuvent être mis sous la forme d'une fonction qui calcule un booléen. *Ssreflect* est aussi une bibliothèque mathématique qui a permis de prouver le théorème des quatre couleurs[33].

Formalisation des Graphes

Notre modèle des systèmes distribués repose sur la théorie des graphes. *Coq* n'ayant pas de bibliothèque standard pour les graphes, nous nous sommes intéressés aux alternatives.

La contribution à *Coq* de Duprat [25] représente les ensembles de sommets et d'arêtes par des propositions pour lesquelles la relation d'appartenance n'est pas décidable. Robillard [70] permet les calculs sur les graphes qu'il définit à l'aide du système de module de *Coq* en vue d'extraire le code. La bibliothèque *Loco* de Castéran et al. [19] se rapproche de cette dernière mais en utilisant des classes de types [74]. La version actuelle de *Loco* vise à remplacer les modules de fonctions finies [26] de la bibliothèque standard de *Coq* par les représentations de Lescuyer [51]. La bibliothèque *Ssreflect* a été utilisée pour réaliser la formalisation du théorème des quatre couleurs.

Après avoir expérimenté la formalisation de *Loco* et *Ssreflect*, nous avons opté pour *Ssreflect*. En effet, les outils pour raisonner sur les graphes qui y sont développés ont facilité la rédaction des preuves qui ont pu être allégées via la réflexion.

Contributions et Organisation de la Thèse

Tout au long de ce mémoire, nous considérerons des systèmes distribués synchrones, anonymes et communiquant par messages. Si aucune précision n'est apportée, chaque système est représenté par un graphe simple et connexe. Nous développons l'aspect analyse et l'aspect preuve dans deux parties indépendantes.

Dans un premier temps, au Chapitre 1, nous rappelons des notions mathématiques, probabilistes et de la théorie des graphes qui nous serviront pour les parties suivantes.

Dans la première partie, nous nous intéressons à des solutions algorithmiques les plus optimales possibles en terme de complexité en temps.

Le Chapitre 2 donne un aperçu de l'état de l'art concernant les résultats sur la complexité en bits des algorithmes distribués résolvant les problèmes du MIS, du couplage maximal et de la coloration pour diverses familles de graphes.

Nous présentons, dans le Chapitre 3, des algorithmes distribués probabilistes optimaux en terme de complexité en bits qui résolvent les problèmes du MIS et du couplage maximal dans les anneaux anonymes.

Puis nous prouvons, dans le Chapitre 4, des bornes inférieures de la complexité en temps et en bits pour résoudre des problèmes classiques de l'algorithmique distribuée probabiliste qui sont : la coloration, le couplage maximal, l'ensemble indépendant maximal et des extensions de ces problèmes. Nous mettons en évidence une méthode qui unifie le schéma des preuves. Cette méthode nous a permis de prouver des résultats existants mais aussi de nouveaux résultats.

Les preuves mathématiques requises pour l'analyse des algorithmes mettent en évidence la difficulté due tant à l'aspect distribué qu'à l'aspect probabiliste. C'est pourquoi nous développons dans la deuxième partie une formalisation des algorithmes de notre modèle en nous intéressant tout d'abord à la preuve en *Coq* d'algorithmes simples. Le développement est basé sur les critères suivants :

- une interface avec des possibilités de calcul sur des exemples et contre-exemples ;
- des énoncés lisibles et acceptés par un utilisateur non spécialiste de *Coq* ;
- des lemmes généraux aidant à l'analyse.

Le Chapitre 5 donne un aperçu de l'état de l'art. Nous détaillons ensuite les fonctionnalités de la bibliothèque *Loco*. Nous poursuivons par des rappels sur les monades et la bibliothèque *Alea*, permettant de raisonner en *Coq* sur les algorithmes probabilistes, sur laquelle se base notre développement. Enfin, nous faisons l'énumération des résultats existants sur quatre algorithmes distribués probabilistes que nous souhaitons étudier et qui résolvent les problèmes de la brisure de symétrie, du rendez-vous, du couplage maximal et de l'ensemble indépendant maximal (MIS).

Le Chapitre 6 traite des algorithmes probabilistes, puis des algo-

rithmes distribués probabilistes. Les algorithmes probabilistes sont définis à travers deux sémantiques dérivées d'une même syntaxe :

- la sémantique ensembliste et relationnelle, tirée du projet *Loco*, qui nous permet de faire des preuves de correction (par le biais d'invariants) ;
- et la sémantique distributionnelle, qui fera le lien avec *Alea*, qui nous permet de prouver des propriétés sur les distributions.

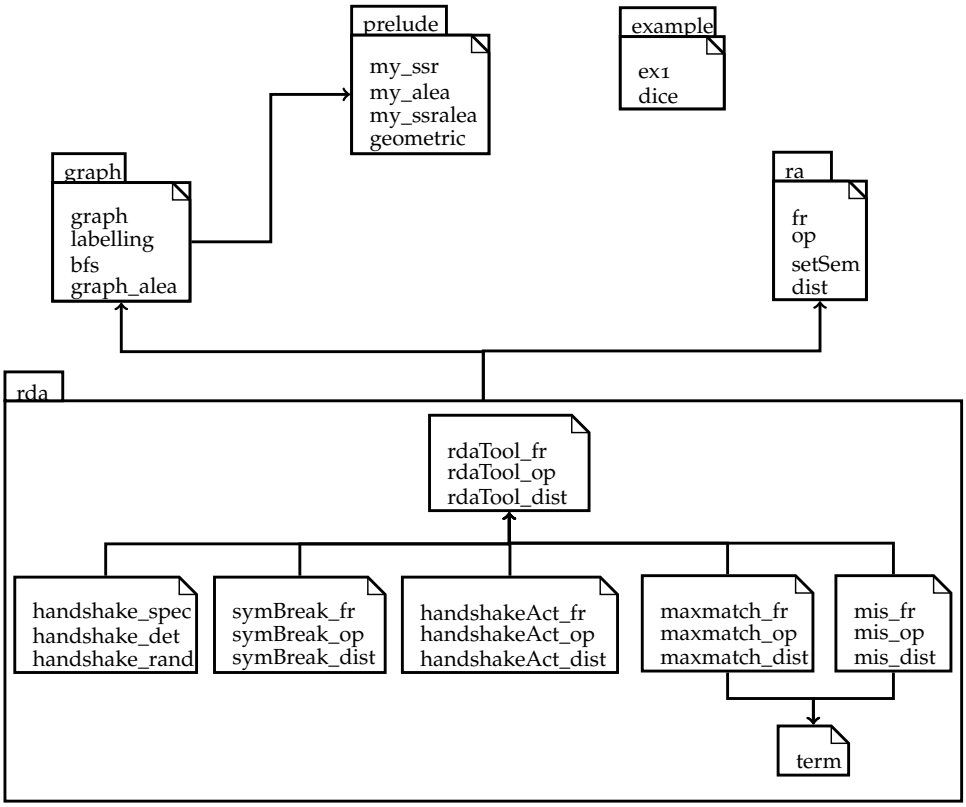
La modélisation formelle des algorithmes distribués probabilistes est réalisée dans l'optique de respecter les hypothèses de notre modèle et de permettre les preuves des propriétés énoncées dans la Section 5.5 page 79.

Nous motivons l'utilisation de l'aspect probabiliste dans le Chapitre 7. Pour cela, nous montrons l'impossibilité de résoudre le problème du rendez-vous en supposant que l'algorithme est déterministe. Nous montrerons ensuite qu'il existe un algorithme probabiliste qui résout ce problème.

Dans le Chapitre 8, nous menons l'analyse formelle des algorithmes distribués probabilistes. Des techniques générales, sous la forme de lemmes que nous avons prouvés, sont présentées : décomposition des calculs, probabilité qu'un évènement se produise ne soit pas nulle, terminaison, etc. Nous appliquons ensuite ces techniques à des algorithmes résolvant les problèmes de la brisure de symétrie, du rendez-vous et du couplage maximal.

Les définitions et lemmes donnés dans la deuxième partie sont tirés de la bibliothèque *RDA* qui est composée comme suit :

- `example` : contient des exemples pour la prise en main d'*Alea* ;
- `prelude` : contient des extensions de *Ssreflect* et d'*Alea* ;
- `graph` : contient des outils relatifs à la théorie des graphes ;
- `ra` : contient notre syntaxe et les sémantiques des algorithmes probabilistes (sémantique ensembliste, opérationnelle et distributionnelle) ;
- `rda` : contient des outils généraux pour les algorithmes distribués (`rdaTool`) et des applications (`symBreak`, `handshake`, `maxmatch`, `mis`) ; la syntaxe et les sémantiques opérationnelles et distributionnelles sont traitées dans les fichiers avec les extensions respectives `*_fr`, `*_op`, `*_dist`.


prelude
my_ssr
my_alea
my_ssralea
geometric
example
ex1
dice
graph
graph
labelling
bfs
graph_alea
ra
fr
op
setSem
dist
rda
rdaTool_fr
rdaTool_op
rdaTool_dist
handshake_spec
handshake_det
handshake_rand
symBreak_fr
symBreak_op
symBreak_dist
handshakeAct_fr
handshakeAct_op
handshakeAct_dist
maxmatch_fr
maxmatch_op
maxmatch_dist
mis_fr
mis_op
mis_dist
term

Préliminaires

1

Sommaire

1.1 Rappels Mathématiques

Les rappels faits dans cette section sont directement tirés du livre de Motwani et Raghavan [65].

1.1.1 Comparaison Asymptotique

Définition 1.1 *Soient deux fonctions $f, g : \mathbb{R} \to \mathbb{R}^+$. On dit que :*
- $f(n) = O(g(n))$ *s'il existe une constante $c > 0$ et un entier N tels que $\forall\, n > N, f(n) \leq cg(n)$.*
- $f(n) = \Omega(g(n))$ *s'il existe une constante $c > 0$ et un entier N tels que $\forall\, n > N, f(n) \geq cg(n)$.*
- $f(n) = \Theta(g(n))$ *si $f(n) = O(g(n))$ et $f(n) = \Omega(g(n))$.*
- $f(n) = o(g(n))$ *si $lim_{n\to\infty} f(n)/g(n) = 0$.*
- $f(n) \sim g(n)$ *si $lim_{n\to\infty} f(n)/g(n) = 1$.*

1.1.2 Inégalité Combinatoire

Définition 1.2 *Les coefficients binomiaux sont définis comme suit.*
Soient $n \geq k \geq 0$,

$$\binom{n}{k} = \binom{n}{n-k} = \frac{n!}{k!(n-k)!}$$

. Si $k > n \geq 0$, on définit $\binom{n}{k} = 0$.
La raison du nom « coefficients binomiaux » est leur présence dans l'expression binomiale :

$$(p+q)^n = \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k}.$$

Proposition 1.1 *Soient $n \geq k \geq 0$,*

1. $\binom{n}{k} \leq \frac{n^k}{k!}$.
2. *Pour n grand,* $\binom{n}{k} \sim \frac{n^k}{k!}$.
3. $\binom{n}{k} \leq \left(\frac{ne}{k}\right)^k$.
4. $\binom{n}{k} \geq \left(\frac{n}{k}\right)^k$.

Proposition 1.2

1. $\forall x, e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$
2. $\forall t, n \in \mathbb{R}^+, \left(1 + \frac{t}{n}\right) \leq e^t \leq \left(1 + \frac{t}{n}\right)^{n+t/2}$

Proposition 1.3 *Soient n réels strictement positifs $x_1, \ldots, x_n$, on a l'inégalité suivante dite inégalité arithmético-géométrique :*

$$\frac{1}{n}\left(\sum_{i=1}^{n} x_i\right) \geq \left(\prod_{i=1}^{n} x_i\right)^{\frac{1}{n}}.$$

Proposition 1.4

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}.$$

1.2 Rappels sur la Théorie des Probabilités

Nous faisons des rappels sur la théorie des probabilités qui nous seront utiles pour modéliser puis analyser les algorithmes distribués probabilistes de notre modèle. La plupart des définitions et résultats énoncés ci-dessous sont directement tirés des livres de Motwani et Raghavan [65], Rosen [72] et Tel [75].

1.2.1 Concepts Fondamentaux

Définition 1.3 ***Expérience.*** *Une expérience est une tâche physiquement ou mentalement concevable issue d'un résultat mesurable.*

Définition 1.4 ***Univers.*** *L'univers est l'ensemble Ω de tous les résultats possibles d'une expérience.*

Définition 1.5 ***Évènement.*** *Un évènement dans un univers Ω est un sous-ensemble de Ω.*

*Soit une famille d'évènements $\{E_j | j \in J\}$, l'**union** $\bigcup_{j\in J} E_j$ est l'ensemble des résultats appartenant au moins à un des E_j ; l'**intersection** $\bigcap_{j\in J} E_j$ est l'ensemble de tous les résultats qui appartiennent à tous les E_j.*

*Le **complément** $\bar{E}$ d'un évènement E est l'ensemble des résultats dans l'univers qui n'appartiennent pas à E.*

*Les évènements E_1 et E_2 sont **disjoints** si $E_1 \cap E_2 = \emptyset$. Les évènements E_1, E_2, E_3, ... sont **disjoints deux à deux** si chaque paire E_i, E_j d'évènements distincts sont disjoints.*

Définition 1.6 ***σ-algèbre.*** *Soit Ω un ensemble. On appelle sigma-algèbre de l'univers Ω un ensemble de sous-ensembles $\mathcal{E}$ contenant l'ensemble vide, stable par passage au complément et par union dénombrable :*

1. $\emptyset \in \mathcal{E}$,
2. $E \in \mathcal{E} \Rightarrow \bar{E} \in \mathcal{E}$,
3. $E_1, E_2, \cdots \in \mathcal{E} \Rightarrow E_1 \cup E_2 \cup \cdots \in \mathcal{E}$.

*Si $\mathcal{E}$ est une σ-algèbre, le couple $(\Omega, \mathcal{E})$ est appelé **espace mesurable**.*

Par exemple, si $\Omega = \{a, b, c, d\}$, une σ-algèbre possible de Ω est $\mathcal{E} = \{\emptyset, \{a, b\}, \{c, d\}, \{a, b, c, d\}\}$.

Définition 1.7 ***Mesure.*** *On appelle mesure sur un espace mesurable $(\Omega, \mathcal{E})$ une application $\mu : \mathcal{E} \to \mathbb{R}^+$ telle que :*

1. $\mu(\emptyset) = 0$,
2. $\mu(\bigcup_n E_n) = \sum_n \mu(E_n)$ *lorsque les E_n constituent une suite d'éléments deux à deux disjoints de la σ-algèbre $\mathcal{E}$. Cette propriété est appelée **σ-additivité**.*

Définition 1.8 ***Probabilité.*** *Une probabilité sur $\mathcal{E}$ (σ-algèbre de Ω) est une mesure de masse totale 1, on la note $\mathbb{P}$:*

$$\mathbb{P}(\Omega) = 1.$$

Proposition 1.5 ***Principe d'inclusion-exclusion.*** *Soient deux évènements E_1 et E_2,*

$$\mathbb{P}(E_1 \cup E_2) = \mathbb{P}(E_1) + \mathbb{P}(E_2) - \mathbb{P}(E_1 \cap E_2).$$

Proposition 1.6 ***Inégalité de Boole.*** *Soient E_1, E_2, ..., E_n des évènements, alors*

$$\mathbb{P}(\bigcup_{k=1}^{n} E_k) \leq \sum_{k=1}^{n} \mathbb{P}(E_k).$$

Proposition 1.7 ***Monotonie.*** *Soient deux évènements E_1 et E_2, on a :*

$$E_1 \subseteq E_2 \Rightarrow \mathbb{P}(E_1) \leq \mathbb{P}(E_2).$$

1.2.2 Indépendance et Dépendance

Définition 1.9 ***Indépendance.*** *Deux évènements E_1 et E_2 sont indépendants si $\mathbb{P}(E_1 \cap E_2) = \mathbb{P}(E_1)\mathbb{P}(E_2)$.*

Les n évènements $E_1, E_2, \ldots, E_n$ sont indépendants si pour tout k ($2 \leq k \leq n$) et pour tous $j_1, j_2, \ldots, j_k$ ($1 \leq j_1 < j_2 < \cdots < j_k \leq n$),

$$\mathbb{P}(E_{j_1} \cap E_{j_2} \cap \cdots \cap E_{j_k}) = \mathbb{P}(E_{j_1})\mathbb{P}(E_{j_2}) \ldots \mathbb{P}(E_{j_k}).$$

Définition 1.10 ***Probabilité conditionnelle.*** *La probabilité conditionnelle de l'évènement E_1 sachant l'évènement E_2 est notée $\mathbb{P}(E_1|E_2)$ et est donnée par la formule :*

$$\frac{\mathbb{P}(E_1 \cap E_2)}{\mathbb{P}(E_2)}$$

en supposant que $\mathbb{P}(E_2) > 0$.

Proposition 1.8 ***Formule des probabilités totales.*** *Soient E_1, E_2, ..., E_k, une partition de l'univers Ω. Alors pour tout évènement E,*

$$\mathbb{P}(E) = \sum_{i=1}^{k} \mathbb{P}(E \cap E_i) = \sum_{i=1}^{k} \mathbb{P}(E|E_i)\mathbb{P}(E_i).$$

Proposition 1.9 ***Formule de Bayes.*** *Soient E_1, E_2, ..., E_k, une partition de l'univers Ω. Alors pour tout évènement E,*

$$\mathbb{P}(E_i|E) = \frac{\mathbb{P}(E_i \cap E)}{\mathbb{P}(E)} = \frac{\mathbb{P}(E|E_i)\mathbb{P}(E_i)}{\sum_{j=1}^{k} \mathbb{P}(E|E_j)\mathbb{P}(E_j)}.$$

Proposition 1.10 ***Formule de composition.*** *Soient E_1, E_2, ..., E_k des évènements satisfaisant $\mathbb{P}(\bigcap_{k=1}^{n-1} E_k) > 0$, alors*

$$\mathbb{P}(E_1 \cap E_2 \cap \cdots \cap E_n) = \mathbb{P}(E_1)\mathbb{P}(E_2|E_1)\mathbb{P}(E_3|E_1 \cap E_2) \ldots \mathbb{P}(E_n|\bigcap_{k=1}^{n-1} E_k).$$

1.2.3 Variable Aléatoire

Les évènements que nous traitons sont, en général, exprimables sous la forme de fonctions à valeurs réelles appelées *variables aléatoires*.

Définition 1.11 ***Variable aléatoire.*** *Soit $(\Omega, \mathcal{E})$ un espace mesurable. Une variable aléatoire X est une fonction à valeur réelle sur un univers, $X : \Omega \to \mathbb{R}$, telle que pour tout $x \in \mathbb{R}$,*

$$\{\omega \in \Omega | X(\omega) \leq x\} \in \mathcal{E}.$$

Cela nous donne une représentation compacte d'évènements complexes puisque $\mathbb{P}(X \leq x)$ est une autre façon d'écrire $\mathbb{P}(\{\omega \in \Omega | X(\omega) \leq x\})$.

Définition 1.12 ***Fonction de distribution.*** *La fonction de distribution $F : \mathbb{R} \to [0,1]$ pour une variable aléatoire X est définie comme $F_X(x) = \mathbb{P}(X \leq x)$.*

Définition 1.13 ***Variable aléatoire discrète.*** *Une variable aléatoire discrète est une fonction sur l'univers dont le co-domaine est soit fini, soit un sous-ensemble dénombrable de $\mathbb{R}$. Nous nous intéresserons aux variables qui sont à valeurs entières.*

Soit X une variable aléatoire discrète qui renvoie uniquement les valeurs 0 ou 1. Cette variable aléatoire X permet d'exprimer les occurrences ou les non-occurrences d'un évènement E, où $E = \{\omega \in \Omega | X(\omega) = 1\}$ et $\bar{E} = \{\omega \in \Omega | X(\omega) = 0\}$. Il est à noter que les notions de probabilité conditionnelle et d'indépendance se reportent sur les variables aléatoires car elles correspondent seulement à une autre façon de noter les évènements.

Définition 1.14 ***Continuité.*** *Une variable aléatoire X est continue si elle a une fonction de distribution F dont la dérivée F' est une fonction intégrable et positive.*

La fonction F' fait référence à la *densité* de la variable aléatoire X. Nous ne considérerons ici que des variables aléatoires discrètes, même si la définition suivante peut s'étendre à des variables aléatoires continues.

Définition 1.15 ***Densité.*** *La densité $p_X : \mathbb{R} \to [0,1]$ d'une variable aléatoire X est définie par $p_X(x) = \mathbb{P}(X = x)$.*

1.2.4 Espérance

Définition 1.16 ***Espérance.*** *L'espérance d'une variable aléatoire X de densité p est définie comme $\mathbb{E}(X) = \sum_x x p_X(x)$, où la sommation se fait sur le co-domaine de X.*

Proposition 1.11 ***Linéarité de l'espérance.*** *Soient $X_1, \ldots, X_k$ des variables aléatoires et $h(X_1, \ldots, X_k)$ une fonction linéaire. Alors,*

$$\mathbb{E}(h(X_1, \ldots, X_k)) = h(\mathbb{E}(X_1), \ldots, \mathbb{E}(X_k)).$$

Proposition 1.12 *Soit X une variable aléatoire à valeur entière positive alors*

$$\mathbb{E}(X) = \sum_{x=1}^{\infty} \mathbb{P}(X \geq x).$$

Définition 1.17 ***Fonction caractéristique.*** *Une fonction caractéristique d'un ensemble X est notée $\mathbb{I}_X$ et est définie comme suit :*

$$\mathbb{I}_X(x) = \begin{cases} 1 & si\ x \in X \\ 0 & sinon. \end{cases}$$

Remarque 1.1 *Considérons une distribution de probabilité $\mathcal{D}$ sur un espace mesurable $(\Omega, \mathcal{E})$ où Ω est fini. Nous introduisons la mesure μ telle que*

$$(\mu\mathcal{D})f = \sum_{\omega \in \Omega} \mathcal{D}(\{\omega\}) f(\omega).$$

Soit E un évènement et $\mathbb{I}_E$ la fonction caractéristique correspondant à une variable aléatoire discrète à valeurs dans $\{0,1\}$ qui dénote l'évènement E. Alors,

$$(\mu\ \mathcal{D})\mathbb{I}_E = \sum_{\omega\in\Omega} \mathcal{D}(\{\omega\})\mathbb{I}_E(\omega).$$

L'espérance de la variable aléatoire $\mathbb{I}_E$ est :

$$\mathbb{E}(\mathbb{I}_E) = 0 * p(0) + 1 * p(1) = \mathbb{P}(\mathbb{I}_E = 1).$$

On retrouve donc la probabilité que l'évènement E se produise pour la distribution $\mathcal{D}$.

1.2.5 Diverses Lois et leurs Propriétés

Loi de Bernoulli

Imaginons qu'on lance une pièce dont la probabilité d'obtenir *face* est p. Soit X la variable aléatoire valant 1 si le résultat est *face* et 0 sinon. Alors X suit la loi de Bernoulli de paramètre p.

Définition 1.18 ***Loi de Bernoulli.*** *Soit p un nombre réel tel que $0 \leq p \leq 1$, une épreuve de Bernoulli de paramètre p est une expérience aléatoire pouvant soit réussir soit échouer. La probabilité de succès est p et celle d'échec est $q = 1 - p$.*

Proposition 1.13 *L'espérance d'une variable aléatoire X de Bernoulli vaut p :*

$$\mathbb{E}(X) = p.$$

Loi Binomiale

Soient X_1, X_2, ..., X_n des variables aléatoires indépendantes et identiquement distribuées dont la distribution suit la loi de Bernoulli de paramètre p. La variable aléatoire $X = X_1 + X_2 + \cdots + X_n$ correspond aux nombres de succès obtenus suite à n expériences successives. La variable aléatoire X suit la loi binomiale de paramètres n et p.

Définition 1.19 ***Loi binomiale.*** *Considérons une épreuve de Bernoulli de paramètre p. Cette épreuve est renouvelée n fois. Soit X le nombre de succès obtenus à l'issue des n épreuves. La loi géométrique, notée $\mathcal{B}(n,p)$, correspond à la distribution de X.*

Proposition 1.14 *La variable aléatoire X suit la loi de probabilité suivante :*

$$\mathbb{P}(X = k) = \binom{n}{k} p^k q^{n-k}$$

où $q = 1 - p$.

Proposition 1.15 *L'espérance du nombre de succès vaut np :*

$$\mathbb{E}(X) = np.$$

Loi Géométrique

Imaginons qu'on lance une pièce jusqu'à ce que *face* apparaisse. En supposant que chaque lancer suit une loi de Bernoulli de paramètre p, la variable aléatoire X correspondant au nombre total de lancers suit une loi géométrique de paramètre p.

Définition 1.20 ***Loi géométrique.*** *Considérons une épreuve de Bernoulli de paramètre p. Cette épreuve est renouvelée jusqu'à l'obtention d'un succès. Soit X la variable aléatoire donnant le nombre d'échecs avant le premier succès. X prend ses valeurs dans l'ensemble* $\{0, 1, 2..\}$*. La* ***loi géométrique*** *correspond à la distribution de X.*

Proposition 1.16 *La probabilité d'avoir k échecs suivis d'un succès est :*

$$\mathbb{P}(X = k) = (1 - p)^k p.$$

Proposition 1.17 *L'espérance du nombre d'échecs avant un succès est* $1/p$ *:*

$$\mathbb{E}(X) = \sum_{k=0}^{\infty} \mathbb{P}(X = k) = \frac{1}{p}.$$

1.2.6 Classification des Algorithmes Probabilistes

Les algorithmes probabilistes sont classés selon leur degré de correction : les algorithmes Las Vegas et Monte Carlo.

Définition 1.21 ***Algorithmes Las Vegas.*** *Un algorithme Las Vegas est un algorithme probabiliste qui*

1. *termine avec une probabilité positive*
2. *et qui est correct lorsqu'il termine.*

Remarque 1.2 *En général, la probabilité de terminaison d'un algorithme Las Vegas est* 1*, des exécutions infinies existent mais ne se produisent que si un certain motif de bits aléatoires est sélectionné infiniment souvent.*

Définition 1.22 ***Algorithmes Monte Carlo.*** *Un algorithme Monte Carlo est un algorithme probabiliste qui*

1. *termine*
2. *et qui est correct avec une probabilité positive.*

Remarque 1.3 *La probabilité d'échec d'un algorithme Monte Carlo est toujours strictement plus petite que* 1*.*

1.2.7 Forte Probabilité

Lorsqu'on parlera de propriété vérifiée avec forte probabilité, cela reviendra à dire avec probabilité $1 - o(1/n)$ où n est la taille de l'entrée.

1.2.8 Générateur de Nombres Pseudo-Aléatoires

Des nombres pseudo-aléatoires sont des nombres générés de façon prédictible mais qui semblent se comporter comme des nombres aléatoires indépendants et uniformément distribués.

Il est difficile d'extraire, des résultats fournis par des générateurs pseudo-aléatoires, des séries de nombres qui suivent une règle. Bien que ces résultats ne soient pas aléatoires, ces générateurs sont utilisés car ils sont facilement implémentables dans des langages de programmation. Une des classes très répandues de générateurs est la méthode linéaire congruentielle introduite par Lehmer. Comme le décrit Knuth [42] dans son livre, la méthode linéaire congruentielle consiste en un choix minutieux des nombres :

- m : le modulo ($m > 0$),
- a : le multiplicateur ($0 \leq a < m$),
- c : l'incrément ($0 \leq c < m$),
- X_0 : la valeur initiale, appelée graine ($0 \leq X_0 < m$).

La séquence de nombres pseudo-aléatoires est définie par la formule de récurrence :

$$X_{n+1} = (aX_n + c) \ mod \ m.$$

La période de ce générateur est au maximum m. De plus, l'opération modulo fait que les termes sont compris entre 0 et $(m-1)$.

La séquence obtenue n'est pas toujours aléatoire pour tous les choix de m, a, c et X_0. Le choix de ces nombres a été étudié et certaines heuristiques ont pu être faites.

1.3 Rappels sur la Théorie des Graphes

Nous rappelons des notions de la théorie des graphes qui nous seront utiles par la suite. Elles sont tirées du livre de Berge [11].

1.3.1 Graphes Non-Orientés

Définition 1.23 ***Graphe non-orienté sans boucle.*** *Un graphe non-orienté* G *est défini par un ensemble de sommets* $\mathsf{V}(\mathsf{G})$*, un ensemble d'arêtes* $\mathsf{E}(\mathsf{G})$ *et par une fonction ext qui associe à chaque arête deux éléments distincts de* $\mathsf{V}(\mathsf{G})$*, appelés ses extrémités.*

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

On notera un tel graphe $\mathsf{G} = (\mathsf{V}, \mathsf{E})$*.*

Le nombre des sommets d'un graphe G *est appelé la* ***taille*** *du graphe et est notée* n_{G} *ou n s'il n'y a pas d'ambiguïté. Le nombre d'arêtes sera noté* m_{G} *ou m s'il n'y a pas d'ambiguïté.*

Définition 1.24 ***Graphe simple non-orienté.*** *Un graphe simple non-orienté* G *est un graphe non-orienté sans boucle tel qu'il y ait au plus une arête entre deux sommets, i.e., pour tous sommets* $u, v \in \mathsf{V}(\mathsf{G})$*,* $|\{e \in \mathsf{E}(\mathsf{G}) | ext(e) = \{u, v\}\}| \leq 1$*.*

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

Définition 1.25 ***Sous-graphe.*** *Un graphe* H *est un sous-graphe du graphe* G *si* $\mathsf{V}(\mathsf{H}) \subseteq \mathsf{V}(\mathsf{G})$ *et* $\mathsf{E}(\mathsf{H}) \subseteq \mathsf{E}(\mathsf{G})$*.*

Un graphe H *est un sous-graphe* ***partiel*** *du graphe* G *si* $\mathsf{V}(\mathsf{H}) = \mathsf{V}(\mathsf{G})$ *et* $\mathsf{E}(\mathsf{H}) \subseteq \mathsf{E}(\mathsf{G})$*.*

Un graphe H *est un sous-graphe* ***induit*** *du graphe* G *si* $\mathsf{V}(\mathsf{H}) \subseteq \mathsf{V}(\mathsf{G})$ *et* $\mathsf{E}(\mathsf{H})$ *est l'ensemble des arêtes de* G *dont les extrémités sont dans* $\mathsf{V}(\mathsf{H})$.

Définition 1.26 ***Voisinage et degré.*** *Dans un graphe* G, *le voisinage d'un sommet* u, *noté* $N_{\mathsf{G}}(u)$, *est l'ensemble des voisins de* u, *i.e.,* $N_{\mathsf{G}}(u) = \{v \in \mathsf{V}(\mathsf{G}) | \exists\ e \in \mathsf{E}(\mathsf{G})$ *tels que* $ext(e) = \{u, v\}\}$.

On note $I_{\mathsf{G}}(u)$ *l'ensemble des arêtes incidentes au sommet* u, *i.e.,* $I_{\mathsf{G}}(u) = \{e \in \mathsf{E}(\mathsf{G}) | u \in ext(e)\}$.

Dans un graphe G, *le degré d'un sommet* u, *noté* $deg_{\mathsf{G}}(u)$, *est le nombre d'arêtes incidentes à* u, *i.e.,* $deg_{\mathsf{G}}(u) = |I_{\mathsf{G}}(u)|$. *En particulier, si* G *est un graphe simple, alors le degré de* u *est son nombre de voisins, i.e.,* $deg_{\mathsf{G}}(u) = |N_{\mathsf{G}}(u)|$.

Définition 1.27 ***Chemin.*** *Dans un graphe* G, *un chemin* Γ *entre deux sommets* u *et* v *de* G *est une suite alternée* $(u_0, e_1, u_1, e_2, \ldots, e_l, u_l)$ *telle que :*

- $\forall i \in [0, l], u_i \in \mathsf{V}(\mathsf{G})$,
- $\forall i \in [1, l], e_i \in \mathsf{E}(\mathsf{G})$,
- $\forall i \in [1, k], ext(e_i) = \{u_{i-1}, u_i\}$,
- $u_0 = u$ *et* $u_l = v$.

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

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

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

Remarque 1.4 *Dans un graphe simple, un chemin sera généralement décrit par la suite de ses sommets* $(u_0, u_1, \ldots, u_l)$, *puisque pour tout* $i \in [1, l]$, *l'arête* e_i *est nécessairement l'arête* $\{u_{i-1}, u_i\}$. *Ainsi, dans un graphe simple* G, *une suite de sommets* $(u_0, u_1, \ldots, u_l)$ *est un chemin si pour tout* $i \in [1, l]$, $\{u_{i-1}, u_i\} \in \mathsf{E}(\mathsf{G})$.

Définition 1.28 ***Connexité.*** *Un graphe* G *est connexe si pour tous sommets* $u, v \in \mathsf{V}(\mathsf{G})$, *il existe un chemin entre* u *et* v *dans* G.

Définition 1.29 ***Acyclicité.*** *Un graphe* G *est acyclique s'il ne contient pas de cycle.*

Définition 1.30 ***Arbre.*** *Un arbre est un graphe connexe et acyclique.*

Définition 1.31 ***Arbre couvrant.*** *Un arbre couvrant d'un graphe* G *est un arbre qui est un sous-graphe partiel de* G.

Définition 1.32 ***Anneau.*** *Un anneau est un graphe constitué d'un cycle simple.*

Définition 1.33 ***Distance.*** *Étant donnés un graphe connexe* G *et deux sommets* $u, v \in \mathsf{V}(\mathsf{G})$, *la distance de* u *à* v *dans* G, *notée* $dist_{\mathsf{G}}(u, v)$, *est la longueur du plus court chemin de* u *à* v.

Le ***diamètre*** *de* G, *noté* $diam_{\mathsf{G}}$ *est la plus grande distance entre deux sommets de* G, *i.e.,* $diam_{\mathsf{G}} = max\{dist_G(u, v) | u, v \in \mathsf{V}(\mathsf{G})\}$.

Définition 1.34 ***Homomorphisme.*** *Un homomorphisme* φ *d'un graphe* G *dans un graphe* H *est une application de* $\mathsf{V}(\mathsf{G})$ *dans* $\mathsf{V}(\mathsf{H})$ *et de* $\mathsf{E}(\mathsf{G})$ *dans* $\mathsf{E}(\mathsf{H})$ *qui préserve les relations d'incidence, i.e., pour toute arête* $e \in \mathsf{E}(\mathsf{G})$, $ext(\varphi(e)) = \varphi(ext(e))$.

Définition 1.35 ***Isomorphisme.*** *Un homomorphisme φ d'un graphe G dans un graphe H est un isomorphisme si φ est bijective. On dit alors que G et H sont isomorphes.*

Dans les définitions suivantes, on considère les homomorphismes entre graphes simples.

Définition 1.36 ***Homomorphisme.*** *Un homomorphisme φ d'un graphe simple G dans un graphe simple H est une application de $\mathsf{V}(\mathsf{G})$ dans $\mathsf{V}(\mathsf{H})$ qui préserve les relations d'adjacence entre sommets, i.e., pour toute arête $\{v, w\} \in \mathsf{E}(\mathsf{G})$, $\{\varphi(v), \varphi(w)\} \in \mathsf{E}(\mathsf{H})$.*

1.3.2 Graphes Orientés

Définition 1.37 ***Graphe orienté.*** *Un graphe orienté D est défini par un ensemble de sommets $\mathsf{V}(\mathsf{D})$, un ensemble d'arcs $\mathsf{A}(\mathsf{D})$ et deux fonctions s_{D} et t_{D} de $\mathsf{A}(\mathsf{D})$ dans $\mathsf{V}(\mathsf{D})$.*

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

*Pour tout sommet $u, v \in \mathsf{V}(\mathsf{D})$, s'il existe un arc $a \in \mathsf{A}(\mathsf{D})$ tel que $s_{\mathsf{D}}(a) = u$ et $t_{\mathsf{D}}(a) = v$, u est un **prédécesseur** de v et v est un **successeur** de u.*

Définition 1.38 ***Voisinage et degré.*** *Dans un graphe orienté D, le voisinage sortant d'un sommet u, noté $N^+_{\mathsf{D}}(u)$, est l'ensemble des successeurs de u, i.e., $N^+_{\mathsf{D}}(u) = \{v \in \mathsf{V}(\mathsf{D}) | \exists\, a \in \mathsf{A}(\mathsf{D})$ tels que $s(a) = u$ et $t(a) = v\}$. On note $I^+_{\mathsf{D}}(u)$ l'ensemble des arcs sortants de u, i.e., $I^+_{\mathsf{D}}(u) = \{a \in \mathsf{A}(\mathsf{D}) | s(a) = u\}$. Le degré sortant de u, noté $deg^+_{\mathsf{D}}(u)$, est le nombre d'arcs sortants de u, i.e., $deg^+_{\mathsf{D}}(u) = |I^+_{\mathsf{D}}(u)|$.*

Le voisinage entrant d'un sommet u dans un graphe orienté D, noté $N^-_{\mathsf{D}}(u)$, est l'ensemble des prédécesseurs de u, i.e., $N^-_{\mathsf{D}}(u) = \{v \in \mathsf{V}(\mathsf{D}) | \exists\, a \in \mathsf{A}(\mathsf{D})$ tels que $s(a) = v$ et $t(a) = u\}$. On note $I^-_{\mathsf{D}}(u)$ l'ensemble des arcs entrants de u, i.e., $I^-_{\mathsf{D}}(u) = \{a \in \mathsf{A}(\mathsf{D}) | t(a) = u\}$. Le degré entrant de u, noté $deg^-_{\mathsf{D}}(u)$, est le nombre d'arcs entrants de u, i.e., $deg^-_{\mathsf{D}}(u) = |I^-_{\mathsf{D}}(u)|$.

Définition 1.39 ***Graphe symétrique.*** *Un graphe orienté symétrique est un graphe orienté D muni d'une involution $Sym : \mathsf{A}(\mathsf{D}) \to \mathsf{A}(\mathsf{D})$ qui à chaque arc $a \in \mathsf{A}(\mathsf{D})$ associe son arc symétrique $Sym(a)$ tel que $s_{\mathsf{D}}(a) = t_{\mathsf{D}}(Sym(a))$.*

La Figure 1.1 donne un exemple de fonction *Sym*.

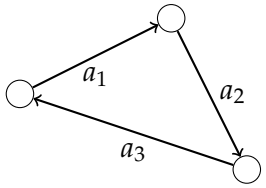


FIGURE 1.1 – *Exemple de graphe orienté symétrique où $Sym(a_1) = a_3$, $Sym(a_2) = a_1$ et $Sym(a_3) = a_2$.*

Définition 1.40 ***Homomorphisme.*** *Un homomorphisme φ d'un graphe orienté D dans un graphe orienté D' est une application de $\mathsf{V}(\mathsf{D})$ dans $\mathsf{V}(\mathsf{D}')$ et de $\mathsf{A}(\mathsf{D})$ dans $\mathsf{A}(\mathsf{D}')$ qui commute avec les fonctions source et cible, i.e., pour tout arc $a \in \mathsf{A}(\mathsf{D})$, $s_{\mathsf{D}'}(\varphi(a)) = \varphi(s_{\mathsf{D}}(a))$ et $t_{\mathsf{D}'}(\varphi(a)) = \varphi(t_{\mathsf{D}}(a))$.*

La Figure 1.2 présente deux graphes homomorphes.

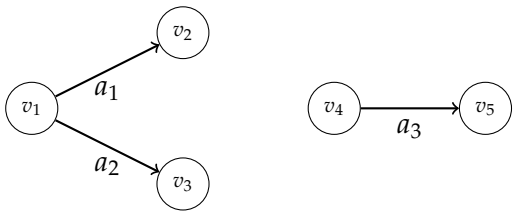


FIGURE 1.2 – *Exemple d'homomorphisme γ entre deux graphes orientés où $\gamma(a_1) = \gamma(a_2) = a_3$, $\gamma(v_1) = v_4$, $\gamma(v_2) = \gamma(v_3) = v_5$.*

Définition 1.41 ***Isomorphisme.*** *Un homomorphisme φ d'un graphe orienté D dans un graphe orienté D' est un isomorphisme si φ est bijective. On dit alors que D et D' sont isomorphes.*

1.3.3 Graphes Étiquetés

Définition 1.42 ***Graphe étiqueté.*** *Soit Λ un ensemble d'étiquettes (fini ou infini). Un graphe étiqueté, noté (G, λ) est un graphe G muni d'une fonction d'étiquetage $\lambda : \mathsf{V}(\mathsf{G}) \cup \mathsf{E}(\mathsf{G}) \to \Lambda$ qui associe une étiquette à chaque sommet $v \in \mathsf{V}(\mathsf{G})$ et à chaque arête $e \in \mathsf{E}(\mathsf{G})$.*

*Lorsque la fonction d'étiquetage n'a pas à être explicitée, le graphe (G, λ) sera dénoté $\mathbf{G}$. On dit que le graphe G est le graphe **sous-jacent** de $\mathbf{G}$.*

Définition 1.43 ***Uniformité.*** *On dit que l'étiquetage d'un graphe (G, λ) est uniforme s'il existe deux étiquettes $\alpha, \beta \in \Lambda$ telles que pour tout sommet $v \in \mathsf{V}(\mathsf{G})$, $\lambda(v) = \alpha$ et pour toute arête $e \in \mathsf{E}(\mathsf{G})$, $\lambda(e) = \beta$.*

Définition 1.44 ***Sous-graphe.*** *Un graphe $\mathbf{H} = (\mathsf{H}, \eta)$ est un sous-graphe (respectivement graphe partiel, respectivement sous-graphe induit) d'un graphe $\mathbf{G} = (\mathsf{G}, \lambda)$ si H est un sous-graphe (respectivement graphe sous-jacent, respectivement sous-graphe induit) de G et pour tout $x \in \mathsf{V}(\mathsf{H}) \cup \mathsf{E}(\mathsf{H}), \lambda(x) = \eta(x)$.*

Définition 1.45 ***Homomorphisme.*** *Un homomorphisme φ d'un graphe étiqueté $\mathbf{G} = (\mathsf{G}, \lambda)$ dans un graphe étiqueté $\mathbf{H} = (\mathsf{H}, \eta)$ est un homomorphisme de G dans H tel que pour tout $x \in \mathsf{V}(\mathsf{G}) \cup \mathsf{E}(\mathsf{G}), \lambda(x) = \eta(\varphi(x))$.*

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

1.3.4 Graphes Orientés Étiquetés

Définition 1.46 ***Graphe orienté étiqueté.*** *Soit Λ un ensemble d'étiquettes (fini ou infini). Un graphe orienté étiqueté, noté (D, λ) est un graphe orienté D muni d'une fonction d'étiquetage $\lambda : \mathsf{V}(\mathsf{D}) \cup \mathsf{A}(\mathsf{D}) \to \Lambda$ qui associe une étiquette à chaque sommet $v \in \mathsf{V}(\mathsf{D})$ et à chaque arc $a \in \mathsf{A}(\mathsf{D})$.*

*Lorsque la fonction d'étiquetage n'a pas à être explicitée, le graphe (D, λ) sera dénoté $\mathbf{D}$ on dit que le graphe orienté D est le graphe orienté **sous-jacent** de $\mathbf{D}$.*

Définition 1.47 ***Homomorphisme.*** *Un homomorphisme φ d'un graphe orienté étiqueté $\mathbf{D} = (\mathsf{D}, \lambda)$ dans un graphe orienté étiqueté $\mathbf{D}' = (\mathsf{D}', \lambda')$ est un homomorphisme de D dans D' tel que pour tout $x \in \mathsf{V}(\mathsf{D}) \cup \mathsf{A}(\mathsf{D}), \lambda(x) = \lambda'(\varphi(x))$.*

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

Définition 1.48 ***Fonction d'étiquetage symétrique.*** *Étant donné un graphe orienté symétrique D, on dit qu'une fonction d'étiquetage λ est symétrique, s'il existe une fonction $Sym : \Lambda \to \Lambda$ telle que pour tout arc a, $Sym(\lambda(a)) = \lambda(Sym(a))$.*

1.3.5 Des Graphes Non-Orientés aux Graphes Orientés

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

Nous obtenons la définition suivante pour les graphes simples.

Définition 1.49 *À partir d'un graphe simple étiqueté $\mathbf{G} = (\mathsf{G}, \lambda)$ muni d'une numérotation de ports δ, on construit un graphe orienté symétrique étiqueté, noté $Dir(\mathbf{G}, \delta') = (Dir(\mathsf{G}), (\lambda', \delta'))$, comme suit.*

L'ensemble des sommets de $\mathsf{V}(Dir(\mathsf{G}))$ est l'ensemble $\mathsf{V}(\mathsf{G})$ et pour chaque arête $\{u, v\} \in \mathsf{E}(\mathsf{G})$, il existe deux arcs $a_{u,v}$ et $a_{v,u}$ dans $\mathsf{A}(Dir(\mathsf{G}, \delta'))$ tels que $s(a_{u,v}) = t(a_{v,u}) = u$, $s(a_{v,u}) = t(a_{u,v}) = v$, $\delta'(a_{(u,v)}) = (\delta_u(v), \delta_v(u))$ et $\delta'(a_{(v,u)}) = (\delta_v(u), \delta_u(v))$. Ces arcs correspondent pour chaque sommet aux ports d'entrée et ports de sortie. Par extension, l'étiquetage δ' des arcs est appelé numérotation de ports.

Pour tout sommet $u \in \mathsf{V}(\mathsf{G})$, $\lambda'(u) = \lambda(u)$ et pour toute arête $\{u, v\} \in \mathsf{E}(\mathsf{G})$, $\lambda'(a_{u,v}) = \lambda'(a_{v,u}) = \lambda(\{u, v\})$.

Notons que le graphe orienté ne contient pas de boucles ou d'arcs multiples. Par la suite, l'objet que nous utiliserons est $(Dir(\mathsf{G}), (\lambda, \delta'))$. Deux exemples de telles constructions sont données dans la Figure 1.3.

Un exemple d'étiquetage du graphe G_2 de la Figure 1.3 est donné dans la Figure 1.4.

1.3.6 Revêtements

Définition 1.50 ***Revêtement.*** *Un graphe non-orienté G est un revêtement d'un graphe non-orienté H via l'homomorphisme φ si φ est un homomorphisme de G sur H tel que pour tout sommet v de V la restriction de φ aux arêtes incidentes à v est une bijection entre les arêtes de v et les arêtes de $\varphi(v)$.*

Remarque 1.5 *Dans le cas d'un graphe simple, la condition devient : pour tout sommet v de V la restriction de φ aux voisins de v est une bijection entre les voisins de v et les voisins de $\varphi(v)$.*

Dans le cas des graphes orientés cette définition devient :

Définition 1.51 ***Revêtement.*** *Un graphe orienté D est un revêtement d'un graphe orienté D' via*

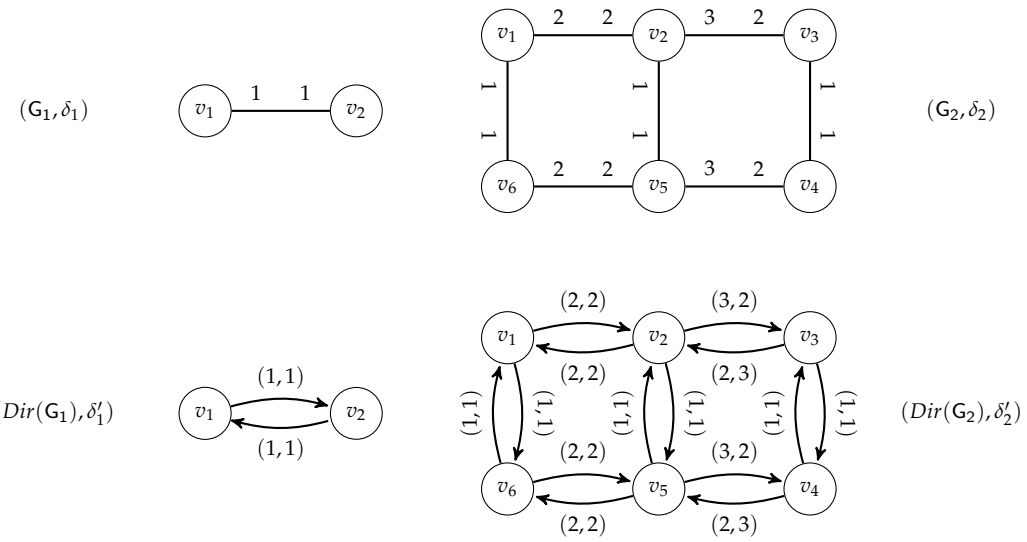


FIGURE 1.3 – *Deux graphes G_1 et G_2 avec des numérotations de ports δ_1 et δ_2 et leur graphes orientés étiquetés associés $Dir(G_1)$ et $Dir(G_2)$ avec leurs numérotations de ports δ'_1 et δ'_2.*

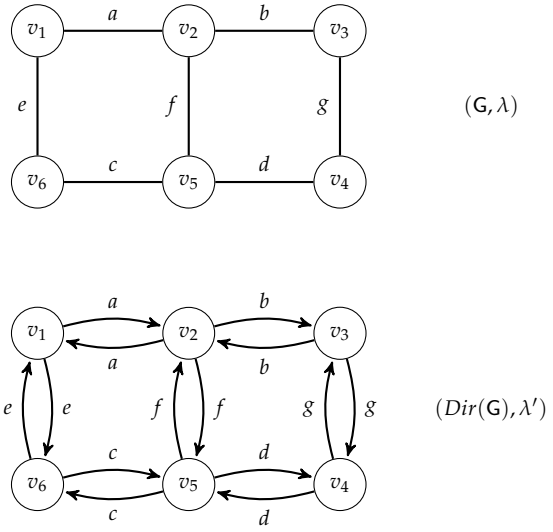


FIGURE 1.4 – *Un graphe G avec un étiquetage λ et son graphe orienté étiqueté associé $Dir(G)$ avec son étiquetage λ'.*

φ si φ est un homomorphisme de D vers D' tel que pour chaque arc $a' \in A(D')$ et pour chaque sommet $v \in \varphi^{-1}(t(a'))$ (resp. $v \in \varphi^{-1}(s(a'))$), il existe un unique arc $a \in A(D)$ tel que $t(a) = v$ (resp. $s(a) = v$) et $\varphi(a) = a'$. Ainsi, pour chaque sommet $v \in V(D)$, la restriction de φ aux arcs incidents à v est une bijection entre les arcs incidents à v et les arcs incidents à $\varphi(v)$.

Remarque 1.6 *La notion de revêtement s'étend naturellement aux graphes étiquetés.*

Définition 1.52 ***Fibre.*** *Soit D un graphe orienté revêtement d'un graphe orienté D'. Une fibre sur un sommet v' (resp. un arc a') de D' est définie comme l'ensemble $\varphi^{-1}(v')$ de sommets de D (resp. l'ensemble $\varphi^{-1}(a')$ d'arcs de D). Les revêtements ont pour propriété que toutes les fibres sur un graphe orienté donné ont la même cardinalité. Cette cardinalité est appelée le* ***nombre de feuillets*** *du revêtement.*

Définition 1.53 ***Revêtement symétrique.*** *Un graphe orienté étiqueté symétrique* $\mathbf{D}$ *est un revêtement symétrique d'un graphe orienté étiqueté symétrique* $\mathbf{D}'$ *via* φ *si* $\mathbf{D}$ *est un revêtement de* $\mathbf{D}'$ *via* φ *et si pour chaque arc* $a \in A(D)$, $\varphi(Sym(a)) = Sym(\varphi(a))$.

Remarque 1.7 *Soit* D *un revêtement symétrique de* D' *via l'homomorphisme* φ. *Tout étiquetage sur* D' *induit naturellement via* φ^{-1} *un étiquetage sur* D. *Réciproquement, un étiquetage sur* D *induit un étiquetage sur* D' *via* φ *si pour chaque fibre, tous ses sommets portent la même étiquette et tous ses arcs portent la même étiquette.*

Un exemple de revêtement est donné dans la Figure 1.5.

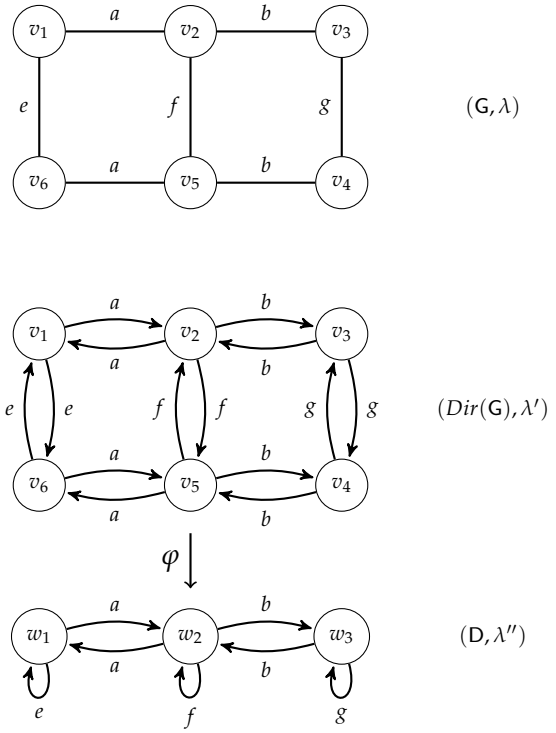


FIGURE 1.5 – *On considère un graphe étiqueté* (G, λ) *et son graphe orienté associé* $(Dir(\mathsf{G}), \lambda')$. *Le graphe* $Dir(\mathsf{G})$ *est un revêtement symétrique via* φ *de* D *où* $\varphi(v_1) = \varphi(v_6) = w_1$, $\varphi(v_2) = \varphi(v_5) = w_2$ *et* $\varphi(v_3) = \varphi(v_4) = w_3$. *Le nombre de feuillets de ce revêtement est* 2. *L'étiquetage* λ' *de* $Dir(\mathsf{G})$ *induit l'étiquetage* λ'' *de* D. *Notons que, réciproquement, l'étiquetage* λ'' *de* D *induit via* φ^{-1} *l'étiquetage* λ' *de* $Dir(\mathsf{G})$ *et donc l'étiquetage* λ *de* G.

Définition 1.54 ***Graphe minimal pour la relation « être revêtement symétrique ».*** *Un graphe orienté étiqueté symétrique* $\mathbf{D}$ *est minimal pour la relation « être revêtement symétrique » s'il n'existe pas de graphe étiqueté symétrique* $\mathbf{D}'$ *non isomorphe à* $\mathbf{D}$ *tel que* $\mathbf{D}$ *est un revêtement symétrique de* $\mathbf{D}'$.

1.3.7 Construction de Reidemeister

La construction que nous donnons maintenant a été présentée par Reidemeister [69] pour décrire tous les revêtements d'un graphe donné. Nous présentons une description donnée par Bodlaender [15] (un exemple est donné dans la Figure 1.6).

– Soit G un graphe.

- Considérons un arbre couvrant (et plus généralement un graphe couvrant) S_T de G.
- Soit α un entier positif. Faire α copies de S_T.
- Pour toute arête dans G qui n'est pas une arête de S_T, nous avons α copies des deux extrémités. À chacune de ces arêtes, associer une permutation de $[1, \alpha]$ et insérer une à une toute copie de la première extrémité sur une unique copie de la seconde suivant cette permutation.

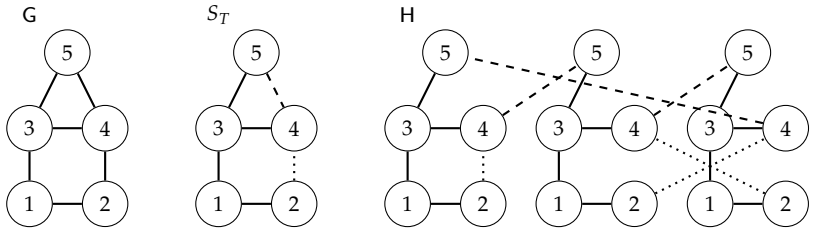


FIGURE 1.6 – *Exemple de construction de Reidemeister pour un graphe* G *avec son arbre couvrant S_T associé (représenté en trait continu). Le graphe* H *est un revêtement à trois feuillets de* G*. Les permutations associées aux arêtes $\{2,4\}$ et $\{4,5\}$ sont $\sigma_{(2,4)} = (1)(2,3)$ (la source, le sommet 2, de la première copie est associée à la cible, le sommet 4 de la première copie ; la source de la deuxième copie est associée à la cible de la troisième copie et la source de la troisième copie est associée à la cible de la deuxième copie) et $\sigma_{(4,5)} = (1,2,3)$ (la source de la première copie est associée à la cible de la deuxième copie, la source de la deuxième copie est associée à la cible de la troisième copie et la source de la troisième copie est associée à la source de la cible de la première copie).*

Reidemeister [69] énonce le résultat suivant :

Théorème 1.1 *Soit $\mathbf{G} = (\mathsf{G}, \rho)$ un graphe simple et S_T un arbre couvrant de $\mathbf{G}$. Un graphe simple $\mathbf{H}$ est un revêtement simple de $\mathbf{G}$ si et seulement s'il existe un entier q et un ensemble $\Sigma = \{\sigma_{(u,v)} | \{u,v\} \in E(\mathbf{H}) \backslash E(S_T)\}$ de permutations de $[1,q]$ (avec la propriété $\sigma(u,v) = \sigma^{-1}(v,u)$) tels que $\mathbf{H}$ est isomorphe au graphe $\mathbf{G}_{S_T,\Sigma} = (\mathsf{G}_{S_T,\Sigma}, \lambda)$ défini de la manière suivante :*

$$\mathsf{V}(\mathsf{G}_{S_T,\Sigma}) = \{(u,i) | v \in V(\mathsf{G}) \wedge i \in [1,q]\},$$
$$\mathsf{E}(\mathsf{G}_{S_T,\Sigma}) = \{\{(u,i),(v,i)\} | \{u,v\} \in \mathsf{E}(S_T)\} \cup \{\{(u,i),(v,\sigma_{(u,v)}(i))\} | \{u,v\} \in \mathsf{E}(\mathsf{G}) \backslash \mathsf{E}(S_T) \wedge i \in [1,q]\}.$$

De plus,
pour tout $(u,i) \in \mathsf{V}(\mathsf{G}_{S_T,\Sigma}), \lambda((u,i)) = \rho(u)$
et pour tout $\{(u,i),(v,j))\} \in \mathsf{E}(\mathsf{G}_{T,\Sigma}), \lambda(\{(u,i),(v,j)\}) = \rho(\{u,v\})$.

Remarque 1.8 *Cette construction fournit un revêtement de* G*. De plus, étant donné un arbre couvrant S_T de* G*, chaque revêtement de* G *peut être obtenu de cette façon à partir de S_T. Le nombre de copies est le nombre de feuillets de ce revêtement.*

Remarque 1.9 *Sauf contre-indication, par la suite, nous manipulerons des graphes simples, connexes et non-orientés.*

Partie I

Analyses d'Algorithmes Distribués Probabilistes

Dans cette partie, nous résolvons certains problèmes de l'algorithmique distribuée de la façon la plus optimale possible.

Dans un premier temps, nous donnons un aperçu de l'état de l'art concernant les résultats sur la complexité en bits des algorithmes distribués résolvant les problèmes du MIS, du couplage maximal et de la coloration pour diverses familles de graphes.

Nous verrons ensuite des algorithmes distribués probabilistes optimaux en terme de complexité en bits qui résolvent les problèmes du MIS et du couplage maximal dans les anneaux anonymes.

Puis nous prouvons des bornes inférieures de la complexité en temps et en bits de quelques algorithmes distribués probabilistes.

État de l'Art

2

Sommaire

2.1 Complexité en Bits et Messages d'un Bit

En algorithmique distribuée, on distingue deux modèles pour la taille des messages : le modèle LOCAL et le modèle CONGEST (cf. [67] page 27). Le premier modèle autorise des messages d'une taille illimitée alors que le second n'autorise que des messages d'une taille $O(\log n)$ (où n est la taille du réseau). Dans les deux modèles, les sommets ont des identifiants uniques.

Le modèle que nous étudions est anonyme (pas d'unicité des identifiants et pas de connaissance des identifiants pour les sommets) et les sommets n'ont pas de connaissance globale du réseau telle que la taille de ce dernier. Ainsi, dans ce contexte, lorsqu'un processus construit un message, sa taille ne peut dépendre de la taille du réseau et il devient alors naturel de considérer des messages composés d'un seul bit ou plus généralement des messages de taille bornée par une constante.

Enfin, la complexité en bits est considérée comme une mesure plus fine que la complexité de communication. Elle a été étudiée pour briser la symétrie ou pour la coloration par Bodlaender et al. [14], Bar-Noy et al. [8], Kothapalli et al. [43] ou par Dinitz et al. [24]. Dinitz et al. [24] expliquent qu'elle peut être vue comme une extension naturelle de la complexité de communication introduite par Yao [77] pour l'analyse des tâches dans un cadre distribué. Une introduction à ce domaine est faite par Kushilevitz et Nisan [47].

2.2 Couplage Maximal

Hanckowiak et al. [35] ont étudié la complexité du problème du couplage maximal et ont présenté un algorithme déterministe en $O(\log^4 n)$ (dans cet article, le modèle considéré permet une orientation des arêtes). Concernant le problème du couplage maximal dans les réseaux anonymes non-orientés, Israeli et Itai [40] ont présenté un algorithme distribué Las Vegas pour les graphes dont la complexité en temps et en bits est $O(\log n)$ avec forte probabilité. Cet algorithme est optimal.

2.3 Ensemble Maximal Indépendant

Peleg [67] a étudié de nombreux exemples d'algorithmes distribués Las Vegas pour le MIS. Le calcul d'un MIS fait l'objet de recherche intensive au niveau des complexités parallèle et distribuée (cf. [3, 54, 7, 52]). Karp et Widgerson [41] ont prouvé que le problème du MIS est de classe NC. Les problèmes de classe NC (pour *Nick's class*) sont des problèmes décisionnels qui peuvent être résolus en temps polylogarithmique sur une machine parallélisée ayant un nombre polynomial de processeurs.

La complexité de ce problème pour certaines classes de graphes tels que les graphes à croissance délimitée est étudiée dans Kuhn et al. [46]. Des résultats ont aussi été obtenus pour les réseaux radios [64]. On doit une contribution majeure à Luby [54] qui donne un algorithme distribué Las Vegas dont l'idée principale est d'obtenir pour chaque sommet un

ordre total local ou une élection locale qui brise la symétrie locale et permet à chaque sommet de décider localement s'il rejoint le MIS ou pas. La complexité en temps est $O(\log n)$ et en bits $O(\log^2 n)$. Récemment, un algorithme distribué Las Vegas a été présenté par Métivier et al. [61]. Cet algorithme améliore la complexité en bits : elle est optimale et égale à $O(\log n)$ avec forte probabilité.

2.4 Anneaux Anonymes

Kothapalli et al. [43] considèrent la famille des anneaux anonymes et montrent que si au plus un bit peut être envoyé sur chaque arête pendant une ronde, alors tout algorithme distribué Las Vegas pour la coloration des sommets (algorithme dans lequel chaque sommet a le même état initial et connaît initialement ses propres arêtes) a besoin de $\Omega(\log n)$ rondes avec forte probabilité pour colorer un anneau de taille n avec un nombre fini de couleurs. Avec les mêmes hypothèses, comme l'expliquent Métivier et al. [60], nous déduisons aussi qu'un algorithme distribué qui calcule un MIS a besoin de $\Omega(\log n)$ rondes avec forte probabilité.

Kothapalli et al. [43] ont aussi considéré la famille des anneaux orientés et ils ont prouvé que la complexité en bits de cette famille est $\Omega(\sqrt{\log n})$ avec forte probabilité.

2.5 Tableau Récapitulatif

Le tableau de la Figure 2.1 résume les résultats pour la complexité en temps et en bit pour le MIS, le couplage maximal et la coloration. Notons que les algorithmes correspondants n'ont pas besoin de connaissance initiale sur le graphe comme sa taille, la position ou l'identité des sommets.

	MIS	Couplage Maximal	Coloration
Graphes Généraux	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$
Anneaux	$\Theta(\sqrt{\log n})$	$\Theta(\sqrt{\log n})$	$\Theta(\log n)$
Anneaux orientés	$\Theta(\sqrt{\log n})$	$\Theta(\sqrt{\log n})$	$\Theta(\sqrt{\log n})$

Figure 2.1 – *Complexité en temps et en bits pour le MIS, le couplage maximal et la coloration.*

3 Algorithmes Distribués Probabilistes Optimaux pour leur Complexité en Bits pour les Problèmes du MIS et du Couplage Maximal dans les Anneaux Anonymes

Sommaire

Dans ce chapitre, nous présentons et analysons des algorithmes distribués de type Las Vegas qui calculent un MIS ou un couplage maximal dans les anneaux anonymes. Avec forte probabilité, ils ont une complexité en temps et en bits de l'ordre de $O(\sqrt{\log n})$. Ces algorithmes sont optimaux modulo une constante multiplicative.

Les résultats présentés dans ce chapitre ont été publiés dans le journal Information and Computation [28].

3.1 Introduction

Dans ce chapitre, nous considérons des anneaux. Même si les anneaux sont des graphes très simples, ils sont utilisés comme cas d'étude dans plusieurs problèmes, comme l'expliquent Attiya et Welch [5], page 31 : « les anneaux sont une structure appropriée pour les systèmes par passage de messages et correspondent à des systèmes de communication physique, par exemple, l'anneau à jeton » (ou *Token Ring*, protocole de réseau local).

Un fait important dû à la symétrie initiale de l'anneau est le suivant : *il n'y a pas d'algorithme distribué déterministe pour les anneaux anonymes qui résout le problème du MIS ou du couplage maximal en supposant que tous les sommets s'éveillent en même temps*, cf. [67].

Étant donné que les sommets de l'anneau sont anonymes et qu'ils ont le même degré, les algorithmes qu'ils exécutent sont identiques. La distribution de probabilité qui décrit leurs comportements est par conséquent la même.

L'intérêt de ce travail repose, non seulement sur les résultats concernant la complexité, mais aussi sur les descriptions et les analyses de ces algorithmes qui suivent le même schéma et donc qui peuvent être généralisées.

Ces résultats montrent une séparation entre les complexités des problèmes tels le MIS ou le couplage maximal par rapport au problème de la coloration. Une coloration nécessite $\Omega(\log n)$ rondes dans les anneaux et avec forte probabilité (cf. [43]) alors qu'il y a un algorithme plus rapide pour le MIS. Cela va à l'encontre d'autres modèles, dans lesquels le MIS ou le couplage maximal sont aussi difficiles que la coloration.

Kothapalli et al. [43] ont prouvé qu'un algorithme Las Vegas peut réaliser une coloration d'un anneau orienté avec une complexité en bits de $O(\sqrt{\log n})$ avec forte probabilité. Ce résultat est optimal à une constante près, ils ont démontré que la complexité en bits de la coloration dans un cycle orienté est $\Omega(\sqrt{\log n})$ avec forte probabilité. Ce qui conduit à la question de savoir si la complexité en bits du MIS ou du couplage maximal est plus petite dans un anneau orienté que la complexité du MIS ou du couplage maximal dans un anneau non-orienté. Comme il est aisé de déduire à partir d'un MIS (ou à partir d'un couplage maximal) une 3-coloration dans un anneau orienté, nous déduisons à partir du résultat de Kothapalli et al. [43] que la réponse est non.

En effet, soit G un anneau et I un MIS de G. Supposons, arbitrairement, que les sommets qui appartiennent à I sont dans l'état 1 et ceux qui appartiennent au complément sont dans l'état 0. Les sommets qui sont dans I prennent la couleur N. Pour les sommets qui sont dans le complément de I, deux configurations de leur voisinage sont possibles : $1-0-1$ ou $1-0-0-1$. Pour la première configuration, le sommet qui n'appartient pas à I prend la couleur B. Concernant la deuxième configuration, comme chaque arête a une orientation, celui qui est source de l'arête prend la couleur B et celui qui est cible prend la couleur G. Un exemple est donné dans la Figure 3.1.

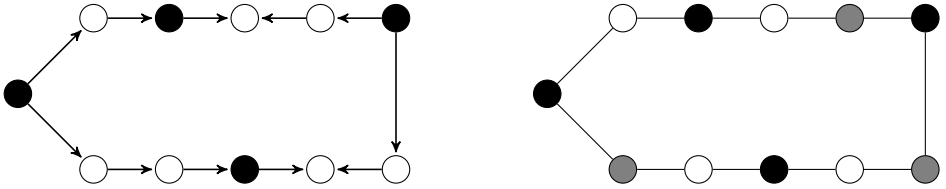


FIGURE 3.1 – *Soit* G *le graphe orienté ci-dessus. L'étiquetage de gauche nous donne un MIS de* G*, les sommets dans le MIS sont représentés en noir et ceux dans le complément en blanc. De cet étiquetage et de l'orientation des arêtes, nous tirons une* 3*-coloration représentée dans l'étiquetage de droite.*

3.2 Un Algorithme pour le Problème du MIS dans les Anneaux

3.2.1 Présentation de l'Algorithme `RingMIS`

Le résultat de l'algorithme `RingMIS` est codé pour chacun des sommets v dans son état local $\lambda(v)$. Un sommet v qui rejoint le MIS met son état $\lambda(v)$ à 1 et un sommet v qui rejoint le complément du MIS le met à 0. Une fois qu'il prend l'état 1 ou 0, un sommet a atteint son état final et devient inactif. Initialement, v est dans un état indéterminé : $\lambda(v) =?$. Un sommet peut prendre un état intermédiaire en mettant son état $\lambda(v)$ à X_0 ou X_1. Par conséquent, pour tout sommet v, on a : $\lambda(v) \in \{?, X_0, X_1, 0, 1\}$.

L'algorithme se déroule en phases, chaque phase est composée de 3 étapes : Tirage (T), Extension (E) et Remplissage (R). Une phase est définie comme étant l'exécution séquentielle de T, E et R, et sera notée TER. L'algorithme est donc une succession de phases TER que l'on notera $(TER)^*$.

L'algorithme est présenté par le biais de règles de réécriture s'appliquant sous certaines contraintes relatives au temps écoulé. Plus précisément, pour une phase TER, les règles de réécriture qui correspondent à T, E et R peuvent s'appliquer si le temps est correct modulo 3. Ce système peut aisément se convertir en un algorithme par passage de messages dans lequel chaque processus enregistre son propre état et un compteur modulo 3. De plus, après chaque étape T, E et R, chaque processus actif a besoin de 6 rondes pour collecter des informations afin qu'il puisse connaître les choix aléatoires ou les nouveaux états dans son voisinage à une distance inférieure ou égale à 6. Ci-dessous se trouve la description de chaque étape.

- Tirage. Chaque sommet dans l'état ? et appartenant à un sous-graphe connexe de taille 7 dont les sommets sont dans l'état ? choisit uniformément au hasard un sommet parmi ses deux voisins ou lui-même. Nous représentons l'action « le sommet v choisit le sommet w » par une flèche allant de v à w. L'état de chacun des sommets change suivant la règle de la Figure 3.2. Cette règle décrit ce qui se passe lorsqu'apparaît dans le graphe le motif se trouvant sur la première ligne. Si un tel motif ne se crée pas, les sommets restent dans le même état et sont prêts à prendre part à la prochaine étape. Notons que lorsqu'un sommet doit décider s'il est dans la configuration de la Figure 3.2, il doit examiner des potentiels 7-uplets dont un seul peut correspondre au motif.

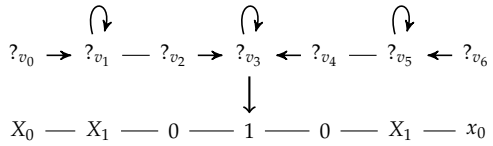


FIGURE 3.2 – *La règle de tirage : cette règle indique que si un sommet v_0, dans l'état ?, choisit un voisin v_1 qui s'est choisi lui-même et qui a un voisin v_2, etc, alors l'état de v_0 devient X_0, celui de v_1 devient X_1, celui de v_2 devient 0, etc.*

Remarque 3.1 *Les sommets dans l'état 0 ou 1 dans la Figure 3.2 sont dans un état final. À la fin de cette étape, les sommets v_2, v_3 et v_4 de cette figure sont dans un état final avec une probabilité d'au moins $(1/3)^7$.*

Remarque 3.2 *Si nous considérons la règle de tirage décrite dans la Figure 3.2, il apparaît clairement qu'un sommet ne peut se trouver dans l'obligation d'appliquer deux règles différentes durant la même ronde (modifier les états n'est pas ambigu).*

Une ***zone finale*** est un ensemble de sommets se trouvant dans un état final et formant un chemin. La propriété suivante est vérifiée dans une zone finale : les sommets d'une zone finale dans l'état 1 forment un ensemble indépendant maximal. Plus précisément, une zone finale est décrite par l'expression rationnelle : $01((0+00)1)^*0$. Un invariant, que l'on nommera par la suite I, est le suivant : une fois que l'étape de tirage T produit une zone finale, à chaque début d'étape, l'anneau est divisé en zones finales délimitées par des sommets qui sont dans un état intermédiaire et séparés par des ? comme le montre la Figure 3.3. On appelle ***zone finale étendue*** une zone finale à laquelle on ajoute les quatre sommets étiquetés X_1 et X_0 qui sont à ses extrémités. L'invariant I est valable jusqu'à ce que chaque sommet soit dans un état appartenant à $\{0,1\}$, et, dans ce cas, l'algorithme est terminé.

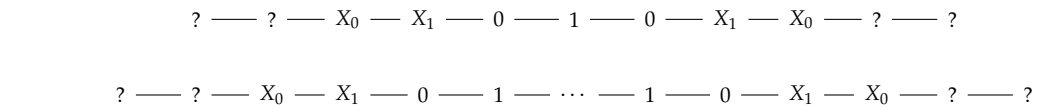


FIGURE 3.3 – *Forme des configurations apparaissant dans l'anneau.*

- Extension. Chaque zone finale a deux côtés et tente de s'étendre de deux sommets de chacun des côtés. L'extension se produit sur un côté s'il y a au moins quatre sommets indéterminés adjacents à ce côté en suivant la règle de la Figure 3.4.
  L'extension dans une direction a un impact sur quatre sommets : deux sommets obtiennent un état final (en changeant leur état X_i en i), les états des deux autres sommets deviennent des états intermédiaires. Les deux derniers sommets qui sont dans l'état ? peuvent ne pas changer et rester ? ou alors peuvent se changer en X_i si une extension a lieu de l'autre côté.
- Remplissage. Lorsque les zones finales étendues sont séparées par au plus six sommets, l'espace entre elles est rempli comme l'indique la Figure 3.5. Ainsi, l'étape de remplissage donne un état final à tous les sommets se trouvant dans de telles zones.

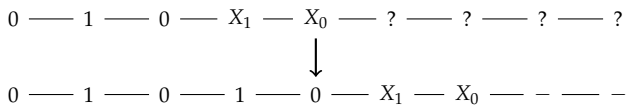


FIGURE 3.4 – *Règle pour l'étape d'extension.*

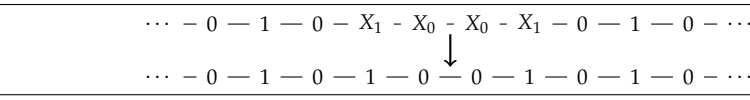
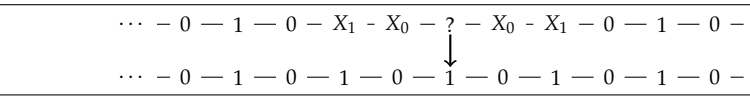
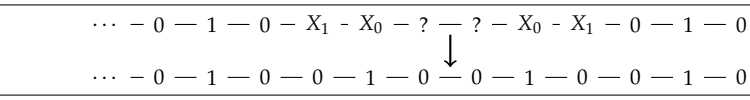
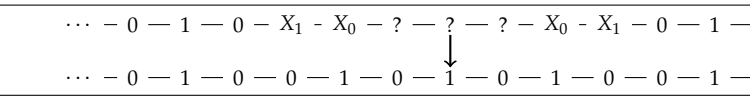
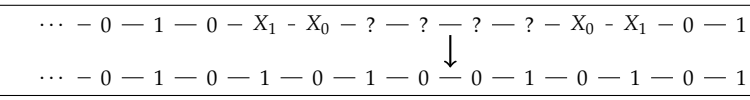
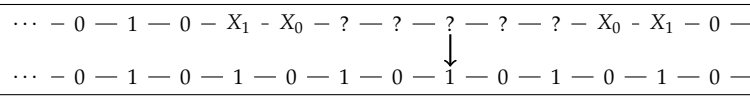
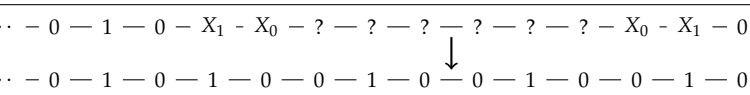
F_0	
F_1	
F_2	
F_3	
F_4	
F_5	
F_6	

FIGURE 3.5 – *Règles pour l'étape de remplissage. La règle F_0 indique que lorsque deux zones finales étendues se touchent, les états des sommets étiquetés X_0 (resp. X_1) deviennent 0 (resp. 1). La règle F_1 indique que s'il y a un seul sommet entre deux zones finales étendues, alors l'état de ce sommet devient 1 et les états des sommets étiquetés X_0 (resp. X_1) deviennent 0 (resp. 1). Et ainsi de suite pour les autres règles.*

Remarque 3.3 *Les règles F_4, F_5 et F_6 ne sont pas nécessaires. Si on les omet, les zones correspondantes seront remplies à la prochaine phase.*

Remarque 3.4 *L'invariant I concernant la configuration de l'anneau décrite dans Figure 3.3 est préservé par chaque étape.*

Remarque 3.5 *Une fois qu'une zone est formée, à chaque étape, le nombre de sommets dans un état final est augmenté d'au moins 4. En effet, si l'étape d'extension se produit alors c'est trivial. Sinon, cela signifie que l'étape de remplissage se produira et dans ce cas, la preuve est directe.*

Remarque 3.6 *L'algorithme est valable seulement pour les graphes qui ont au moins 7 sommets. Pour de plus petits graphes, on peut appliquer un autre algorithme se réalisant en temps constant.*

3.2.2 Analyse de l'Algorithme

Nous avons le lemme suivant :

Lemme 3.1 *Soit v un sommet qui entre dans le MIS ou son complément au temps $t_0 > 0$. Alors, pour tout $t \geq 0$, après t phases, tous les sommets à distance au moins $2t$ de v sont soit dans le MIS soit dans son complément.*

Démonstration. La preuve se déduit directement de la Remarque 3.5 puisque toute zone finale est incrémentée d'au moins quatre sommets à la fin de chaque phase. □

Théorème 3.1 *L'algorithme* `RingMIS` *calcule un MIS dans un anneau de taille n en* $O(\sqrt{\log\ n})$ *rondes avec forte probabilité.*

Démonstration. Pour tout $u \in V$, et tout $t \geq 0$, soit $P(u,t)$ le chemin de taille $2t+1$ centré sur u. Et pour tout $i \in \{1,\cdots,t\}$, soit $E(u,i)$ (resp. $\overline{E(u,i)}$) l'évènement « *un sommet dans le chemin* $P(u,2(t-i))$ *entre dans l'ensemble à la phase i* » (resp. « *aucun sommet dans le chemin* $P(u,2(t-i))$ *n'entre dans l'ensemble à la phase i* »).

Soit v un sommet qui n'est pas encore dans le MIS ni dans son complément. D'après le Lemme 3.1, v entrera dans l'ensemble ou dans son complément dans les t phases qui suivent sauf si aucun des évènements $E(v,i)$, pour tout $i \in \{1,\cdots,t\}$, ne se produit.

D'autre part, tout sommet u dans $P(v,2t)$ à distance 6 ou plus de toute zone finale entrera dans l'ensemble au cours d'une phase s'il est au centre du motif de la Figure 3.2 page 39, et ce avec probabilité au moins $\pi = \left(\frac{1}{3}\right)^7$ indépendamment de ce qui peut arriver à l'extérieur de ces sept sommets.

Par conséquent :

$$\mathbb{P}r\left(\overline{E(u,i)}\right) \leq (1-\pi)^{\frac{4(t-i)+1}{7}}.$$

Maintenant, soit $p_{>t}(v)$ la probabilité que v n'entre pas dans l'ensemble ou dans son complément dans les t phases, nous avons :

$$\begin{aligned} p_{>t}(v) &\leq \prod_{i=1}^{t} \mathbb{P}r\left(\overline{E(u,i)}\right) \qquad (3.1)\\ &\leq \prod_{i=1}^{t} (1-\pi)^{\frac{4(t-i)+1}{7}} \\ &= (1-\pi)^{\frac{2t^2-t}{7}} \leq (1-\pi)^{\frac{t^2}{7}}. \end{aligned}$$

En prenant $t = 175\sqrt{\log n}$, nous obtenons :

$$\begin{aligned} p_{>175\sqrt{\log n}}(v) &\leq (1-\pi)^{\frac{1}{7}175^2 \log n} \\ &\sim e^{-\frac{1}{7}175^2\pi \log n} \text{ quand } n \to \infty \\ &= o\left(\frac{1}{n^2}\right) \text{ quand } n \to \infty. \qquad (3.2) \end{aligned}$$

Remarque 3.7 $\frac{1}{7}175^2\pi \simeq 2,000457$.

Nous obtenons ainsi qu'un sommet dans le chemin $P(v,2t_0)$, où $t_0 = 175\sqrt{\log n}$, entrera dans l'ensemble indépendant maximal dans les $175\sqrt{\log n}$ phases suivantes avec une probabilité $1 - o\left(n^{-2}\right)$. De plus, l'extension de la zone finale de ce sommet inclura v dans les $175\sqrt{\log n}$ phases suivantes (puisqu'une zone finale s'étend de deux sommets à la fin de chaque phase). En tenant compte des n sommets du graphe, nous voyons que la probabilité qu'un sommet reste indéterminé après $350\sqrt{\log n}$ phases est $o\left(n^{-1}\right)$. □

Nous en déduisons le corollaire :

Corollaire 3.1 *L'algorithme* `RingMIS` *calcule un MIS dans un anneau de taille n en $O(\sqrt{\log\ n})$ rondes en moyenne.*

Démonstration. Si un sommet reste indéterminé après $350\sqrt{\log\ n}$ phases, soit tous les sommets sont indéterminés et le temps restant est exactement le temps initial (en moyenne), soit quelques sommets sont déterminés et tous les sommets seront dans un état final dans, au plus, $n/4$ phases supplémentaires. Il s'ensuit que le temps moyen est aussi $O(\sqrt{\log\ n})$. □

3.3 Un Algorithme pour le Problème du Couplage Maximal

3.3.1 Présentation de l'Algorithme `RingMM`

Comme pour le problème du MIS, l'algorithme pour le problème du couplage maximal, `RingMM`, est une séquence de la forme $(TER)^*$: Tirage, Extension et Remplissage. La sortie est toujours définie pour chaque sommet v par la variable $\lambda(v)$. Un sommet v rejoignant le couplage maximal met sa variable $\lambda(v)$ à 1. Un sommet v ne rejoignant pas le couplage maximal met $\lambda(v)$ à 0. Une fois qu'un sommet a pris l'état 0 ou 1, il est dans un état final et devient inactif. Initialement, v est dans un état indéterminé : $\lambda(v)$ =?. Un sommet peut prendre un état intermédiaire en donnant la valeur X à $\lambda(v)$. Au final, on a : $\lambda(v) \in \{?, X, 0, 1\}$.

Chaque extrémité d'une arête qui entre dans le couplage maximal sait avec qui elle est liée grâce aux numéros de ports. La représentation d'une arête entrant dans le couplage maximal est comme suit : $1 \overset{*}{-} 1$.

Dans le cas du problème du couplage maximal, une ***zone finale*** est définie comme un ensemble de sommets formant un chemin et étant dans un état final. Cette zone vérifie la propriété suivante : le sous-ensemble formé par les arêtes étiquetées avec $*$ est un couplage maximal. Plus précisément, la zone est décrite par l'expression suivante : $(1 \overset{*}{-} 1) + ((1 \overset{*}{-} 1) + (0 - 1 \overset{*}{-} 1))^*$. Ici, l'invariant I de l'algorithme est basé sur la configuration donnée dans la Figure 3.6

$$X \overset{*}{—} X — 1 \overset{*}{—} 1 — X \overset{*}{—} X$$

$$X \overset{*}{—} X — 1 \overset{*}{—} 1 — [(1 \overset{*}{-} 1) + ((1 \overset{*}{-} 1) + (0 - 1 \overset{*}{-} 1))^*] — 1 \overset{*}{—} 1 — X \overset{*}{—} X$$

Figure 3.6 – *Forme des configurations apparaissant dans l'anneau.*

Tout comme pour le MIS, l'algorithme pour le couplage maximal est présenté comme un système de règles de réécriture. Ce système peut être converti en un algorithme par passage de messages dans lequel chaque processus enregistre son propre état et un compteur modulo 3. De plus, à chaque étape T, E et R, chaque processus actif échange des messages d'un bit pour connaître les résultats des choix aléatoires et des états des sommets à distance 1, 2, jusqu'à 5, dans le but de reconstituer le sous-graphe correspondant.

Remarque 3.8 *Une arête est étiquetée avec* $1 \stackrel{*}{-} 1$ *pour représenter le fait qu'elle est dans le couplage maximal. Cette information est connue par chacun des deux sommets grâce à la numérotation de ports.*

Comme précédemment, nous décrivons les différentes règles de réécriture pour chacune des étapes.

- Tirage. Chaque sommet avec un état ? choisit uniformément au hasard un sommet parmi ses deux voisins. Nous représentons l'action « un sommet v choisit un sommet w » par une flèche allant de v à w. Deux sommets qui se sont choisis mutuellement changent leur état ? en 1 ou X selon les deux règles suivantes :
  - l'état ? est changé en 1 si l'arête est entourée des deux côtés par une arête dont les extrémités se sont choisies,
  - l'état ? est changé en X si l'arête est entourée d'un côté par une arête dont les extrémités se sont choisies et de l'autre côté par une arête dont les extrémités ne se sont pas choisies mutuellement,
  - aucun changement dans les autres cas.

  La règle décrivant le tirage est illustrée dans la Figure 3.7.

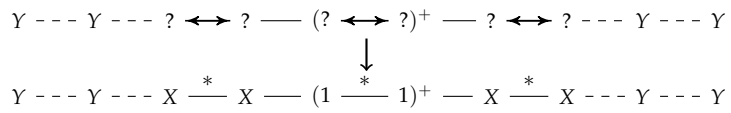


FIGURE 3.7 – *Règle pour le tirage. La paire* $(Y \cdots Y)$ *peut être l'un des motifs suivants :* $(-? \to ? \to)$, $(\leftarrow ? \leftarrow ? -)$, $(\leftarrow ? - ? \to)$, $(X \stackrel{*}{-} X)$, $(-? \to X \stackrel{*}{-})$, $(\stackrel{*}{-} X - ? \to)$, *etc.*

Remarque 3.9 *Si une arête est étiquetée* $1 \stackrel{*}{-} 1$*, alors elle est dans le couplage maximal et son état est final.*

- Extension. L'extension se produit dans une direction s'il y a au moins quatre sommets indéterminés adjacents à l'extrémité suivant la règle de la Figure 3.8.

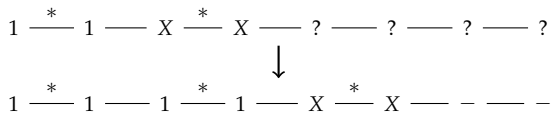


FIGURE 3.8 – *La règle pour l'extension.*

Remarque 3.10 *Les états des deux derniers sommets dans un état ? peuvent rester dans le même état ou peuvent changer en X. Cela est représenté dans la figure par un* $-$*. En effet, la règle peut avoir lieu de l'autre côté de l'anneau, ce qui implique un changement d'état.*

- Remplissage. Lorsque les zones finales sont séparées d'au plus cinq sommets, l'espace entre elles est rempli comme indiqué sur la Figure 3.9.
  La règle F_0 décrit le comportement quand deux zones finales sont adjacentes. La règle F_1 décrit le comportement lorsque deux zones finales sont séparées par un sommet dans l'état ?. Plus généralement, F_i $(i \leq 5)$ décrit le comportement lorsque deux zones finales sont séparées par i sommets. L'état intermédiaire X et l'état indéterminé

F_0	$\cdots - 1 \overset{*}{-} 1 - X \overset{*}{-} X - X \overset{*}{-} X - 1 \overset{*}{-} 1 - \cdots$ $\downarrow$ $\cdots - 1 \overset{*}{-} 1 - 1 \overset{*}{-} 1 - 1 \overset{*}{-} 1 - 1 \overset{*}{-} 1 - \cdots$
F_1	$\cdots - 1 \overset{*}{-} 1 - X \overset{*}{-} X - ? - X \overset{*}{-} X - 1 \overset{*}{-} 1 - \cdots$ $\downarrow$ $\cdots - 1 \overset{*}{-} 1 - 1 \overset{*}{-} 1 - 0 - 1 \overset{*}{-} 1 - 1 \overset{*}{-} 1 - \cdots$
F_2	$\cdots - 1 \overset{*}{-} 1 - X \overset{*}{-} X - ? - ? - X \overset{*}{-} X - 1 \overset{*}{-} 1 - \cdots$ $\downarrow$ $\cdots - 1 \overset{*}{-} 1 - 1 \overset{*}{-} 1 - 1 \overset{*}{-} 1 - 1 \overset{*}{-} 1 - 1 \overset{*}{-} 1 - \cdots$
F_3	$\cdots - 1 \overset{*}{-} 1 - X \overset{*}{-} X - ? - ? - ? - X \overset{*}{-} X - 1 \overset{*}{-} 1 - \cdots$ $\downarrow$ $\cdots - 1 \overset{*}{-} 1 - 0 - 1 \overset{*}{-} 1 - 0 - 1 \overset{*}{-} 1 - 0 - 1 \overset{*}{-} 1 - \cdots$
F_4	$\cdots - 1 \overset{*}{-} 1 - X \overset{*}{-} X - ? - ? - ? - ? - X \overset{*}{-} X - 1 \overset{*}{-} 1 - \cdots$ $\downarrow$ $\cdots - 1 \overset{*}{-} 1 - 1 \overset{*}{-} 1 - 1 \overset{*}{-} 1 - 1 \overset{*}{-} 1 - 1 \overset{*}{-} 1 - 1 \overset{*}{-} 1 - \cdots$
F_5	$\cdots - 1 \overset{*}{-} 1 - X \overset{*}{-} X - ? - ? - ? - ? - ? - X \overset{*}{-} X - 1 \overset{*}{-} 1 - \cdots$ $\downarrow$ $\cdots - 1 \overset{*}{-} 1 - 1 \overset{*}{-} 1 - 1 \overset{*}{-} 1 - 0 - 1 \overset{*}{-} 1 - 1 \overset{*}{-} 1 - 1 \overset{*}{-} 1 - \cdots$

FIGURE 3.9 – *Règles pour le remplissage.*

? sont changés en états finaux (1 ou 0) en assurant un couplage maximal.

Remarque 3.11 *L'invariant I concernant la configuration de l'anneau décrit dans la Figure 3.6 est préservé par chaque phase. À la fin de l'algorithme, il ne reste plus que des sommets en état final.*

Remarque 3.12 *Comme dans le MIS, à chaque phase, le nombre de sommets d'une zone finale est incrémenté par au moins 4.*

Remarque 3.13 *L'algorithme est valable seulement pour les graphes d'au moins 6 sommets.*

3.3.2 Analyse de l'Algorithme

On a le théorème suivant :

Théorème 3.2 *L'algorithme* `RingMM` *calcule un couplage maximal dans un anneau de taille n en* $O(\sqrt{\log\ n})$ *rondes avec forte probabilité et en moyenne.*

Démonstration. Comme pour la preuve du Théorème 3.1 page 41, l'idée principale est que lorsqu'un sommet v entre dans son état final au temps t_0, *i.e.*, $\lambda(v)$ appartient à l'ensemble $\{0, 1\}$, tous les sommets à distance au plus $2t$ en partant de v entrent dans leur état final au temps au plus $t_0 + t$. D'autre part, tout sommet à distance 4 ou plus d'une zone finale entrera dans le couplage maximal dans une phase si son état est 0 ou 1 dans l'une des configurations décrites dans la Figure 3.7 et ce avec probabilité au moins $\pi = \frac{1}{2^6}$ indépendamment de ce qui se passe à l'extérieur de la configuration. Soit $p_{>t}(v)$ la probabilité pour qu'un sommet v obtienne son état final, une adaptation de la preuve du Théorème 3.1 nous mène à :

$$p_{>30\sqrt{\log n}}(v) \quad = \quad o\left(\frac{1}{n^2}\right) \text{ quand } n \to \infty. \tag{3.3}$$

D'où, la probabilité qu'un sommet reste indéterminé après $60\sqrt{\log n}$ phases est $o\left(n^{-1}\right)$.

La seconde affirmation est aussi prouvée en utilisant le même argument que celui utilisé pour le Corollaire 3.1 page 42. □

3.4 Calcul d'un 2-MIS

Nous avons présenté et analysé des algorithmes distribués Las Vegas qui calculent un MIS ou un couplage maximal pour des anneaux anonymes. Leur complexité en bit et en temps sont $O(\sqrt{\log\ n})$. Ces algorithmes sont optimaux modulo une constante multiplicative.

La description et l'analyse de ces algorithmes peuvent être généralisées. Nous illustrons cette idée à travers le calcul d'un 2-MIS.

Un ensemble 2-indépendant est un sous-ensemble K de V tel que la distance entre deux sommets de K est au moins 3. Un ensemble 2-indépendant K est dit maximal (2-MIS) si tout sommet de G est dans K ou à distance 1 ou 2 d'un élément de K.

Comme pour l'algorithme `RingMIS`, la sortie de l'algorithme est définie sur chaque sommet v par une variable $\lambda(v)$ qui représente l'état de v. Un sommet v rejoignant le 2-MIS met à 1 son état $\lambda(v)$ et un sommet v ne rejoignant pas le 2-MIS le met à 0. Une fois qu'un sommet a pris la valeur 0 ou 1, il est dans un état final et devient inactif. Initialement, v est dans un état indéterminé : $\lambda(v) =?$. Un sommet peut prendre un état intermédiaire X_0 ou X_1. Par conséquent, on a : $\lambda(v) \in \{?, X_0, X_1, 0, 1\}$.

L'algorithme `Ring2MIS` se déroule en phases, chaque phase est composée de 3 étapes : Tirage (T), Extension (E) et Remplissage (R). Une phase est définie comme une exécution séquentielle de T, E et R : TER. L'algorithme est donc une séquence de phases : $(TER)^*$.

L'algorithme est présenté comme un ensemble de règles de réécriture. Pour une phase TER, les règles de réécriture qui correspondent à T, E et R sont applicables si le temps est correct modulo 3. Ce système peut aisément être converti en un algorithme par passage de messages dans lequel chaque processus enregistre son propre état et un compteur modulo 3. De plus, à chaque étape T, E et R, chaque processus actif échange des messages d'un bit pour connaître les résultats des choix aléatoires et des états des sommets à distance 1, 2 jusqu'à 10, tout cela dans le but de reconstituer le sous-graphe correspondant.

Voici la présentation de chacune des étapes.

- Tirage. Chaque sommet dans un état ? et appartenant à un sous-graphe connexe de l'anneau de taille 11 dont les sommets sont dans l'état ? choisit uniformément au hasard un sommet parmi : ses deux voisins ou lui-même. Nous représentons l'action « le sommet v choisit le sommet w » par une flèche allant de v à w. Il advient que l'état de chaque sommet change selon la Figure 3.10
- Extension. Chaque zone finale essaie de s'étendre de 3 sommets dans chacune des deux directions. L'extension se produit dans une direction s'il y a au moins 6 sommets indéterminés adjacents à l'extrémité selon la règle suivante (nous indiquons seulement l'état des sommets en abrégé) : $??????X_0X_0X_100100X_1X_0X_0??????$ devient $???X_0X_0X_100100100100X_1X_0X_0???$.

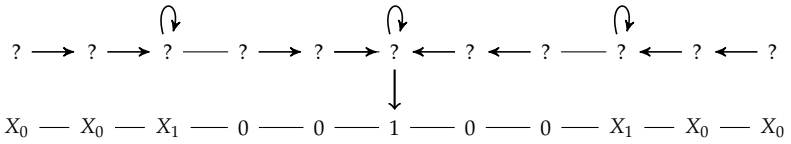


FIGURE 3.10 – *La règle de tirage.*

L'extension dans une direction n'a d'impact que sur 6 sommets : 3 sommets obtiennent un état final (en changeant X_i en i), les états des 3 autres sommets deviennent intermédiaires. Les 3 derniers sommets avec l'état ? peuvent ne pas changer et rester ? ou changer en X_i à cause de l'extension qui pourrait se produire de l'autre côté.

– Remplissage. Lorsque les zones finales étendues sont séparées d'au plus 10 sommets, l'espace entre elles est rempli comme suit (nous donnons seulement les états de la fin et du début des zones finales) :
  – F_0 : $100X_1X_0X_0X_0X_0X_1001$ devient 100100001001,
  – F_1 : $100X_1X_0X_0?X_0X_0X_1001$ devient 1001001001001,
  – F_2 : $100X_1X_0X_0??X_0X_0X_1001$ devient 01001000010010,
  – F_3 : $100X_1X_0X_0???X_0X_0X_1001$ devient 100100010001001,
  – F_4 : $100X_1X_0X_0????X_0X_0X_1001$ devient 1001001001001001,
  – F_5 : $100X_1X_0X_0?????X_0X_0X_1001$ devient 10010000100001001,
  – F_6 : $100X_1X_0X_0??????X_0X_0X_1001$ devient 100100100001001001,
  – F_7 : $100X_1X_0X_0???????X_0X_0X_1001$ devient 1001001001001001001,
  – F_8 : $100X_1X_0X_0????????X_0X_0X_1001$ devient 10010000100100001001,
  – F_9 : $100X_1X_0X_0?????????X_0X_0X_1001$ devient 100100100010001001001,
  – F_{10} : $100X_1X_0X_0??????????X_0X_0X_1001$ devient 1001000010000100001001.

Remarque 3.14 *Comme pour le MIS, les règles F_6, F_7, F_8, F_9 et F_{10} ne sont pas nécessaires. Si elles sont omises, les zones correspondantes seront remplies à la prochaine phase.*

Finalement, on peut prouver :

Théorème 3.3 *L'algorithme* `Ring2MIS` *calcule un 2-MIS dans un anneau de taille n en $O(\sqrt{\log\ n})$ rondes avec forte probabilité et en moyenne.*

4 Bornes Inférieures pour la Complexité en Temps et en Bits de quelques Algorithmes Distribués Probabilistes

Sommaire

Nous nous intéressons dans ce chapitre à des algorithmes distribués probabilistes qui résolvent les problèmes classiques tels la coloration, le couplage maximal ou l'ensemble indépendant maximal.

Nous considérons des réseaux anonymes où les sommets communiquent par message d'un seul bit. Nous présentons une méthode générale, basée sur les revêtements, pour prouver des bornes inférieures pour la complexité en bits et donc pour le temps d'exécution nécessaire pour résoudre ces problèmes.

Dans ce cadre, nous obtenons de nouvelles preuves de quelques résultats déjà connus ainsi que de nouveaux résultats. Nous donnons aussi un résultat d'impossibilité par rapport à l'existence d'un algorithme distribué Las Vegas qui casserait la symétrie à distance k pour $k \geq 3$.

Les résultats présentés dans ce chapitre ont été exposés à la conférence SOFSEM 2014 [29].

4.1 Introduction

Pour de nombreux problèmes sur les graphes, des bornes inférieures de la complexité en bits et du temps d'exécution d'un algorithme distribué probabiliste peuvent être obtenues simplement en considérant des graphes non-connexes. Ces résultats peuvent sembler non satisfaisants puisque le plus souvent la communauté s'intéresse à des graphes connexes. Nous présentons ici une méthode générale, basée sur les revêtements, permettant de prouver des bornes inférieures pour la complexité en temps sur des graphes connexes ou non et nous l'appliquons à des problèmes comme la coloration, le couplage maximal, le MIS ou encore sur des généralisations de ces problèmes.

Les problèmes du MIS ou de la coloration peuvent être généralisés à une distance k pour tout entier positif k. Plus précisément, soit $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ un graphe et k un entier positif, un ensemble k-indépendant est un sous-ensemble I de V tel que la distance entre deux sommets de I est supérieure ou égale à $k+1$. Si I est maximal pour cette propriété, alors I est un ensemble k-indépendant maximal (k-MIS en abrégé). Une coloration à distance k de G est une coloration des sommets de G telle que deux sommets différents reliés par un chemin d'au plus k arêtes sont de couleurs différentes.

Les principales contributions de ce travail sont des constructions générales, basées sur les revêtements, qui présentent d'une façon unifiée des preuves sur les bornes inférieures pour la complexité en temps et la complexité en bits pour quelques problèmes de graphes. Quelques unes de ces bornes inférieures sont déjà connues, d'autres sont nouvelles.

Plus précisément, grâce aux revêtements, nous construisons des familles infinies de graphes connexes ou non-connexes K (soit n le nombre de sommets de K) munis d'une fonction de numérotation de ports telles que tout algorithme distribué Las Vegas qui utilise des messages d'un bit ne peut briser certaines symétries dans K (*i.e.*, certains sommets restent dans le même état) après $c\sqrt{\log n}$ ou $c \log n$ rondes (pour une certaine constante c) avec forte probabilité.

De ces constructions et résultats, nous déduisons que :

- résoudre les problèmes comme ceux du MIS [1], de la coloration[1], du couplage maximal [1], du 2-MIS ou de la coloration à distance 2 prend $\Omega(\log n)$ rondes avec forte probabilité pour une famille infinie de graphes non-connexes ;
- résoudre les problèmes du MIS ou du couplage maximal prend $\Omega(\sqrt{\log n})$ rondes avec forte probabilité pour une famille infinie d'anneaux ;
- résoudre les problèmes comme le MIS[1], la coloration [1], le couplage maximal, le 2-MIS ou la coloration à distance 2 prend $\Omega(\log n)$ rondes avec forte probabilité pour une famille infinie de graphes connexes.

Remarque 4.1 *Le nombre de sommets des graphes dans chaque famille infinie que nous avons construite pour obtenir des bornes inférieures est de la forme $\alpha n_G + 1$ avec $\alpha \geq 1$ où n_G est le nombre de sommets d'un graphe donné G.*

1. Résultats déjà connus.

Nous déduisons aussi que l'algorithme distribué Las Vegas pour le couplage maximal (resp. coloration, resp. MIS) présenté par Israeli et Itai [40] (resp. Métivier et al. [60], resp. Métivier et al. [61]) est optimal (pour la complexité en temps et en bits) modulo une constante multiplicative. Plus précisément, la complexité en bits des solutions présentées dans ces articles est $O(\log n)$ avec forte probabilité pour les graphes anonymes ayant n sommets.

Remarque 4.2 *L'algorithme pour la coloration présenté par Métivier et al. [60] peut être généralisé pour obtenir un algorithme pour la coloration à distance* 2 *avec une complexité en bit* $O(\log n)$ *avec forte probabilité pour les graphes anonymes ayant n sommets.*

Ces résultats peuvent se résumer :

Théorème 4.1 *La complexité en bits pour le problème du MIS, le problème de la coloration, le problème du couplage maximal et le problème de la coloration à distance* 2 *est* $\theta(\log n)$ *avec forte probabilité pour les graphes anonymes avec n sommets.*

Si nous considérons le cas particulier des anneaux, nous avons prouvé ([28], Chapitre 3) que $O(\sqrt{\log n})$ rondes sont suffisantes avec forte probabilité pour calculer un MIS ou un couplage maximal où n est le nombre de sommets de l'anneau. On en déduit donc que la complexité en bits pour le problème du MIS ou du couplage maximal est $\theta(\sqrt{\log n})$ avec forte probabilité pour les anneaux anonymes avec n sommets.

La dernière section donne un résultat d'impossibilité sur l'existence des algorithmes distribués Las Vegas pour briser des « symétries à distance k » dans des graphes anonymes pour $k \geq 3$, comme le problème du k-MIS ou le problème de la coloration à distance k.

4.2 Préliminaires

Cette section aborde les notions de revêtements. Elle explique comment les revêtements sont utilisés dans le contexte des algorithmes distribués.

4.2.1 Graphes (Orientés) Étiquetés et Algorithmes Distribués Synchrones

Soit $\mathbf{G} = (\mathsf{G}, \lambda)$ un graphe étiqueté avec une numérotation de ports δ. Soit $\mathbf{D} = Dir(\mathbf{G}, \delta')$ le graphe orienté étiqueté correspondant à $(Dir(\mathsf{G}), (\lambda, \delta'))$ (cf. Définition 1.49 page 23). Soit $\mathcal{A}$ un algorithme distribué synchrone. Nous parlons indifféremment d'une exécution de $\mathcal{A}$ sur $(\mathbf{G}, \delta)$ ou sur $\mathbf{D}$. L'état de chaque processus est représenté par l'étiquette $\lambda(v)$ du sommet correspondant à v. Soit $\mathbf{D}' = (Dir(\mathbf{G}'), \delta')$ un graphe orienté étiqueté obtenu par l'application d'un pas de $\mathcal{A}$ à $\mathbf{D}$. Nous rappelons que le système est synchrone, par conséquent chaque sommet de D a un nouvel état calculé par une fonction de transition qui dépend de l'état du sommet et des messages qu'il a reçus. Cette transition est notée par :

$$(Dir(\mathbf{G}), \delta') \underset{\mathcal{A}}{\Longrightarrow} (Dir(\mathbf{G}'), \delta')$$

ou indifféremment

$$(\mathbf{G}, \delta) \underset{\mathcal{A}}{\Longrightarrow} (\mathbf{G}', \delta').$$

Soit v un sommet de $\mathbf{G}$. Nous notons par $(v, \lambda(v)) \underset{\mathcal{A}}{\Longrightarrow} (v, \lambda'(v))$ la transition associée au sommet v de G.

Soit r un entier positif. Une séquence $(\mathbf{D}_i)_{0 \leq i \leq r}$ de graphes orientés étiquetés est appelée une ***$\mathcal{A}$-exécution*** (ou une ***exécution*** lorsque $\mathcal{A}$ se déduit clairement du contexte) de taille r si $\mathbf{D}_{i+1}$ s'obtient à partir de $\mathbf{D}_i$ en un pas d'une exécution de $\mathcal{A}$. Elle est notée par $\mathbf{D}_i \mathcal{A} \mathbf{D}_{i+1}$ pour tout $0 \leq i < r$. Une exécution de taille 1 est un ***pas***. De plus, si $\mathcal{A}$ est un algorithme distribué probabiliste synchrone et si l'exécution d'un pas a une probabilité p, alors on la notera : $\mathbf{G} \underset{p}{\Longrightarrow} \mathbf{G}'$.

Nous supposons que les choix des sommets sont indépendants.

Remarque 4.3 *Soit $\mathcal{A}$ un algorithme distribué probabiliste synchrone. Soit G un graphe. Soit δ une numérotation de ports de G. Soit λ un étiquetage de G. Nous avons : $(\mathbf{G}, \delta) \underset{p}{\Longrightarrow} (\mathbf{G}', \delta)$ (avec $\mathbf{G}' = (\mathsf{G}, \lambda')$) si et seulement si p est égal au produit sur $\mathsf{V}(\mathsf{G})$ des probabilités de transitions $(v, \lambda(v)) \underset{\mathcal{A}}{\Longrightarrow} (v, \lambda'(v))$.*

4.2.2 Revêtements et Algorithmes Distribués Synchrones

Cette section présente notre principal outil : les revêtements symétriques et leur utilisation dans les algorithmes distribués synchrones. La notion de revêtement symétrique (cf. Définition 1.53 page 25) nous permet d'exprimer des « similarités » entre deux graphes orientés et de prouver l'existence d'exécutions d'un algorithme qui ne brisent pas certaines symétries à l'intérieur de certains sous-ensembles de sommets d'un graphe (et du graphe orienté associé) : à savoir à l'intérieur des fibres. Des définitions et des propriétés principales sur les revêtements sont données par Boldi et Vigna [16].

Deux exemples de graphes orientés étiquetés symétriques sont donnés dans la Figure 4.1.

Remarque 4.4 *Soit $\mathbf{D}$ un revêtement étiqueté symétrique de $\mathbf{D}'$ via φ. Soit δ' une numérotation de ports de $\mathbf{D}'$ et soit δ une numérotation de ports de $\mathbf{D}$ induite par φ^{-1}. Soit $\mathcal{A}$ un algorithme distribué synchrone. Nous considérons l'exécution d'une ronde de $\mathcal{A}$ sur $(\mathbf{D}', \delta')$. Cette ronde peut être relevée en $(\mathbf{D}, \delta)$ de la façon suivante :*

1. *« le sommet $v' \in \mathsf{V}(\mathsf{D}')$ envoie des messages à ses voisins » devient « chaque sommet de $\varphi^{-1}(v') \subseteq \mathsf{V}(\mathsf{D})$ envoie des messages à ses voisins » (les mêmes messages sont envoyés depuis v' et depuis chaque sommet w de $\varphi^{-1}(v')$ à travers les mêmes numéros de ports vers les voisins de v' et w) ;*
2. *« le sommet $v' \in \mathsf{V}(\mathsf{D}')$ reçoit des messages de ses voisins » devient « chaque sommet w de $\varphi^{-1}(v') \subseteq \mathsf{V}(\mathsf{D})$ reçoit des messages de ses voisins à travers les mêmes numéros de ports que v' » ;*
3. *« le sommet $v' \in \mathsf{V}(\mathsf{D}')$ réalise des calculs locaux » devient « chaque sommet de $\varphi^{-1}(v') \subseteq \mathsf{V}(\mathsf{D}')$ réalise des calculs locaux » (les mêmes calculs locaux sont réalisés sur v' et sur chaque sommet de $\varphi^{-1}(v')$, par conséquent, le sommet v' et les sommets de $\varphi^{-1}(v')$ sont dans le même état).*

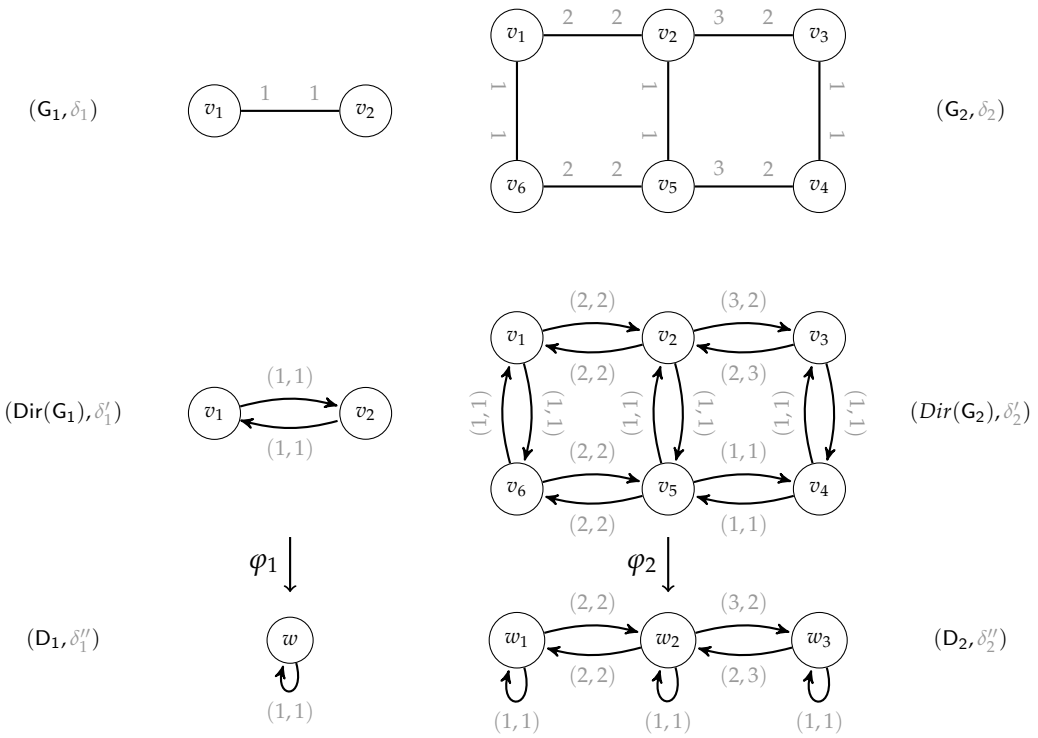


FIGURE 4.1 – *Le graphe orienté $Dir(\mathsf{G}_1)$ (resp. $Dir(\mathsf{G}_2)$) est un revêtement symétrique via φ_1 (resp. φ_2) de D_1 (resp. D_2) où φ_1 associe les sommets de $Dir(\mathsf{G}_1)$ sur l'unique sommet de D_1 et $\varphi_2(v_1) = \varphi_2(v_6) = w_1$, $\varphi_2(v_2) = \varphi_2(v_5) = w_2$ et $\varphi_2(v_3) = \varphi_2(v_4) = w_3$. Le nombre de feuillets de ces deux revêtements est 2. La numérotation de ports δ''_1 (resp. δ''_2) de D_1 (resp. D_2) induit via φ_1^{-1} (resp. φ_2^{-1}) la numérotation de ports δ'_1 (resp. δ'_2) de $Dir(\mathsf{G}_1)$ (resp. $Dir(\mathsf{G}_2)$) et donc la numérotation de ports δ_1 (resp. δ_2) de G_1 (resp. G_2).*

Finalement, nous obtenons le lemme classique suivant (cf. [4, 20]) :

Lemme 4.1 *Soit $\mathbf{D}$ un revêtement étiqueté symétrique de $\mathbf{D}'$ via φ. Soit δ' une numérotation de ports de $\mathbf{D}'$ et soit δ une numérotation de ports de $\mathbf{D}$ induite par φ^{-1}. Alors toute exécution d'un algorithme distribué $\mathcal{A}$ sur $(\mathbf{D}', \delta')$ peut être relevée en une exécution sur $(\mathbf{D}, \delta)$, de telle façon qu'à la fin de l'exécution, pour tout sommet $v \in \mathsf{V}(\mathsf{D})$, v est dans le même état que $\varphi(v)$.*

Nous nous intéressons ici à certains problèmes comme le calcul d'un MIS, la coloration des sommets ou le calcul d'un couplage maximal. Dans chacun de ces cas, nous avons à briser certaines symétries à l'intérieur du graphe étiqueté. Pour briser les symétries à l'intérieur d'un graphe étiqueté, il suffit de distinguer un sommet. C'est précisément le but d'un algorithme d'élection. Un algorithme distribué résout le problème de l'élection s'il termine toujours et si dans sa configuration finale, exactement un processus est marqué *élu* et tous les autres processus sont *non-élus*. De plus, il est supposé que lorsqu'un sommet prend l'état *élu* ou *non-élu*, il reste dans cet état jusqu'à la fin de l'exécution de l'algorithme. Le lien entre le problème de l'élection et les revêtements est le suivant [20] :

Théorème 4.2 *Étant donné un graphe connexe G, il existe un algorithme d'élection pour G si et seulement si $Dir(\mathsf{G})$ est minimal pour la relation « être revêtement symétrique ».*

Remarque 4.5 *Un graphe orienté étiqueté symétrique* $\mathbf{D}$ *est* ***minimal pour la relation « être revêtement symétrique »*** *s'il n'existe pas de graphe étiqueté symétrique* $\mathbf{D}'$ *non isomorphe à* $\mathbf{D}$ *tel que* $\mathbf{D}$ *est un revêtement symétrique de* $\mathbf{D}'$.

La technique générale utilisée pour les résultats d'impossibilité des algorithmes distribués [4] est la suivante. Si D et D' sont deux graphes orientés et que D est un revêtement de D' via φ, alors tous les calculs sur D' induisent un calcul sur D via φ^{-1}. C'est de cette façon qu'est prouvé le Théorème 4.2 car plusieurs sommets peuvent être élus simultanément par une exécution de l'algorithme.

Nous étudions des problèmes sur les graphes qui nécessitent de briser des symétries initiales ; ces symétries sont précisément présentes dans des graphes orientés qui ne sont pas minimaux pour la relation « être revêtement symétrique » et correspondent aux sommets à l'intérieur d'une fibre. La symétrie initiale dans un graphe se retrouve dans les sommets d'une fibre ayant le même état initial. Pour résoudre des problèmes comme la coloration, le MIS, le couplage maximal, etc., nous avons besoin de briser la symétrie à l'intérieur d'une fibre. Soit $\mathcal{A}$ un algorithme probabiliste, nous montrons que des exécutions de $\mathcal{A}$, ayant une probabilité strictement positive, maintiennent les sommets à l'intérieur d'une fibre dans le même état. Par conséquent, elles ne brisent pas la symétrie initiale et ne résolvent pas les problèmes cités précédemment (Propositions 4.1, 4.2 page 57 et 4.3 page 61).

Ainsi, par la suite, nous considérerons des graphes orientés non minimaux pour la relation « être revêtement ».

4.3 Obtenir des Bornes Inférieures en Considérant des Graphes Non-Connexes

Dans cette section, nous prouvons des bornes inférieures en considérant des familles de graphes non-connexes qui ne sont pas minimales pour la relation « être revêtement symétrique ». Nous obtenons un résultat général que nous illustrons à travers des applications pour la coloration, le MIS, le couplage maximal, le 2-MIS ou la coloration à distance 2.

Proposition 4.1 *Soit* G *un graphe ayant* n_G *sommets. Supposons que* $Dir(\mathsf{G})$ *ne soit pas minimal pour la relation « être revêtement symétrique ». Soit* δ_1 *une numérotation de ports de* G. *Soit* (K, δ) *le graphe* K *avec la numérotation de ports* δ *formé par* α *copies de* (G, δ_1) *et soit* $n = \alpha n_{\mathsf{G}}$. *Alors, il existe une constante* $c > 0$ *telle que, pour tout algorithme distribué Las Vegas* $\mathcal{A}$ *qui utilise des messages de* 1 *bit, il y a au moins une des copies de* (G, δ_1) *telle que pour chaque fibre F, tous les sommets de F sont dans le même état après* $c \log n$ *rondes avec forte probabilité.*

Afin d'illustrer cette proposition, la Figure 4.2 présente la famille des graphes K où G est le graphe complet à deux sommets.

Démonstration. Soit $\mathcal{A}$ un algorithme distribué Las Vegas, et soit G un graphe tel que $Dir(\mathsf{G})$ n'est pas minimal pour la relation « être revêtement symétrique ». Soit δ_1 une numérotation de ports de G, $\mathsf{D} = Dir(\mathsf{G})$ et δ_2 la numérotation de ports correspondant à D. Soit D' un graphe orienté non isomorphe à D tel que D est un revêtement symétrique de D' via le

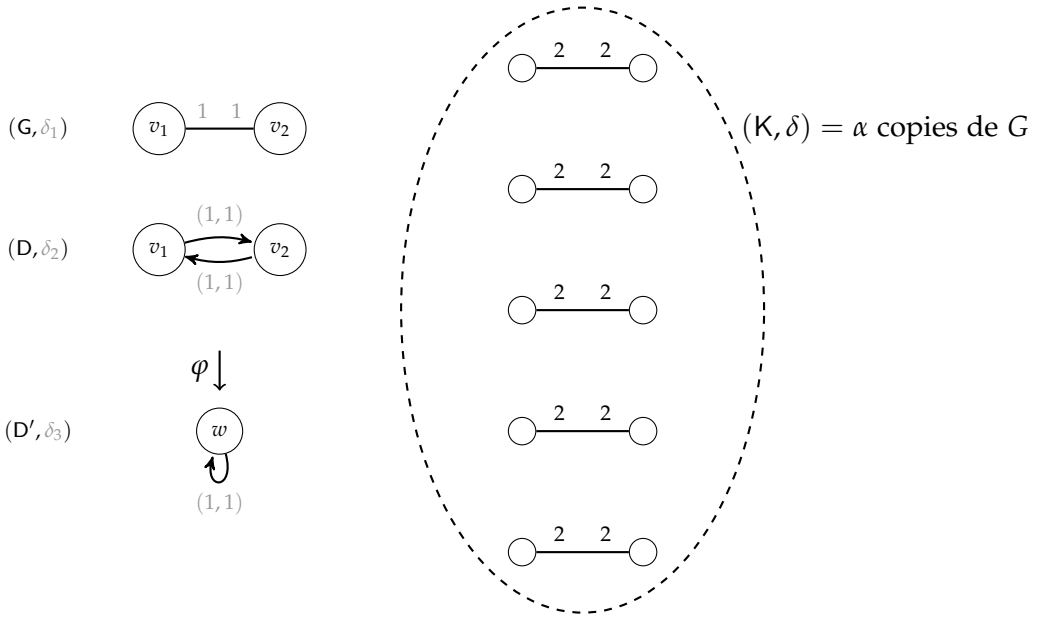


FIGURE 4.2 – *Le graphe complet* G *avec la numérotation de ports* δ_1 *et son graphe orienté* $\mathsf{D} = Dir(\mathsf{G})$ *avec sa numérotation de ports* δ_2 *qui est revêtement de* (D', δ_3). *Le graphe* K *est constitué de* α *copies de* G.

morphisme φ et soit δ_3 une numérotation de ports de D' induite par δ_2 via φ. Soit (K, δ) le graphe formé des α copies de (G, δ_1).

Une exécution $((\mathbf{D}'_i, \delta_3))_{0 \leq i \leq \ell}$ de $\mathcal{A}$ sur $(\mathbf{D}', \delta_3)$ (avec $\mathbf{D}'_0 = \mathbf{D}'$) induit, via φ^{-1}, une exécution $((\mathbf{D}_i, \delta_2))_{0 \leq i \leq \ell}$ de $\mathcal{A}$ sur $(\mathbf{D}, \delta_2)$ (comme l'explique la Remarque 4.4 page 51) avec $\mathbf{D}_0 = \mathbf{D}$. Une telle exécution dans D' implique que l'exécution induite dans G alloue le même état à chaque fibre de G.

Parmi les exécutions associées à $(\mathbf{D}', \delta_3)$, nous choisissons l'exécution $\mathcal{E}$ ayant une transition de plus grande probabilité pour chaque pas de D'. Elle induit alors l'exécution suivante pour chaque $0 \leq i < \ell$: $(\mathbf{D}_i, \delta_2) \underset{p_i}{\Longrightarrow} (\mathbf{D}_{i+1}, \delta_2)$. Nous rappelons que chaque message de $\mathcal{A}$ utilise soit le bit 0 soit le bit 1, de plus la probabilité d'envoi des messages par tout sommet v dans $\mathsf{V}(\mathsf{D}')$ est la même pour tous les sommets $\varphi^{-1}(v) \in \mathsf{V}(\mathsf{D}')$, donc de probabilité supérieure ou égale à $1/2$. Ainsi, pour chaque i : $p_i \geq \frac{1}{2^{2m_\mathsf{G}}}$, où m_G est le nombre d'arêtes de G. Finalement, nous considérons l'exécution de $\mathcal{A}$ induite par l'exécution $\mathcal{E}$ sur chaque copie de (G, δ_1) de (K, δ). La probabilité que les fibres de G soient dans le même état après k rondes est plus grande que la probabilité de l'exécution $\mathcal{E}$ avec $\ell = k$ qui est $p_1 * \cdots * p_k \geq \left(\frac{1}{2^{2m_G}}\right)^k$.

Nous définissons la variable aléatoire T qui compte le nombre de rondes nécessaires pour obtenir dans chaque copie de G dans K une fibre telle que les sommets de cette fibre ne sont pas tous dans le même état. De même, nous définissons la variable aléatoire T_G qui compte le nombre de rondes nécessaires pour obtenir dans G une fibre telle que les sommets de cette fibre ne sont pas tous dans le même état.

Alors, pour tout $k \geq 1$, nous avons :

$$\begin{aligned}
\mathbb{P}(T > k) &= 1 - \mathbb{P}(T \leq k) \\
\mathbb{P}(T > k) &\geq 1 - (\mathbb{P}(T_{\mathsf{G}} \leq k))^{\alpha} \\
\mathbb{P}(T > k) &\geq 1 - \left(1 - \left(\frac{1}{2^{2m_{\mathsf{G}}}}\right)^{k}\right)^{\alpha} \\
&\sim 1 - e^{-\frac{\alpha}{2^{2km_{\mathsf{G}}}}} \text{ quand } \alpha \to \infty. \qquad (4.1)
\end{aligned}$$

D'où, en prenant $k = \frac{1}{4m_{\mathsf{G}}} \log_2 \alpha$:

$$\begin{aligned}
\mathbb{P}\left(T > \frac{1}{4m_{\mathsf{G}}} \log_2 \alpha\right) &> 1 - \frac{1}{e^{\sqrt{\alpha}}} \qquad (4.2) \\
&= 1 - \frac{1}{e^{\sqrt{\frac{n}{n_{\mathsf{G}}}}}} \qquad (4.3) \\
&= 1 - o\left(\frac{1}{n}\right), \text{ quand } n \to \infty.
\end{aligned}$$

Le résultat suit. □

Corollaire 4.1 *Pour tout algorithme distribué Las Vegas , il existe une famille infinie $\mathcal{F}$ de graphes non-connexes telle que $\mathcal{A}$ a une complexité en bit $\Omega(\log n)$ avec forte probabilité sur $\mathcal{F}$ pour résoudre le problème de la coloration ou le problème du MIS.*

Démonstration. Soit G le graphe complet de taille 2 où $\mathsf{V} = \{v_1, v_2\}$, comme défini sur la Figure 4.2. Le graphe orienté D n'est pas minimal pour la relation « être revêtement symétrique ». Maintenant, considérons le graphe K comme défini dans la proposition précédente (avec $n_G = 2$, $m_G = 1$ et $\alpha = n/2$). Alors l'expression (4.2) devient :

$$\mathbb{P}\left(T > \frac{1}{4}\log_2 \frac{n}{2}\right) = 1 - o\left(\frac{1}{n}\right), \text{ lorsque } n \to \infty. \qquad (4.4)$$

Après $c \log n$ rondes (avec $c = \frac{1}{4}$) les deux sommets de la copie de G donnée par la proposition ont reçu la même information (des séquences identiques de messages via les mêmes ports) et ont fait les mêmes transitions (comme l'explique la Remarque 4.4 page 51) et donc ont pu prendre la même décision pour la couleur avec probabilité non nulle pour le pas suivant ; donc, si l'algorithme s'arrête à cette étape, il se pourrait qu'il donne un résultat incorrect.

La même preuve est valable pour le problème du MIS et le corollaire suit.

□

Remarque 4.6 *Une preuve directe pour le Corollaire 4.1 peut être obtenue du fait que la symétrie peut se briser dans le graphe complet à deux sommets avec probabilité $1/2$, et donc en considérant $n/2$ copies de ce graphe, nous déduisons une borne inférieure $\Omega(\log n)$. Cependant, nous mentionnons le Corollaire 4.1 avec cette preuve pour donner une première illustration simple d'une application de la Proposition 4.1 page 53.*

Corollaire 4.2 *Pour tout algorithme distribué Las Vegas $\mathcal{A}$, il existe une famille infinie $\mathcal{F}$ de graphes non-connexes telle que $\mathcal{A}$ a une complexité $\Omega(\log n)$ avec forte probabilité sur $\mathcal{F}$ pour résoudre le problème du couplage maximal.*

Démonstration. La preuve est similaire à la preuve du corollaire précédent. En effet, il suffit de prendre G égal à un triangle (le graphe complet avec $n_G = m_G = 3$, voir Figure 4.3). Alors, les mêmes arguments tiennent en prenant $c = 1/12$. □

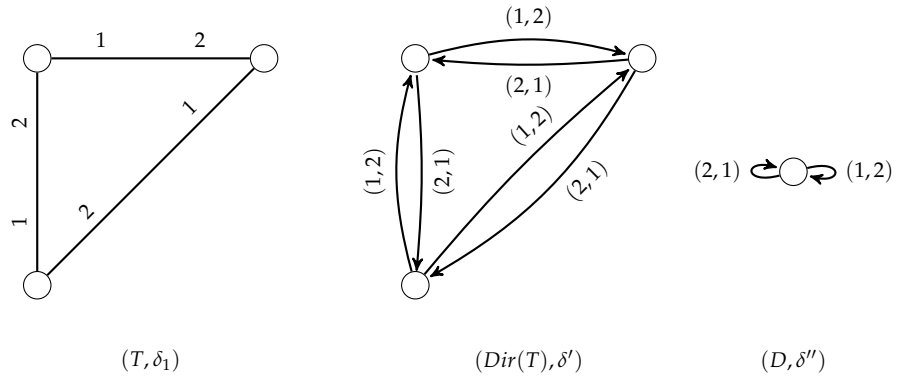


FIGURE 4.3 – *Un triangle T avec une numérotation de ports δ et son graphe orienté étiqueté associé $(Dir(T), \delta')$ qui est un revêtement de (D, δ''). Il y a une seule fibre qui contient tous les sommets de $Dir(T)$.*

Corollaire 4.3 *Pour tout algorithme distribué Las Vegas $\mathcal{A}$, il existe une famille infinie $\mathcal{F}$ de graphes non-connexes telle que $\mathcal{A}$ a une complexité $\Omega(\log n)$ avec forte probabilité sur $\mathcal{F}$ pour résoudre le problème du 2-MIS et le problème de la coloration à distance 2.*

Démonstration. Le problème du 2-MIS (resp. coloration à distance 2) est le même que le problème du MIS (resp. coloration) dans un graphe formé de plusieurs copies du graphe complet à 2 sommets. Ainsi, le corollaire suit, en prenant pour graphe G celui de la Figure 4.2 page 54. □

4.4 Obtenir des Bornes Inférieures de la Forme $\Omega(\sqrt{\log n})$ pour des Graphes Connexes

Dans cette section, nous décrivons une construction générale pour obtenir des bornes inférieures de la forme $\Omega(\sqrt{\log n})$ pour résoudre de façon distribuée des problèmes sur les graphes pour des familles particulières de graphes connexes. Nous illustrons cette construction avec une famille infinie d'anneaux.

4.4.1 Première Construction

Nous utilisons un cas particulier de la construction de Reidemeister (Section 1.3.7 page 25). Soient G un graphe connexe, e une arête sur G et α un entier positif. Soit G^e le sous-graphe de G obtenu en supprimant l'arête e. Nous définissons le graphe $R(G, e, \alpha)$ comme étant le graphe obtenu de la façon suivante :

- faire α copies de G^e notées G^e_i pour $1 \leq i \leq \alpha$;
- pour tout i tel que $1 \leq i \leq \alpha$ et $i+1$ calculé modulo α, connecter la première extrémité de e dans G^e_i à la seconde extrémité de e dans G^e_{i+1}.

Cette construction est étendue de façon naturelle aux graphes avec numérotation de ports.

Les Figures 4.4 et 4.5 donnent un exemple de cette construction avec le triangle.

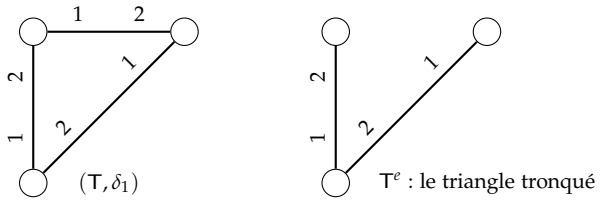


FIGURE 4.4 – *Un triangle* T *avec une numérotation de ports* δ *et le triangle tronqué associé.*

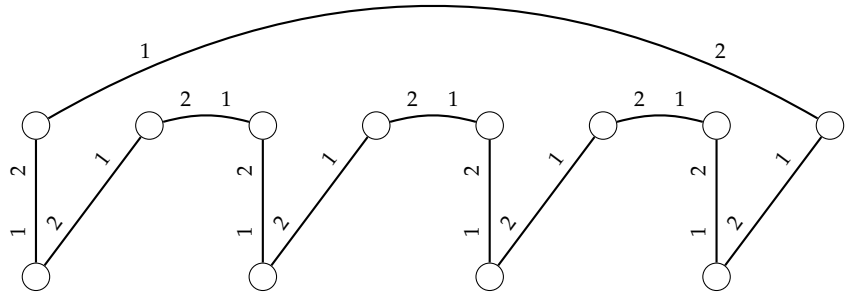


FIGURE 4.5 – *La construction de Reidemeister avec 4 copies du triangle tronqué* T^e *de la Figure 4.4. Nous obtenons* $R(\mathsf{T}, e, 4)$ *et la numérotation de ports induite par* δ *: un revêtement du triangle* (T, δ) *avec 4 feuillets.*

Remarque 4.7 *Le graphe* $R(\mathsf{G}, e, \alpha)$ *est connexe si et seulement si* G^e *est connexe.*

4.4.2 Les Bornes Inférieures

Nous obtenons une borne inférieure à partir de ce type de construction :

Proposition 4.2 *Soit* G *un graphe non minimal pour la relation « être revêtement » ayant* n_{G} *sommets et au moins un cycle et muni d'une numérotation de ports* δ*. Soit* e *une arête de* G *telle que* G^e*, le graphe obtenu en supprimant l'arête* e*, est connexe. Soient* α *un entier positif et* $n = \alpha n_{\mathsf{G}}$*. Soit* δ' *une numérotation induite par* δ *sur* $R(\mathsf{G}, e, \alpha)$*. Alors il existe une constante* $c > 0$ *telle que, pour tout algorithme distribué Las Vegas* $\mathcal{A}$ *qui utilise des messages de 1 bit, il y a au moins une copie de* G^e *sur* $R(\mathsf{G}, e, \alpha)$ *pour laquelle chaque fibre a tous ses sommets dans le même état après* $c\sqrt{\log n}$ *rondes avec forte probabilité.*

Démonstration. Soient $\mathcal{A}$ un algorithme distribué Las Vegas, G un graphe non minimal pour la relation « être revêtement » et δ une numérotation de ports de G. Soit $\mathsf{D} = Dir(\mathsf{G})$ et soit D' non isomorphe à D tel que D est un revêtement symétrique de D' via le morphisme φ, δ_1 la numérotation de

ports de D correspondant à δ et δ_2 la numérotation de ports de D′ induite par φ.

Soient e une arête telle que G^e est connexe et δ_3 la numérotation de ports de $R(\mathsf{G}, e, \alpha)$ induite par δ. Le graphe $R(\mathsf{G}, e, \alpha)$ est un revêtement de G via un morphisme ψ. La Figure 4.6 donne un exemple suivant les mêmes notations.

Comme dans la preuve de la Proposition 4.1 page 53, soit $p > 0$ la probabilité de l'exécution la plus probable dans D′. Par relèvement, p est la probabilité que tous les sommets d'une fibre dans G soient dans le même état particulier après un pas de $\mathcal{A}$.

Dans $R(\mathsf{G}, e, \alpha)$, considérons une séquence de $2j+1$ copies consécutives $\mathsf{G}^e_{-j}, \cdots, \mathsf{G}^e_0, \cdots, \mathsf{G}^e_j$ de G^e telle que pour chaque fibre J des copies tous les sommets de J sont dans le même état s_J. Soit $(R_i)_{i>0}$ une exécution de $\mathcal{A}$ sur $R(G, e, \alpha)$ induite via $(\varphi \circ \psi)^{-1}$ par une exécution dans G.

Maintenant, soit E_i l'évènement : pour chaque fibre J, tous les processus de J dans $\mathsf{G}^e_{-(j-i+1)}, \cdots, \mathsf{G}^e_{j-i+1}$ sont dans le même état après les pas 1 à i de l'exécution. En conditionnant par $E_1, \cdots, E_{i-1}$, E_i a une probabilité au moins $p^{2(j-i)+3}$, donc :

$$\begin{aligned}
\mathbb{P}\left(E_1 \wedge \cdots \wedge E_{j-1}\right) &= \mathbb{P}\left(E_1\right)\mathbb{P}\left(E_2 \mid E_1\right)\cdots\mathbb{P}\left(E_{j-1} \mid, E_1 \wedge \cdots E_{j-1}\right) \\
&\geq \prod_{i=1}^{j-1} p^{2(j-i)+3} \\
&= p^{\sum_{i=1}^{j-1} 2(j-i)+3} \\
&= p^{3(j-1)+2\sum_{i=1}^{j-1} i} \\
&= p^{3(j-1)+j(j-1)} \\
&= p^{(j-1)(j+3)}. \qquad (4.5)
\end{aligned}$$

Si tous les évènements $E_1, \cdots,$ et E_{j-1} se sont réalisés, alors l'algorithme ne peut pas se terminer après $j-1$ rondes car les sommets dans les fibres de G^e_{-1}, G^e_0 et G^e_1 ont reçu les mêmes séquences de messages et se comportent de la même façon et donc doivent avoir une probabilité non-nulle de ne pas briser la symétrie.

En divisant $R(\mathsf{G}, e, \alpha)$ en $\lfloor \alpha/(2j+1) \rfloor$ ensembles de copies consécutives de taille $2j+1$, nous voyons que chacun de ces ensembles, indépendamment les uns des autres, a une probabilité au moins $p^{(j-1)(j+3)}$ de produire les évènements $E_1, \cdots, E_{j-1}$. Ainsi, la probabilité qu'aucun ensemble ne produise ces évènements est : $\left(1 - p^{(j-1)(j+3)}\right)^{\lfloor \alpha/(2j+1) \rfloor}$.

Maintenant, nous choisissons j aussi grand que possible pour que $p^{(j-1)(j+3)} > (\log^2 \alpha)/\alpha$. Cela donne $j = \theta\left(\sqrt{\log \alpha}\right) = o\left(\log \alpha\right)$. La probabilité qu'aucun ensemble ne produise ces évènements est alors au plus :

$$\left(1 - \log^2 \alpha/\alpha\right)^{\lfloor \alpha/(2j+1) \rfloor} \leq e^{-\log^2 \alpha/o(\log \alpha)}$$

$$= e^{-\log \alpha/o(1)} = e^{-(\log(n/n_{\mathsf{G}}))/o(1)} = o((n/n_{\mathsf{G}})^{-1}) = o(n^{-1}).$$

Notons que $j = \theta\left(\sqrt{\log \alpha}\right) = \theta\left(\sqrt{\log n/n_{\mathsf{G}}}\right) = \theta\left(\sqrt{\log n}\right)$. D'où la conclusion que l'algorithme doit utiliser plus de $j-1$, soit $\theta\left(\sqrt{\log n}\right)$, rondes avec probabilité $1 - o(n^{-1})$ pour réussir à briser la symétrie. □

4.4.3 La Complexité pour le Problème du MIS et le Problème du Couplage Maximal pour les Anneaux

En appliquant la Proposition 4.2 page 57 au triangle ($\mathsf{G} = \mathsf{T}$) comme l'explique la Figure 4.6, nous obtenons :

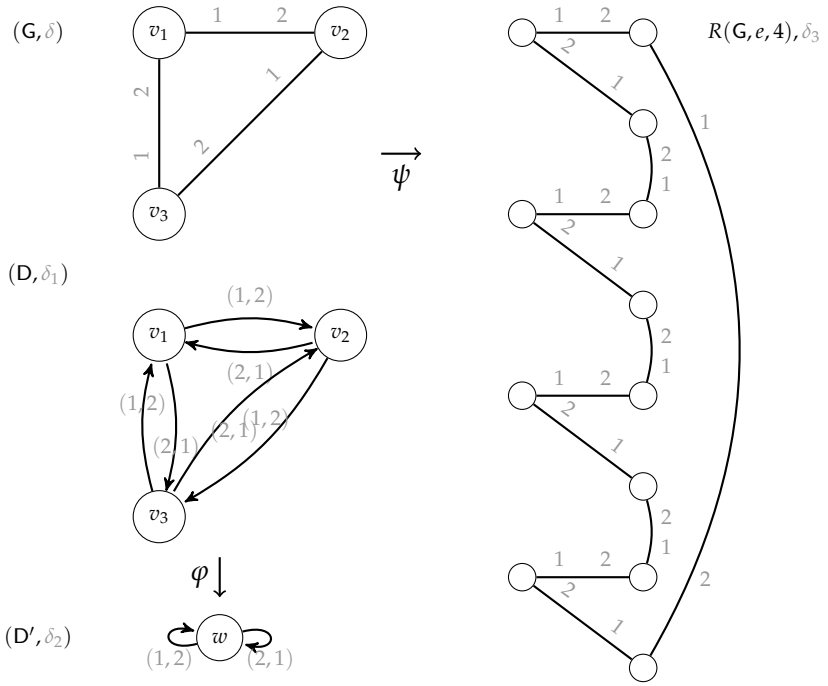


FIGURE 4.6 – *Le graphe complet* G *avec la numérotation de ports* δ *et son graphe orienté* $\mathsf{D} = Dir(\mathsf{G})$ *avec sa numérotation de ports* δ_1 *qui est revêtement de* (D', δ_2). *Le graphe* $R(\mathsf{G}, e, 4)$ *revêtement de G via* ψ *avec sa numérotation de ports* δ_3.

Corollaire 4.4 *Pour tout algorithme distribué Las Vegas* $\mathcal{A}$, *il existe une famille infinie d'anneaux* $\mathcal{R}$ *telle que* $\mathcal{A}$ *a une complexité en bits* $\Omega(\sqrt{\log n})$ *avec forte probabilité sur* $\mathcal{R}$ *pour calculer un MIS ou un couplage maximal.*

4.5 Obtenir des Bornes Inférieure de la Forme $\Omega(\log n)$ pour des Graphes Connexes

Dans cette section, nous décrivons une construction d'une famille infinie de graphes connexes pour obtenir une borne de la forme $\Omega(\log n)$. Ensuite, nous l'appliquons aux problèmes suivants : MIS, coloration, couplage maximal, 2-MIS et coloration à distance 2.

4.5.1 Deuxième Construction

Nous présentons une famille infinie de graphes connexes pour laquelle tout algorithme distribué Las Vegas ne réussit pas à briser la symétrie à l'intérieur des fibres pour une numérotation de ports choisie aléatoirement parmi un ensemble possible de numérotations.

Soit $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ un graphe tel que $\mathsf{D} = Dir(\mathsf{G})$ n'est pas minimal pour

la relation « être revêtement ». Les Figures 4.7 et 4.8 illustrent les constructions en prenant pour G une grille 3×2.

Soit D′ un graphe orienté symétrique tel que D est un revêtement de D′ via l'homomorphisme φ et D n'est pas un graphe orienté isomorphe à D′. Soit F une fibre de D et soit w le sommet de D′ tel que $F = \varphi^{-1}(w)$. Soit b la taille de la fibre $F = f_1, \ldots, f_b$. Comme D n'est pas isomorphe à D′ la taille b de F est supérieure ou égale à 2. Nous pouvons noter aussi que F est un sous-ensemble de sommets de G.

Soit α un entier positif. Nous notons H le graphe formé par α copies de G.

Soit u un sommet qui n'appartient pas à H. Nous définissons maintenant le graphe K obtenu à partir de H en ajoutant un nouveau sommet, le sommet u, et en ajoutant une arête entre u et tous les sommets de toutes les copies de la fibre F. Soit n le nombre de sommets de K, nous avons : $n = \alpha n_G + 1$.

Nous considérons une numérotation de ports de D′ ; elle induit via φ^{-1} une numérotation de ports sur D et donc sur G et sur H. Nous assignons à chaque port (v, u) où v est un sommet de la fibre F de degré d le nombre $d + 1$. Les nombres des ports incidents à u sont choisis aléatoirement, uniformément parmi les permutations de $[1, \alpha \cdot b]$.

Soit $\mathcal{A}$ un algorithme distribué Las Vegas. Le but de cette section est de prouver que si $\mathcal{A}$ s'arrête sur K avec forte probabilité dans un temps inférieur à $c \log n$, il y a une forte probabilité de donner un résultat incorrect. Il s'ensuit qu'il existe une numérotation de ports pour laquelle l'algorithme ne calcule pas un résultat correct dans un temps inférieur ou égal à $c \log n$ avec forte probabilité.

Soit $\mathsf{D}'_{w'}$ le graphe obtenu en ajoutant un nouveau sommet, noté w', au graphe D′ et en ajoutant une arête entre w' et w (l'image de la fibre F par φ). Soit D_F le graphe obtenu en ajoutant b nouveaux sommets f'_i, $1 \leq i \leq b$, à D et en ajoutant une arête entre f_i et f'_i ($1 \leq i \leq b$). De la même manière nous définissons G_F. Il est facile de vérifier que D_F est un revêtement de $\mathsf{D}'_{w'}$ via le morphisme γ défini par : la restriction de γ à D est égale à φ et $\gamma(f'_i) = w'$ (pour $1 \leq i \leq b$).

Une exécution $(\mathbf{D}'_{w'i})_{0 \leq i \leq \ell}$ de $\mathcal{A}$ sur $\mathbf{D}'_{w'}$ (avec $\mathbf{D}'_{w'0} = \mathbf{D}'_{w'}$) induit une exécution $(\mathbf{D}_{Fi})_{0 \leq i \leq \ell}$ de $\mathcal{A}$ sur $\mathbf{D}_F$ (avec $\mathbf{D}_{F0} = \mathbf{D}_F$) via γ. Elle induit aussi une exécution sur G_F. Pour ce genre d'exécution, les sommet de D_F, et donc de G_F, qui appartiennent à la même fibre sont dans le même état à chaque étape.

Soit D_f et G_f les graphes obtenus à partir de D_F et G_F en fusionnant les sommets f'_1,...f'_b sur le même sommet f. Cela implique que, dans D_f et G_f il y a une arête entre les sommets $f_1, \ldots, f_b$ et f.

Maintenant, considérons une exécution de $\mathcal{A}$ sur D_f (donc sur G_f) telle que :

- pour les sommets de D_f différents de f, cette exécution est induite par une exécution de $\mathcal{A}$ sur $\mathsf{D}'_{w'}$,
- le sommet f envoie le même bit à tous les sommets de F.

Il apparaît que pour chacune des fibres de D_f (et donc de G_f), les sommets de cette fibre sont dans le même état après chaque pas de cette exécution. Au final, les pas se déroulent comme si c'était une exécution induite par la relation « être revêtement ». Par définition, une exécution sur D_f (et donc

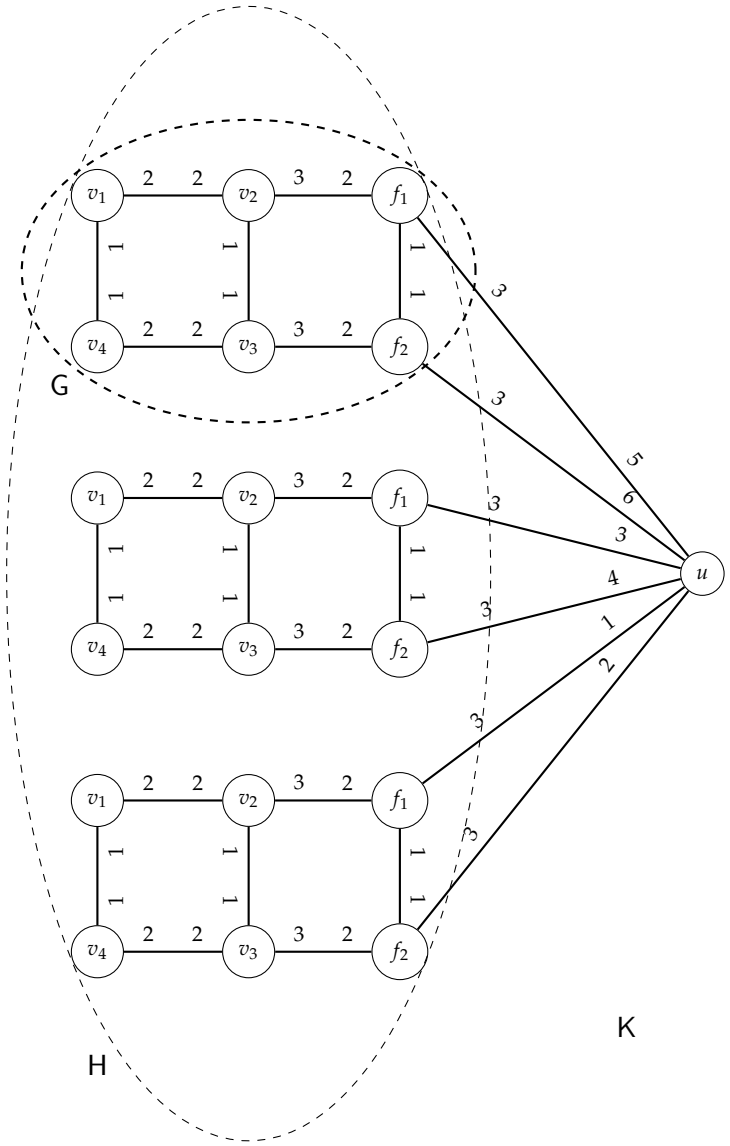


FIGURE 4.7 – *Construction à partir du graphe* G *(grille* 3×2*) du graphe* H *(composé de* $\alpha = 3$ *copies de* G*) et du graphe* K *(ajout du sommet* u*) et de leurs numérotations de ports associées.*

sur G_f) qui satisfait cette propriété (pour chaque fibre, les sommets de cette fibre sont dans le même état après chaque pas de cette exécution) est appelée ***uniforme***, et par extension, D_f (et donc G_f) est uniforme.

4.5.2 Borne Inférieure

À partir de cette construction, nous établissons le résultat suivant :

Proposition 4.3 *Soit $\mathcal{A}$ un algorithme distribué Las Vegas et soit* G *un graphe connexe ayant n_{G} sommets. Supposons que $Dir(\mathsf{G})$ ne soit pas minimal pour la relation « être revêtement symétrique ». Soit α un entier positif. Soit* K *le graphe connexe défini ci-dessus (un sommet ajouté à α copies de* G*) et avec la numérotation de ports associée. Soit $n = \alpha n_{\mathsf{G}}$.*

Il existe une constante $c > 0$ telle qu'après $c \log n$ rondes de l'exécution de

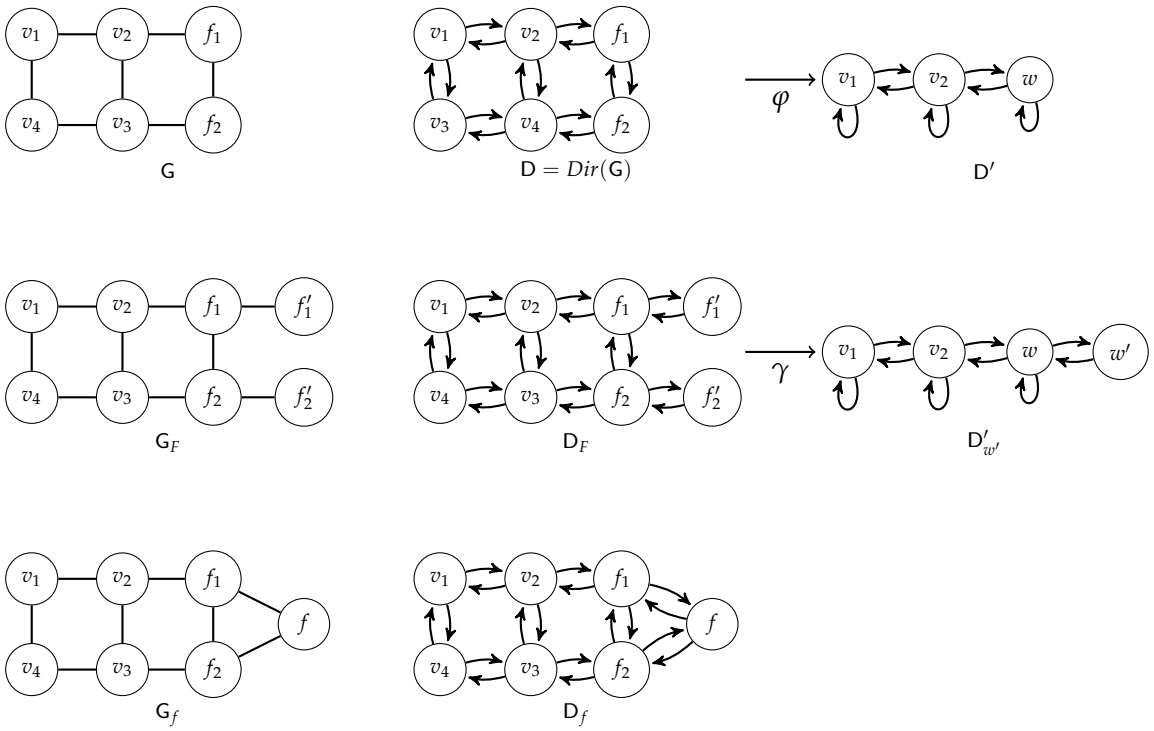


FIGURE 4.8 – *Le graphe orienté* D *(resp.* D_F*) est un revêtement symétrique via* φ *(resp.* γ*) de* D′ *(resp.* $D'_{w'}$*) ; le nombre de feuillets de ces deux revêtements est* 2. D_f *est obtenu en fusionnant tous les sommets de la fibre* $\{f'_1, f'_2\}$.

l'Algorithme $\mathcal{A}$ *sur* K*, avec forte probabilité, il y a une copie de* G *dans* K *pour laquelle l'exécution est uniforme.*

Démonstration. Soit $\mathcal{A}$ un algorithme distribué Las Vegas. Soit n un entier positif. Soit K un graphe connexe défini comme ci-dessus avec la numérotation de ports associée.

Nous choisissons une constante $c_1 > 0$, dépendant de G, et nous montrons par récurrence qu'après r rondes de $\mathcal{A}$, il y a un sous-ensemble de K, que nous noterons U_r, composé de copies uniformes de G et de taille au moins nc_1^r.

- L'état uniforme U_0 est constitué par l'ensemble de toutes les copies de G qui sont initialement uniformes.
- Au pas $r+1$ de l'exécution, considérons, pour les sommets des copies de G dans U_r, un pas induit par la relation « être revêtement » sur $D'_{w'}$, plus particulièrement le pas ayant la plus forte probabilité. Pour l'ensemble des sommets des autres copies et pour u, nous prenons un pas quelconque.
  Il y a une probabilité supérieure ou égale à p_1, pour une certaine constante positive p_1 dépendant seulement de G, pour que tout sommet dans une copie uniforme fasse le pas choisi. Nous avons : $p_1 \geq \frac{1}{2^{2m_G}}$, où m_G est le nombre d'arête de K.
  Nous considérons le sous-ensemble U'_r de U_r consistant en toutes les copies de G dans lesquelles tout sommet fait ce choix. Le nombre de sommets de ce sous-ensemble, $|U'_r|$, est une variable aléatoire dont la distribution est binomiale avec paramètres $|U_r|$ et p_1. D'où la probabilité que $|U'_r| > p_1|U_r|/2$ est $1 - O(|U_r|^{-2})$.

Remarque 4.8 *La loi binomiale de paramètres n, p consiste en la répétition n fois de l'épreuve de Bernoulli de paramètre p (qui a deux issues possibles : succès avec probabilité p et échec avec probabilité $1-p$). On appelle X la variable aléatoire indiquant le nombre de succès, on a :*

$$\begin{aligned}\mathbb{P}(X \leq k) &= \sum_{i=0}^{k} \binom{n}{i} p^i (1-p)^{n-i} \\ &\leq e^{-\frac{(np-k)^2}{2pn}}. \end{aligned} \tag{4.6}$$

L'algorithme ne permet l'envoi que d'un seul bit. Soit b ($b = 0$ ou $b = 1$) un message qui est envoyé par u à au moins la moitié de ses voisins se trouvant dans U'_r. Cela implique des exécutions uniformes sur certaines des copies de G. Nous voyons que chaque sommet dans U'_r a une probabilité de recevoir le bit b supérieure à une constante même si $p_1|U_r|/2$ copies ont déjà reçu le message b. D'où l'existence d'une constante c_2 telle qu'au moins $c_2 p_1 |U_r|/2$ copies dans U'_r reçoivent le message b avec probabilité $1 - O(|U_r|^{-2})$. Cela donne un nouvel ensemble uniforme de taille $c_2 p_1 |U_r|/2$ après la ronde r. Ainsi, nous choisissons $c_1 = c_2 p_1/2$, ce qui donne une suite géométrique de raison c_1 et le résultat suit.

Par le choix d'un c convenable tel que $|U_{c \log n}| = \Omega(n^{1/2})$, nous obtenons le résultat escompté puisque la somme de toutes les probabilités d'échecs est $o(n^{-1})$. □

4.5.3 Exemples d'Application

Comme application à cette proposition, nous considérons un algorithme qui avec forte probabilité termine ses communications en temps $c \log n$. Le résultat de l'algorithme peut être ensuite décidé localement. Nous obtenons alors une contradiction à partir de l'existence d'un ensemble uniforme : avec une probabilité supérieure à $1/2$, un tel ensemble existe et donc deux sommets dans la même fibre d'une copie de G se trouvant dans l'ensemble ont reçu les mêmes séquences de messages et donc pourraient prendre la même décision. Suivant la forme du graphe G, deux sommets dans la même copie prenant la même décision pourraient impliquer que le résultat décidé est incorrect.

Corollaire 4.5 *Pour tout algorithme distribué Las Vegas $\mathcal{A}$, il existe une famille infinie $\mathcal{C}$ de graphes connexes telle que $\mathcal{A}$ a une complexité en bits $\Omega(\log n)$ avec forte probabilité sur $\mathcal{C}$ pour résoudre le problème de la coloration.*

Démonstration. Prenons G comme étant le graphe G_1 de la Figure 4.1 page 52, *i.e.*, une seule arête $\{v_1, v_2\}$. La fibre F est $\{v_1, v_2\}$. Le sommet u est connecté à chaque copie de v_1 et v_2. Considérons une copie de G dans un état uniforme. La proposition suivante est alors garantie dans cette copie : v_1 et v_2 ont envoyé et reçu des séquences de messages identiques sur chacun des ports et donc ils ont une distribution identique quant aux actions possibles à la ronde suivante. Ainsi, s'ils n'ont pas décidé leur couleur à cette étape, ils ont une probabilité non nulle de choisir une couleur impropre. □

Corollaire 4.6 *Pour tout algorithme distribué Las Vegas $\mathcal{A}$, il existe une famille infinie $\mathcal{C}$ de graphes connexes telle que $\mathcal{A}$ a une complexité en bits $\Omega(\log n)$ avec forte probabilité sur $\mathcal{C}$ pour résoudre le problème du couplage maximal.*

Démonstration. Prenons G comme un triangle $\{v_1, v_2, v_3\}$ (cf. Figure 4.3 page 56). La fibre F est $\{v_1, v_2, v_3\}$. Le sommet u est connecté à chaque copie de v_i ($1 \leq i \leq 3$). Un couplage maximal dans un tel graphe doit contenir exactement une arête de chacune des copies de G. Considérons une copie de G dans l'ensemble uniforme, toutes les copies de v_i ont envoyé et reçu des séquences de messages identiques sur chaque port et donc elles ont une distribution identique pour les actions possibles à la prochaine ronde. Ainsi, s'il est décidé qu'une arête du triangle est dans le couplage, alors deux arêtes du même triangle pourraient être dans ce couplage, ce qui contredirait la définition de couplage. □

Corollaire 4.7 *Pour tout algorithme distribué Las Vegas $\mathcal{A}$, il existe une famille infinie $\mathcal{C}$ de graphes connexes telle que $\mathcal{A}$ a une complexité en bits $\Omega(\log n)$ avec forte probabilité sur $\mathcal{C}$ pour résoudre le problème du MIS.*

Démonstration. Prenons G comme un carré $\{u_1, u_2, u_3, u_4\}$ (cf. Figure 4.9). Soit la fibre $F = \{u_2, u_4\}$. Le sommet u est connecté à chaque copie de u_2 et u_4. Dans un tel graphe, pour avoir un MIS, dans chaque copie, soit u_1, soit u_3 doit se trouver dans le MIS. Considérons une copie de G dans l'ensemble uniforme, par le même argument que précédemment, si u_1 peut décider d'être dans le MIS alors u_3 le décide aussi et vice-versa, ce qui mène à une contradiction. □

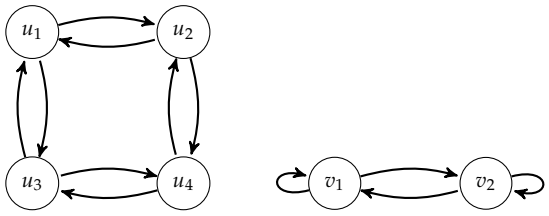


FIGURE 4.9 – *Le graphe orienté $Dir(\mathsf{G})$ associé à un carré qui est le revêtement de D avec deux feuillets via l'homomorphisme φ défini par : $\varphi(u_1) = \varphi(u_3) = v_1$ et $\varphi(u_2) = \varphi(u_4) = v_2$.*

Corollaire 4.8 *Pour tout algorithme distribué Las Vegas $\mathcal{A}$, il existe une famille infinie $\mathcal{C}$ de graphes connexes telle que $\mathcal{A}$ a une complexité en bits $\Omega(\log n)$ avec forte probabilité sur $\mathcal{C}$ pour résoudre les problèmes du 2-MIS et de la coloration à distance 2.*

Démonstration. Prenons G comme une grille trois par deux (cf. Figure 4.10) avec u connecté aux copies de v_1 et de v_6 qui sont dans la même fibre (cf. Figure 4.7 page 61). Considérons une copie de G dans un ensemble uniforme ; que u soit ou pas dans le 2-MIS, au moins un sommet de la grille doit être dans le 2-MIS. Mais, pour $1 \leq i \leq 6$, si la copie de v_i peut décider d'être dans le 2-MIS, alors la copie de v_{7-i} peut décider d'être dans le 2-MIS.

Pour le problème de coloration à distance 2, nous considérons une copie de G dans un état uniforme. Dans cette copie, pour $1 \leq i \leq 6$, si la copie de v_i décide une couleur alors la copie de v_{7-i} peut décider la même couleur. □

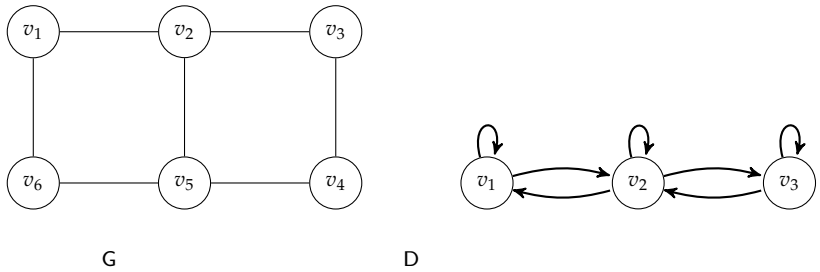


FIGURE 4.10 – *Le graphe orienté Dir*(G) *est un revêtement pour* D *avec* 2 *feuillets via l'homomorphisme* φ *défini par :* $\varphi(v_1) = \varphi(v_6) = w_1$, $\varphi(v_2) = \varphi(v_5) = w_2$, *et* $\varphi(v_3) = \varphi(v_4) = w_3$.

4.6 Preuves d'impossibilité

Cette section présente des résultats d'impossibilité sur l'existence d'algorithmes distribués Las Vegas $\mathcal{A}$ pour briser les symétries à distance k pour $k \geq 3$ dans les graphes anonymes.

D'après la Figure 4.11, une exécution de $\mathcal{A}$ sur le triangle T induit, via le morphisme φ^{-1}, une exécution de $\mathcal{A}$ dans l'hexagone H. Maintenant, si un sommet v de T a un état final s, alors il y a deux sommets de H qui ont le même état final s, et donc la même couleur pour le problème de la coloration ou l'appartenance au MIS pour le problème du MIS ce qui représente une contradiction.

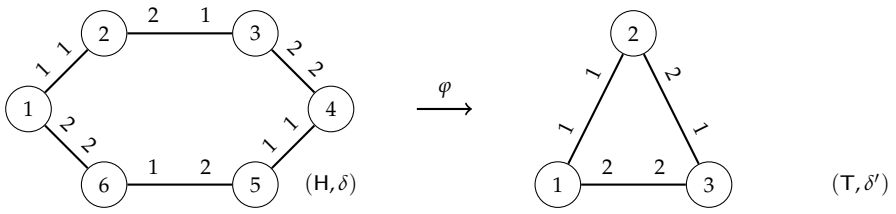


FIGURE 4.11 – *Le graphe* (H, δ) *est un revêtement du graphe* (T, δ'), *à deux feuillets, via l'homomorphisme* φ *qui associe un sommet d'étiquette i de* H *au sommet d'étiquette i de* T.

D'où le résultat :

Proposition 4.4 *Soit k un entier positif tel que* $k \geq 3$. *Il n'y a pas d'algorithme distribué Las Vegas pour résoudre le problème de la coloration à distance k ou le problème du k-MIS.*

Démonstration. Soit k un entier positif tel que $k \geq 3$. Par contradiction, nous supposons qu'il existe un algorithme distribué Las Vegas $\mathcal{A}$ qui résout le problème de la coloration à distance 2. Nous considérons un hexagone H et un triangle T. Le graphe (H, δ) est un revêtement de (T, δ') via le morphisme φ défini dans la Figure 4.11.

Une exécution $(\mathbf{T}_i)_{0 \leq i \leq \ell}$ de $\mathcal{A}$ sur (T, δ') (avec $\mathbf{T}_0 = (\mathsf{T}, \delta')$) induit une exécution $(\mathbf{H}_i)_{0 \leq i \leq \ell}$ de $\mathcal{A}$ sur (H, δ) (avec $\mathbf{H}_0 = (\mathsf{H}, \delta)$) via φ^{-1}.

Maintenant, si un sommet v de T a une couleur finale c alors il existe deux sommets de H qui ont la couleur c, contradiction.

En utilisant la même construction, nous prouvons que s'il existe un algorithme distribué Las Vegas $\mathcal{A}$ qui résout le problème du k-MIS alors nous obtenons une contradiction : il existe dans T un sommet dans un état qui indique qu'il appartient au k-MIS et donc il existe deux sommets dans H dans un état qui indique qu'ils appartiennent au k-MIS. □

Le même type de construction nous donne le résultat suivant :

Lemme 4.2 *Il n'y a pas d'algorithme distribué Las Vegas qui détecte, pour tout graphe étiqueté, si deux sommets à distance au plus* 3 *ont la même étiquette.*

Partie II

Preuves Formelles d'Algorithmes Distribués Probabilistes

Dans cette partie, nous cherchons à modéliser les algorithmes distribués de notre modèle en nous intéressant à des problèmes simples.

Dans un premier temps, nous donnons un aperçu de l'état de l'art concernant la modélisation des algorithmes distribués, probabilistes et distribués probabilistes. Ensuite nous faisons quelques rappels sur les outils sur lesquels nous nous sommes basés pour faire la modélisation : les monades, la bibliothèque *Alea*, la bibliothèque *Loco*. Enfin nous énonçons les algorithmes, et résultats associés, que nous étudierons au cours de cette partie. Ils résolvent les problèmes de la brisure de symétrie, du rendez-vous et du MIS.

Nous décrirons ensuite la modélisation formelle des algorithmes probabilistes, puis des algorithmes distribués probabilistes que nous avons faite. Pour cela, nous définissons une syntaxe et des sémantiques qui nous permettent de simuler, de prouver et d'analyser.

Puis nous nous intéresserons au problème du rendez-vous. Nous motivons le recours aux algorithmes probabilistes en exhibant une preuve d'impossibilité : il n'existe pas d'algorithme déterministe qui, quelque soit le graphe pris en entrée, résout le problème du rendez-vous. Nous montrons ensuite qu'il existe un algorithme probabiliste qui résout le problème du rendez-vous.

Enfin, nous détaillons les outils mis en place dans la bibliothèque qui permettent l'analyse des algorithmes distribués probabilistes. Nous appliquons ces outils à des cas concrets.

Existant

5

Sommaire

5.1 Bref État de l'Art

L'état de l'art est divisé en trois sections. Nous regardons d'abord l'existant dans le domaine du distribué, puis dans le domaine des algorithmes probabilistes et enfin nous nous intéressons aux algorithmes distribués probabilistes.

5.1.1 Algorithmes Distribués

Chou [21] montre la correction d'algorithmes distribués grâce à l'assistant de preuve *HOL*. Il s'intéresse au modèle asynchrone. Les algorithmes distribués sont modélisés par des systèmes de transitions d'étiquetage et les spécifications sont exprimées en termes de logique temporelle. Il montre des propriétés sur des algorithmes concrets en prouvant leur simulation avec des algorithmes abstraits via des lemmes de réduction en translatant les propriétés dont la forme devient des triplets de Hoare.

Méry et al. [59] décrivent les systèmes de calculs locaux à partir de leurs spécifications formelles par des étapes successives de raffinement par le biais du formalisme `Event-B` [17, 2]. Une tâche est représentée par une machine abstraite, chaque raffinement est prouvé avec l'aide de la plate-forme `Rodin` [71] pour aboutir à une implémentation du processus exécuté par chaque sommet. Cette approche dite « correcte par construction » est utilisée par Tounsi [76] pour certifier les algorithmes distribués et les exécuter au sein de l'environnement Visidia [10].

Castéran et Filou [18] ont développé une bibliothèque, *Loco*, sur les graphes étiquetés et les systèmes de ré-étiquetages de graphes. Elle permet à l'utilisateur de spécifier des tâches et de prouver la correction de systèmes de ré-étiquetage par rapport à leur tâche ainsi que des résultats d'impossibilité. La version la plus récente [19] permet l'exécution d'algorithmes distribués (probabilistes). Les algorithmes sont représentés de façon à permettre des exécutions finies, des preuves de correction et des calculs de distribution.

Küfner et al. [45] prouvent des propriétés à l'aide de l'assistant de preuve *Isabelle* d'algorithmes distribués tolérants aux pannes. Ils prouvent des résultats positifs sur leur correction. Ils étudient plus particulièrement le problème du consensus. Ils utilisent un modèle asynchrone par passage de messages et considèrent que les processus peuvent faire des fautes. Ils représentent les algorithmes par une machine d'états où opèrent des règles de transitions qui sont des étapes de calculs entre des configurations. Une configuration à un instant t est composée d'un tableau des états locaux des processeurs, d'un historique des messages et d'un historique des émissions. Le raisonnement se fait en inspectant le code et en utilisant des invariants et l'historique.

Deng et Monin [23] vérifient en *Coq* des protocoles de population auto-stabilisants. Ils servent à décrire les réseaux mobiles *ad hoc* qui sont des nœuds mobiles interagissant avec les autres pour faire des calculs. Le comportement des protocoles auto-stabilisants est basé sur des séquences infinies d'exécution. Le modèle est représenté par un graphe orienté et le comportement est décrit par un ensemble de règles d'interaction. Les propriétés à vérifier concernent la partie auto-stabilisante : la convergence et

la clôture. Elles sont exprimées à l'aide de la logique temporelle. Le système est muni d'un détecteur général qui donne des informations sur le réseau à tous les sommets et est plus ou moins fiable.

5.1.2 Algorithmes Probabilistes

Kozen [44] cite deux approches différentes à la sémantique du langage probabiliste : l'approche opérationnelle et l'approche fonctionnelle.

Approche Opérationnelle. Cette approche est basée sur l'utilisation d'un nombre arbitraire de variables aléatoires indépendantes qui suivent une loi de probabilité (tirage pile ou face, distribution uniforme). Cette approche repose sur la transformation monadique suivante. Soit Ω un ensemble de séquences infinies de valeurs aléatoires indépendantes. Un programme probabiliste retournant une valeur de type A sera représenté comme une fonction de type $\Omega \to A \times \Omega$. En effet, si on considère l'état global ω de type Ω du programme, ce programme calcule une valeur de type A à l'aide d'un préfixe ω_1 de ω, et retourne, en plus de la valeur calculée, l'état courant ω_2 qui correspond au suffixe de $\omega (= \omega_1\omega_2)$ non consommé. Utiliser cette approche nécessite de modéliser la distribution de probabilité sur Ω.

Hurd [37] représente un programme probabiliste comme étant une monade d'état où l'état est une suite infinie de booléens. Cette interprétation sous la forme d'une fonction de type $\beta \to \mathbb{B}^\infty \to (\beta \times \mathbb{B}^\infty)$ a été implémentée dans l'assistant de preuve *HOL*. Le programme prend en entrée une suite infinie de booléens tirés uniformément au hasard pour faire le calcul et retourne le résultat ainsi que le reste de la suite de booléens (la suite privée des booléens consommés).

Approche Fonctionnelle. Dans cette approche, les programmes probabilistes sont vus comme des transformateurs de mesures. Audebaud et Paulin-Mohring [6] utilisent une transformation monadique pour représenter les distributions de probabilité, ce qui permet de prouver que la probabilité qu'un programme vérifie une certaine propriété est égale à une certaine expression ou valeur. Ils ont développé *Alea*, une bibliothèque pour raisonner sur les programmes probabilistes. *Alea* est utilisée dans des applications qui nécessitent des outils probabilistes tels que CertyCript/EasyCrypt [9] qui aide à construire et vérifier des preuves dans le domaine de la cryptographie.

5.1.3 Algorithmes Distribués Probabilistes

Plusieurs approches prennent en compte le double paradigme des systèmes distribués probabilistes : l'aspect probabiliste et le non-déterminisme dû au temps de réponse variant d'un processeur à l'autre. Elles requièrent des modèles avec des choix non-déterministes entre plusieurs distributions de probabilités. Ces choix peuvent être faits par un ordonnanceur ou un adversaire. Pour spécifier des propriétés d'algorithmes distribués probabilistes, on peut utiliser la logique temporelle

avec des opérateurs probabilistes et un seuil. Norman [66] fait un résumé de ces approches.

Segala [73] sépare les deux aspects non-déterministe et probabiliste en introduisant un adversaire, dès lors le système ressemble à une chaîne de Markov, il est alors possible de raisonner sur les probabilités. Il utilise des automates probabilistes temporisés qui modélisent les transitions possibles de l'algorithme. La vérification automatique d'automates temporisés est faite en construisant un quotient d'états finis du système. Ce quotient prend la forme d'un système de transition d'états étiquetés qui représentent tous les comportements de l'automate temporisé et qui peut être analysé en utilisant les techniques du *Model-checking*. Il prend comme exemple le consensus.

Le *Model-checking* est un outil utilisé pour assurer la correction des systèmes, mais, utilisé sur de grand systèmes, cela mène à une explosion de la complexité en espace. Il existe des méthodes pour réduire cette explosion. Une analyse qualitative des algorithmes distribués probabilistes est rendue possible grâce au *model-checker PRISM*[48]. Kwiatkowska et al. [49] utilisent ce *model-checker* et un assistant de preuve, *Cadence* [58], pour obtenir des preuves automatisées. Cet assistant de preuve permet une vérification de systèmes larges et complexes en réduisant le problème de vérification à de plus petits sous-problèmes qui peuvent être résolus automatiquement par des *model-checkers*. Le protocole de consensus a été prouvé pour sa partie non-probabiliste par *Cadence* et pour sa partie probabiliste par *PRISM*.

Hurd et al. [38] montrent comment formaliser en logique d'ordre supérieur la théorie de *pGCL*, un langage pour raisonner sur des choix probabilistes ou faits par un ordonnanceur. Ils ont implémenté un générateur de condition de vérification. Ils ont élaboré une grammaire et des commandes pour dérouler les algorithmes distribués probabilistes comme des calculs. Ils utilisent des ordonnanceurs pour les choix non-déterministes. À l'aide de post-conditions, ils déduisent les pré-conditions qui correspondent aux espérances d'obtenir les post-conditions. Pour faire les calculs sur les boucles *while* ils utilisent le plus grand point fixe qui existe car la sémantique est monotone. Ils prouvent l'algorithme d'exclusion mutuelle de Rabin : N processus s'exécutent en même temps et de temps en temps doivent accéder à une zone critique, l'algorithme élit un processus unique qui peut entrer dans la zone critique. Cependant, ils utilisent une interprétation de l'algorithme qui ne consiste pas à modéliser les N processeurs de façon concurrente mais qui compte le nombre de processeurs.

Notre modèle est synchrone, ainsi nous ne prenons en compte que l'aspect probabiliste. Sur de grands systèmes, le *model-checking* mène à l'explosion de l'espace mémoire. De plus nous souhaitons quantifier sur les graphes et les algorithmes. Nous nous tournons alors vers une alternative du *model-checking* : les assistants de preuve. Malgré la perte des l'automatisation des preuves, *Coq* est un assistant de preuve interactif qui permet d'écrire des tactiques, ce qui donne une semi-automatisation.

5.2 La Bibliothèque *Loco*

La bibliothèque *Loco* contient un simulateur de systèmes de calculs locaux introduits par Litovsky et al. [53] pilotés par des algorithmes probabilistes (hiérarchie des modes de synchronisation : *LC0*, *LC1*, *LC2* ; élection locale ; rendez-vous). Un intérêt de l'algorithme pour le problème du rendez-vous est qu'il est réellement utilisé dans l'exécution des algorithmes *LC0*. Elle contient des preuves de propriétés génériques sur les classes de calculs locaux, des preuves d'impossibilité, des transformations certifiées d'algorithmes développées dans la thèse de Filou [27].

La syntaxe utilisée pour les algorithmes distribués probabilistes dans *Loco* est basée sur la monade libre de Spiwack. Elle définit à partir de cette monade des sémantiques ensembliste et opérationnelle. Cette dernière permet de simuler les algorithmes. Parmi eux, on retrouve une version du rendez-vous qui sert aussi de brique de base pour des algorithmes nécessitant des synchronisations tels que les algorithmes décrits par des calculs locaux.

Le développement de la partie probabiliste de la bibliothèque *Loco* a commencé suite au développement de notre bibliothèque *RDA*. D'un point de vue technique, ces deux développements se distinguent par l'utilisation de l'extension *Ssreflect* dans *RDA* pour représenter les graphes. La bibliothèque *Loco* comprend une implémentation des graphes sous la forme de dictionnaires [51].

À terme, nous souhaitons fusionner les développements. Pour cela, il faut rajouter notamment des lemmes sur la théorie des graphes (relatifs aux parcours d'arbres) aux lemmes existants dans *Loco*. D'autre part, alors que *Loco* utilise des *Setoids* qui nécessitent notamment des preuves supplémentaires (comme des *morphismes*), les résultats des algorithmes dans *RDA* sont des fonctions finies, ce qui a permis d'avoir recours à l'égalité de Leibniz et de simplifier les preuves. Ainsi, il faudra passer des preuves de *Ssreflect* utilisant ce mécanisme à *Coq*.

Remarque 5.1 *Alors que la bibliothèque RDA concerne l'aspect analyse probabiliste, la bibliothèque Loco s'intéresse plus à la sémantique et à la simulation des algorithmes distribués probabilistes. Afin de laisser l'opportunité à l'utilisateur de simuler les programmes, nous avons adapté la sémantique opérationnelle de Loco.*

Ainsi, avant d'analyser nos algorithmes, nous procédons à leur simulation afin de détecter de potentielles erreurs de définition. Pour cela, nous avons implémenté un générateur standard de nombres aléatoires (Section 1.2.8 page 18) où $m = 255$, $a = 137$ *et* $c = 187$. *Ces simulations se trouvent des les fichiers* `*_op.v` *de l'archive de RDA [30].*

5.3 Les Monades

Les monades en Haskell [1] peuvent être vues comme des descriptions de calculs qui peuvent être composés.

Les monades sont des structures composées d'un constructeur de type et de deux opérateurs de base : `bind` et `return`.

- Le constructeur de type : si `M` est le nom de la monade et `t` un type de données, alors `M t` est le type correspondant dans la monade.

- La fonction `return` : de type `t → M t`.
  À partir d'un élément de type `t`, elle construit un élément de type monadique `M t`.
- L'opération `bind`, représentée par l'opérateur infixe »= : de type `(M t) → (t → M u) → (M u)`.

On suppose que le type monadique est muni d'une opération d'équivalence dénotée ici par le symbole ≡. Les monades doivent vérifier les propriétés ci-dessous.

- Identité gauche :
  `return a »= f ≡ f a.`
- Identité droite :
  `m »= return ≡ m.`
- Associativité :
  `(m »= f) »= g ≡ m »= (fun x ⇒ f x »= g).`

Pour améliorer l'aspect du code utilisé par les monades, Haskell fournit un sucre syntaxique : la *do-notation*. Nous en donnons ici une forme se rapprochant d'Ocaml. Soit un bloc de code `stmts`, l'opération `bind` se traduit comme suit :

```
x »= fun v ⇒ do {<stmts>}   =   Flet v = x in {<stmts>}.
```

Les propriétés précédentes deviennent en utilisant la *do-notation* :

- Identité gauche :
```
Flet x' = x in
    f x'          ≡   f x .
```
- Identité droite :
```
Flet x = m in
    x             ≡   m .
```
- Associativité :
```
Flet y = {Flet x = m in        Flet x = m in
            f x } in     ≡        Flet y = f x in
    g y                             g y.
```

Les applications les plus communes de monades sont :

- utilisation pour représenter les échecs (monade `Maybe` ou `Option`),
- utilisation pour traiter le non-déterminisme (monade `List`),
- utilisation pour les *I/O* (*Input/Ouput*), entrée/sortie (monade `IO`).

5.4 La Bibliothèque *Alea*

La bibliothèque *Alea* , développée en *Coq*, permet de raisonner sur les algorithmes probabilistes. Ses auteurs, Audebaud et Paulin-Mohring [6], y considèrent les programmes probabilistes comme des transformateurs de mesures en utilisant une transformation monadique pour représenter les distributions de probabilité.

Si la distribution des sorties possibles d'un programme probabiliste est connue, lorsqu'on considère P comme étant une propriété sur la sortie, il est alors possible de calculer la probabilité que le résultat du programme

satisfasse P. L'objectif est donc d'estimer la distribution des valeurs retournées par un programme probabiliste.

5.4.1 Formalisation en *Coq* des Mesures

Nous avons vu lors des rappels sur la théorie des probabilités que distribution de probabilités et mesures sont liées : pour une distribution de probabilité $\mathcal{D}$ et un évènement E, $\mathbb{P}(E) = (\mu\ \mathcal{D})\mathbb{I}_E$. Ainsi, dans *Alea*, la distribution de probabilité d'un algorithme probabiliste $\mathcal{A}$ retournant des valeurs de type A se définit comme un enregistrement *Coq* de type `distr` A.

```
Record distr (A:Type) : Type := {
   mu : M A;
   mu_stable_inv : stable_inv mu;
   mu_stable_plus : stable_plus mu;
   mu_stable_mult : stable_mult mu;
   mu_continuous : continuous mu}.
```

Cet objet est défini comme la donnée d'une fonction $\mu : (A \to [0,1]) \to [0,1]$ ayant les propriétés suivantes : pour tout scalaire $k \in [0,1]$; pour toutes fonctions f, f_1, f_2, …, f_n de type $(A \to [0,1])$ telles que $f_1 \leq 1 - f_2$:

$$\begin{cases} \mu(1-f) \leq 1 - \mu(f) & \text{stabilité par inversion,} \\ \mu(f_1 + f_2) = \mu(f_1) + \mu(f_2) & \text{stabilité par addition,} \\ \mu(k * f) = k * \mu(f) & \text{stabilité par multiplication,} \\ \mu(sup_n\ f_n) \leq sup_n(\mu\ f_n) & \text{continuité.} \end{cases}$$

5.4.2 Interprétation Monadique des Algorithmes Probabilistes

Le type `M t` représente le type $(\texttt{t} \to [0,1]) \to [0,1]$. Les deux opérateurs de base `return`, et `bind` sont définis ci-dessous. Ces définitions satisfont les propriétés que toute monade doit avoir (identité gauche et droite, associativité).

- `return` : $\tau \to M\tau$
  `= fun (x : τ) ⇒ fun (f : τ → [0,1]) ⇒ (f x).`
- `bind` : $M\tau \to (\tau \to M\sigma) \to M\sigma$
  `= fun (ma : Mτ) ⇒ fun (mb : τ → Mσ) ⇒`
  `fun (f : σ → [0,1]) ⇒ ma (fun (x:τ) ⇒ (mb x f)) .`

Deux primitives probabilistes sont introduites :

- `random` : à partir d'un entier naturel n, cette fonction renvoie un nombre entre 0 et n avec la probabilité uniforme $1/(n+1)$,
- `flip` : cette fonction renvoie soit *vrai* soit $faux$ avec probabilité $1/2$.

Les accès à ces primitives dans un programme probabiliste sont indépendants les uns des autres.

Voici la définition de leur mesure :

- `flip : M bool`
  `= fun (f : bool → [0,1]) ⇒` $\frac{1}{2}(f$ `true`$) + \frac{1}{2}(f$ `false`$)$.
- `random n : M nat`
  `= fun (f : nat → [0,1]) ⇒` $\sum_{i=0}^{n} \frac{1}{n+1}(f\ i)$.

À partir de ces opérateurs, les programmes probabilistes sont construits à l'aide de :

- `Munit` a : cette fonction retourne la distribution de Dirac au point a où $a : A$;
- `Mlet` x `=` d_1 `in` d_2 : cette fonction évalue d_1, lie le résultat à x et ensuite évalue d_2 où d_1 est un programme probabiliste de type `distr` A et d_2 est un programme probabiliste de type `distr` B.

Remarque 5.2 *Nous ne donnons pas ici la syntaxe Coq telle qu'elle est dans un soucis de lisibilité du code. Pour plus de détails, les lecteurs sont invités à se référer au code des bibliothèques Alea et RDA.*

Certains programmes probabilistes ne terminent pas mais terminent éventuellement (*i.e.*, avec probabilité 1). Dans la bibliothèque *Alea*, le type `distr` A inclut les sous-probabilités (telles que leur mesure est strictement inférieure à 1) qui permettent notamment de modéliser les programmes ne terminant pas. Le type `distr` A a une structure d'ordre partiel complet et tout opérateur monotone et continu de type `(`$A \rightarrow$ `distr` B`)` $\rightarrow A \rightarrow$ `distr` B a un point fixe. Considérons le programme suivant :

$$\texttt{let rec } \mathcal{A}\ (a : A) = \ldots$$

Ce programme a pour type `A` $\rightarrow$ `distr A`. En partant d'un état initial `init:A` on peut mesurer la probabilité qu'il termine, donnée par l'expression $\mu\ (\mathcal{A}$ `init`$)\ \mathbb{I}$. Nous utiliserons l'opérateur `Fix` pour déclarer les algorithmes récursifs.

5.4.3 Preuve en *Coq*

L'interprétation des programmes probabilistes en tant que mesures nous permet de prouver des résultats en utilisant des transformations sur la structure du programme. En voici deux principales :

Lemme 5.1 `(ALEA.Munit_simpl [6]).`
$\forall\ (P : \tau)\ (f : \tau \rightarrow [0,1]),\ (\mu\ P)\ f = (f\ P)$`.`

Lemme 5.2 `(ALEA.Mlet_simpl [6]).`
$\forall\ (P\ Q : \tau)\ (f : \tau \rightarrow [0,1])$,
$(\mu$ `(Mlet` $x = P$ `in` $(Q\ x))\ f = (\mu\ P)$ `(fun` $x \Rightarrow (\mu\ (Q\ x))\ f)$`.`

Finalement on formalise la probabilité pour une expression e de réaliser un évènement Q.

Définition 5.1 *Soit Q une propriété et soit $\mathbb{I}_Q$ sa fonction caractéristique. La probabilité que le résultat de l'expression e satisfasse Q est représentée par $\mu\ e\ \mathbb{I}_Q$.*

5.5 Quelques Algorithmes Distribués Probabilistes

Nous présentons ici quatre algorithmes qui vont nous servir d'exemples pour la suite. Ces algorithmes $\mathcal{A}_{sb}$, $\mathcal{A}_{hs}$, $\mathcal{A}_{mm}$, $\mathcal{A}_{mis}$ sont des solutions simples à des problèmes classiques : brisure de symétrie, rendez-vous,

couplage maximal, ensemble indépendant maximal. Ils sont pour la plupart tirés et analysés par Zemmari [78].

L'étude de ces solutions est la première étape menant à la modélisation. Ainsi dans cette section nous décrivons les algorithmes que nous souhaitons étudier et nous énonçons les résultats que nous souhaitons prouver en *Coq*.

5.5.1 Brisure de Symétrie

Chaque processus n'interagit avec les autres qu'en fonction de sa vue locale. Lorsqu'il y a des symétries entre certains sommets, il n'est pas possible de les différencier de façon déterministe. Nous étudions plus particulièrement la symétrie observée dans un graphe complet à deux sommets. Nous souhaitons que leur état diffère l'un de l'autre.

On considère un graphe composé de deux sommets liés entre eux. L'algorithme $\mathcal{A}_{sb}$ est le suivant : pour chaque sommet v du graphe, le sommet v tire aléatoirement un bit et l'envoie à son voisin ; il recommence jusqu'à ce que son bit tiré soit différent de celui reçu.

La Figure 5.1 montre une exécution possible de l'algorithme $\mathcal{A}_{sb}$. Le sommet v_1 a tiré la suite de bits 101001 et le sommet v_2 a tiré la suite de bits 001001, le dernier bit échangé étant le premier de la suite.

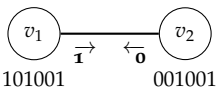


FIGURE 5.1 – *Exemple d'une exécution de l'algorithme $\mathcal{A}_{sb}$.*

L'analyse de l'algorithme nous donne :

Théorème 5.1 *Soit X la variable aléatoire qui indique le nombre de rondes nécessaires pour que les deux sommets se différencient, on a :*

$$\mathbb{E}(X) = \sum_{k=0}^{\infty} \mathbb{P}(X > k) = 2.$$

Ce résultat nous indique qu'en moyenne il faut deux rondes pour que les deux sommets se différencient.

5.5.2 Rendez-vous

Le but du rendez-vous est de créer des liens de communication exclusifs entre des voisins dans un réseau. Par la suite, nous étudions un algorithme distribué probabiliste pour le problème du rendez-vous dans un graphe G décrit et analysé dans [62].

L'algorithme $\mathcal{A}_{hs}$ (cf. Algorithme 1) est le suivant : pour chaque sommet v, v choisit uniformément au hasard un voisin, l'informe qu'il l'a choisi et informe les autres voisins qu'ils n'ont pas été choisis.

Définition 5.1 *(Rendez-vous entre deux sommets)*

Supposons que, après une exécution de $\mathcal{A}_{hs}$, v ait choisi son $i^{ème}$ voisin et que w ait choisi son $j^{ème}$ voisin, il y a un rendez-vous *entre v et w si le $i^{ème}$ voisin de v est w et le $j^{ème}$ voisin de w est v.*

Algorithme 1: L'algorithme $\mathcal{A}_{hs}$ pour le rendez-vous

```
Algorithme A_hs
 Chaque sommet v répète infiniment :
 v choisit uniformément au hasard l'un de ses voisins c(v)
 v envoie 1 c(v)
 v envoie 0 à ses voisins différents de c(v)
 v reçoit les messages de tous ses voisins
 (* Il y a rendez-vous si v reçoit 1 de c(v) *)
```

La Figure 5.2 illustre le résultat d'une exécution possible de $\mathcal{A}_{hs}$. Deux rendez-vous apparaissent : v_3 est associé à v_4 et v_2 à v_5.

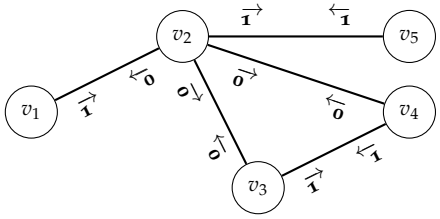


FIGURE 5.2 – *Exemple d'une exécution de l'algorithme $\mathcal{A}_{hs}$.*

Nous faisons l'analyse d'une ronde de cet algorithme.

Définition 5.2 *Nous notons $\mathcal{HS}(e)$ l'évènement "il y a un rendez-vous sur l'arête e". Nous définissons $\mathcal{H}(e)$ comme étant la fonction caractéristique de $\mathcal{HS}(e)$.*

Le but de notre modélisation est de prouver formellement en *Coq* le résultat suivant [62] :

Théorème 5.2 *La probabilité $\mathbb{P}(\exists\ e \in E,\ \mathcal{H}(e))$ d'avoir au moins un rendez-vous après une exécution de $\mathcal{A}_{hs}$ est supérieure ou égale à la constante $(1 - e^{-1/2})$.*

Remarque : À la fin d'une ronde, la probabilité d'avoir au moins un rendez-vous est supérieure à $1 - e^{-1/2} \approx 0.39$. Par conséquent, désignons par T le nombre de fois qu'il faut répéter $\mathcal{A}_{hs}$ pour avoir au moins un rendez-vous dans le réseau, T est une variable aléatoire géométrique de paramètre $p \geq 0.39$, d'où son espérance $1/p \leq 2.54$. On en déduit que pour obtenir un rendez-vous dans un graphe, 3 itérations de l'algorithme $\mathcal{A}_{hs}$ sont, en moyenne, suffisantes.

5.5.3 Couplage Maximal

L'algorithme $\mathcal{A}_{mm}$ pour résoudre le couplage maximal consiste à répéter l'algorithme du rendez-vous $\mathcal{A}_{hs}$ sur le sous-graphe actif, un couple de sommets devenant inactif s'il est en rendez-vous. On a :

Théorème 5.3 *L'algorithme $\mathcal{A}_{mm}$ termine avec probabilité 1.*

5.5.4 Ensemble Indépendant Maximal

L'algorithme étudié ici n'est pas optimal comme ceux présentés dans la première partie de ce manuscrit. Nous nous intéressons dans cette partie

davantage à la formalisation et nous cherchons à montrer que nos outils peuvent s'appliquer. Cet algorithme a les mêmes propriétés que l'algorithme pour le couplage maximal précédent : il termine avec probabilité 1 et lorsqu'il termine, il est correct.

L'algorithme $\mathcal{A}_{mis}$ se décompose en plusieurs phases. À chaque phase, chaque processeur u encore dans le graphe génère un nombre aléatoire $x(u)$. Un processeur u est inclus dans le MIS si $x(u)$ est un minimum local : $x(u) < x(v)$ pour tout voisin v de u. Nous considérons que les nombres générés sont dans l'ensemble $[0, c]$. Quand tous les minima locaux ont été inclus dans le MIS, chaque processeur survivant (non inclus dans le MIS et non voisin d'un sommet dans le MIS), génère une nouvelle variable aléatoire, et encore une fois, les minima locaux sont inclus dans le MIS et ainsi de suite. L'algorithme s'arrête quand il n'y a plus de processeur survivant.

Théorème 5.4 *L'algorithme $\mathcal{A}_{mis}$ termine avec probabilité 1 lorsque $c > 0$.*

6 Modélisation Formelle des Algorithmes Distribués Probabilistes

Sommaire

Nous traitons dans un premier temps les algorithmes probabilistes, puis dans un second temps les algorithmes distribués probabilistes. Les algorithmes probabilistes sont définis à travers deux sémantiques dérivées d'une même syntaxe :

- la sémantique ensembliste et relationnelle, tirée du projet *Loco*, qui nous permet de faire des preuves de correction (par le biais d'invariants) ;
- et la sémantique distributionnelle, qui fera le lien avec *Alea*, qui nous permet de prouver des propriétés sur les distributions.

La modélisation formelle des algorithmes distribués probabilistes est réalisée dans l'optique de respecter les hypothèses du modèle (voir Introduction page 2) et de permettre les preuves des propriétés énoncées dans la Section 5.5 page 79.

6.1 Algorithmes Probabilistes

6.1.1 Exemples d'Algorithmes Probabilistes Simples

Afin de se familiariser avec la bibliothèque *Alea*, la première étape a été de concevoir de petits exemples et d'obtenir des résultats simples. Ce qui ressort de cette mise en pratique est la difficulté de simplifier les calculs dans *Alea*.

Lancer d'un Dé.

L'exemple que nous traitons ici est le lancer d'un dé numéroté de 0 à 5. La distribution des nombres tirés au hasard entre 0 et 5 est définie comme suit :

$$\mu(random\ 5)\ f = \sum_{i=0}^{5} \frac{1}{6}(f\ i).$$

La fonction caractéristique de la propriété « *est égal à 2 ou à 4* » est la suivante :

$$\mathbb{I}_{.=2\vee.=4} \begin{cases} 0 \mapsto 0 \\ 1 \mapsto 0 \\ 2 \mapsto 1 \\ 3 \mapsto 0 \\ 4 \mapsto 1 \\ 5 \mapsto 0. \end{cases}$$

La probabilité de tirer au hasard un chiffre, compris entre 0 et 5, égal à 2 ou à 4 est alors :

$$\begin{array}{lcl} \mu_{random\ 5}\ \mathbb{I}_{.=2\vee.=4} & = & \sum_{i=0}^{5} \frac{1}{6}(\mathbb{I}_{i=2\vee i=4}\ ) \\ & = & \frac{1}{6}*0+\frac{1}{6}*0+\frac{1}{6}*1+\frac{1}{6}*0+\frac{1}{6}*1+\frac{1}{6}*0 \\ & = & \frac{1}{3}. \end{array}$$

Lancer de deux Dés.

Un autre exemple est l'étude de la somme du lancer de deux dés, en voici l'algorithme :

Définition 6.1
```
Definition throw_dice :=
let k = random 5 in
   let k' = random 5 in
     return (k + k' + 2).
```

Un lemme de simplification nous donne le résultat suivant :

Lemme 6.1
```
(example.dice.throw_dice_simpl).
∀ (f: nat → [0,1]), μ (throw_dice) f == ∑_{i=0}^{5} ∑_{j=0}^{5} 1/36 * f (2 + i + j).
```

Démonstration. L'idée de la preuve repose sur l'étape :
$\forall\ n\ f,\ (\mu_{random}\ n)\ f = \sum_{k=0}^{n} \frac{1}{1+n}(f\ k)$. □

Pour connaître la probabilité que la somme soit égale à 11, il s'agit d'écrire la fonction caractéristique qui à une variable associe le booléen

qui vaut vrai si elle est égale à 11 et faux sinon. On doit vérifier que cette probabilité est 2/36 (il y a deux façons d'obtenir 11 sur 36) :

Lemme 6.2 `(example.dice.dice_11).`
$\forall$ `(n:nat),` μ `throw_dice` $\mathbb{I}_{.=11} = \frac{2}{36}$.

6.1.2 Extension à la Bibliothèque *Alea*

Des lemmes applicables à tout algorithme probabiliste ont été nécessaires dans nos preuves plus particulières sur les algorithmes distribués probabilistes. Nous présentons ici l'extension que nous avons réalisée d'*Alea*.

Probabilité Non Nulle.

La probabilité qu'un évènement se produise lors d'un algorithme probabiliste, *i.e.*, que la sortie de l'algorithme satisfasse la propriété sous-jacente de l'évènement, est non nulle s'il existe une exécution possible de l'algorithme, *i.e.*, un résultat que l'algorithme pourrait retourner, dans laquelle l'évènement est vérifié. Cette exécution est appelée témoin. Ainsi, pour montrer qu'une probabilité est non nulle, il suffit d'exhiber un témoin. La réciproque est vraie si on considère que l'algorithme retourne des éléments se trouvant dans un ensemble fini.

Lemme 6.3 `(prelude.my_alea.proba_not_null).`
`Considérons un algorithme probabiliste` A `de type distr T. Soit` t`:T un témoin et` E`:T`$\rightarrow$`[0,1] un évènement,`

`si` $\mu\ A\ \mathbb{I}_{.=t} > 0$ `et` $(E\ t) > 0$ `alors` $\mu\ A\ \mathbb{I}_{(E\ .)} > 0$.

Lemme 6.4 `(prelude.my_ssralea.proba_not_null2).`
`Considérons un algorithme probabiliste` A `dont le type de sortie` T `est fini et` E`:T`$\rightarrow$`[0,1] un évènement,`

`si` $\mu\ A\ \mathbb{I}_{(E\ .)} > 0$

`alors il existe un témoin t:T tel que` $\mu\ A\ \mathbb{I}_{.=t} > 0$ `et` $(E\ t) > 0$.

Loi Géométrique.

La loi géométrique se définit à partir de la loi de Bernoulli. Cette dernière a été développée dans la bibliothèque *Alea*. Notre définition de la loi géométrique est donc basée sur la définition `bernoulli` d'*Alea*.

Définition. Nous définissons l'algorithme `(geom p x)`, qui prend en entrée la probabilité de succès p et un entier naturel x et qui retourne x si l'épreuve de Bernoulli a réussi, sinon se rappelle avec pour entrée $x+1$. Ainsi pour raisonner sur le nombre d'échecs avant un succès de probabilité p il suffit de traiter `(geom p 0)`.

Définition 6.2
```
Fix geom (p:U) (x:nat)
if (bernoulli p) then (return n) else (geom p (x+1)).
```

Terminaison. Nous avons, dans un premier temps, prouvé que l'algorithme `geom` termine avec probabilité 1.

Lemme 6.5 `(prelude.geometric.geom_term).`
`∀ x p, μ (geom p x) 𝕀 = 1.`

Probabilité d'avoir k Échecs. Nous avons ensuite montré que la probabilité de réussir au premier coup est p :

Lemme 6.6 `(prelude.geometric.geom_eq_x).`
`∀ x p, μ (geom p x) 𝕀_{=x} = p.`

Ce qui entraîne que la probabilité d'avoir k échecs puis un succès :

Lemme 6.7 `(prelude.geometric.geom_eq_k).`
`∀ k, p, μ (geom p 0) 𝕀_{=k} = p(1-p)^k.`

Espérance. Afin de définir l'espérance d'une variable aléatoire, nous commençons par la définition de la somme `sumgk`.

Définition 6.3 *Definition sumgk (x:U) (n:nat) :=*
$\sum_{k=0}^{n}(k+1)\ x^k$.

Les calculs de simplification et de limites étant fastidieux à obtenir, nous avons remis à plus tard la preuve formelle du lemme suivant (mineur dans le cadre de notre développement) et nous en donnons ici une démonstration.

Lemme 6.8 `(prelude.geometric.sumgk_lim).` *Admitted*
`∀ p,` $\lim_{n\to\infty}$ `p×(sumgk (1-p) n) =` $\frac{1}{p}$.

Démonstration. Soient $0 < p < 1$ et $q = 1 - p$, par récurrence sur n, on obtient :

$$p \times (sumgk\ q\ n) = p \times \frac{1-(n+2)q^{n+1}+(n+1)q^{n+2}}{(1-q)^2}.$$

La limite quand n tend vers l'infini est alors $1/p$. □

De cette somme en découle la définition de la loi géométrique, et des lemmes précédents, nous prouvons que la limite de la somme géométrique est $1/p$.

Définition 6.4 *Definition geom_exp (p:U) (n:nat) :=*
sumgk (μ (geom p 0) 𝕀_{=k}) n.

Lemme 6.9 `(prelude.geometric.geom_exp_lim).`
`∀ p,` $0 < p \to \lim_{n\to\infty}$ `(geom_exp p n) =` $\frac{1}{p}$.

Lemme 6.10 `(prelude.geometric.geom_exp_lim2).`
`∀ p p',` $0 < p' \to p' \leq p \to$ `∀ n, (geom_exp p n) ≤` $\frac{1}{p'}$.

6.1.3 Syntaxe des Algorithmes Probabilistes

Nous exprimons les algorithmes probabilistes en *Coq* sous forme monadique à l'aide de trois opérateurs : `return`, `bind` et la primitive aléatoire `random` (cf. Définition 6.5).

Définition 6.5
```
Inductive FR (B:Type): Type :=
| Freturn (b:B)
| Fbind {A :Type}(a:FR A)(f : A → FR B)
| Frandom (n:nat)(f : nat → FR B).
```

Afin de simplifier l'expression de ces algorithmes nous utiliserons les notations suivantes :

- `Flet x = f1 in f2` : abréviation pour `Fbind`;
- `Flet x = (random n) in f` : abréviation pour `Frandom`.

Nous pouvons nous lancer dans l'étude de leur analyse. La sémantique ensembliste nous permet de prouver la correction des algorithmes. La sémantique distributionnelle nous permet de raisonner sur les propriétés des distributions des algorithmes.

La monade d'ensemble décrit l'ensemble des résultats intermédiaires et finaux de notre algorithme. Il est donc possible de prouver des propriétés de correction de l'algorithme en raisonnant sur cet ensemble.

En voici la sémantique :

Définition 6.6
```
Fixpoint Setsem B: Type(m :FR B) : Ensemble B :=
match m with
    |Freturn b => fun x => x = b
    |Fbind A a f => fun x => exists y, Setsem a y ∧ Setsem (f y) x
    |Frandom n f => fun x => exists i, (i ≤ n) ∧ Setsem (f i) x
end.
```

Nous définissons la sémantique de la monade de distribution à l'aide des opérateurs `Munit`, `Mlet` et `Random` définis dans le code de la bibliothèque *Alea*.

Définition 6.7
```
Fixpoint Distsem B: Type(m :FR B) : distr B :=
match m with
    |Freturn b => Munit b
    |Fbind _ a f => Mlet (Distsem a) (fun x => (Distsem (f x)))
    |Frandom n f => Mlet (Random n) (fun k => (Distsem (f k )))
end.
```

6.2 Algorithmes Distribués Probabilistes

Nous avons vu dans l'introduction que nous nous plaçons sous les hypothèses d'un modèle synchrone, anonyme où les communications se font par passage de messages. Nous rappelons qu'un système distribué est représenté par un graphe fini simple (connexe) $G = (V, E)$ dont les sommets (l'ensemble V) correspondent aux processus et les arêtes (l'ensemble E) aux liens de communication.

Considérons un graphe $G = (V, E)$. Un algorithme distribué est défini comme une fonction prenant en paramètre un algorithme local et l'appliquant à chaque sommet. Nous décrivons dans cette section ce qui nous a amené à notre modélisation.

6.2.1 Modélisation des Graphes

Nous représentons les graphes simples comme un ensemble fini de sommets `V:finType` et une relation d'adjacence `Adj:rel V` qui à deux sommets associe un booléen pour exprimer la présence d'un lien. Nous rajoutons deux hypothèses : `girrefl` pour indiquer qu'il n'y a pas de boucle et `gsym` pour indiquer que c'est un graphe non orienté.

Nous sommes amenés à utiliser des ensembles de sommets ou d'arêtes. Étant donnée une fonction d'énumération d'ensemble fini, `enum`, chaque fois que nous avons besoin d'une énumération explicite d'un ensemble S, nous utilisons les notations `(enum S)` ou encore $s_1 \dots s_l$ où s_i est un élément de S. Le type d'une telle séquence sera noté par la suite *seq* S.

6.2.2 Vue Locale et Vue Globale

Nous différencions la vue locale (donnée en paramètre à l'algorithme local) de la vue globale (nécessaire à l'analyse). Lors de l'analyse de l'algorithme, des propriétés sur le graphe, dans sa globalité, seront étudiées, pour cela nous avons besoin de distinguer les sommets et les arêtes. Ainsi une vue globale du graphe contient des identifiants sur les sommets. Un exemple de graphe se trouve sur la Figure 6.2.

Cependant, par hypothèse, le système est anonyme et un processeur n'a pas d'information sur la taille du graphe. Ainsi, l'information relative à un sommet ne contient ni son identifiant ni d'identifiants d'autres sommets. Un sommet exécute un algorithme localement avec des informations locales.

La Figure 6.1 nous donne la vue locale d'un sommet à l'initialisation. Il apparaît clairement qu'un sommet ne peut distinguer que ses liens de communication, il peut donc les numéroter. Nous notons δ la fonction de numérotation des ports qui à chaque sommet associe la séquence de ses voisins. La fonction δ doit satisfaire deux hypothèses : la fonction δ ne lie que des sommets adjacents et la séquence de voisins associés à un sommet ne contient pas d'élément redondant.

- `Hypothesis Hδ1 : ∀ v w, Adj v w = v ∈ (δ w).`
- `Hypothesis Hδ2 : ∀ v, uniq (δ v).`

Enfin pour lier le local au global, il nous faudra passer d'une vue à l'autre. La Figure 6.3 est un exemple de numérotation des liens en correspondance avec les identifiants des sommets.

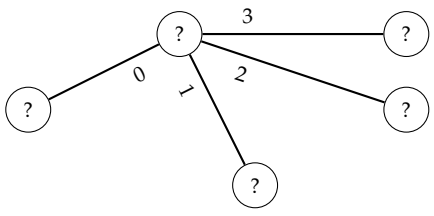


FIGURE 6.1 – *Vue locale d'un sommet.*

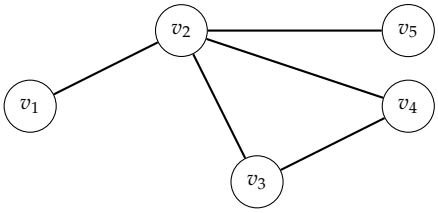


FIGURE 6.2 – *Représentation d'un réseau ayant 5 processus.*

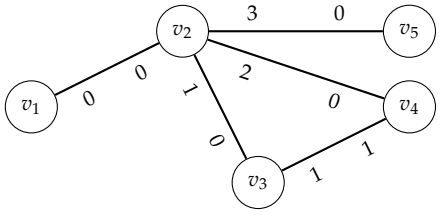


FIGURE 6.3 – *Représentation d'un réseau avec une fonction de numérotation des ports.*

La Figure 6.4 représente une ronde d'un algorithme où des sommets envoient des booléens à leurs voisins. Pour chaque port (v, w), la valeur

envoyée par v à destination w est affichée sous une flèche pointant de v vers w. Par exemple, nous pouvons voir, en mettant en parallèle la numérotation des ports de la Figure 6.3, que le sommet v_2 envoie $\mathbf{1}$ à son voisin numéro 3 correspondant à v_5 d'un point de vue global.

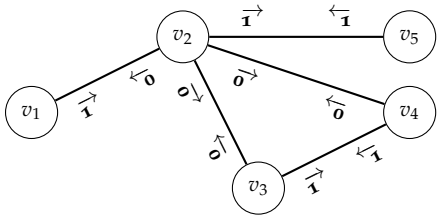


FIGURE 6.4 – *Résultat de l'exécution d'une ronde où les messages sont des booléens.*

6.2.3 Paramètres d'un Algorithme Distribué Probabiliste

Fonctions Globales

Fonction de Numérotation des Ports. Par hypothèse, un processeur sait différencier ses liens de communication. Nous définissons une ***fonction de numérotation des ports*** $\delta : \mathsf{V} \to \mathit{seq}\ \mathsf{V}$ qui associe à chaque sommet une séquence ordonnée de ses voisins.

Exemple 6.1 *Soit v un sommet, $\delta(v) = [v_1, v_2]$ signifie que v a deux voisins, le premier étant v_1 et le second étant v_2.*

Fonction d'Étiquetage. Un algorithme distribué probabiliste est la transformation d'un étiquetage initial en un nouvel étiquetage.

Chaque sommet a un état représenté par une ***fonction d'étiquetage*** $\lambda : \mathsf{V} \to \Lambda$ où Λ est le type des étiquettes des sommets.

Nous définissons un ***port*** comme le couple de deux sommets adjacents. On note P_G l'ensemble des ports du graphe G. S'il n'y a pas d'ambiguïté, il sera noté P.

Remarque 6.1 *Le couple (v, w) représente un port où v et w sont deux sommets adjacents.*

Nous modélisons l'échange des messages, d'un point de vue global, par une fonction d'étiquetage des ports sur le graphe G. On note Ψ le type des étiquettes de ports. Une ***fonction d'étiquetage de ports*** $\psi : \mathsf{P} \to \Psi$ associe à chaque port son étiquette.

Nous notons $\sigma = (\lambda, \psi)$ la paire de fonctions d'étiquetage qui associe à chaque sommet (respectivement port) son état. Le type d'une telle paire, correspondant à l'état global du graphe, sera noté par la suite `State`.

Nous notons simplement $\psi(v, w)$ l'état stocké sur le port (v, w) et $\lambda(v)$ celui stocké sur le sommet v.

Un algorithme distribué prend en entrée un état initial.

Algorithme Local

De l'étude d'algorithmes simples, menée dans la Section 5.5 page 79, nous constatons qu'il arrive que les algorithmes se découpent en étapes. Dans une étape, se passent plusieurs rondes, chacune des rondes contenant des instructions qui leur sont propres. Les étapes sont répétées jusqu'à ce qu'une condition d'arrêt soit atteinte.

Nous définissons le type des algorithmes locaux comme : `FLocT:=`$\Lambda \times$ (`seq` Ψ) $\times$ (`seq` Ψ) $\rightarrow$ `FR`($\Lambda \times$ (`seq` Ψ)). À partir de son propre état, de l'état de sa zone d'écriture et de l'état de sa zone de lecture, un sommet v peut modifier son propre état et l'état de sa zone de réécriture. Cette expression peut contenir la primitive `random` d'où son type de retour.

Un algorithme distribué prend alors en entrée une liste d'algorithmes locaux `(LR:seq FLocT)`.

6.2.4 Passage du Local au Global et vice-versa

Définition 6.1 ***Zone d'écriture.***
La zone d'écriture d'un sommet v est l'ensemble des ports qu'il peut modifier, c'est-à-dire les ports qui lui sont liés (de la forme (v, w) où w est l'un de ses voisins). En d'autres termes, il s'agit de la zone de ré-étiquetage d'un sommet.

Définition 6.2 ***Zone de lecture.***
La zone de lecture d'un sommet v est l'ensemble des ports desquels il peut avoir une information, c'est-à-dire les ports qui sont liés à ses voisins (de la forme (w, v) où w est l'un de ses voisins). Il ne peut pas modifier ces étiquettes, il peut juste les lire.

Pour simuler l'envoi ou la réception de messages sur des canaux de communication nous ré-étiquetons les ports. Un sommet ne connaissant que son état local enverra des messages sur ses canaux. Il y aura alors une correspondance entre un port et le couple d'un sommet et d'un entier afin de représenter le lien par lequel un sommet dépose son message. Si le sommet v envoie un message à son $i^{\text{ème}}$ voisin, il va envoyer son message par le canal (v, i). Supposons que w est le $i^{\text{ème}}$ voisin de v, le port correspondant au canal est (v, w).

Exemple 6.2 *Dans la Figure 6.3 page 89, v_5 est le voisin numéro 3 de v_2 et v_2 est le voisin numéro 0 de v_5 ; le port (v_2, v_5) correspond au canal $(v_2, 3)$, de même le port (v_5, v_2) correspond au canal $(v_5, 0)$.*

Remarque 6.2 *Notons que les deux représentations sont utiles. La première est utilisée pour préserver l'anonymat pour écrire des interactions locales au sujet d'un sommet et de son voisinage, et ainsi pour définir par exemple l'algorithme local. La seconde est utilisée pour exprimer avec une vue globale du graphe une situation entre deux sommets v et w.*

Nous définissons deux fonctions qui modélisent la réception de messages de tous les voisins (`read`) et l'envoi de messages à tous les voisins (`write`) :

– `read : State` $\times$`V` $\rightarrow \Lambda \times$ (`seq` Ψ) $\times$ (`seq` Ψ) : soient σ et v,

`(read` σ v`)` renvoie l'état de v, l'état de sa zone de lecture et l'état de sa zone d'écriture extraits de σ.

- `write : State` $\times \mathsf{V} \times \Lambda \times$ `(seq` Ψ`)` $\rightarrow$ `State` : soient σ, v, λ, ψ, `(write` σ v λ ψ`)` renvoie un nouvel état obtenu à partir de l'ancien état σ où a été mis à jour l'état du sommet v par λ et sa zone d'écriture par ψ.

6.2.5 Représentation d'un Algorithme Distribué Probabiliste

Une ronde `FRound` est l'application d'un algorithme de `LR` à chacun des sommets du graphe selon sa numérotation de ports δ. Une étape `FStep` est l'application des rondes prenant successivement en entrée les algorithmes de `LR`. Nous simulons un algorithme Monte Carlo par la fonction `FMC` qui prend notamment en entrée le nombre d'itérations d'étapes.

Plus précisément, la Définition 6.8 nous donne l'implémentation en *Coq*.

Définition 6.8

```
Definition FLocT := Λ → (seq Ψ) → (seq Ψ) → FR(Λ*seq Ψ).

Fixpoint FRound (seqV:seq V)(res: State)(LR:FLocT):FR State:=
match seqV with
|nil ⇒ Freturn res
|v::t ⇒ Flet s = (FRound t res LR) in
          Flet p = (LR (read res v)) in
             Freturn (write s v p)
end.

Fixpoint FStep (LRs:seq FLocT)(seqV:seq V)(res:State):FR State:=
match LRs with
| nil ⇒ Freturn res
|a1::a2 ⇒ Flet y = (FRound seqV res a1) in (FStep a2 seqV y)
end.

Fixpoint FMC(n:nat) (LRs:seq FLocT) (seqV:seq V)(res:State):FR
State:=
match n with
|O ⇒ Freturn res
| S m ⇒ Flet y = (FStep LCs seqV res) in (FMC m LCs seqV y)
end.
```

Remarque 6.3 *Les algorithmes Monte Carlo se basent sur la décrémentation du nombre d'itération passé en entrée. Ce n'est pas le cas des algorithmes Las Vegas qui peuvent potentiellement contenir des exécutions infinies. Notre syntaxe actuelle ne permet pas l'expression de tels algorithmes. Une perspective est d'étendre cette syntaxe pour les prendre en compte. Toutefois, à l'aide de la bibliothèque Alea, nous avons développé des outils nous permettant de les définir et d'en faire l'analyse comme nous le verrons au Chapitre 8 page 105.*

Le Rendez-vous : Preuve d'Impossibilité et Correction

7

Sommaire

Nous motivons l'utilisation de l'aspect probabiliste en montrant l'impossibilité de résoudre le problème du rendez-vous en supposant que l'algorithme est déterministe. Nous montrerons ensuite qu'il existe une algorithme probabiliste qui résout ce problème.

L'étude de l'algorithme distribué probabiliste qui résout le problème du rendez-vous dans le cas où le graphe considéré est connexe a été étudié[31]. Dans ce chapitre, la seule hypothèse faite sur le graphe est qu'il contient au moins une arête.

7.1 Preuve par Invariant

Tel [75] indique que la technique basique pour montrer que la propriété P est toujours vraie (c'est-à-dire vraie pour chaque configuration de chaque exécution de l'algorithme) est de montrer que P est un invariant. Ainsi, nous définissons un invariant et des lemmes relatifs suivant la définition qu'a donnée Tel. Dans cette section nous considérons un type `B`.

Un invariant s'appuie sur la notion de stabilité (voir Définition 7.1) : suite à un pas de calcul f, si la propriété P était vraie avant ce pas et qu'elle le reste après alors elle est stable (ou préservée) par f.

Définition 7.1
```
Definition Stable (P: B → Prop) (f:B → FR B) :=
∀ s, P s →
∀ s', In (Setsem (f s)) s' → P s'.
```

La propriété P est un invariant si elle est vérifiée dans l'état initial et si elle est stable :

Définition 7.2
```
Definition Invariant (P: B → Prop) (f: B → FR B) (init : B) :=
P init ∧ Stable P f.
```

Pour définir nos théorèmes relatifs à l'impossibilité ou à la correction, il nous faut montrer qu'aucun état qui puisse être atteint ne satisfait P ou bien que tout état atteint satisfait P. Pour cela, nous définissons la propriété d'accessibilité :

Définition 7.3
```
Definition reachFrom (f: B → FR B) (i s: B) :=
∃ n, In (Setsem (iter n (fun s ⇒ let x = s in (f x)) (return i)))
s.
```

Nous avons ajouté à notre bibliothèque le résultat classique suivant sur les états accessibles lorsqu'une propriété est stable :

Lemme 7.1
```
(ra.setSem.reachInd).
∀ (P: B → Prop)(f: B → FR B) (i: B),
Invariant P f i →
∀ s, reachFrom f i s → P s.
```

7.2 Spécification du Problème du Rendez-vous

Nous donnons les spécifications du problème du rendez-vous. Soit Λ le type des étiquettes des sommets et Ψ le type des étiquettes des ports.

7.2.1 Couplage

Une solution au problème du rendez-vous donne un couplage du graphe. Le couplage, défini par la fonction `matching` dans la Définition 7.4, comporte deux aspects :

- l'adjacence (fonction `synchAdj`) : deux sommets en rendez-vous doivent être adjacents ;
- la symétrie (fonction `synchSym`) : si un sommet v est en rendez-vous avec un sommet w alors le sommet w doit être en rendez-vous avec le sommet v.

Plus précisément, soit G un graphe simple non orienté dont l'ensemble de sommets est V et la relation d'adjacence `Adj`. Soit $s := \mathsf{V} \rightarrow \texttt{seq}\ \mathsf{V}$ une fonction qui à chaque sommet v associe soit `None` pour spécifier que v n'a pas de rendez-vous, soit `Some` w pour spécifier que v a un rendez-vous avec w. Le couplage se définit de la façon suivante :

Définition 7.4
```
Definition synchAdj s := ∀ v,
match (s v) with
|Some w ⇒ Adj v w
|_ ⇒ true
end.

   Definition synchSym s := ∀ v,
match (s v) with
|Some w ⇒ (s w) == Some v
|_ ⇒ true
end.

   Definition matching s :=
(synchAdj s) ∧ (synchSym s).
```

7.2.2 Présence d'un Rendez-vous

On définit le fait qu'il y ait un rendez-vous entre deux sommets dans la propriété `hsBetween`. L'existence d'un tel rendez-vous est définie par `hsExists`.

Soit G un graphe simple non orienté dont l'ensemble des sommets est V et sa relation d'adjacence est `Adj`. Soit $s := \mathsf{V} \rightarrow \texttt{seq}\ \mathsf{V}$ une fonction qui à chaque sommet v associe soit `None` pour spécifier que v n'a pas de rendez-vous, soit `Some` w pour spécifier que v a un rendez-vous avec w. On définit :

Définition 7.5
```
Definition hsBetween s v w :=
(Adj v w) && ( s v == Some w) && (s w == Some v).

   Definition hsExists s := ∃ v w, hsBetween s v w.
```

7.2.3 Uniformité

Par hypothèse, le graphe G est anonyme. Tel [75] définit un réseau anonyme comme un réseau dans lequel d'uniques identités ne sont pas disponibles pour distinguer les processeurs ; on suppose aussi qu'on n'a pas recours à un sommet distingué. Par conséquent tous les processeurs avec le même degré sont identiques.

Ainsi, on suppose qu'à l'initialisation, l'état du graphe est tel que chaque sommet de même degré a une vue uniforme : ces sommets ont le même état et sur chacun de leurs liens de communication, ils ont reçu un même message et ont envoyé un même message. Nous définissons cette propriété ci-dessous.

Soit G un graphe simple et non orienté où V est l'ensemble des sommets de G et `Adj` sa relation d'adjacence. Soit δ sa fonction d'étiquetage vérifiant les hypothèses `H`δ`1` et `H`δ`2` (Section 6.2.2 page 89). Soit `s` un étiquetage. La vue uniforme `UniformView` est valable pour deux sommets ayant même degré défini localement via la fonction δ.

Définition 7.6 *Definition UniformView* V *Adj* G δ *Hδ1 Hδ2* σ :=
∀ *v w*, |δ *v*| = |δ *w*| → *read* σ *v* = *read* σ *w*.

Afin de ne pas prendre en compte la numérotation des ports, ou encore que le résultat soit valable quelque soit la numérotation des ports, on suppose qu'à l'initialisation le graphe est *uniforme*, fonction `Uniform`, c'est-à-dire que tous les sommets portent la même étiquette et tous les ports portent la même étiquette. Nous considérons un étiquetage $\sigma = (\lambda, \psi)$:

Définition 7.7 *Definition Uniform* V *Adj* G (λ,ψ) :=
∀ v_1 v_2, (λ v_1) = (λ v_2) ∧ ∀ p_1 p_2, (ψ p_1) = (ψ p_2).

On ajoute le lemme ci-dessous : un état uniforme est un état avec vue uniforme.

Lemme 7.2 `(rda.handshake_spec.uniformUniformView).`
`∀σ, Uniform σ → UniformView σ.`

7.2.4 Fonction Globale du Rendez-vous

Nous définissons la fonction qui à chaque sommet v associe soit `None` pour spécifier que v n'a pas de rendez-vous, soit `Some` w pour spécifier que v a un rendez-vous avec w.

Un sommet doit pouvoir savoir s'il a rendez-vous ou non et à travers lequel de ses ports. Nous supposons alors qu'il y a une fonction locale, `hsPort`, qui associe un numéro de port à une vue locale. De cette fonction, nous définissons une fonction globale `hsPortR` qui doit satisfaire l'hypothèse `hsp1`. Nous pouvons alors définir la fonction `assNeigh`.

Soit G un graphe simple et non orienté dont l'ensemble des sommets est V et sa relation d'adjacence est `Adj`. Soit δ sa fonction d'étiquetage vérifiant les hypothèses `Hδ1` et `Hδ2` (Section 6.2.2 page 89). Soit σ un étiquetage. On définit :

Définition 7.8
```
Variable hsPort : Λ → seq Ψ → seq Ψ → option nat.

   Definition hsPortR δ σ v :=
(hsPort (read δ σ v)).

   Hypothesis hsp1 : ∀ σ v i,
(hsPortR sigma v) = Some i → i < (deg G v).

   Definition assNeigh δ σ v : (option V) :=
match (hsPortR δ σ v) with
|Some i ⇒ Some (nth (δ v) i)
|_ ⇒ None
end.
```

7.2.5 Propriétés de l'Algorithme

Soit G un graphe simple non orienté ayant un ensemble de sommets V et une relation d'adjacence `Adj` et muni d'une numérotation de ports δ.

Soit `LR` la séquence de règles locales applicables à chaque sommet pour un algorithme résolvant le problème du rendez-vous. Nous définissons un pas de calcul, `nextState`, comme suit.

Définition 7.9 *Variable LR : FLocT Λ Ψ.*

Definition nextState δ σ:= FStep δ LR (enum V) σ.

On attend de l'algorithme qu'il soit cohérent (`consistent`), c'est-à-dire que si v sait qu'il a rendez-vous avec un de ses voisins qui correspond dans le graphe à w via la fonction `hsPort`, alors w est aussi en rendez-vous avec v.

Définition 7.10 *Definition consistent δ hsPort σ :=*
synchSym (assNeigh hsPort δ σ).

Enfin, on souhaite montrer la propriété `hsEventually` : un rendez-vous peut se produire à partir d'un état initial.

Définition 7.11 *Definition hsEventually LR δ hsPort initState:= ∃ σ,*
reachFrom (nextState LR δ) initState σ ∧
hsExists (assNeigh hsPort δ sigma).

7.2.6 Spécification de l'Algorithme

Soit Λ le type des étiquettes des sommets et Ψ celui des étiquettes des ports.

Remarque 7.1 *Une hypothèse importante sur le graphe est qu'il contient au moins une arête. En effet, s'il ne contient que des sommets isolés, il ne pourra jamais y avoir de rendez-vous. Le graphe étant supposé non vide, nous noterons par p_0 un des ports du graphe lorsque nous en aurons besoin (un port par défaut). Le port p_0 dépend de la fonction d'adjacence mais lorsqu'elle peut se déduire facilement nous l'omettrons.*

Pour résumer, un algorithme résolvant le problème du rendez-vous a les composants suivants :

- `HsR` : séquence de règles locales appliquées à chaque sommet ;
- `HsP` : fonction locale du rendez-vous ;
- `HsI` : état initial.

Les hypothèses sur ces composants sont les suivantes :

- `HsI1` : l'état initial est cohérent ;
- `HsI2` : l'état initial est uniforme ;
- `HsP1` : la fonction globale du rendez-vous renvoie des numéros de ports inférieurs au degré ;
- `HsRind` : la cohérence est préservée par un pas de l'algorithme, en effet nous sommes dans un réseau fiable et nous souhaitons que si un sommet pense qu'il est en rendez-vous, ce rendez-vous existe bien.

Ci-dessous la structure d'un tel algorithme :

Définition 7.12 *Record hsAlgo Λ Ψ :={*

*(** Local rules *)*
HsR : seq (FLocT Λ Ψ);

*(** Handshake function *)*
HsP : Λ → seq Ψ → seq Ψ → option nat;

```
   (** Initial state *)
HsI : ∀ V Adj G, State Adj;

   (** Hypotheses *)

HsI1:∀ V Adj G δ
(Hδ1:∀ v w,Adj v w = w ∈(δ v))(Hδ2:∀ v,uniq(δ v)) p0,
consistent δ HsP (HsI G);

HsI2:∀ V Adj G,
Uniform (HsI G);

HsP1:∀ V Adj G δ
(Hδ1:∀ v w,Adj v w = w ∈(δ v))(Hδ2:∀ v,uniq(δ v)) p0 σ v i,
(hsPortR δ HsP σ v) = Some i → i < (deg G v);

HsRind:∀ V Adj G δ
(Hδ1:∀ v w,Adj v w=w ∈(δ v))(Hδ2:∀ v,uniq(δ v)) p0,
Stable (fun σ ⇒ consistent δ HsP σ) (nextState HsR δ).
}.
```

Le but de l'algorithme est de réaliser des rendez-vous, c'est-à-dire que pour tout graphe, il existe une exécution dont l'un des états contient un rendez-vous :

```
Définition 7.13  Definition hsRealisation Λ Ψ (A: hsAlgo Λ Ψ) :=
∀ V Adj G δ
(Hδ1:∀ v w, Adj v w = w ∈ (δ v)) (Hδ2:∀ v, uniq(δ v)) p0,
hsEventually (HsR A) δ (HsP A) (HsI A G).
```

7.3 Preuve d'Impossibilité

Nous avons vu que ce qui différencie un algorithme déterministe d'un algorithme probabiliste est l'usage de la fonction `random`. Nous voyons dans cette section l'intérêt des algorithmes distribués probabilistes en montrant qu'il n'existe pas d'algorithme déterministe probabiliste qui résout le problème du rendez-vous.

La propriété « *être déterministe* » est définie dans la Définition 7.14. Les algorithmes sont exprimés à l'aide de listes d'algorithmes locaux, nous exprimons pour ces objets aussi la propriété de déterminisme (`Adet`).

```
Définition 7.14  Fixpoint Deterministic {B:Type}(e : FR B):Prop :=
match e with
| Freturn b ⇒ True
| Fbind A a f ⇒ Deterministic a ∧ ∀ b, Deterministic (f b)
| _ ⇒ False
end.
```

```
Définition 7.15  Fixpoint Adet (l:seq FLocT) :=
match l with
|nil ⇒ True
|t::q ⇒ (∀ lv lp1 lp2, Deterministic (t lv lp1 lp2)) ∧ (Adet q)
end.
```

Nous voulons prouver le lemme suivant :

Lemme 7.1 *Dans notre modèle, il n'existe pas d'algorithme déterministe qui résout le problème du rendez-vous quelque soit le graphe* G *et la fonction de numérotation de port* δ.

Plus formellement :

Lemme 7.3 `(rda.handshake_det.NotReal). ∀ Λ Ψ (A:hsAlgo Λ Ψ), Adet (HsR A) → ∼ (hsRealisation A).`

Nous donnons ici les étapes principales de la preuve du Lemme 7.3. Le développement reprenant les lemmes nommés dans cette preuve est disponible sur la page web de la bibliothèque *RDA* [30].

Démonstration. Nous voulons montrer qu'il n'existe pas d'algorithme distribué déterministe dans notre modèle qui permet de créer un rendez-vous dans n'importe quel graphe ayant n'importe quelle numérotation de ports. Pour cela, nous supposons l'existence d'un tel algorithme et nous montrons que pour un graphe particulier et une numérotation particulière cet algorithme ne produit aucun rendez-vous. Le graphe traité est celui décrit dans la Figure 7.1. Nous procédons par récurrence sur le nombre de rondes.

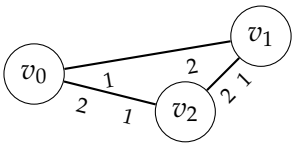


FIGURE 7.1 – *Contre-exemple pour la preuve d'impossibilité :* $(\mathsf{V}, \mathsf{E}, \delta)$.

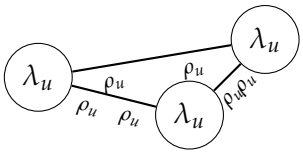


FIGURE 7.2 – *État global initial.*

- Initialisation. Par hypothèse nous savons qu'à l'initialisation l'état a une vue uniforme locale et est cohérent.
- Stabilité. Nous prouvons que la vue uniforme est préservée par une ronde : `Lemme UniformViewStablehs`. Ainsi, nous obtenons par définition que la vue uniforme est un invariant.
- Récurrence. Soit σ un état cohérent ayant une vue uniforme. Nous prouvons que (`Lemme NoHS`) pour toute vue locale de chacun des sommets, `hsPort` est égal à `None`. La preuve est basée sur le fait que si un sommet v a rendez-vous avec un autre, disons son premier voisin, alors ce premier voisin doit aussi avoir rendez-vous avec son premier voisin qui dans notre configuration ne correspond pas à v. Ceci contredit alors l'hypothèse de cohérence.
- Conclusion. Nous savons que la vue uniforme et la cohérence sont préservées par une ronde. Nous avons montré l'uniformité et la cohérence de l'état initial. Nous avons prouvé que pour un état arbitraire σ qui est cohérent et uniforme, il n'y a pas possibilité de créer un rendez-vous. On en déduit qu'aucun rendez-vous ne peut se faire durant une ronde et donc durant l'exécution de l'algorithme.

□

7.4 Preuve de Correction

7.4.1 L'Algorithme Probabiliste

De l'Algorithme 1 page 80 nous tirons la définition de l'algorithme local `randHSLoc` (cf. Définition 7.16) : un sommet choisit uniformément

au hasard un voisin et informe chacun de ses voisins s'il a été choisi ou non.

Les étiquettes des sommets sont de type `option nat` et celles des ports sont de type `bool`. On considère un graphe G ayant un ensemble de sommets V et une relation d'adjacence `Adj` et muni d'une numérotation de ports vérifiant les hypothèses `H`δ`1` et `H`δ`2` (Section 6.2.2 page 89). On nomme `State` le type des étiquetages d'un tel graphe.

Pour la définition de l'algorithme local, nous avons recours à la fonction `(randSendChosen` n l`)` qui renvoie une séquence de booléens de la taille de son argument l tous mis à **0**, sauf le $n^{\text{ème}}$ élément qui est mis à **1**.

Définition 7.16
```
Definition randHSLoc (λ:Λ) (ψout ψin:seq Ψ) : FR (Λ × seq Ψ) :=
 match |ψin| with                  (*suivant le nombre de voisin*)
   |O ⇒ Freturn (None, nil)        (*sommet isolé*)
   |S n ⇒ Flet k = (random n) in
           Freturn(None,randSendChosen(k+1) ψin)
 end.
```

Remarque 7.2 *Ici, l'étiquette du sommet n'intervient pas. Un sommet est en rendez-vous s'il reçoit* ***1*** *de celui qu'il a choisi.*

Nous définissons une ronde pour le rendez-vous, `randHSRound`, grâce à notre définition d'une ronde `FRound`, en la paramétrant par l'algorithme local `randHSLoc` :

Définition 7.17
```
Definition randHSRound (σ: State):= FRound δ σ randHSLoc.
```

7.4.2 Simulation de l'Algorithme Probabiliste

À l'aide de la sémantique opérationnelle nous avons pu simuler l'algorithme. L'interprétation du résultat de la simulation se trouve dans `rda.handshake_op`.

Pour faire la simulation, nous considérons le graphe dans la Figure 7.3. La Figure 7.3 nous donne la numérotation des ports et la Figure 7.4 l'état des sommets et des ports. Nous simulons l'algorithme avec une graine égale à 6 en utilisant notre générateur (cf. Remarque 5.1 page 76). Le résultat obtenu est bien en accord avec ce que l'on attend de l'algorithme de rendez-vous, il est dessiné dans la Figure 7.5. On peut y voir qu'il y a un rendez-vous entre v_1 et v_2.

7.4.3 Définition Formelle de l'Algorithme Probabiliste

Pour définir formellement notre algorithme, nous définissons les composants (`randHsR`, `randHsP`, `randHsI`) et nous prouvons les hypothèses (`randHsI1`, `randHSI2`, `randHSP1`, `randHsRind`) pour pouvoir faire une structure `hsAlgo` (Définition 7.12 page 98).

La séquence de règle `randHsR` correspond à une seule règle locale qui est `randHSLoc` (Définition 7.16).

La fonction de rendez-vous `randHsP` renvoie `None` si le sommet n'a pas de rendez-vous ou `Some i` si le sommet est en rendez-vous sur son

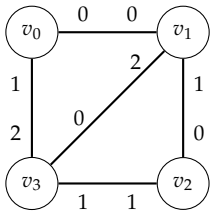


FIGURE 7.3 – *Graphe avec une numérotation possible de ses ports.*

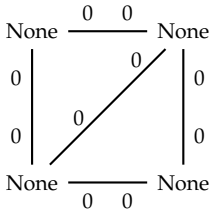


FIGURE 7.4 – *Graphe initial étiqueté.*

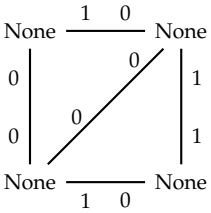


FIGURE 7.5 – *Graphe final étiqueté.*

$i^{\text{ème}}$ port qui correspond à celui qu'il a choisi. Pour cela on définit la fonction `agreed` qui renvoie vrai si le rang de **1** dans la zone d'écriture correspond au rang de **1** dans la zone de lecture (*i.e.*, si on a une arête étiquetée sur ses deux ports avec des **1**).

L'état initial est celui où toutes les étiquettes des sommets sont à `None` et toutes celles des ports valent **0**.

Définition 7.18
```
Definition randHsR : FLocT Λ Ψ :=
(randHSLoc::nil).

   Definition randHsP λ ψout ψin : option nat :=
if (agreed ψout ψin) then Some (index true ψout) else None.

   Definition randHsI V Adj G: State Adj :=
(finfun [ffun v ⇒ None], finfun [ffun p ⇒ false]).
```

Nous prouvons les hypothèses que doit satisfaire tout algorithme résolvant le problème du rendez-vous pour tout graphe : la cohérence de l'état initial, l'uniformité pour l'état initial, le domaine de la fonction de rendez-vous et la préservation de la cohérence par un pas de calcul.

Lemme 7.4
```
Lemma rda.handshake_rand.randHsI1 V Adj G δ
(Hδ1:∀ v w,Adj v w=w ∈(δ v))(Hδ2:∀ v,uniq(δ v)) p0:
consistent randHsP (randHsI G) δ.

Lemma rda.handshake_rand.randHsI2 V Adj G:
Uniform (randHsI G).

Lemma rda.handshake_rand.randHsP1 V Adj G δ
(Hδ1:∀ v w,Adj v w=w ∈(δ v))(Hδ2:∀ v,uniq(δ v)) p0 σ v:
hsPortR randHsP σ v = Some i → i < deg G v.

Lemma rda.handshake_rand.randHsRind V Adj G δ
(Hδ1:∀ v w,Adj v w=w ∈(δ v))(Hδ2:∀ v,uniq(δ v)) p0:
Stable (fun σ ⇒ (consistent randHsP σ δ) (nextState randHsR δ).
```

Nous pouvons alors construire notre algorithme :

Définition 7.19 *Definition randhs : (hsAlgo Λ Ψ) :=*
(Build_hsAlgo randHsI1 randHsI2 randHsP1 randHsRind).

Enfin, nous prouvons le lemme ci-dessous : cet algorithme n'est pas déterministe. Cette propriété diffère de l'hypothèse faite dans le cas du résultat d'impossibilité qui considère des algorithmes déterministes.

Lemme 7.5 `(rda.handshake_rand.NonADet). ∼ Adet (HsR randhs).`

7.4.4 Correction de l'Algorithme Probabiliste

Invariant du Couplage

Nous montrons que l'algorithme produit toujours des couplages. Cette propriété se déduit facilement de la propriété de cohérence de l'algorithme.

Lemme 7.6
```
(rda.handshake_rand.randHsInvariant_matching).
Invariant (fun σ⇒matching (fun v⇒assNeigh (HsP randhs) v σ δ))
          (nextState (HsR randhs) δ) (HsI randhs G).
```

Réalisation d'un Rendez-vous

Nous montrons qu'il existe au moins une exécution de l'algorithme qui crée un rendez-vous.

Lemme 7.7 `(rda.handshake_rand.Real) : hsRealisation randhs.`

Pour montrer ce lemme, sachant qu'il existe toujours au moins une arête $\{u, v\}$ dans le graphe, nous considérons l'étiquetage tel que les ports de cette arête portent l'étiquette **1** et les autres ports des sommets u et v portent l'étiquette **0**. Pour tous les autres sommets, nous étiquetons par défaut le premier port à **1** et tous les autres à **0**. On montre que cet étiquetage peut être obtenu par une exécution de l'algorithme `randhs` et que cet étiquetage contient un rendez-vous (celui sur l'arête $\{u, v\}$).

Probabilité d'obtenir un Rendez-vous

Nous montrons que la probabilité d'obtenir un rendez-vous est supérieure à une certaine constante quelque soit le graphe. Une étude plus approfondie de cet algorithme est similaire à celle de l'algorithme présentée dans la Section 8.5.2 page 120.

Théorème 7.1
```
(rda.handshake_rand.rand_hsexists) ∀ σ,
μ(Distem (randHSRound σ)) (fun s⇒hsExists(assNeigh randHsP δ s))
≥ 1 − e^{−1/2}
```

Modélisation pour l'Analyse

8

Sommaire

Dans ce chapitre, nous traitons l'analyse formelle des algorithmes distribués probabilistes. Des techniques générales sont énoncées et nous appliquons ces méthodes sur les algorithmes résolvant les problèmes de la brisure de symétrie, du rendez-vous, du couplage maximal et du MIS.

Dans tout ce chapitre, nous considérons un graphe G dont l'ensemble des sommets est V et la relation d'adjacence est `Adj` et muni d'une fonction de numérotation des ports δ vérifiant les deux hypothèses `H`δ`1` et `H`δ`2` (cf. Section 6.2.2 page 89). Lorsque certains arguments de fonctions *Coq* se déduisent du contexte, nous ne les noterons pas afin d'alléger l'écriture du code.

8.1 Les Trois Algorithmes Distribués Probabilistes

Dans cette section, nous exprimons dans notre syntaxe les algorithmes du rendez-vous, du couplage maximal et du MIS. Nous dénoterons par *algorithme papier* les algorithmes tirés de la littérature. Nous en donnons ensuite une définition semi-formelle avant de les exprimer dans notre syntaxe.

8.1.1 Rendez-vous

Nous considérons ici un algorithme un peu plus complexe que celui présenté dans la Section 7.4.1 page 100. Au lieu d'appliquer l'algorithme sur tous les sommets, l'algorithme local ne sera appliqué que sur les sommets actifs. Cela permettra de procéder à la construction d'un couplage maximal si on itère cet algorithme comme nous le verrons dans la Section 8.1.2 page 109.

Description Semi-Formelle

L'analyse de l'algorithme $\mathcal{A}_{hs}$ (Section 5.5.2 page 80) se fait sur une ronde. Nous définissons (Algorithme 2) alors semi-formellement l'algorithme local qui correspond `hs_local` et une ronde qui applique cet algorithme à tous les sommets `hs`.

Nous souhaitons utiliser l'algorithme qui résout le rendez-vous dans la description de l'algorithme qui résout le couplage maximal. Cela entraîne que l'algorithme $\mathcal{A}_{hs}$ ne s'applique que sur les sommets actifs. De plus, on voudrait savoir si un sommet est dans un rendez-vous et avec qui, s'il est isolé ou s'il est encore en train de calculer.

L'état local d'un sommet v est alors :
- `None` : le sommet est encore en activité ;
- `Some i` : si $i <$`deg` v alors le sommet est en rendez-vous avec son $i^{\text{ème}}$ voisin sinon il est isolé.

Les messages sont codés sur des bits. L'état local d'un port (v, w) est :
- **1** : v a choisi w,
- **0** : v n'a pas choisi w.

À l'initialisation tous les sommets sont actifs (étiquettes des sommets à `None`) et le font savoir à leurs voisins (étiquettes des ports à **1**).

Description formelle

Chaque sommet exécute l'algorithme local `HSLoc` (cf. Définition 8.1) : si un sommet est actif et s'il a des voisins, il choisit uniformément au hasard un voisin et informe tous ses voisins s'ils ont été choisis ou pas.

Algorithme 2: L'algorithme papier pour le rendez-vous

```
Algorithme hs
   Chaque sommet v applique simultanément l'algorithme
local : (hs_local v)

Algorithme (hs_local v)
   si v est actif alors
     si v a des voisins actifs alors
       v choisit uniformément au hasard l'un de ses
       voisins actifs, le i^ème voisin
       v envoie 0 à tous ses voisins sauf au i^ème à qui il
       envoie 1
       v prend pour état Some i
     sinon v prend pour état Some d où d est son degré
```

Nous verrons dans la section suivante que ces deux conditions sont nécessaires à l'itération.

Pour la définition de l'algorithme local, nous avons recours aux trois fonctions suivantes.

- La fonction `(active λ)` renvoie vrai si l'étiquette λ vaut `None`, faux sinon.
- La fonction `(numberActive` ψ_{in}`)` renvoie le nombre de **1** que la zone de lecture ψ_{in} contient, c'est-à-dire le nombre de voisins actifs.
- La fonction `(sendChosen` n l`)` renvoie une séquence de booléens de la taille de son argument l tous mis à **0**, sauf un élément dont le rang est égal au rang du $n^{\text{ème}}$ élément vrai de l.

Définition 8.1

```
Definition HSLoc (λ:Λ) (ψ_out ψ_in:seq Ψ) : FR (Λ × seq Ψ) :=
if (active λ) then
  match (numberActive ψ_in) with
      |O ⇒ return (Some |ψ_out|,nseq |ψ_out| false)
      |S n ⇒ let k = (random n) in return(λ,sendChosen(k+1)ψ_in))
  end
else return (λ, ψ_out).
```

Le graphe est composé de sommets actifs et de sommets inactifs. Nous souhaitons qu'un rendez-vous se produise dans le sous-graphe actif mais pas dans le sous-graphe inactif. Initialement, nous supposons que l'étiquetage `initState` passé en paramètre est tel que tous les voisins actifs ont envoyé **1** à leurs voisins et les voisins inactifs ont envoyé **0**. La calcul local fait par un sommet v consiste à choisir un nombre entre 0 et le nombre de ses voisins actifs décrémenté de 1. L'étiquette d'un sommet est égale à `None` si ce sommet est inactif ou à `Some` i s'il a choisi son $i^{\text{ème}}$ voisin. Si le sommet n'a choisi personne mais est isolé, alors il portera l'étiquette `Some` i où i est son degré. Si le sommet est inactif alors il ne fait aucune modification.

Remarque 8.1 *Le fait de permettre de ne traiter que le sous-graphe actif est une généralisation du problème du rendez-vous. En effet, pour traiter le problème tel qu'il a été résolu dans la Section 7.4.1 page 100, il suffit de considérer que tous les sommets sont actifs. Cet ajout d'état nous permettra de traiter l'itération et plus particulièrement de résoudre le problème du couplage maximal.*

Nous avons simulé le calcul local pour un sommet v qui consiste à choisir un de ses voisins actifs et à lui envoyer **1** et **0** aux autres.

Nous définissons une ronde pour le rendez-vous, où `sV` correspond à une énumération des sommets du graphe, comme suit :

Définition 8.2 *Definition HSRound (sV:seq V) (σ: State) := FRound sV σ HSLoc.*

8.1.2 Couplage Maximal

Afin d'obtenir un couplage maximal, il suffit d'itérer l'algorithme du rendez-vous sur les sommets actifs, jusqu'à ce que plus aucun sommet ne soit actif.

Description Semi-Formelle

À partir de l'Algorithme $\mathcal{A}_{mm}$ (Section 5.5.3 page 81) , nous définissons semi-formellement l'Algorithme 3.

Algorithme 3: L'algorithme papier pour le couplage maximal

```
Algorithme mm
   Chaque sommet v répète, tant qu'il est actif,
l'algorithme local : (mm_local v)

Algorithme (mm_local v)
   v exécute l'algorithme hs_local
   Si v reçoit 1 du voisin qu'il a choisi alors il
devient inactif
```

L'état initial et l'état local des sommets sont les mêmes que ceux utilisés par l'algorithme `hs`. Les messages envoyés sont encodés sur des bits.

Description Formelle

Nous voyons que cet algorithme doit se faire en deux phases : tout d'abord chaque sommet essaie de réaliser un rendez-vous, ensuite s'il a réussi, il devient inactif et le fait savoir à ses voisins. Ainsi cet algorithme repose sur deux algorithmes locaux successifs exprimés dans la Définition 8.3.

Définition 8.3 *Definition MMLoc1 (λ:Λ) (ψ_{out} ψ_{in}:seq Ψ) : FR (Λ × seq Ψ) := HSLoc λ ψ_{out} ψ_{in}.*

```
Definition MMLoc2 (λ:Λ) (ψout ψin:seq Ψ) : FR (Λ × seq Ψ) :=
if (active λ) then
  if (agreed ψout ψin) then
     return ( Some (index true ψout) , nseq |ψout| 0)
  else return (None, nseq |ψout| true)
else return (λ, ψout).
```

Le premier algorithme local `MMLoc1` consiste à faire un choix et à envoyer son choix, ce qui correspond à l'algorithme `HSLoc`.

Le second algorithme local `MMLoc2` met à jour l'activité d'un sommet. Si deux sommets actifs v et w se sont donnés rendez-vous, c'est qu'ils ont les étiquettes de leurs ports de sortie mis à **1** (sur les ports (v, w) et (w, v))

et **0** pour le reste ; ils mettent alors à jour leur statut avec le numéro de port qu'ils ont choisi pour le rendez-vous et envoient **0** aux voisins pour signaler qu'ils sont inactifs. S'ils sont actifs et qu'ils n'ont pas réussi à faire de rendez-vous alors ils restent actifs et le signalent à leurs voisins en envoyant **1**. S'ils sont inactifs, ils ne modifient rien.

La Définition 8.4 correspond à l'application de ces deux algorithmes locaux à chaque sommet de façon successive.

Définition 8.4 *Definition MMStep (sV:seq V) (σ:State):= FStep (MMLoc1::MMLoc2::nil) sV σ.*

Il est possible de répéter cet algorithme un certain nombre de fois. En voici la définition :

Définition 8.5 *Definition MMMC (n:nat) (sV:seq V) (σ:State):= FMC n (MMLoc1::MMLoc2::nil) sV res.*

Au début de l'algorithme, le graphe initial ne contient que des sommets actifs (état `None`) et que des messages à **0** pour indiquer que personne n'a fait de choix. À la fin de l'algorithme, s'il y a un couplage maximal le graphe est composé uniquement de sommets inactifs (état à `Some x` où x indique le choix s'il y en a un ou un nombre supérieur au degré du sommet s'il n'a choisi personne) et ne contient que des messages à **0**.

8.1.3 MIS

Description Semi-Formelle

À partir de l'Algorithme $\mathcal{A}_{mis}$ (Section 5.5.4 page 81) , nous définissons semi-formellement l'Algorithme 4. Il prend en paramètre un entier c. Suivant ce paramètre, l'algorithme est plus ou moins rapide. L'algorithme local ne s'applique que sur les sommets actifs.

Algorithme 4: L'algorithme papier pour le MIS

Algorithme (mis c)
Chaque sommet v répète simultanément, tant qu'il est actif, l'algorithme local : (mis_local v c)

Algorithme (mis_local v c)
v choisit uniformément au hasard un nombre entre 0 et c et l'envoie à ses voisins
v reçoit $c(i)$ de chacun de ses voisins actifs
Si pour tous les voisins actifs i de v on a $c > c(i)$ alors v rentre dans le MIS, devient inactif et envoie **1** à tous ses voisins, sinon il envoie **0**
Si v reçoit **1** alors il rentre dans le complément du MIS, devient inactif et envoie **0** à tous ses voisins, sinon il envoie **1**
Si v reçoit **0** de w alors il met à jour sa liste de voisins actifs en y supprimant w.

Une fois encore nous voyons qu'il y a des sommets actifs et inactifs.

L'état local d'un sommet v est alors :
– `None` : le sommet est encore en activité ;
– `Some` **1** : le sommet est dans le MIS ;
– `Some` **0** : le sommet est dans le complément du MIS.
Les messages sont codés sur des bits. L'état local d'un port (v, w) est :
– **1** : v a choisi w,
– **0** : v n'a pas choisi w.
À l'initialisation tous les sommets sont actifs (étiquettes des sommets à `None`) et le font savoir à leurs voisins (étiquettes des ports à **1**).

Description Formelle

L'algorithme du MIS repose sur trois rondes successives où les algorithmes locaux sont décrits dans la Définition 8.6.

Définition 8.6

```
Definition MISLoc1 (λ:Λ) (ψout ψin:seq Ψ) : FR (Λ × seq Ψ) :=
if (active λ) then
  let l = (getRandSeq ψin) in (return (λ, l))
else return (λ, nseq (seq.size ψin) 0).

Definition MISLoc2 (λ:Λ) (ψout ψin:seq Ψ) : FR (Λ × seq Ψ) :=
if (active λ) then
  if (supNeigh ψout ψin) then
    return (Some true, nseq (seq.size ψin) 1)
  else return (None, nseq (seq.size ψin) 0)
else return (λ, nseq (seq.size ψin) 0).

Definition MISLoc3 (λ:Λ) (ψout ψin:seq Ψ) : FR (Λ × seq Ψ) :=
if (active λ) then
  if (hasRec1 ψin) then
    return (Some false, nseq (seq.size ψin) 0)
  else return (None, nseq (seq.size ψin) 1)
else return (λ, nseq (seq.size ψin) 0).
```

L'algorithme local `MISLoc1` s'applique à un sommet actif et consiste à envoyer un nombre aléatoire différent à chacun de ses voisins grâce à la fonction `getRandSeq`.

L'algorithme local `MISLoc2` appliqué à chaque sommet le désigne, le cas échéant, en tant que maximum local à distance 1 (fonction `supNeigh`). Si c'est le cas, il rentre dans le MIS (son état passe à `Some true`) et le signale à ses voisins en envoyant **1**. Si ce n'est pas le cas, il reste actif (`None`) et envoie **0** à ses voisins.

L'algorithme local `MISLoc3` s'applique aux sommets actifs. Si ces sommets ont reçu un **1** (`hasRec1`) alors, ils entrent dans le complément du MIS (statut à `Some false`) et envoient **0** à leur voisin pour signifier qu'ils sont devenus inactifs. Si ce n'est pas le cas, ils restent actifs et envoient **1** à leurs voisins.

La succession de ces trois algorithmes (cf. Définition 8.7) correspond à l'algorithme de la Section 5.5.4 page 81.

Définition 8.7

```
Definition MISStep (sV:seq V) (σ:State):=
FStep n(MISLoc1::MISLoc2::MISLoc3::nil) sV σ.
```

L'itération de cet algorithme n fois est définie comme suit :

Définition 8.8 *Definition MISMC (n:nat) (sV : seq V) (σ:State):= FMC n (MISLoc1::MISLoc2::MISLoc3::nil) sV σ.*

8.2 Sémantique Distributionnelle

Nous faisons de nombreuses simplifications quant à la syntaxe définie dans la Section 6.2 page 88. Nous avons les fonctions correspondantes dans la sémantique distributionnelle, syntaxiquement la première lettre `F` est remplacée par un `D`. Nous avons alors défini les fonctions suivantes et prouvé les lemmes d'équivalence correspondant :

- `DRound`,
- `DStep`,
- `DMC`.

À titre d'exemple, nous montrons la simplification pour la fonction `DRound` et le lemme d'équivalence permettant de passer de la définition générique à la spécifique.

Définition 8.9

```
Definition DLocT:= Λ → (seq Ψ) → (seq Ψ) → distr (Λ*seq Ψ).

Fixpoint DRound (sV:seq V)(σ:State)(LR:DLocT):distr (State) :=
   match sV with
    |nil ⇒ Dreturn σ
    |v::t ⇒ Dlet s = (DRound t σ LR) in
      Dlet p = (LR (read σ v) in
          Dreturn (write s v p)
  end.
```

Remarque 8.2 *Les fonctions `Dreturn` et `Dlet` correspondent dans la bibliothèque Alea respectivement à `Munit` et `Mlet`.*

Lemme 8.1

```
Variable DLr: DLocT.
Variable FLr: FLocT Λ Ψ.

Hypothesis LocalRule1: ∀ ls l1 l2,
(Distsem (FLr ls l1 l2)) = (DLr ls l1 l2).

Lemma rda.rdaTool_dist.DF_eq1 : ∀ (sV: seq V) (σ:State),
Distsem (FRound sV σ FLr) = DRound sV σ DLr.
```

Nous nous intéresserons par la suite uniquement à l'analyse et utiliserons la syntaxe simplifiée.

8.3 Validité du modèle

Un algorithme local opère par changements locaux et utilise des primitives de randomisation. Pour exprimer ces algorithmes dans un langage fonctionnel nous avons eu recours à la bibliothèque *Alea* [6].

Les algorithmes probabilistes requièrent la primitive `(Random n)` qui retourne un entier naturel compris entre 0 et n avec une probabilité uniforme $\frac{1}{n+1}$. Les appels à cette fonction sont indépendants deux à deux.

Dans une ronde `DRound`, l'algorithme local probabiliste `LR` est appliqué à une séquence de sommets de G. Le résultat obtenu est une distribution de fonctions d'étiquetages sur les sommets et sur les ports.

8.3.1 Aspect Distribué.

L'algorithme simule l'envoi des messages en mettant à jour l'état σ pour chaque sommet. La fonction `read` appliquée à un sommet v ne donne que l'information locale relative à ce sommet : son propre état, et les états de ses ports d'entrée et sortie dans l'ordre donné par la numérotation des ports. La fonction `write` appliquée à un sommet v met à jour l'état global en ne récrivant que sur la zone de réécriture du sommet v. Ces fonctions sont déterministes. Il est à noter que comme les zones de réécriture sont disjointes deux à deux (les ré-étiquetages ne s'entrelacent pas), deux appels à la fonction `write`, chacun appliqué à des sommets différents, permutent. Il est ainsi équivalent d'appliquer cette fonction d'abord au sommet v puis au sommet w que de l'appliquer d'abord à w puis à v.

Lemme 8.2 `(graph.labelling.write_comm).` $\forall\ v\ w,\ v \neq w \Rightarrow$

$$(\texttt{write}\ (\texttt{write}\ \sigma\ w\ c_2)\ v\ c_1) = (\texttt{write}\ (\texttt{write}\ \sigma\ v\ c_1)\ w\ c_2).$$

Dans notre expression, une ronde dépend de l'implémentation de la fonction d'énumération des sommets que l'on passe en entrée : `(enum V)`. Elle décrit séquentiellement la simulation de l'application de la fonction locale de façon simultanée sur tous les sommets. En fait, notre système est distribué, cela implique que plusieurs sommets peuvent réétiqueter leur zone de réécriture en même temps. Nous pouvons alors séquentialiser l'algorithme. Pour cela, nous avons à montrer que l'ordre d'application de l'algorithme local n'a aucune influence sur le résultat. Cette propriété est assurée par le fait que le résultat est insensible à l'ordre dans lequel on applique la fonction `write` lorsque les entrées sont différentes. Ainsi, le résultat sera le même que celui obtenu si les sommets avaient exécuté l'algorithme en même temps.

Lemme 8.3 `(rda.rdaTool_dist.DRoundCommute3).`
Soit σ un état de G et Lr une règle locale *discrétisable* (pouvant s'écrire sous forme d'une somme).
Soit lv une séquence de sommets de G. Soit lv' une permutation de lv, nous avons :

$$\texttt{Dround}\ lv\ \sigma\ \texttt{LR} = \texttt{Dround}\ lv'\ \sigma\ \texttt{LR}.$$

8.3.2 Aspect Probabiliste.

L'aspect probabiliste a été évoqué dans le chapitre précédent. Nous rappelons ici une définition et deux lemmes importants.

Dans *Alea*, une expression probabiliste $(e : \tau)$ est interprétée comme une distribution dont le type est $(\tau \to [0,1]) \to [0,1]$. Ce type est noté `distr` τ. Nous utiliserons la notation $(\mu\ e)$ pour représenter la mesure associée à l'expression e.

Définition 8.10 *Soit Q une propriété et soit $\mathbb{I}_Q$ sa fonction caractéristique. La probabilité que le résultat de l'expression e satisfasse Q est représentée par $\mu\ e\ \mathbb{I}_Q$.*

La plupart des preuves sont basées sur ces deux transformations :

Lemme 8.4 `(ALEA.Munit_simpl [6]).`
$\forall\ (P : \tau)\ (f : \tau \to [0,1]),\ (\mu\ (Dreturn P))\ f = (f\ P)$.

Lemme 8.5 `(ALEA.Mlet_simpl [6]).`
`∀ (P Q : τ) (f : τ → [0,1]),`
`(μ (Dlet x = P in (Q x)) f = (μ P) (fun x ⇒ (μ (Q x)) f).`

8.4 Résultats Généraux

La fonction `DRound` opère séquentiellement sur un ensemble de sommets, plus précisément, c'est une application de la fonction locale `LR` sur une séquence de sommets : `Dround (`$v_1 :: v_2 :: \ldots :: v_n$`)` revient à exécuter `LR` v_1 `(LR` v_2 `(..LR` v_n σ`)..)`.

La fonction `LR` est locale, c'est-à-dire qu'elle affecte seulement un sommet et ses ports relatifs. De plus, elle peut contenir des primitives probabilistes dont les appels sont indépendants les uns des autres. Nous avons développé les techniques de preuves suivantes : permutabilité, composition, résultat local transformé en global, probabilité non nulle. Nous avons aussi considéré des algorithmes qui peuvent se terminer avec probabilité 1.

Nous utilisons la dénomination `is_discrete` pour exprimer qu'une règle est discrétisable, c'est-à-dire peut s'écrire sous la forme d'une somme. La propriété `Term` décrit qu'une fonction termine, c'est-à-dire que sa mesure avec la fonction $\mathbb{1}$ vaut 1. Nous utilisons la fonction `is_disjoint` pour spécifier que deux ensembles sont disjoints.

8.4.1 Permutabilité de l'Entrée

Tous les sommets effectuent les mêmes tâches en simultané. Leurs actions affectent leurs ports. Les zones d'influence des sommets sont disjointes deux à deux. Par conséquent, il n'y a pas de différence s'ils exécutent leurs tâches les uns après les autres. Pour simuler un algorithme distribué, nous pouvons le décrire comme un algorithme local appliqué à chaque sommet l'un après l'autre et ensuite prouver que l'ordre n'est pas important.

Nous avons prouvé (cf. Lemme 8.3) que pour toutes les séquences s_1 et s_2 telles que s_2 est une permutation de s_1, `(DRound` s_1`)` renvoie le même résultat que `(DRound` s_2`)`.

Ainsi, si l'on considère l'étiquetage σ et la séquence $(v :: s)$ où v est un sommet et s une suite de sommets où v n'apparaît pas, cela revient au même d'incorporer le résultat de la fonction locale appliquée à v dans `DRound` s σ que d'incorporer le résultat de `DRound` s σ dans le résultat de la fonction locale appliquée à v. Plus formellement, soit `LR` une règle locale discrétisable :

Lemme 8.6
```
(rda.rdaTool_dist.DRoundcons2).
 ∀ s σ, (∀ w, is_discrete (LR (read σ w))) →
  DRound s σ LR=
    if (null s) then σ
      else Dlet c = LR (read σ (head s) ) in
        write   (DRound (tail s) σ LR) (head s) c.
```

La preuve de ce lemme est basée sur la discrétisation de la mesure de la fonction locale `LR`, qui est sa réécriture en une somme finie.

8.4.2 Composition

Une façon de prouver des propriétés sur DRound est de procéder par récurrence sur la séquence de sommets. Par exemple, nous prouvons qu'une ronde termine avec probabilité 1, en supposant que la règle locale LR termine :

Lemme 8.7 (rda.rdaTool_dist.DRound_total).
∀ σ s, (∀ v, Term (LR (read v))) →
(μ (DRound s σ LR)) $\mathbb{I} = 1$.

Démonstration. Par récurrence sur s. Supposons que la propriété est vérifiée pour s', nous montrons qu'elle l'est pour $s = (v :: s')$, soit :

$$\forall\ \sigma\ s'\ v, (\mu\ (\texttt{DRound}\ v :: s'\ \sigma))\ \mathbb{I} = 1.$$

En utilisant la définition de DRound, cette expression devient :

$$\begin{array}{l}\forall\ \sigma\ s'\ v,\ (\mu\ (\texttt{Dlet r = DRound s' } \sigma \texttt{ in} \\ \quad \texttt{Dlet c = LR (read } \sigma \texttt{ v) in} \\ \quad\quad \texttt{write r v c))}\ \mathbb{I} = 1.\end{array}$$

Les transformations des lemmes 8.5 et 8.4 donnent :

$$\begin{array}{l}\forall\ \sigma\ s'\ v,\ (\mu\ \ (\texttt{DRound s' } \sigma)) \\ \texttt{(fun r} \Rightarrow\ (\mu\ \texttt{(LR (read } \sigma \texttt{ v)))}\ \mathbb{I}) = 1.\end{array}$$

Si LR termine, alors comme la définition de la fonction caractéristique $\mathbb{I}$ est (fun $x \Rightarrow 1$), il nous faut montrer que :

$$\forall\ \sigma\ v,\ (\mu\ \texttt{DRound}\ s'\ \sigma)\ \mathbb{I} = 1.$$

Ce qui est l'hypothèse de récurrence. D'où le résultat. □

8.4.3 Résultat Local Transformé en Résultat Global

Sous certaines hypothèses, il est possible de reporter un résultat local en un résultat global :

Lemme 8.8 (rda.rdaTool_dist.FLocalGlobal).
Soit v un sommet et $x \in [0,1]$, si :
- ∀ w s λ ψ, LR (read s w) = LR (read (write s u λ ψ) w),
- ∀ w, is_discrete (LR (read σ w)),
- ∀ w, Term (LR (read σ w)),
- ∀ u ≠ w, is_disjoint (WriteArea u) (WriteArea v),
- ∀ w s λ ψ, $\mathbb{I}_{Elw}$ (λ, ψ) = $\mathbb{I}_{Egw}$ (write s w λ ψ),

alors :
μ(LR (read σ v))$\mathbb{I}_{Elv} = x \Rightarrow \mu$(DRound (enum V) σ LR)$\mathbb{I}_{Egv} = x$.

8.4.4 Terminaison

Les algorithmes peuvent s'itérer un nombre fini de fois quitte à ne renvoyer un résultat juste qu'avec une certaine probabilité. Ce sont des algorithmes de type Monte Carlo. Les algorithmes de type Las Vegas renvoient toujours un résultat correct, mais certaines de leurs exécutions peuvent être infinies. Notre modélisation donne le choix à l'utilisateur d'étudier les algorithmes sous forme Monte Carlo ou Las Vegas. Les algorithmes Monte Carlo ont déjà été définis de façon syntaxique. Nous allons nous intéresser dans cette section à l'implémentation sous la forme Las Vegas.

Description.

Les algorithmes Las Vegas peuvent avoir des exécutions infinies. La condition d'arrêt que nous considérons est locale à chaque sommet. Ainsi un sommet, dans un état actif, exécute son calcul local, alors qu'un sommet dans un état inactif ne fait rien. Lorsqu'un sommet est dans un état inactif, d'autres sommets peuvent eux être actifs. Or nous procédons par ronde et appliquons l'algorithme local à chaque sommet. Ainsi, nous avons traduit l'opération « ne rien faire » par le renvoi de la vue locale actuelle, *i.e.*, la fonction identité. L'inconvénient est alors que lorsque tous les sommets sont inactifs, l'algorithme s'exécute toujours alors qu'un état stationnaire est atteint. L'étiquetage ne change plus.

Dans ce cas-là pourtant, lorsqu'un état final est atteint, nous considérons que l'algorithme est terminé. C'est pourquoi pour analyser l'algorithme nous l'interprétons en vérifiant avant de lancer une ronde de l'algorithme local sur tous les sommets s'il existe encore un sommet actif. S'il n'en existe plus nous renvoyons l'état actuel que nous pourrons alors étudier.

La définition 8.11 nous présente la classe d'algorithmes distribués probabilistes que nous étudions. La fonction globale `FGlobal` contient une propriété de terminaison locale qui est vérifiée sur tous les sommets avant de lancer ou pas une ronde, ou plus généralement la fonction `TermB` qui est vérifiée sur l'étiquetage dans sa globalité.

Définition 8.11
```
Fix DLV (sV: seq V)(σ: State) (LR: seq DLocT)
(TermB: State → bool) : distr (State) :=
  if (TermB σ) then Dreturn σ
  else Dlet r = (DStep LR sV σ) in
    DLV sV r LR TermB.
```

Algorithmes avec Exécution Infinie

Un algorithme distribué probabiliste répète une ronde jusqu'à ce qu'une certaine propriété soit vérifiée par l'étiquetage du graphe. En général, cette propriété est que tous les sommets arrêtent d'interagir entre eux, *i.e.*, jusqu'à ce qu'ils soient inactifs.

Faire ressortir un variant qui se décrémente à chaque ronde permet de prouver la terminaison. Or nous avons vu que pour certains algorithmes, il peut exister des exécutions infinies. Pour traiter ce genre de programmes, la bibliothèque *Alea* permet de considérer les limites. Ainsi, lorsqu'une fonction récursive (comme celle de la Définition 8.11) est introduite, elle est interprétée comme un point fixe et calculée comme étant le plus petit point fixe.

Terminaison avec Probabilité 1

Certes, toutes les exécutions ne se terminent pas toujours. Cependant, de nombreux algorithmes probabilistes ont la propriété de se terminer avec probabilité 1.

Nous avons cherché à caractériser ces algorithmes (par une classe plus

large : transformation d'états voir Définition 8.12) en mettant à la disposition de l'utilisateur le Lemme 8.9.

Définition 8.12 *Variable rd : State → distr State.*
Variable termB : State → bool

Fix fglobal σ :=
if (termB σ) then Dreturn σ
else Dlet r = (rd s) in
fglobal r.

Lemme 8.9 (rda.term.termglobal).
Pour toute transformation probabiliste d'un étiquetage en un autre rd : State → distr State, tout étiquetage σ, toute propriété de terminaison TermB, toute fonction (cardTermB: State → nat), tout réel c appartenant à $[0,1]$ et toute propriété sur les états (PR:State→bool), si on a :

1. ∀ s, Term (rd s)
2. ∀ s, cardTermB s = 0 → TermB s = true
3. $0 < c$
4. ∀ s, 0 < cardTermB s → PR s → c ≤ μ (rd s) $\mathbb{I}_{(cardTermB\ .<cardTermB\ s)}$
5. ∀ s, PR s → μ rd $\mathbb{I}_{(cardTermB\ s<cardTermB\ .)} = 0$
6. ∀ s f, PR s → μ (rd s) $\mathbb{I}_{PR.\ \wedge\ f.}$ = μ (rd s) $\mathbb{I}_f$
7. PR σ

alors
Term (fglobal rd TermB σ).

Nous obtenons alors le lemme `rda.rdaTool_dist.DPLV_total` en prenant pour transformation d'étiquetage `rd` la fonction suivante, où `LR` est la séquence de règles de calculs locaux : `DStep LR (enum V)`. Ainsi pour prouver qu'un algorithme distribué probabiliste Las Vegas termine avec probabilité 1, il suffit de montrer que la probabilité qu'un certain variant (qui s'il est nul implique la terminaison) sur l'étiquetage a une probabilité non nulle de décrémenter et a une probabilité nulle d'augmenter après une étape `DStep LR (enum V)`. La propriété `PR` permet de caractériser les étiquetages : l'avant-dernière hypothèse montre qu'elle n'influe pas sur le calcul des probabilités et la dernière hypothèse montre qu'elle est vraie pour l'étiquetage initial. En résumé, `PR` est une propriété qui est toujours vraie.

8.5 Applications

8.5.1 Brisure de Symétrie

Ce problème est le tout premier que nous avons traité pour l'analyse. Il consiste à briser la symétrie entre deux voisins dans un graphe complet à deux sommets. Localement, ces sommets ne peuvent se différencier. Une ronde consiste pour un sommet à tirer à pile ou face, à informer ses voisins et à mettre à jour son statut. La modélisation générique étant faite, il nous faut définir `LR`, l'algorithme local en *Coq*, que nous appellerons `sb_local`.

On considère donc un graphe avec deux sommets liés entre eux 0 et 1. Un sommet a pour étiquette locale (λ_1, λ_2) le couple suivant : son statut (actif ou inactif) et une séquence de booléens, qui correspond à ses tirages.

Les messages envoyés sont des booléens. Initialement, chaque sommet est actif, n'a aucun tirage et n'a envoyé aucun message.

L'algorithme local exécuté par chaque sommet v consiste en :

- si le dernier message envoyé est différent de celui reçu, alors v devient inactif, sinon il tire uniformément au hasard un booléen b ;
- b est concaténé à λ et b est écrit sur le port de sortie.

Cet algorithme `sb_local` est présenté dans la Définition 8.13. Chaque sommet n'a qu'un voisin, ainsi les séquences de messages sur les ports d'entrée et sortie ψ_{in} et ψ_{out} n'ont au plus qu'un élément. On y accède avec la fonction `hd`.

Description Semi-Formelle

On considère un graphe composé de deux sommets liés entre eux. L'algorithme `sb_local` appliqué à chaque sommet est le suivant : v tire aléatoirement un bit et l'envoie à son voisin ; il recommence jusqu'à ce que son bit tiré soit différent de celui reçu. L'état des sommets est la suite de bits tirés. L'algorithme global `sb` est décrit par l'Algorithme 5.

Algorithme 5: L'algorithme papier pour la brisure de symétrie

```
Algorithme sb
   Chaque sommet v applique simultanément l'algorithme
local (sb_local v) jusqu'à ce que les bits tirés soient
différents

Algorithme (sb_local v)
   v tire uniformément au hasard un bit b ∈ {0,1}
   v envoie b à son voisin
```

Description formelle

Définition 8.13 *Definition sb_local (λ:Λ) (ψ_{out} ψ_{in}:seq Ψ) : distr ($\Lambda \times$ seq Ψ) :=*
if (hd ψ_{in} == hd ψ_{out}) then
Mif Flip then
Dreturn ((false, (true$::\lambda_2$)), [$::$true])
else Dreturn ((false, (false$::\lambda_2$)), [$::$false])
else Dreturn ((true, λ_2), nil).

Remarque 8.3 *Notre algorithme utilise ici la fonction `Mif` d'Alea qui avec une distribution de booléens traite le cas vrai dans le `then` et le cas faux dans le `else`.*

Deux résultats ont été montrés. Afin de les exprimer nous avons recours aux définitions suivantes :

Définition 8.14 *Soit $\mathcal{SB}(v,w)$ l'évènement « l'étiquette de v est différente de l'étiquette de w ». Nous définissons par $\mathcal{S}(v,w)$ la fonction caractéristique de $\mathcal{SB}(v,w)$.*

Le premier résultat est que la probabilité que les sommets se différencient est $1/2$.

Lemme 8.10 $\forall \lambda\ \psi_{out}\ \psi_{in}$,
(hd ψ_{in} == hd ψ_{out}) $\Rightarrow$
μ (sb_local λ ψ_{out} ψ_{in}) $\mathcal{S}(0,1) = 1/2$.

On peut voir une ronde comme une épreuve de Bernoulli. Ainsi l'espérance est une variable aléatoire qui suit la loi géométrique de paramètre 1/2 qui est la probabilité qu'il y ait un succès, *i.e.*, que les sommets se différencient. Le Lemme 6.9 page 87 nous donne :

Lemme 8.11 $\lim_{n\to\infty}$ `(geom_exp 1/2 n)` $= 2$.

Pour obtenir une brisure de symétrie, nous répétons l'algorithme `sb_local`.

Définition 8.15 *Definition sb (σ: State) :=*
DLV σ sb_local (fun (λ_1,λ_2) ψ_{in} $\psi_{out} \Rightarrow \lambda_1 =$ ***0***).

En général, l'étiquetage initial (`initState`) est tel que chaque sommet est actif, a une liste de bits vide et n'a envoyé aucun message. L'algorithme défini par la Définition 8.15 a pour condition de terminaison que les deux sommets soient inactifs. Les deux sommets deviennent inactifs lorsque le message reçu est différent du message envoyé.

Voici l'analyse que nos résultats généraux ont permis de mener.

Terminaison.

Lemme 8.12 `(rda.symBreak.sb_term).` μ `(sb initState)` $\mathbb{I} = 1$.

Démonstration. La mesure correspond à la borne supérieure de la suite $p_0 = 0$ et $p_{i+1} = 1/2 + 1/2p_i$. Il a été prouvé dans *Alea* que $p_n = 1 - 1/2^n$. La borne supérieure $(sup\ p_n)$ est égale à 1. □

Correction.

Afin de montrer qu'à la fin de toute exécution les étiquettes de bits de chaque sommet sont différentes, il nous faut dans un premier temps définir la propriété caractéristique décidable `neq_c_dec`.

Définition 8.16 *Definition neq_c λ := (λ* ***1****) <> (λ* ***0****).*

Lemme 8.13 `(rda.symBreak.neq_c_dec).`
$\forall$`(σ: State){(neq_c σ.1)} + { (neq_c σ.1)}.`

Ensuite, il faut montrer que l'algorithme termine :

Lemme 8.14 `(rda.symBreak.Sb_breaks).`
μ `symBreak (fun k ⇒` $\mathbb{I}_{neq\_c\_dec}$`=1.`

Tirage de longueur supérieure à k.

La probabilité que le nombre de bits à tirer pour que les deux étiquettes diffèrent soit supérieur à k vaut $1/2^k$.

Lemme 8.15 `(rda.symBreak.probltlg).`
$\forall\ k,\ \mu$ `symBreak` $\mathbb{I}_{fun\ s\ \Rightarrow\ k<|(s\ \mathbf{1}).1|} = (1/2)^k$.

Espérance.

Nous pouvons prouver directement que l'espérance pour briser la symétrie vaut 2 :

Lemme 8.16 `(rda.symBreak.expectancySb).`
$\lim_{n\to\infty} \sum_{k=1}^{n} \mu$ `symBreak` $\mathbb{I}_{fun\ s \Rightarrow k<|(s\ \mathbf{1}).1|}$`==2.`

Démonstration. Il suffit de prouver le résultat de la série géométrique de paramètre `1/2` :

$$\lim_{n\to\infty} \sum_{k=1}^{n} \mu \text{ symBreak } \mathbb{I}_{fun\ s \Rightarrow k<|(s\ \mathbf{1}).1|} = \lim_{n\to\infty} \sum_{k=1}^{n} 1/2^k = 2.$$

□

8.5.2 Rendez-vous

Voici la définition du rendez-vous en sémantique distributionnelle. L'équivalence avec la Définition syntaxique 8.1 page 108 a été prouvée.

Définition 8.17
```
Definition DHSLoc (λ:Λ) (ψout ψin : seq Ψ) : dist (Λ * seq Ψ) :=
  if (active λ) then
    match (numberActive ψin) with
      |O ⇒ Dreturn (Some |ψout|, nseq |ψout|) false)
      |S n ⇒ Dlet k = (Random n)
         Dreturn (λ, sendChosen k.+1 ψin) in
    end
  else Dreturn (λ, ψout).

Definition DHS (sV: seq V) (σ:State) :=
DRound sV σ DHSLoc.
```

Application des Techniques Générales.

Permutabilité. La permutabilité découle directement du Lemme 8.6 page 114. En effet, l'hypothèse pour appliquer ce lemme est basée sur la discrétisation de la mesure de la fonction locale. Ici, c'est bien le cas, car l'écriture de la fonction `DHSLoc` est basée sur celle de `random` qui est elle-même discrète.

Composition. Nous avons vu qu'une technique générale ressort clairement de la preuve du Lemme 8.7 page 115. L'expression probabiliste peut être décomposée en la mesure sur un sommet et la mesure sur le reste. Par conséquent, si nous voulons prouver une propriété vérifiée par un sommet v, nous pouvons avoir recours à cette technique. Par exemple, soit $P(v,w)$ la propriété « *v choisit w* ». Soit sV la séquence de sommets du graphe. On a :

Lemme 8.17 `(rda.hsAct_dist.DHS_degv_global).` `∀ G σ {v,w},`
`(μ (DHS sV σ))` $\mathbb{I}_{P(v,w)} = 1/deg(v)$`.`

Démonstration. Nous avons prouvé que l'ordre importe peu (cf. Lemme 8.3 page 113). Soit $(sV\backslash v)$ la sous-séquence de sV dont les éléments égaux à v ont été supprimés. Nous pouvons écrire :

$$\forall\ G\ \sigma\ \{v,w\},\ (\mu\ (\text{DHS}\ (v :: (sV\backslash v))\ \sigma)\, \mathbb{I}_{P(v,w)} = 1/deg(v).$$

Nous pouvons ensuite appliquer la technique de composition. □

Résultat Local Transformé en Résultat Global. Comme la fonction LR est discrétisable et termine, et comme chaque sommet peut seulement changer l'étiquette de ses ports, il est possible de reporter quelques résultats locaux vers des globaux comme celui-ci :

- localement : la probabilité pour un sommet v de choisir son $i^{\text{ème}}$ voisin est $1/deg(v)$;
- globalement : la probabilité pour v de choisir w est $1/deg(v)$.

En effet, le fait pour un sommet de choisir localement un de ses ports est exprimé globalement par le fait qu'il choisisse un de ses voisins. Ainsi, nous avons la même probabilité dans les deux cas grâce au Lemme 8.8 page 115.

Analyse du Succès.

Nous étudions la probabilité de succès, c'est-à-dire la probabilité d'obtenir au moins un rendez-vous entre deux sommets voisins actifs dans un graphe $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ après une exécution de notre algorithme. Nous voulons prouver le Théorème 8.1 : cette probabilité est supérieure à $(1 - e^{-1/2})$. Nous supposons que le graphe d'entrée G contient au moins une arête dont les deux extrémités sont actives. Nous la nommerons, sans perte de généralité, e_1.

Ce résultat a été prouvé en Coq. Le lecteur est invité à se rapporter à [30] pour les détails des preuves en *Coq*.

Théorème 8.1 (hsAct.DHS_deg).
$\forall\ sV\ \sigma,\ \mu\ (DHS\ sV\ \sigma)\ (\exists\ e,\ \mathcal{H}(e)) \geq 1 - e^{-1/2}$.

Pour la suite de cette section, nous utiliserons le symbole $\mathbb{P}$ comme abréviation de la distribution $\mu(DHS\ sV\ \sigma)$. Il nous faut donc prouver :

$$\mathbb{P}(\exists\ e \in \mathsf{E},\ \mathcal{H}(e)) \geq 1 - e^{-1/2}.$$

La preuve de ce théorème est basée sur les deux résultats suivants : les Lemmes 8.18 et 8.19. Nous détaillons ci-dessous les étapes essentielles de la preuve :

Démonstration. **Preuve du Théorème 8.1**

Nous avons d'une part :
$\mathbb{P}(\exists\ e \in \mathsf{E},\ \mathcal{H}(e)) \quad = 1 - \mathbb{P}(\prod_{j=1}^{m}\ \overline{\mathcal{H}(e_j)})$.
D'autre part :

$$\begin{aligned}
\mathbb{P}(\prod_{j=1}^{m}\ \overline{\mathcal{H}(e_j)}) \quad &= \prod_{i=1}^{m}(1 - \mathbb{P}(\mathcal{H}(e_i) | \prod_{j=1}^{i-1} \overline{\mathcal{H}(e_j)})) \\
&\leq \prod_{i=1}^{m}(1 - \mathbb{P}(\mathcal{H}(e_i))) && \text{(cf. Lemme 8.18)} \\
&\leq \prod_{i=1}^{m}(1 - \frac{\sum_{j=1}^{m}\mathbb{P}(\mathcal{H}(e_j))}{m}) \\
&\leq (1 - \frac{\frac{1}{2}}{m})^m && \text{(*)} \\
&\leq e^{-\frac{1}{2}}.
\end{aligned}$$

Remarque 8.4 *(*) Cette étape consiste à prouver :* $\sum_{j=1}^{m}\mathbb{P}(\mathcal{H}(e_j)) \leq \frac{1}{2}$.
Et $\sum_{j=1}^{m}\mathbb{P}(\mathcal{H}(e_j)) = \sum_{j=1}^{m}\frac{1}{deg(e_j^1)*deg(e_j^2)}$ *où* $e_j = (e_j^1, e_j^2)$.
Le résultat classique suivant nous mène à la conclusion :
$\sum_{j=1}^{m}\frac{1}{deg(e_j^1)*deg(e_j^2)} \geq \frac{1}{2}$.

Remarque 8.5 *(*) Une autre remarque de cette étape porte sur l'utilisation des sommes dans Alea. Il faut s'assurer qu'aucun terme, ni leur somme ne dépasse* 1*. De plus, la manipulation des calculs sur l'ensemble* $[0,1]$ *est parfois fastidieuse. Nous avons alors eu recours à une transformation de cette expression dans l'espace des réels positifs, nous avons fait nos calculs avec moins de contraintes (car on peut dépasser* 1*) puis nous sommes repassés dans* $[0,1]$ *en nous assurant au préalable que le résultat était bien compris entre* 0 *et* 1*.*

□

Le Lemme 8.18 est utilisé pour la seconde étape décrite plus haut.

Lemme 8.18 (prelude.my_alea.Mcond_prodConjBound). Soit $\delta(e,e')$ le booléen valant **1** si les arêtes e et e' ne sont pas adjacentes et **0** sinon. Si les hypothèses suivantes sont vraies :

1. $\forall i \in 1..m,\ \mathbb{P}(\prod_{j=i+1}^{m} \overline{\mathcal{H}(e_j)}) \neq 0$
2. $\forall i \in 1..m,\ \mathcal{DHS}(e_i)$ et $\bigwedge_{j=i+1|\sim\delta(e_i,e_j)}^{m} \overline{\mathcal{DHS}(e_j)}$ sont indépendants
3. $\forall i \in 1..m, \mathcal{H}(e_i) * \prod_{j=i+1|\delta(e_i,e_j)}^{m} (\overline{\mathcal{H}(e_j)}) = \mathcal{H}(e_i)$

alors pour tout i dans $1..m$ et toute arête e,

$$\mathbb{P}(\mathcal{H}(e)) \leq \mathbb{P}(\mathcal{H}(e) | \prod_{j=i+1}^{m} \overline{\mathcal{H}(e_j)}).$$

Démonstration. La preuve de ce lemme repose sur une partition de E en deux sous-ensembles : le premier est composé des arêtes adjacentes à e et le second est composé de celles qui ne le sont pas :

$$\text{soit } A = \prod_{j=i+1|\delta(e_i,e_j)}^{m} \overline{\mathcal{H}(e_j)} \text{ et } B = \prod_{j=i+1|\sim\delta(e_i,e_j)}^{m} \overline{\mathcal{H}(e_j)}.$$

Nous pouvons écrire grâce à l'hypothèse 1. l'expression : $\frac{\mathbb{P}(B)}{\mathbb{P}(A*B)}$. Ensuite, nous avons montré que :

$$1 \leq \frac{\mathbb{P}(B)}{\mathbb{P}(A*B)}.$$

L'hypothèse 2. nous mène à :

$$\mathbb{P}(\mathcal{H}(e)) \leq \frac{\mathbb{P}(\mathcal{H}(e)*B)}{\mathbb{P}(A*B)}.$$

Finalement, l'hypothèse 3. nous donne le résultat :

$$\mathbb{P}(\mathcal{H}(e)) \leq \frac{\mathbb{P}(\mathcal{H}(e)*A*B)}{\mathbb{P}(A*B)} = \mathbb{P}(\mathcal{H}(e) | \prod_{j=i+1}^{m} \overline{\mathcal{H}(e_j)}).$$

□

Remarque 8.6 *Le Lemme 8.18 est l'application d'un lemme plus général qui prend en entrée un ensemble fini (ici, il est appliqué avec l'ensemble fini des arêtes). Pour plus de détails, le lecteur est invité à lire le développement en Coq.*

Remarque 8.7 *L'hypothèse 1. est nécessaire pour éviter une division par zéro. Elle n'était pas mentionnée dans la preuve et fut exigée comme obligation de preuve par Coq lors du développement.*

Remarque 8.8 *Les hypothèses 2. et 3. sont vérifiées par l'algorithme* `DHS`.

Lemme 8.19 `(rda.hsAct_dist.hs1).`
`Pour tout sous-ensemble` $S = \{e_{i+1},\ldots,e_m\}$ `d'arêtes, la probabilité qu'aucun rendez-vous ne se produise dans` S `n'est pas nulle, c'est-à-dire :`

$$\forall i \in 1..m,\ \mathbb{P}(\prod_{j=i+1}^{m} \overline{\mathcal{H}(e_j)}) \neq 0.$$

Démonstration. Considérons l'ensemble d'arêtes $E = e_1, \ldots, e_m$.

Pour prouver que cette probabilité n'est pas nulle, nous mettons en évidence un témoin qui est une exécution possible de l'algorithme et dans lequel il n'y a pas de rendez-vous sur les arêtes $e_{i+1}, \ldots, e_m$ (Lemme 6.3 page 86) où $i \in 1..m$.

Nous avons prouvé qu'il est toujours possible de construire une fonction de parenté représentant un arbre enraciné de n'importe quel graphe connexe $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ tel que la racine soit une extrémité de l'arête e_1.

Cependant, le graphe actif n'est pas forcément connexe. C'est pourquoi nous considérons le sous-graphe composé de tous les sommets actifs accessibles depuis l'arête e_1. La probabilité cherchée est inférieure à celle de ce sous-graphe (car les calculs locaux dans deux composantes connexes sont indépendants).

Nous en extrayons une fonction totale où la racine est associée à l'autre extrémité de l'arête e_1. Cet étiquetage peut être obtenu par notre algorithme. De plus, il assure qu'il n'y aura pas de rendez-vous dans le graphe à part peut-être en e_1 ce qui n'est pas un problème car nous considérons seulement les arêtes e_{i+1}, .., e_m. C'est pourquoi nous avons besoin comme hypothèse d'avoir au moins une arête active. □

Remarque 8.9 *Pour prouver que cette probabilité n'est pas nulle, nous avons utilisé le Lemme générique 6.3 page 86.*

8.5.3 Couplage Maximal

Voici la définition du couplage maximal en sémantique distributionnelle. L'équivalence avec la Définition syntaxique 8.3 page 109 a été prouvée.

Une application directe pour la terminaison est celle du couplage maximal. En effet, pour le réaliser, il faut itérer l'algorithme du rendez-vous sur les sommets actifs, et ceux qui sont en rendez-vous deviennent inactifs. L'état en argument est l'état initial où tous les sommets sont actifs.

Définition 8.18
```
Definition DMMLoc1 (λ:Λ) (ψout ψin:seq Ψ) : FR (Λ × seq Ψ) :=
  if (active λ) then
    if (agreed ψout ψin) then
       Dreturn ( Some (index true ψout) , ψout)
    else Dreturn (None, map (fun x ⇒ true) ψout)
  else Dreturn (λ, ψout).

  Definition DMMLoc2 (λ:Λ) (ψout ψin:seq Ψ) : FR (Λ × seq Ψ) :=
HSLoc λ ψout ψin.

  Definition termB (f: State) : bool :=
[∀ v, active (f.1 v)].
```

```
   Definition DMMLV (sv: seq V) (σ: State) :=
DLV sV σ (DMMLoc1::DMMLoc2::nil) termB.
```

Cet algorithme termine :

Lemme 8.20 `(rda.maxmatch_dist.DMMLV_term).`
$\forall\ \sigma,\ \mu(\texttt{DMMLV (enum V) } \sigma)\ \mathbb{I} = 1.$

Démonstration. Nous avons utilisé pour prouver ce lemme le résultat du Lemme 8.9 page 117. Nous avons montré dans un premier temps que la probabilité d'avoir au moins un rendez-vous en une ronde est égale à une certaine variable (Théorème 8.1 page 121) qui correspond au c de notre lemme de terminaison. Il suffit de prendre comme variant qui décrémente à chaque ronde le nombre de sommets actifs. La propriété `PR` des ces étiquetages que doit vérifiée l'initialisation et qui doit être stable après chaque étape est que tout sommet actif envoie **1** à tous ses voisins et tout sommet inactif envoie **0**. Il ne reste plus qu'à prouver les 7 hypothèses du Lemme 8.9. □

8.6 Perspectives

L'aspect probabiliste est traité grâce à la bibliothèque *Alea*. Nous avons étudié les problèmes du rendez-vous, de la brisure de symétrie et du couplage maximal.

Plusieurs résultats génériques ont été obtenus en *Coq* avec ce modèle et ont été appliqués aux problèmes : la permutabilité, la composition, la transformation d'un résultat local en global, la preuve que la probabilité de certains évènements est non nulle, la terminaison. Le développement de la bibliothèque permettant de raisonner sur ce modèle d'algorithmes distribués se trouve sur la page web [30].

Une autre application pour la terminaison est celle de l'ensemble maximal indépendant. Pour le réaliser, il faut itérer la succession de trois algorithmes locaux : `DMISLoc1` attribue des numéros au hasard sur les ports de la zone d'écriture de chaque sommet à l'aide de la fonction `DgetRandSeq`; `DMISLoc2` élit le maximum local et `DMISLoc3` termine la mise à jour de l'activité des sommets. Ces définitions sont équivalentes aux définitions syntaxiques 8.6 page 111.

Définition 8.19
```
Fixpoint DgetRandSeq (l: seq Ψ) : distr (seq Ψ) :=
  match l with
    |nil ⇒ Dreturn nil
    |t ::q ⇒ Dlet l' = (DgetRandSeq q) in
           Dlet x = (Random c) in (Dreturn (x.+1::l')))    end.

   Definition DMISLoc1 (λ:Λ) (ψ_out:seq Ψ) (ψ_in:seq Ψ):
distr (Λ * seq Ψ) :=
  if (active λ) then
    Dlet l = (DgetRandSeq ψ_in) in (Dreturn (λ, l))
  else Dreturn (λ, nseq (seq.size ψ_in) 0).

   Definition DMISLoc2 (λ:Λ) (ψ_out:seq Ψ) (ψ_in:seq Ψ):
distr (Λ * seq Ψ) :=
  if (active λ) then
    if (supNeigh ψ_out ψ_in) then
      Dreturn (Some true, nseq (seq.size ψ_in) 1)
```

```
    else Dreturn (None, nseq (seq.size ψin) 0)
  else Dreturn (λ, nseq (seq.size ψin) 0).

   Definition DMISLoc3 (λ:Λ) (ψout:seq Ψ) (ψin:seq Ψ):
distr (Λ * seq Ψ) :=
   if (active λ) then
    if (hasRec1 ψin) then
      Dreturn (Some false, nseq (seq.size ψin) 0)
    else Dreturn (None, nseq (seq.size ψin) 1)
  else Dreturn (λ, nseq (seq.size ψin) 0).

   Definition DMISStep (seqV : seq V ) (res: State) :=
   DPStep δ 0 p0 (DMISLoc1::DMISLoc2::DMISLoc3::nil) seqV res.

   Definition DMISMC (n:nat) (seqV : seq V ) (res: State) :=
   DPMC δ 0 p0 n (DMISLoc1::DMISLoc2::DMISLoc3::nil) seqV res.
```

Afin de prouver la terminaison, il suffirait de définir l'algorithme sous la version Las Vegas comme suit :

Définition 8.20

```
Definition termB (f: State) : bool :=
[∀ v, active (f.1 v)].

   Definition DMISLV (sv: seq V) (σ: State) :=
DLV sV σ (DMISLoc1::DMISLoc2::DMISLoc3::nil) termB.
```

Puis il faudrait prouver le lemme suivant en utilisant la même méthode que pour le couplage maximal, c'est-à-dire en prouvant d'abord que la probabilité d'avoir un sommet entrant dans le MIS en une ronde est strictement positive.

Lemme 8.21 $\forall\ \sigma,\ \mu(\texttt{DMISLV (enum V)}\ \sigma)\ \mathbb{I} = 1.$

Conclusion

L'étude, dans un premier temps, d'algorithmes optimaux (bornes supérieures et bornes inférieures) nous a montré le besoin de certifier des analyses délicates. Pour cela nous avons modélisé formellement le modèle en vue de faire des preuves formelles d'algorithmes simples pour commencer.

Bornes Supérieures

Nous avons mis au point des algorithmes distribués probabilistes qui résolvent le problème du MIS et du couplage maximal dans les anneaux anonymes de taille n en $O(\sqrt{log\ n})$ rondes. Cette complexité en bits est optimale. La méthode utilisée suit le même schéma en trois étapes qui se répètent jusqu'à la fin de l'algorithme. Elle consiste à créer un motif dans l'anneau à plusieurs endroits. Certains des sommets de ce motif prennent alors un état local final qui leur permet de répondre localement au problème, ce qui définit un sous-graphe où les sommets ont terminé. La deuxième étape consiste à étendre ce sous-graphe. La troisième étape consiste à fusionner deux sous-graphes qui se trouvent à faible distance. Ces trois étapes sont répétées jusqu'à ce qu'il n'y ait plus que des sommets ayant atteints leur état final.

Cette étude est faite sur les anneaux. Peut-on l'étendre à d'autres classes de graphe ?

Bornes Inférieures

Nous avons mis au point des constructions générales, basées sur les revêtements, qui présentent d'une façon unifiée des preuves sur les bornes inférieures pour la complexité en temps et la complexité en bits pour quelques problèmes de graphes. De ces constructions et résultats, nous déduisons que :

- résoudre les problèmes comme ceux du MIS, de la coloration, du couplage maximal, du 2-MIS ou de la coloration à distance 2 prend $\Omega(\log n)$ rondes avec forte probabilité pour une famille infinie de graphes non connexes ;
- résoudre les problèmes du MIS ou du couplage maximal prennent $\Omega(\sqrt{\log n})$ rondes avec forte probabilité pour une famille infinie d'anneaux ;
- résoudre les problèmes comme le MIS, la coloration, le couplage maximal, le 2-MIS ou la coloration à distance 2 prend $\Omega(\log n)$ rondes avec forte probabilité pour une famille infinie de graphes connexes.

En étendant la construction de Reidemeister aux graphes orientés, le résultat de [43] se déduit clairement : résoudre le problème de la coloration

des sommets prend $\Omega(\sqrt{\log n})$ rondes avec forte probabilité pour une famille infinie d'anneaux orientés. Peut-on faire de même pour d'autres problèmes ou d'autres classes de graphes ?

Preuve Formelle

Nous avons développé une syntaxe qui permet de définir les algorithmes distribués probabilistes de notre modèle par la déclaration des règles locales. Nous avons mis au point des outils qui permettent de simuler, prouver la correction et analyser ces algorithmes.

Les méthodes et résultats généraux que nous avons obtenus pour l'analyse sont les suivants :

- l'ordre des exécutions de l'algorithme local par les sommets n'interfère pas dans le résultat final ;
- chaque étape de calcul est la composition du calcul d'un sommet avec les calculs des autres sommets ;
- comment obtenir des résultats globaux (du point de vue du graphe) à partir d'un résultat local (du point de vue d'un sommet) ;
- comment prouver que la probabilité d'un évènement est non nulle ;
- comment prouver que les algorithmes distribués probabilistes de notre modèle terminent avec probabilité 1.

Ces résultats nous ont permis de mener l'analyse d'algorithmes résolvant les problèmes de la brisure de symétrie, du rendez-vous et du couplage maximal. Nous avons montré comment les utiliser pour prouver la terminaison d'un algorithme pour le MIS que nous avons énoncé.

L'étape suivante consiste en la semi-automatisation des preuves par le biais de tactiques. Par exemple, si l'utilisateur souhaite faire l'analyse d'une propriété portant sur un sommet ou sur des parties disjointes du système distribué, une tactique appropriée pourrait lui permettre de prouver cette propriété en générant des sous-preuves liées à l'indépendance et à la modification locale du graphe. Ou encore pour prouver une propriété qui repose sur un invariant, une tactique permettrait de mettre en place le contexte afin de montrer que la propriété implique l'invariant.

Un lemme majeur est la preuve de terminaison avec probabilité 1 de nos algorithmes sous certaines hypothèses. Nous aimerions avoir d'autres propriétés de ce genre. La première partie fait l'analyse en complexité. Afin de la relier plus à la deuxième partie, nous souhaitons développer des outils pour étudier la complexité de nos algorithmes. Pour cela, en plus des algorithmes locaux et du graphe de départ étiqueté, il nous faut prendre en paramètre un entier naturel destiné à compter le nombre de rondes. Ainsi notre algorithme retournerait le graphe étiqueté de sortie (ou sa distribution) et le nombre de rondes pour qu'il termine (ou la distribution de ce nombre).

Bertot [12] décrit diverses sémantiques d'un langage simple de programmation. Parmi elles, nous retrouvons la sémantique d'axiomatisation qui prend en compte les propriétés qui sont satisfaites par les variables du programme avant et après exécution ; il y a aussi la sémantique dénotationnelle qui prend en compte les fonctions récursives. De ces sémantiques, on obtient un générateur d'obligations de preuves (preuves de

correction, de plus faible pré-condition, de plus petit point fixe pour la terminaison). On retrouve cette même volonté d'établir des obligations de preuves dans des logiciels comme *Why3* [22] ou *Rodin* [71]. Il serait alors intéressant d'enrichir notre syntaxe actuelle en ajoutant aux règles de calculs locaux, les conditions de terminaison ainsi que des invariants (déterministes et/ou probabilistes) ce qui permettrait de générer automatiquement les obligations de preuves de correction, de plus faible pré-condition, d'analyse probabiliste et de terminaison.

Bibliographie

[1] http ://www.haskell.org/haskellwiki/monad.

[2] Jean-Raymond Abrial. *Modeling in Event-B : System and Software Engineering*. Cambridge University Press, 2010.

[3] Noga Alon, László Babai, et Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set. *Journal of Algorithms*, 7(4) :567–583, 1986.

[4] Dana Angluin. Local and global properties in networks of processors (extended abstract). Dans *STOC*, pages 82–93, 1980.

[5] Hagit Attiya et Jennifer Welch. *Distributed Computing*. Wiley, 2004.

[6] Philippe Audebaud et Christine Paulin-Mohring. Proofs of randomized algorithms in coq. *Sci. Comput. Program.*, 74(8) :568–589, 2009.

[7] Baruch Awerbuch, Andrew V. Goldberg, Michael Luby, et Serge A. Plotkin. Network decomposition and locality in distributed computation. Dans *Proceedings of the 30th ACM Symposium on FOCS*, pages 364–369. ACM Press, 1989.

[8] Amotz Bar-Noy, Joseph Naor, et Moni Naor. One-bit algorithms. *Distributed Computing*, 4 :3–8, 1990.

[9] Gilles Barthe, Juan Manuel Crespo, Benjamin Grégoire, César Kunz, et Santiago Zanella Béguelin. Computer-aided cryptographic proofs. Dans *ITP*, pages 11–27, 2012.

[10] Michel Bauderon et Mohamed Mosbah. A unified framework for designing, implementing and visualizing distributed algorithms. *Graph Transformation and Visual Modeling Techniques (First International Conference on Graph Transformation)*, 72(3) :13–24, 2003.

[11] Claude Berge. *Graphes*. Gauthiers-villars, Paris, 3e édition, 1983.

[12] Yves Bertot. Theorem proving support in programming language semantics. *CoRR*, abs/0707.0926, 2007.

[13] Yves Bertot et Pierre Castéran. *Interactive Theorem Proving and Program Development. Coq'Art : The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer Verlag, 2004. URL `http://www.labri.fr/publications/l3a/2004/BC04`.

[14] H. L. Bodlaender, S. Moran, et M. K. Warmuth. The distributed bit complexity of the ring : from the anonymous case to the non-anonymous case. *Inf. and comput.*, 114(2) :34–50, 1994.

[15] Hans L. Bodlaender. The classification of coverings of processor networks. *J. Parallel Distrib. Comput.*, 6(1) :166–182, 1989.

[16] Paolo Boldi et Sebastiano Vigna. Fibrations of graphs. *Discrete Mathematics*, 243(1-3) :21–66, 2002.

[17] Dominique Cansell et Dominique Méry. Dans *Logics of Specification Languages*, Chapitre The Event-B Modelling Method : Concepts and Case Studies, pages 47–152. Springer Verlag, 2007.

[18] Pierre Castéran et Vincent Filou. Tasks, types and tactics for local computation systems. *Stud. Inform. Univ.*, 9(1) :39–86, 2011.

[19] Pierre Castéran, Vincent Filou, et Allyx Fontaine. Loco : A Coq Library on Local Computation Systems. http ://www.labri.fr/∼casteran/Loco.

[20] Jérémie Chalopin et Yves Métivier. An efficient message passing election algorithm based on mazurkiewicz's algorithm. *Fundam. Inform.*, 80(1-3) :221–246, 2007.

[21] Chin-Tsun Chou. Mechanical verification of distributed algorithms in higher-order logic. *The Computer Journal*, 38(2) :152–161, 1995.

[22] Équipe de projett Toccata Ã INRIA Saclay-Île-de France / LRI Univ Paris-Sud 11 / CNRS. Why3. http ://why3.lri.fr/.

[23] Yuxin Deng et Jean-François Monin. Verifying self-stabilizing population protocols with coq. Dans *TASE*, pages 201–208, 2009.

[24] Yefim Dinitz, Shlomo Moran, et Sergio Rajsbaum. Bit complexity of breaking and achieving symmetry in chains and rings. *Journal of the ACM*, 55(1), 2008.

[25] Jean Duprat. A Coq toolkit for graph theory. Rapport de recherche 2001-15, LIP ENS Lyon, 2001.

[26] Jean-Christophe Filliâtre et Pierre Letouzey. Functors for proofs and programs. Dans *ESOP*, pages 370–384, 2004.

[27] Vincent Filou. *Une étude formelle de la théorie des calculs locaux à l'aide de l'assistant de preuve Coq*. PhD thesis, 2012. URL `http://www.theses.fr/2012BOR14708`. Thèse de doctorat dirigée par Castéran, Pierre et Mosbah, Mohamed Informatique Bordeaux 1 2012.

[28] Allyx Fontaine, Yves Métivier, John Michael Robson, et Akka Zemmari. Optimal Bit Complexity Randomised Distributed MIS and Maximal Matching Algorithms for Anonymous Rings. *Information and Computation*, xx(xx) :xx, Octobre 2013. URL `http://hal.archives-ouvertes.fr/hal-00871822`.

[29] Allyx Fontaine, Yves Métivier, John Michael Robson, et Akka Zemmari. On Lower Bounds for the Time and the Bit Complexity of

some Probabilistic Distributed Graph Algorithms. Dans V. Geffert, B. Preneel, B. Rovan, J. Stuller, et A M. Tjoa, éditeurs, *40th International Conference on Current Trends in Theory and Practice of Computer Science*, volume 8327 de *Lecture Notes in Computer Science*, pages 235–245, Slovaquie, Janvier 2014. Springer. URL `http://hal.archives-ouvertes.fr/hal-00873468`.

[30] Allyx Fontaine et Akka Zemmari. Rda : A Coq Library on Randomised Distributed Algorithms. http ://www.labri.fr/perso/fontaine/RDA.

[31] Allyx Fontaine et Akka Zemmari. Une Analyse Formelle en Coq d'un Algorithme Distribué Probabiliste résolvant le Problème du Rendez-Vous. Dans Damien Pous et Christine Tasson, éditeurs, *JFLA - Journées francophones des langages applicatifs*, Aussois, France, Février 2013. Damien Pous and Christine Tasson. URL `http://hal.inria.fr/hal-00779700`.

[32] Michael R. Garey et David S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979. ISBN 0716710447.

[33] Georges Gonthier. The four colour theorem : Engineering of a formal proof. Dans *ASCM*, page 333, 2007.

[34] Georges Gonthier, Assia Mahboubi, et Enrico Tassi. A Small Scale Reflection Extension for the Coq system. Rapport de recherche RR-6455, INRIA, 2008. URL `http://hal.inria.fr/inria-00258384`.

[35] Michal Hanckowiak, Michal Karonski, et Alessandro Panconesi. On the distributed complexity of computing maximal matchings. *SIAM J. Discrete Math.*, 15(1) :41–57, 2001.

[36] HOL. http ://hol.sourceforge.net/.

[37] Joe Hurd. *Formal Verification of Probabilistic Algorithms*. PhD thesis, University of Cambridge, 2002. URL `http://www.gilith.com/research/papers`.

[38] Joe Hurd, Annabelle McIver, et Carroll Morgan. Probabilistic guarded commands mechanized in *hol*. *Electr. Notes Theor. Comput. Sci.*, 112 : 95–111, 2005.

[39] Isabelle. https ://www.cl.cam.ac.uk/research/hvg/Isabelle/.

[40] Amos Israeli et Alon Itai. A fast and simple randomized parallel algorithm for maximal matching. *Inf. Process. Lett.*, 22(2) :77–80, 1986.

[41] Richard M. Karp et Avi Widgerson. A fast parallel algorithm for the maximal independent set problem. Dans *Proceedings of the 16th ACM Symposium on Theory of computing (STOC)*, pages 266–272. ACM Press, 1984.

[42] Donald E. Knuth. *The art of computer programming, volume 2 (3rd ed.) : seminumerical algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997. ISBN 0-201-89684-2.

[43] Kishore Kothapalli, Melih Onus, Christian Scheideler, et Christian Schindelhauer. Distributed coloring in $O(\sqrt{\log n})$ bit rounds. Dans *20th International Parallel and Distributed Processing Symposium (IPDPS 2006), Proceedings, 25-29 April 2006, Rhodes Island, Greece*. IEEE, 2006.

[44] Dexter Kozen. Semantics of probabilistic programs. *J. Comput. Syst. Sci.*, 22 :328–350", 1981.

[45] Philipp Küfner, Uwe Nestmann, et Christina Rickmann. Formal verification of distributed algorithms : From pseudo code to checked proofs. Dans *Proceedings of the 7th IFIP TC 1/WG 202 International Conference on Theoretical Computer Science*, TCS'12, pages 209–224, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-33474-0. URL http://dx.doi.org/10.1007/978-3-642-33475-7_15.

[46] Fabian Kuhn, Thomas Moscibroda, Tim Nieberg, et Roger Wattenhofer. Fast deterministic distributed maximal independent set computation on growth-bounded graphs. Dans *DISC*, pages 273–287, 2005.

[47] Eyal Kushilevitz et Noam Nisan. *Communication complexity*. Cambridge University Press, 1999.

[48] Marta Z. Kwiatkowska, Gethin Norman, et David Parker. Prism : Probabilistic symbolic model checker. Dans *Computer Performance Evaluation / TOOLS*, pages 200–204, 2002.

[49] Marta Z. Kwiatkowska, Gethin Norman, et Roberto Segala. Automated verification of a randomized distributed consensus protocol using cadence smv and prism. Dans *CAV*, pages 194–206, 2001.

[50] Christian Lavault. *Évaluation des algorithmes distribués*. Hermes, 1995.

[51] Stéphane Lescuyer. First-class containers in coq. *Stud. Inform. Univ.*, 9(1) :87–127, 2011.

[52] Nathan Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21 :193–201, 1992.

[53] Igor Litovsky, Yves Métivier, et Eric Sopena. Graph relabelling systems and distributed algorithms. Dans H. Ehrig, H.J. Kreowski, U. Montanari, et G. Rozenberg, éditeurs, *Handbook of graph grammars and computing by graph transformation*, volume 3, pages 1–56. World Scientific, 1999.

[54] Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15 :1036–1053, 1986.

[55] Nancy A. Lynch. A hundred impossibility proofs for distributed computing. Dans *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing*, PODC '89, pages 1–28, New York, NY, USA, 1989. ACM. ISBN 0-89791-326-4. URL http://doi.acm.org/10.1145/72981.72982.

[56] Nancy A. Lynch. *Distributed algorithms*. Morgan Kaufman, 1996.

[57] The Coq development team. *The Coq proof assistant reference manual*. LogiCal Project, 2004. URL `http://coq.inria.fr`. Version 8.0.

[58] K.L. McMillan. A methodology for hardware verification using compositional model checking. *Science of Computer Programming*, 37 :279 – 309, 2000. ISSN 0167-6423. URL `http://www.sciencedirect.com/science/article/pii/S0167642399000301`.

[59] Dominique Méry, Mohamed Mosbah, et Mohamed Tounsi. Refinement-based verification of local synchronization algorithms. Dans *FM*, pages 338–352, 2011.

[60] Yves Métivier, John Michael Robson, Nasser Saheb-Djahromi, et Akka Zemmari. About randomised distributed graph colouring and graph partition algorithms. *Inf. Comput.*, 208(11) :1296–1304, 2010.

[61] Yves Métivier, John Michael Robson, Nasser Saheb-Djahromi, et Akka Zemmari. An optimal bit complexity randomized distributed mis algorithm. *Distributed Computing*, 23(5-6) :331–340, 2011.

[62] Yves Métivier, Nasser Saheb, et Akka Zemmari. Analysis of a randomized rendezvous algorithm. *Inf. Comput.*, 184(1) :109–128, 2003.

[63] Yves Métivier et Eric Sopena. Graph relabelling systems : A general overview. *Computers and Artificial Intelligence*, 16(2), 1997.

[64] Thomas Moscibroda et Roger Wattenhofer. Maximal independent set in radio networks. Dans *Proceedings of the 25 Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 148–157. ACM Press, 2005.

[65] Rajeev Motwani et Prabhakar Raghavan. Algorithms and theory of computation handbook. Chapitre Randomized Algorithms, pages 12–12. Chapman & Hall/CRC, 2010. ISBN 978-1-58488-822-2. URL `http://dl.acm.org/citation.cfm?id=1882757.1882769`.

[66] Gethin Norman. Analysing randomized distributed algorithms. Dans *Validation of Stochastic Systems*, pages 384–418, 2004.

[67] David Peleg. *Distributed computing - A Locality-sensitive approach*. SIAM Monographs on discrete mathematics and applications, 2000.

[68] PVS. http ://pvs.csl.sri.com/.

[69] Kurt Reidemeister. *Einführung in die Kombinatorische Topologie*. Brunswick. F. Vieweg, 1932.

[70] Benoit Robillard. Formalization of a generic graph library for Coq : Application to the formal verification of Dijkstra's shortest path algorithm. Rapport de recherche 2012, 2012.

[71] Project RODIN. Rigorous open development environment for complex systems. http ://rodin-b-sharp.sourceforge.net/, 2004. 2004–2007.

[72] Kenneth H. Rosen. *Handbook of Discrete and Combinatorial Mathematics, Second Edition*. Chapman & Hall/CRC, 2nd édition, 2010. ISBN 158488780X, 9781584887805.

[73] Roberto Segala. Verification of randomized distributed algorithms. Dans *European Educational Forum : School on Formal Methods and Performance Analysis*, pages 232–260, 2000.

[74] Matthieu Sozeau et Nicolas Oury. First-class type classes. Dans *Proceedings of the 21st International Conference on Theorem Proving in Higher Order Logics*, TPHOLs '08, pages 278–293, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-71065-3. URL `http://dx.doi.org/10.1007/978-3-540-71067-7_23`.

[75] Gerard Tel. *Introduction to distributed algorithms*. Cambridge University Press, 2000.

[76] Mohamed Tounsi. *Preuves d'algorithmes distribués par raffinement*. PhD thesis, 2012. URL `http://www.theses.fr/2012BOR14545`. Thèse de doctorat dirigée par Mosbah, Mohamed Informatique Bordeaux 1 2012.

[77] Andrew Chi-Chih Yao. Some complexity questions related to distributed computing. Dans *Proceedings of the 11th ACM Symposium on Theory of computing (STOC)*, pages 209–213. ACM Press, 1979.

[78] Akka Zemmari. *Présentation et Analyse de Quelques Algorithmes distribués Probabilistes*. Habilitation à diriger des recherches, spécialité informatique, Université Bordeaux 1, Octobre 2009. 103 pages.