Conception et Analyse de quelques Algorithmes Distribués Probabilistes

El Mehdi Stouti

Numéro d'ordre : 20 année 2016

UNIVERSITÉ ABDELMALEK ESSAADI
FACULTÉ DES SCIENCES
TETOUAN

Centre d'Etudes Doctorales
Sciences et Technologies

Formation Doctorale : Mathématiques, Physique, et Nouvelles Technologies

THÈSE

pour obtenir le titre de

Docteur en Sciences

de l'Université Abdelmalek Essaadi
Spécialité : Informatique

Présentpée et soutenue par

El Mehdi STOUTI

Conception et analyse de quelques algorithmes distribués probabilistes

soutenue le 15 juillet 2016

Jury :

Prsident :	Abdelhamid BENKADDOUR	-	FS - Tétouan
Rapporteurs :	Mohamed EL KAMILI	-	FS Dhar Mahraz - Fès
	Ali EL MERZOUQI	-	FS - Tétouan
	Mohamed EL MEROUANI	-	FP - Tétouan
Examinateur :	Youness TABII	-	ENSA - Tétouan
Co-Directeur :	El Hibaoui ABDELAAZIZ	-	FS - Tétouan
Directeur :	Kamal Eddine EL KADIRI	-	FS - Tétouan

À ma très chère et douce mère et mon très cher père
À tous ceux qui me sont chers...

Résumé

Un système distribué est un environnement où plusieurs processus collaborent pour réaliser un objectif commun. Dans un réseau, les différents processus ne peuvent communiquer directement qu'avec un nombre limité d'autres processus, leurs « voisins ». L'algorithmique distribuée a pour but de décrire quelles sont les tâches qui peuvent être réalisées dans de tels systèmes. Autrement dit, elle cherche à déterminer quels sont les comportements globaux qui peuvent être obtenus dans de tels systèmes où les comportements des différents processus ont des effets locaux. Un élément du réseau est un « nœud » et ce qui permet de distinguer un nœud d'un autre peut être un identifiant (comme une adresse IP sur Internet), mais plus généralement, c'est la position de chaque nœud dans le réseau qui permet de le distinguer. Ce qui caractérise les systèmes distribués est qu'il n'existe pas a priori de système de centralisation qui peut coordonner globalement les différents processus.

Les travaux de cette thèse s'intègrent dans ce contexte et présentent l'évolution et l'analyse des performances des algorithmes probabilistes entièrement distribués et décentralisés et ce dans des systèmes distribués anonymes, asynchrones et de topologies différentes.

Dans un premier temps nous proposons et étudions un algorithme d'élection probabiliste uniforme dans des structures de type « graphes à grilles triangulaires ». Cet algorithme est basé sur l'utilisation des délais aléatoires associés aux sommets supprimables (les sommets simpliciaux). Ces délais sont indépendants et peuvent être générés localement par des sommets au fur et à mesure qu'ils deviennent supprimables. Pour la classe de graphes citée ci-dessus, notre résultat principal est l'introduction d'un algorithme d'élection totalement équitable, c'est-à-dire que quelque soit l'emplacement d'un sommet dans un graphe, il a la même probabilité d'être « élu » que tous les autres sommets.

Dans un second temps, nous introduisons et analysons deux algorithmes distribués probabilistes de construction d'arbre couvrant minimal. Ces algorithmes sont basés essentiellement sur un algorithme de rendez-vous. Tout d'abord, chaque arête sur laquelle un rendez-vous à eu lieu est considérée comme un sous-arbre couvrant. A chaque tour de l'exécution de l'algorithme, les sous-arbres couvrants sont fusionnés. La construction de l'arbre couvrant minimal s'arrête lorsque tous les sous-arbres couvrants sont fusionnés en un seul. Nous montrons par la suite que le graphe construit est un arbre couvrant minimal.

Finalement, nous présentons une plate-forme de simulation des algorithmes distribués permettant d'implémenter, tester et vérifier les algorithmes distribués codés sous forme de calculs locaux et spécialement ceux décrits sous forme de règles de réécriture.

Mots clefs : *Algorithmes distribués probabilistes, système distribué, analyse d'algorithmes, élection uniforme, arbre couvrant minimal, simulation*

Abstract

A distributed system is an environment where multiple processes can work together to achieve a common goal. Processes can only directly communicate with their neighbours. Distributed algorithms aims to describe the tasks that can be executed in such systems. In other words, it seeks to determine what are the global behaviors that can be obtained in such systems, where processes behaviors have local effects. Network elements are called "nodes" and they are distinguished from each other by identifiers (such as an IP address on the Internet), but more generally, it is the position of each node in the network that distinguishes it. What characterizes distributed systems is that there is no centralized system that can comprehensively coordinate the different processes.

The work of this thesis fit into this context and presents the evolution and performance of the algorithms analysis. These probabilistic algorithms are fully distributed and decentralized, executed in distributed systems that are anonymous, asynchronous and with different topologies.

Firstly, we propose and study a uniform probabilistic election algorithm applied on graphs with triangular grids. This algorithm is based on using random delays associated with deletable nodes (simplicial nodes). These delays are independent and can be generated locally by vertices as they become deletable. For the class of graphs cited above, our main result is the introduction of an election algorithm that is totally fair, i.e. regardless of the vertex location in the graph, it will have the same probability of being elected as the other vertices.

Secondly, we introduce and analyze two probabilistic distributed algorithms for a minimum spanning tree construction. These algorithms are essentially based on the Handshake algorithm. First, each edges on which a meeting took place (or isolated vertex) is considered a sub-spanning tree. Each execution of the algorithm, all sub-spanning trees are merged. The execution continues until all sub-spanning trees are merged into one. Thereafter, we prove that the constructed graph is a minimum spanning tree.

Finally, we present a simulation platform for distributed algorithms which allows us to implement, test, and verify distributed algorithms programmed in the form of local computations, especially those described in the rewrite rules.

Keywords : *Distributed probabilistic algorithms, distributed system, algorithm analysis, uniform election, minimum spanning tree, simulation*

Remerciements

A l'issu de la rédaction de cette recherche, je suis convaincu que la thèse est loin d'être un travail solitaire. En effet, je n'aurais jamais pu réaliser ce travail doctoral sans le soutien d'un grand nombre de personnes dont la générosité, la bonne humeur et l'intérêt manifestés à l'égard de ma recherche m'ont permis de progresser dans cette phase délicate de « *l'apprenti-chercheur* ».

En premier lieu, je tiens à remercier mes encadrants M. Kamal Eddine El Kadiri et M. Abdelaaziz El Hibaoui, qui ont dirigé mes travaux au long de mes années de thèse. Je les remercie pour leur grande disponibilité ainsi que les commentaires et suggestions pertinentes dont il m'ont fait bénéficier.

Mes remerciements vont aussi à M. Ali Elmerzouqi pour avoir accepté de participer au jury de ma thèse et pour l'ambiance de travail très agréable qu'il a su créer au département informatique grâce à sa très grande ouverture d'esprit.

Mon profond respect et ma gratitude se dirigent aussi vers M. Mohamed El Kamili et M. Mohamed El Merouani d'avoir accepté de rapporter avec rigueur mes travaux de thèse. Je remercie également M. Abdelhamid Benkaddour et M. Youness Tabii d'avoir accepté de m'honorer en faisant parti du jury.

Enfin, je voudrais rendre hommage à toutes les personnes qui n'ont pas hésité à m'aider d'une manière ou d'une autre durant toute la période de ma thèse.

Table des matières

Table des figures

List of Algorithmes

Introduction générale

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

Dans la suite de cette introduction, nous présenterons d'abord le contexte général de nos travaux de recherche et les problématiques à traiter, ainsi que les objectifs escomptés de ces travaux et les contributions apportées dans le cadre de cette thèse, avant de terminer par présenter l'organisation du présent document.

Contexte et problématiques

Les progrès technologiques des ordinateurs et le haut débit permettent à différentes applications sur des ordinateurs distincts (et distants) de coopérer pour effectuer des tâches coordonnées. La possibilité de connecter un ensemble de machines en réseau local introduit la notion de partage de ressources : les utilisateurs découvrent alors les avantages de partager une imprimante, des disques, etc.
Cette notion entraîne alors d'autres besoins. Les systèmes distribués recouvrent actuellement un champ très vaste. Citons, par exemple, les clusters de machines, les grilles de calculs parallèles ou distribués, les systèmes à objets, le déploiement d'applications Web, le stockage d'informations dans des réseaux de pairs etc.

La question qui surgit est : *"comment concevoir des algorithmes afin de réussir à faire collaborer toutes ces entités pour accomplir une tâche globale ?"*
De cette question nous pouvons dériver les sous-questions suivantes :

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

- *Comment gérer les ressources de manière globale sans pour autant utiliser des algorithmes centralisés qui posent problème en cas de tolérance aux pannes ?*
- *Comment gérer la cohérence et la synchronisation entre les machines du réseau ?* Les informations doivent rester cohérentes sur tout le système, ce qui suppose, en cas de modification d'un état local, de diffuser la modification à toutes les autres entités. Ce problème se pose notamment pour la gestion du temps global.

Un algorithme distribué $\mathcal{A}$ est un algorithme qui s'exécute sur plus d'une machine ou processeur. En d'autres termes, un algorithme distribué $\mathcal{A}$ sur un système distribué $\mathcal{S}$ n'est autre que la caractérisation des transitions locales à réaliser séquentiellement par chaque processus de $\mathcal{S}$ à la réception d'un message [Zem09]. Le choix du meilleur algorithme pour une tâche particulière peut être un processus compliqué, qui exige souvent des analyses mathématiques sophistiquées. La partie de l'informatique qui comprend l'étude de telles questions est appelée analyse des algorithmes. Une étude théorique aide à bien comprendre la structure du problème pour en dégager les propriétés qui seront la base de l'algorithme. Ce n'est pas toujours suffisant pour obtenir une bonne efficacité (complexité théorique). Une étude algorithmique peut s'avérer nécessaire. L'existence et l'expression de ces algorithmes dépendent des hypothèses techniques sur les réseaux comme par exemple le mode de communication (par message ou par registre), la synchronisation partielle de processus voisins, l'anonymat des processeurs. Ce cadre rend difficile la lisibilité de certains algorithmes, il ne permet pas toujours de comprendre leurs puissances et leurs limites et de faire des preuves mathématiques de leurs propriétés.

Concernant un panorama plus général sur l'algorithmique distribuée, nous pourrons nous reporter aux références : [Lyn96, Tel91, Tel94, Tel00] et en particulier à [Lav95b] pour une revue assez détaillée des différentes hypothèses et différents modèles que nous pouvons utiliser pour décrire un algorithme distribué.

Dans cette thèse, la notion de système distribué englobe tous les systèmes composés de plusieurs entités autonomes (processus, ordinateurs, etc.) communiquant chacune avec ses voisins au moyen de canaux de communication ou de variables partagées. Chaque entité évolue alors en fonction de son état propre et de celui de ses voisins.

Objectifs et Contributions

Nous abordons dans ce document certains aspects des thèmes ci-dessous en liaison avec les algorithmes distribués :

- **Algorithmes probabilistes d'élection uniforme**

Un des paradigmes de l'informatique distribuée est l'élection où un processus unique du systèmes distribué $\mathcal{S}$ est privilégié grâce à un choix distribué de tous les processus coopérants de $\mathcal{S}$. L'élection dans un réseau consiste à atteindre une configuration dans laquelle un seul et unique processus est dans un état prédéterminé, $\boxed{élu}$, et tous les autres dans un état prédéterminé différent du précédent, $\boxed{battu}$. Le problème de l'élection est à la base des principaux mécanismes de contrôle utilisés dans les systèmes distribués.
Ainsi, nous pouvons se poser, par exemple, les questions suivantes : *"Comment choisir une machine pour gérer les conflits nécessairement rencontrés lorsque des utilisateurs partagent une ressource (l'exclusion mutuelle)" ?*
Un autre problème qui peut se poser est lorsque cette machine tombe en panne : *"Comment désigner sans l'intervention des utilisateurs une machine qui prendra le relais de la machine défectueuse ? Plus précisément, comment rendre les machines autonomes ?"*
Il devient par exemple difficile de choisir une machine s'elles ont les mêmes propriétés et qu'elles sont indiscernables.
De nombreux algorithmes pour élire un sommet dans un graphe sont connus depuis les années soixante-dix [Lan77].

Il est, par ailleurs, connu que dans certaines topologies de réseau, l'élection distribuée n'est plus faisable [Ang80]. Cependant, seuls les algorithmes probabilistes permettent de contourner cette difficulté en donnant une solution partielle très acceptable d'un point de vue pratique [DFJ+98, MR95, GSB94]. En effet, Il existe plusieurs situations où l'utilisation d'algorithmes probabilistes peut aider à trouver une solution. Par exemple dans les problèmes suivants : l'élection d'un chef dans un réseau fortement symétrique [IR90, BSV+96, YK99], l'exclusion mutuelle [Dij74, Rab82], ou l'orientation d'anneau [IJ93]. Pour ces algorithmes, il est connu qu'aucun algorithme déterministe ne peut fournir une solution, alors qu'il y a des algorithmes probabilistes simples et élégants.

Le concept d'équité ou de l'équitabilité (en anglais fairness) est une propriété importante, voir [DIM95]. Nous pouvons par exemple utiliser cette propriété pour garantir que toutes les machines, qui collaborent pour faire un calcul complexe, ont la même quantité de données à traiter.

Plusieurs travaux ont été menés sur le problème d'élection qui incluent le concept d'équité. Dans [MSDZ05], les auteurs ont proposé un algorithme complètement distribué permettant de réaliser une élection équitable dans les arbres : si l'arbre est de taille $n > 0$, alors tous ses sommets ont la même probabilité $\frac{1}{n}$ d'être élus. Par la suite les auteurs ont cherché s'il y a la possibilité d'adapter ledit algorithme

à d'autres classes de graphes, ce qui a aboutit à un algorithme donnant la même chance d'être élu à tous les sommets d'un k-arbre Dans [HSdZ+04], après ils ont étendu l'algorithme au cas des polyominoïdes dans [HSZ05] et [HRSZ10]. En suite ils l'ont adapté pour que la probabilité, pour un sommet de l'arbre, d'être élu soit guidée par une loi de probabilité donnée. Ce qui a fait l'objet au résultats présentés dans [MSDZ09].

Un des objectifs de notre recherche est de proposer et analyser un algorithme d'élection uniforme sur des familles de graphes non encore étudiées. Cependant, nous employons la propriété d'équité dans son sens strict : tous les processeurs ont la même probabilité d'être élus, de sorte que lorsque l'algorithme est réitéré plusieurs fois, les ressources soient allouées *équitablement*. Ainsi et dans un premier temps, nous exposerons les règles qui permettent de générer et de manière distribuée la famille de graphes à grilles triangulaires. Nous présenterons ensuite les différentes règles du processus d'éliminations de notre algorithme. Ces règles sont exécutées en parallèle par tout nœud simplicial du réseau. Dans un second temps, nous analysons analytiquement ce processus. Pour ce faire nous le modélisons par un processus markovien de mort en temps continu. Ce travail a fait l'objet de la publication [SHH13b] et a été présenté comme communication dans [HSH13][SHKH13][HSKD13][SHK12].

- **Construction d'arbre couvrant minimal**

  Le problème d'arbre couvrant minimal est un problème d'optimisation combinatoire très connu, il consiste à relier tous les sommets d'un graphe connexe non orienté, sans créer aucun cycle. Les différentes méthodes pour chercher l'arbre couvrant minimal ont joué un rôle central dans l'analyse et la conception des algorithmes informatiques. Il existe de nombreux algorithmes qui permettent de calculer un arbre couvrant de poids minimal. La différence est que chacun est conçu pour s'exécuter dans un environnement particulier (centralisé / distribué, synchrone / asynchrone, orienté / non orienté, valué / non valué, etc.). Nous citerons par exemple, l'algorithme de Borůvka [BVP11], de Prim [Mar02] et de Kruskal [NL12].

  Vue l'importance que représente l'arbre couvrant minimal dans les réseaux (routage optimal, etc.), nous nous sommes intéressés à étudier ce problème. Notre objectif est de concevoir et analyser deux algorithmes distribués probabilistes pour la construction d'un arbre couvrant minimal. Ces algorithmes sont basés sur l'algorithme de rendez-vous. Ce dernier permet de produire k rendez-vous qui sont considérés dans notre cas comme des sous-arbres, où k est la taille du couplage maximal produit [HMR+06]. L'étape suivante et qui constitue notre apport consiste

à fusionner de manière distribuée ces sous-arbres sans créer de cycle pour obtenir un seul arbre, tout en intégrant les sommets isolés (ceux qui n'ont pas été assignés par des rendez-vous). Nous prouvons par la suite que le graphe résiduel est acyclique, contient tous les sommets du graphe initial et utilise les arêtes de poids minimal, c'est donc l'arbre couvrant minimal recherché. Pour enrichir notre étude, nous présentons une étude détaillée sur le nombre de messages échangés. Cette étude montre que l'algorithme que nous proposons présente une meilleure performance que ceux qui nous sommes connus en terme d'échange de messages. Nous caractérisons le nombre prévu de phases nécessaires pour réduire le nombre de forme de l'arbre couvrant de n à 1. Nous prouvons que ce nombre attendu est délimitée entre $O(\log_2(n))$ et $O(n)$. Ce travail a été couronné par la publication des articles [HSHD14][SBK15] et a été présenté comme communication dans [SHH13a][SHHK14][SBK14].

- **Simulation des algorithmes distribués probabilistes**

  Selon Pascal Cantot [Can01], le terme *simulation* est utilisé pour qualifier un ensemble de techniques et de principes permettant l'étude de systèmes réels à partir d'un modèle aussi fidèle que possible, qui reflète au mieux les aspects du système réel que nous voulons étudier, compte tenu du besoin et des contraintes de moyens et de délais de développement.

  La simulation s'est appliquée dans de nombreux domaines (militaire, aéronautique, spatial, etc.) dans le but d'atteindre des objectifs divers tels que le prototypage, l'évaluation de performance, l'aide à la prise de décision ou encore l'entraînement. Les techniques de simulation proposent des avantages indéniables sur de vraies expériences pour des systèmes réels. Les expériences sur le système réel peuvent être coûteuses, dangereuses voire impossibles. Par exemple, le système réel est inexistant ou encore l'échelle de temps de la dynamique de son évolution est incompatible avec l'expérimentateur humain (soit beaucoup trop grande ou beaucoup trop petite). De plus, la manipulation facile des modèles (modification de paramètres) et la suppression des effets indésirables (perturbations sur le modèle réel) permettent d'avoir une vision plus complète du système étudié.

  Notre objectif est de développer une plate-forme de simulation d'algorithmes distribués, spécialement ceux qui sont probabilistes. Une plate-forme où c'est possible de dessiner des graphes (réseaux), d'implémenter des algorithmes distribués, de les tester et de les vérifier, de visualiser leur exécution. Pour ce faire, nous commençons par traiter des notions relatives aux domaines de la simulation en introduisant les concepts de bases de la simulation. Nous évoquons ensuite, les problématiques de la simulation distribuée ainsi que les différents standards et architectures créés afin de

mettre en place une bonne plate-forme de simulation contourne les problématiques rencontrées par les plate-formes existantes. Ce travail a donné naissance aux communications nationales et internationales [BSNH15][SBN+15][SBF15], il est en cours de publication (soumis au journal : Computer and System Sciences [BSNH16]).

Organisation de la thèse

Cette thèse est structurée en deux grandes parties. La première partie est divisée en deux chapitres d'état de l'art, alors que la seconde est composée de trois chapitres de contributions. Le document s'achève par une conclusion générale.

- **Partie I : Survol de la théorie des graphes et de l'algorithmique distribuée**

  - ► *Chapitre 1 : « Préliminaires »*

    Afin de comprendre le reste du document, ce chapitre présente un fondement mathématique sur les notions de probabilité, de la théorie des graphes et le modèle d'algorithmique distribuée nécessaire pour la compréhension du reste du document. Il expose ensuite comment un algorithme distribué peut être modélisé au moyen de réétiquetage de graphes. Il présente finalement les modèles stochastiques qui sont utilisés pour décrire les algorithmes probabilistes étudiés.

  - ► *Chapitre 2 : « Généralités sur les algorithmes distribués »*

    Ce chapitre introduit quelques rappels sur les systèmes distribués, leur définition et les modèles de calcul fréquemment rencontrés dans la littérature. Puis, il décrit la motivation de l'utilisation d'un système distribué et expose les différents types de topologies. Il présente ensuite, les problèmes soulevés par de tels systèmes. Il détaille, avant de terminer, un panel non exhaustif des problèmes classiques de l'algorithmique distribuée. Enfin, il présente les algorithmes distribués probabilistes indiquant ce qui change par rapport au cas déterministe.

- **Partie II : Algorithmes distribués probabilistes d'élection et de calcul d'arbre couvrant**

  - ► *Chapitre 3 : « Élection uniforme dans les graphes à grilles triangulaires »*

    Dans ce chapitre, nous introduisons et analysons un algorithme distribué probabiliste d'élection uniforme dans les graphes à grilles triangulaires. Cet algorithme probabiliste, basé sur des calculs locaux, est fondé sur les éliminations

successives des sommets actifs du graphe d'entrée jusqu'à ce que ce graphe soit réduit à un seul sommet. Ce dernier sera considéré comme le sommet $\boxed{élu}$. L'étude du modèle uniforme met en jeu quelques outils élémentaires ou avancés de la théorie des probabilités tels que des techniques d'évaluation de la somme et du maximum pour des variables aléatoires indépendantes, et ce pour évaluer la durée d'élection. Cette durée étant considérée comme une mesure de complexité. En fin, nous étudions l'équité de notre algorithme en fournissant des outils nécessaires pour son analyse.

- *Chapitre 4 : « Construction distribuée d'un arbre couvrant minimal »*

  Dans ce chapitre nous présentons quelques notations et définitions nécessaires pour comprendre le reste de ce chapitre. Nous exposons aussi notre modèle et les hypothèses. La suite de ce chapitre sera consacrée à la description de deux algorithmes pour la construction d'arbre couvrant minimal et à analyser ces algorithmes. Pour le deuxième algorithme, nous montrons que l'arbre couvrant obtenu est *minimal*.

- *Chapitre 5 : « Conception et réalisation d'une plate-forme de simulation d'algorithmes distribués probabilistes »*

  Dans ce chapitre, nous commençons par traiter des notions relatives aux domaines de la simulation en introduisant les concepts de bases de la simulation. Nous évoquons ensuite les problématiques de la simulation distribuée, ainsi que les différents standards et architectures existantes pour mettre en place de telles simulations. À la fin de ce chapitre, nous présenterons notre plate-forme de simulation permettant d'implémenter, visualiser, tester et vérifier les algorithmes distribués codés sous forme de calculs locaux et spécialement ceux décrits sous forme de règles de réécriture.

Première partie

Survol de la théorie des graphes et de l'algorithmique distribuée

Chapitre 1

Preliminaires

Sommaire

Dans ce chapitre, nous présentons les définitions et notations utilisées fréquemment dans ce mémoire : les graphes [Ber85] [Ros00] et le modèle d'algorithmique distribuée [Tel00] [Lyn96] [Lav95a]. Nous présentons ensuite comment un algorithme distribué peut être modélisé au moyen de réétiquetage de graphes. Nous présentons finalement les modèles probabilistes qui seront utilisés pour décrire les algorithmes que nous allons proposer et étudier [Fel50] [Fel66] [Wil91].

1.1 Notions de la théorie des graphes

Dans notre travail, nous nous servons des graphes pour modéliser les réseaux de communications. Dans cette section, nous rappelons quelques notions fondamentales de la théorie des graphes. L'ensemble des définitions est empruntées du livre de C. Berge [Ber85] et celui de Rosen [Ros00].

1.1.1 Définitions élémentaires

Un *graphe non orienté* est un couple (V, E), où V est l'ensemble de sommets et E est l'ensemble de paires d'éléments distincts de V, appelés *arêtes*.

Un graphe *valué* noté $G = (V, E, w)$, où V et E sont définis comme ci-dessus et la valuation est une fonction $w : E \to R$ qui, à toute arête $e \in E$, associe sa valuation (ou poids) $w(e)$.

Le nombre de sommets d'un graphe G est appelé *taille* de G.

Soit $G = (V, E)$ un graphe quelconque. Un *sous-graphe* du graphe G est un graphe $H = (V', E')$ dont l'ensemble des sommets V' (respectivement l'ensemble des E') est un sous-ensemble de V (respectivement de E) tel que pour toute arête $e = \{u, v\}$ de E', les deux sommets u et v appartiennent à V'.

Un *sous-graphe couvrant* du graphe G est un sous-graphe de G qui contient tous les sommets de G.

Le graphe résultant de la suppression d'une arête e d'un graphe $G = (V, E)$ est le graphe G privé de l'arête e.

La suppression d'un sommet v d'un graphe $G = (V, E)$ est l'opération consistant à supprimer le sommet v et toutes les arêtes incidentes à v.

Deux graphes $G_1 = (V_1, E_1)$ et $G_2 = (V_2, E_2)$ sont dits *sommets disjoints* si $V_1 \cap V_2 = \emptyset$. Ils sont dits arêtes disjoints si $E_1 \cap E_2 = \emptyset$.

Un *chemin* P de v_1 à v_i dans G est une suite $P = v_1, e_1, v_2, e_2, \ldots, e_{i-1}, v_i$ de sommets et d'arêtes, telle que, pour $1 \leqslant j < i$, e_j est incidente à v_j et $v_{j+1}, i - 1$ est la *longueur* de P. Si $v_1 = v_i$, alors P est un *cycle*. Deux sommets v et v' sont dits *connectés* ou *liés*, s'il existe un chemin de v à v'.

Un graphe est *connexe* si deux sommets quelconques u et v sont connectés.

Une *composante* connexe du graphe G est un sous-graphe connexe maximal du graphe G.

Remarque 1.1.1. Les définitions que nous avons donné ci-dessus sont relatives aux graphes non orientés. La plupart d'entre elles restent les mêmes pour les graphes orientés.

1.1.2 Arbres et forêts

Un *arbre* T est un graphe connexe sans cycle. Un arbre T' est un facteur d'un arbre T, si soit $T' = T$, soit T' est un facteur d'un arbre obtenu par la suppression d'une feuille de T.

Un *arbre couvrant* d'un graphe G est un sous-graphe couvrant de G qui est un arbre.

Une *forêt* est un graphe dont les composantes connexes sont des arbres. Une *forêt couvrante* d'un graphe G est une forêt qui contient l'ensemble des sommets de G.

1.1.3 Classes particulières de graphes

Un graphe est *complet* si pour toute paire de sommets $\{u, v\}$, l'arête $\{u, v\}$ existe. Nous notons K_n le graphe complet de taille n (ce graphe possède donc $(n \times (n-1))/2$ arêtes).

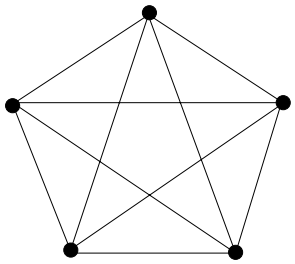


Figure 1 – Exemple de graphe complet (K_5)

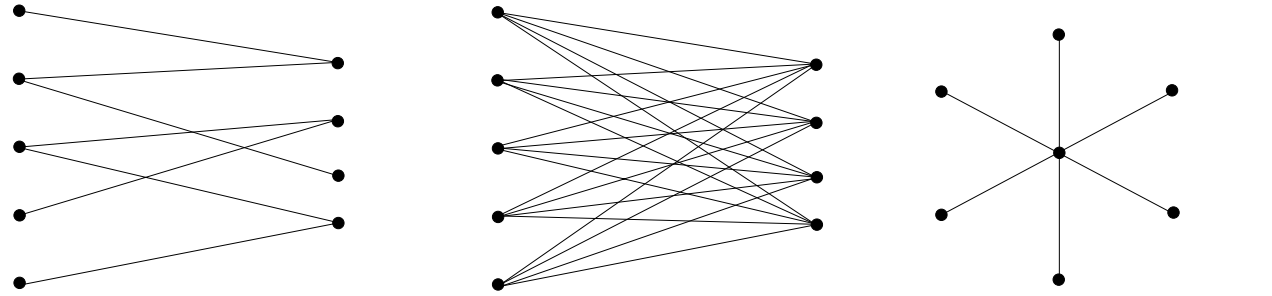
Un graphe G est *biparti* si V peut être partitionné en deux classes disjointes V_1 et V_2 telles que aucune arête de E ne relie deux sommets de V_1 ou deux sommets de V_2. Le graphe G est *biparti complet* si tout sommet de V_1 est adjacent à tout sommet de V_2. On note $K_{n,m}$ le graphe non orienté biparti complet tel que $|V_1| = n$ et $|V_2| = m$.

Une *étoile* de taille n est le graphe $K_{1,n-1}$.

Pour un entier positif r et un sommet v de G, la boule $B(v, r)$, de centre v et de rayon r est définie par induction sur r comme suit :

$$B(v, 0) = \{u\}, \text{et } B(v, r+1) = B(v, r) \cup \bigcup_{w \in B(v,r)} N(w).$$

Où $N(w)$ sont les voisins du sommet w.



a : Graphe biparti

b : Graphe biparti complet ($K_{5,4}$)

c : Graphe en étoile ($K_{1,n-1}$)

FIGURE 2 – Exemple de graphes bipartis

1.1.4 Distances sur un graphe

Soit $G = (V, E)$ un graphe connexe quelconque. La distance entre deux sommets quelconques u et v, notée $d(u, v)$, est la longueur du plus court chemin de u à v. Le *diamètre* du graphe G est la distance maximale entre deux de ses sommets :

$$D(G) = \max_{u,v \in V} \{d(u, v)\}.$$

L'*excentricité* d'un sommet v, notée $e(v)$, d'un graphe connexe G est la distance maximale entre v et les autres sommets de G. Le rayon du graphe G, noté $r(G)$, est alors l'excentricité minimum sur tous ses sommets :

$$e(u) = \max_{v \in V} \{d(u, v)\} \text{ et } r(G) = \min_{u \in V} e(u).$$

Un sommet v est un *centre* d'un graphe connexe G si c'est un élément de l'ensemble des sommets d'excentricité minimum, c'est-à-dire de l'ensemble :

$$c(G) = \{v \mid e(v) = \min_{w \in V} e(w)\}.$$

Soit $T = (V, E)$ un arbre, pour un sommet v, nous définissons $Dist(v, T)$ par :

$$Dist(v, T) = \sum_{w \in V} d(v, w).$$

Un sommet de V est un *sommet médian*, si c'est un élément du sous-ensemble $m(T)$, défini par :

$$m(T) = \{v \mid Dist(v, T) = \min_{w \in V} Dist(w, T)\}.$$

1.2 Systèmes de réécriture de graphe

1.2.1 Graphes étiquetés

Les réécritures de graphes [LMS99] constituent un prolongement des travaux menés par P. Rosenstiehl et al. [FHR72] dans les années 70, par D. Angluin [Ang80] dans les années 80 et plus récemment par Yamashita et Kameda [YK96] concernant les propriétés locales et les propriétés globales dans les réseaux et les graphes.

Dans tout ce qui suit, nous considérons seulement les graphes connexes simples où des sommets et des arêtes sont étiquetés avec des étiquettes appartenant à un alphabet L. Cet alphabet peut être fini ou infini.

Définition 1.2.1. Soit $\mathcal{L}$ un ensemble dont les éléments sont appelés états. Un *graphe $\mathcal{L}$-étiqueté* est un couple (G, λ) où G est un graphe et λ est une fonction de $V \cup E$ vers $\mathcal{L}$.

$$\lambda : V \cup E \longrightarrow \mathcal{L}$$

$$(v, e) \longrightarrow X \in \mathcal{L}$$

Définition 1.2.2. Le graphe G est appelé le *graphe sous-jacent*, et λ est une fonction d'étiquetage de G.
La classe des graphes étiquetés sur un alphabet fini L sera notée $\mathcal{G}_L$.

Définition 1.2.3. Soit (G, λ) et (G', λ') deux graphes étiquetés. (G, λ) est un sous-graphe étiqueté de (G', λ'), noté par $(G, \lambda) \subseteq (G', \lambda')$, si G est un sous-graphe de G' et λ est la restriction de λ' à $V(G) \cup E(G)$.

Définition 1.2.4. L'application $\varphi : V(G) \to V(G')$ est un homomorphisme du graphe étiqueté de (G, λ) vers (G', λ'), si φ est un homomorphisme de G vers G' préservant les étiquettes, c'est à dire, pour tous sommets $v, w \in V(G), \lambda'(\varphi(v)) = \lambda(v),$ et $\lambda'(\{\varphi(v), \varphi(w)\}) = \lambda(\{v, w\})$.
φ est un isomorphisme du graphe étiqueté s'il est bijectif.

Définition 1.2.5. Une *occurrence* de (G, λ) dans (G', λ') est un isomorphisme φ entre (G, λ) et un sous-graphe (H, η) de (G', λ').

1.2.2 Systèmes de réécriture de graphe

Nous décrivons dans cette section les notions formelles des systèmes de réécriture de graphe, comme présentées dans [LMS99] [FHR72] [Ang80] [YK96].

Définition 1.2.6. Une règle de *réécriture* est un triplet $R = (G_R, \lambda_R, \lambda'_R)$ tel que (G_R, λ_R) et (G_R, λ'_R) sont deux graphes étiquetés.

Définition 1.2.7. Un *système de réécriture* de graphe (SRG) est un triplet $\mathcal{R} = (\mathcal{L}, I, P)$ où $\mathcal{L}$ est un ensemble d'états, I est un sous-ensemble de $\mathcal{L}$ appelé l'ensemble des *états initiaux* et P un ensemble fini de règles de réécriture.

Définition 1.2.8. Une $\mathcal{R}$-*étape de réécriture* est un quintuplet $(G, \lambda, R, \phi, \lambda')$ tel que R est une règle de réécriture et ϕ est à la fois une occurrence de (G_R, λ_R) dans (G, λ) et une occurrence de $(G_{\mathcal{R}}, \lambda'_R)$ dans (G, λ').
Une étape de réécriture est notée $(G, \lambda) \rightarrow_{R,\phi} (G, \lambda')$.

Définition 1.2.9. Une *séquence* de $\mathcal{R}$-*réécriture* est un uplet $(G, \lambda_0, R_0, \varphi_0, \lambda_1, R_1, \varphi_1, \lambda_2, \ldots, \lambda_{n-1}, R_{n-1}, \varphi_{n-1}, \lambda_n)$ tel que pour tout $i, 0 \leq i < n, (G, \lambda_i, R_i, \varphi_i, \lambda_{i+1})$ est une étape de $\mathcal{R}$-réécriture.
L'existence de telles séquences de réécriture sera notée par $(G, \lambda_0) \xrightarrow[\mathcal{R}]{*} (G, \lambda_n)$.

Le calcul s'arrête quand l'étiquetage du graphe est tel qu'aucune règle de réécriture ne peut être appliquée.

Définition 1.2.10. Un graphe étiqueté (G, λ) est alors dit $\mathcal{R}-$*irréductible* s'il n'existe aucune occurrence de (G_R, λ_R) dans (G, λ) pour toute règle de réécriture R de P.

Pour tout graphe étiqueté (G, λ) dans $\mathcal{G}_1$ nous notons par $Irred_R((G, \lambda))$ l'ensemble de tous les graphes étiquetés $\mathcal{R}-$irréductible (G, λ_0) tels que $(G, \lambda) \xrightarrow[\mathcal{R}]{*} (G, \lambda')$. Intuitivement parlant, l'ensemble des $Irred_R((G, \lambda))$ contient tous les étiquetages finaux qui sont obtenus à partir des graphes $I-$étiquetés par application des règles de réécriture de P et peuvent être vu comme un ensemble de tous les résultats possibles des calculs codés par le système $\mathcal{R}$. Nous nous intéressons à l'étude de la probabilité d'obtenir une solution particulière parmi un ensemble de solutions possibles.

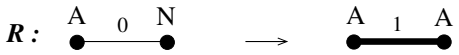
Un système de réécriture est défini par la donnée d'un ensemble fini de telles règles et par une structure de contrôle local : ce contrôle local autorise ou bien interdit l'application de certaines règles. Ainsi nous allons définir des réécritures contrôlées par "les contextes interdits" : nous ne pouvons appliquer une règle que si dans l'environnement immédiat de l'application certains contextes sont absents. Le comportement du réseau est défini par un étiquetage initial et par une suite de pas de calcul.

Deux étapes de réécriture sont dites indépendantes si elles sont exécutées dans des sous-graphes disjoints. Dans ce cas, ces deux étapes peuvent être appliquées dans n'importe quel ordre et même simultanément. Ces règles locales de réécriture sont appliquées jusqu'à ce que la "stabilité" soit atteinte, c'est-à-dire aucune règle n'est applicable. La configuration résultante est dite de forme normale. En effet, une exécution de l'algorithme est ainsi

déterminée par une succession de règles de $\mathcal{R}$ appliquée en une boule de rayon fixe.

Exemple 1.2.1. Illustrons les systèmes de réécriture de graphe en considérant un algorithme distribué simple qui calcule un arbre recouvrant d'un réseau. Nous supposons l'existence d'un processeur distingué unique dans le réseau qui est dans un état "actif" (codé par l'étiquette A), tous les autres processeurs sont dans un un état "neutre" (étiquette N) et tous les canaux de communications sont dans un état "passif" (étiquette 0). Au début, l'arbre ne contient que le sommet actif. A n'importe quelle étape du calcul, un sommet actif peut activer un de ses voisins neutres et marque le canal de communication correspondant par l'étiquette 1. Le calcul s'arrête dès que tous les processeurs sont activés. L'arbre recouvrant est alors obtenu en considérant tous les canaux de communications avec l'étiquette 1.

Le système est donné par $\mathcal{R}_1 = (\mathcal{L}, \mathcal{I}, \mathcal{P})$ défini par $\mathcal{L} = \{N, A, 0, 1\}$, $\mathcal{I} = \{N, A, 0\}$ et $\mathcal{P} = \{R\}$ où R est la règle de réécriture suivante : A chaque fois qu'un nœud étiqueté A est



lié par une arête étiquetée 0 à un nœud dont l'état est N, alors la règle R nous permet de réécrire le sous-graphe correspondant.

Un exemple de calcul utilisant cette règle est donné dans la figure 3. Les étapes de réécriture peuvent s'exécuter simultanément dans des parties disjointes du graphe. Quand le graphe devient irréductible, c'est à dire aucune règle ne peut s'appliquer, un arbre recouvrant, constitué des arêtes étiquetées 1, est alors calculé.

1.2.3 Systèmes de réécriture de graphe avec contextes interdits

L'idée que nous développons ici est d'empêcher l'application d'une règle de réécriture toutes les fois où les occurrences correspondantes sont "inclues" dans certaines configurations spéciales, appelées contextes. Plus formellement, nous avons :

Définition 1.2.11. Soit (G, λ) un graphe étiqueté. Un *contexte* de (G, λ) est un triplet (H, μ, ψ) tel que (H, μ) est un graphe étiqueté et μ une occurrence de (G, λ) dans (H, μ).

Définition 1.2.12. Une *règle de réécriture avec contextes interdits* est un quadruplet $R = (G_R, \lambda_R, \lambda'_R, F_R)$ tel que $(G_R, \lambda_R, \lambda'_R)$ est une règle de réécriture et F_R est un ensemble fini de contextes de (G_R, λ_R). Une règle de réécriture peut être appliquée dans un sous-graphe si et seulement si ce sous-graphe n'est pas inclus dans une occurrence de l'un de ses contextes interdits [LMS95][LMS99].

$$\mathcal{R} : Left \longrightarrow Right \ ; \ \{F_C\}$$

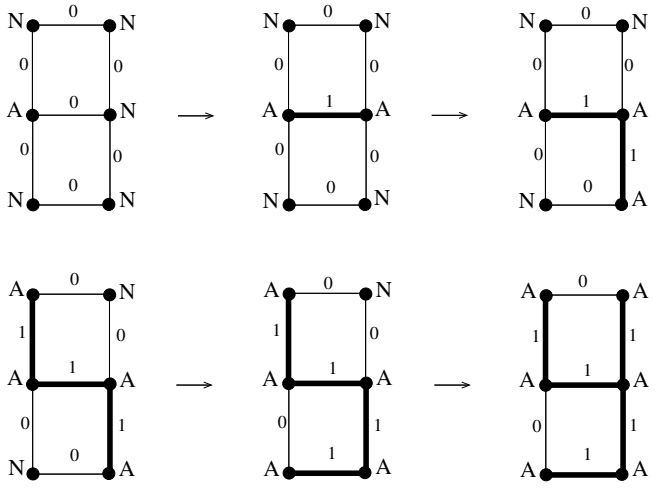
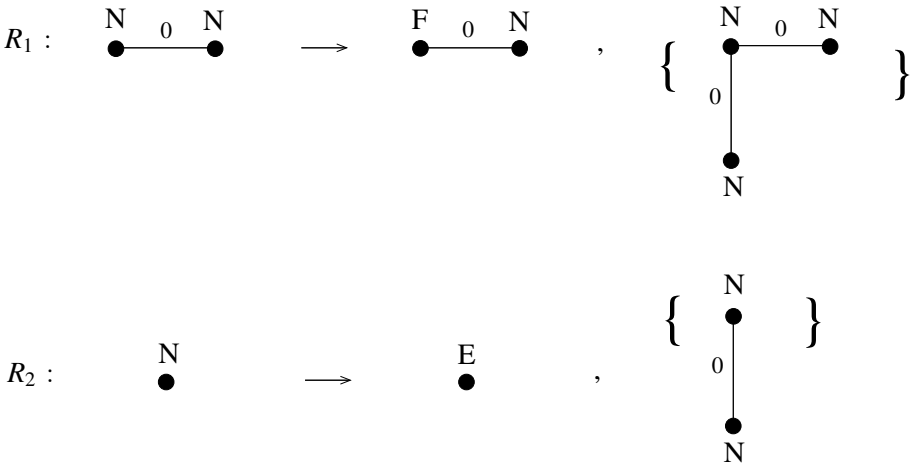


FIGURE 3 – Calcul distribué d'un arbre recouvrant

Définition 1.2.13. Un *système de réécriture de graphe avec contextes interdits* ($SRGCI$) est un triplet $\mathcal{R} = (\mathcal{L}, \mathcal{I}, \mathcal{P})$ défini comme un SRG sauf que $\mathcal{P}$ est un ensemble de règles de réécriture avec contextes interdits.

Définition 1.2.14. Une étape de $\mathcal{R} - rcriture$ est un 5-uplet $(G, \lambda, R, \varphi, \lambda)$ tel que R est une règle de réécriture avec contextes interdits de P, φ est une occurrence de (G_R, λ_R) dans (G, λ) et une occurrence de (G_R, λ'_R) dans G, λ', et pour tout contexte (H_i, μ_i, ψ_i) de (G_R, λ_R), il n'y a aucune occurrence φ_i de H_i, μ_i dans (G, λ) tel que $\varphi_i(\psi_i(G_R, \lambda_R)) = \varphi(G_R, \lambda_R)$.

Exemple 1.2.2. Dans cet exemple, nous donnons un système de réécriture de graphe qui code un algorithme d'élection dans un arbre. Soit $\mathcal{R}_2 = (L, I, P)$ le $SRGCI$ défini par $L = \{N, F, E, 0\}, I = \{N, 0\}$ et $P = \{R_1, R_2\}$ où R_1, R_2 sont les règles de réécriture avec contextes interdits suivants :



Appelons une *feuille* tous les nœuds étiquetés N ayant exactement un voisin avec l'étiquette N. La règle R_1 consiste à "enlever" une feuille de l'arbre (puisque le contexte interdit assure que ce nœud n'a pas d'autre voisin étiqueté N) en lui donnant l'étiquette F. Ainsi, si (G, λ) est un arbre étiqueté où tous les nœuds sont étiquetés N et toutes les arêtes sont étiquetées 0 alors la procédure d'élimination mène à un unique sommet étiqueté N qui deviendra élu grâce à la règle R_2. Il n'est pas difficile de voir que tous les nœuds de l'arbre peuvent être "élus" par cet algorithme.

Exemple 1.2.3. Comme pour l'exemple 1.2.1, nous supposons qu'initialement un seul nœud soit étiqueté A, tous les autres sommets sont étiquetés N et toutes les arêtes sont étiquetées 0. L'idée générale est que le sommet initialement étiqueté A gardera son état jusqu'à la fin du calcul, par contre les autres nœuds seront activés et aurons l'étiquette A'. Dès qu'un nœud étiqueté A' n'est plus "utile" pour le calcul, il atteindra l'état final (étiquette F).

A chaque étape de calcul, un nœud u actif (étiqueté A ou A') agira comme suit :

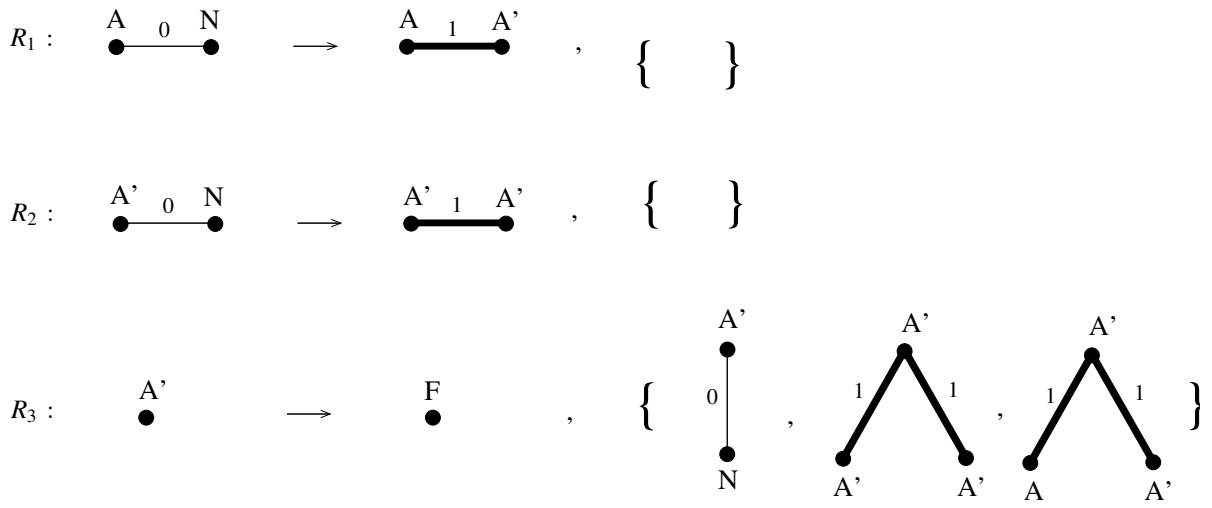
1. Si u a un nœud voisin v étiqueté N, alors u va activer ce nœud : u gardera son étiquette, v devient actif (étiquette A') et l'arête $\{u, v\}$ prend l'étiquette 1.
2. Si u est étiqueté A', u n'a pas de nœud voisin étiqueté N et il est tel que tous ses nœuds voisins excepté un, dont l'arête qui les lie est étiquetée 1, sont étiquetés F, alors u devient étiqueté F.

A tout moment, le sous-graphe induit par les arêtes étiquetées 1 et les nœuds étiquetés A et A' est un arbre. Intuitivement, la deuxième étape signifie que le nœud u est une feuille de cet arbre.

Ainsi, cet algorithme s'exécute en deux phases (qui peuvent se chevaucher) : dans la

première, l'arbre grandit jusqu'à ce que tous les sommets soient atteints ; dans la deuxième phase, l'arbre va diminuer (en perdant ses feuilles) jusqu'à ce qu'il ne reste que le sommet initialement étiqueté A. Ce sommet est capable de détecter que l'algorithme a fini quand tous ses voisins seront étiquetés F.

L'algorithme peut être codé par le système de réécriture de graphe avec contextes interdits $\mathcal{R}_3 = (L, I, P)$ défini par $L = \{N, A, A', F, 0, 1\}, I = \{N, A, 0\}, P = \{R_1, R_2, R_3\}$ où R_1, R_2 et R_3 sont les règles de réécriture avec contextes interdits suivants :



Les règles R_1 et R_2 n'ont pas de contextes interdits : il n'y a pas de restriction pour les appliquer. La règle R_3 a 3 contextes interdits.

La figure 4 décrit un exemple de calcul utilisant cet algorithme.

Définition 1.2.15. Un système de réécriture de graphe $\mathcal{R}$ est dit noethérien s'il n'existe pas de séquence infinie de $\mathcal{R}-$réécriture de graphe tel qu'initialement, l'étiquetage du graphe satisfait l'ensemble des états initiaux [LMS99].

1.3 Notions de la théorie des probabilités

Dans certains systèmes, nous sommes amenés à introduire la notion de probabilités, et ce pour plusieurs raisons. Elle permet tout d'abord de modéliser de façon probabiliste certains comportements réels après une observation statistique, comme par exemple sur des canaux non fiables, la probabilité qu'un message soit perdu.

Les probabilités peuvent également être introduites pour des raisons d'efficacité. L'ajout de probabilités est même parfois nécessaire. En effet, il existe des problèmes pour lesquels il a été prouvé qu'il n'existe pas de solution déterministe. C'est souvent le cas dans des

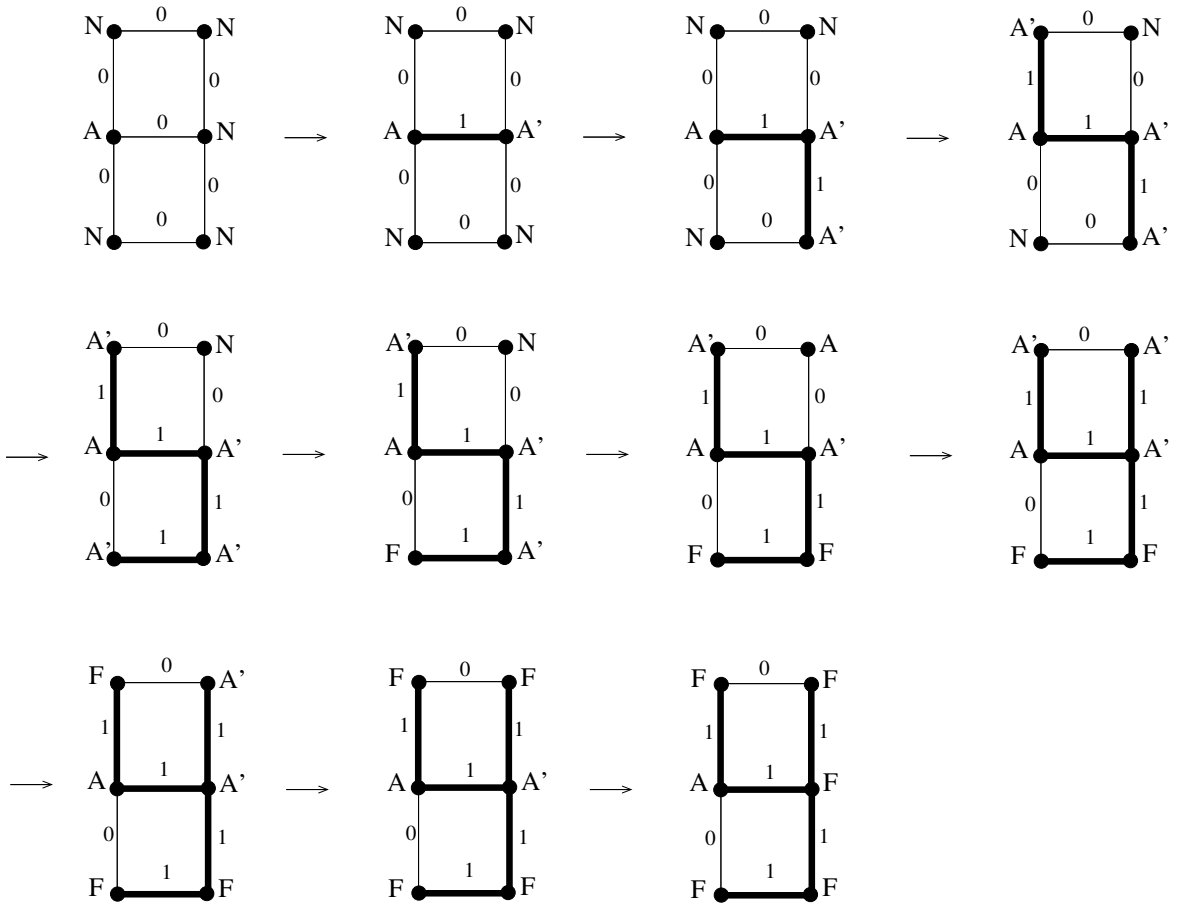


Figure 4 – Calcul distribué d'un arbre recouvrant avec détection locale de la terminaison globale globale

systèmes symétriques où les probabilités sont requises pour rompre la symétrie. Un exemple bien connu est celui du dîner des philosophes [Dij72], pour lequel Lehmann et Rabin [LR81] ont montré qu'il n'existe pas de solution déterministe, symétrique et totalement distribuée. Ils donnent alors une solution probabiliste à ce problème [LR94]. L'algorithme proposé permet de résoudre un problème d'allocation de ressource insoluble dans le cadre déterministe. Informellement, N philosophes sont assis autour d'une table, une baguette étant disposée entre deux philosophes "voisins". Pour manger, un philosophe a besoin de tenir simultanément ses deux baguettes, à savoir celle partagée avec son voisin de gauche et celle partagée avec son voisin de droite. Le but de l'algorithme étant d'assurer que si un philosophe a faim, alors en un temps fini un philosophe (pas nécessairement le même) va manger.

La modélisation mathématique pour étudier un problème peut être issue du domaine des probabilités. Notre but n'est évidemment pas de donner une introduction relativement complète à la théorie des probabilités, mais de présenter les outils et méthodes bien connus dont nous aurons besoin dans la suite de cette thèse. Les références en la matière ne manquent pas. Parmi elles nous pouvons citer les ouvrages [Fel50, Fel66].

Les notations utilisées en probabilité, $Pr(E), \mathbb{E}(X)$ et $var(X)$, sont les notations usuelles respectives de la probabilité d'un événement E, de l'espérance et de la variance d'une variable aléatoire $(v.a.)X$.

1.3.1 Variables aléatoires et lois de probabilités

Une *v.a. (variable aléatoire)* est une application mesurable définie sur l'ensemble des résultats possibles d'une expérience aléatoire, telle qu'il soit possible de déterminer la probabilité qu'elle prenne une valeur donnée ou qu'elle prenne une valeur dans un intervalle donné. Cette localisation conduit à associer à toute v.a. une loi de probabilité sur $\mathbb{R}$.
Les variables aléatoires $X_1, \ldots, X_n$ sont dites indépendantes si pour tout n-uplet $(A_1, \ldots, A_n)$ de boréliens de $\mathbb{R}$, les évènements "$X_1 \in A_1, \ldots, X_n \in A_n$" sont indépendants. Une suite $(X_n)_{n\geq 1}$ de variables aléatoires indépendantes est telle que pour tout n les variables aléatoires $(X_1, \ldots, X_n)$ sont indépendantes.

Nous appelons loi de la v.a. X la loi de probabilité P_X sur $\mathbb{R}$, définie pour tout borélien A de $\mathbb{R}$ par :

$$P_X(A) = Pr(X \in A) \ .$$

Variables aléatoires discrètes

Nous disons qu'une v.a. est discrète si elle ne prend qu'un nombre fini ou dénombrable

de valeurs :

$$X \in \{x_k, k \in K \subset \mathbb{N}\} .$$

Dans le cas où une v.a. est discrète, la loi de la v.a. X est la loi de probabilité sur l'ensemble des valeurs possibles de X qui affecte la probabilité $Pr(X = x_k)$ au singleton $\{x_k\}$.

Les lois discrètes les plus courantes sont : la loi uniforme, la loi de Bernoulli et la loi binomiale.

La loi uniforme sur un ensemble fini est la loi des "tirages au hasard" dans cet ensemble, ou équiprobabilité. Elle donne la même probabilité $1/n$ à tous les éléments de l'ensemble, avec n est le cardinal de cet ensemble.

Variables aléatoires continues

Soit X une v.a. à valeurs dans $\mathbb{R}$ et f_X une densité de probabilité sur $\mathbb{R}$. Nous disons que X est une v.a. continue de densité f_X si pour tout intervalle I de $\mathbb{R}$ nous avons :

$$Pr(X \in I) = \int_I f_X(x)\ dx.$$

La loi de la v.a. X est la loi continue sur $\mathbb{R}$, de densité f_X.

Les lois continues les plus fréquentes sont : la loi uniforme, la loi exponentielle, la loi normale, la loi gamma et la loi student.

1.3.2 Fonction de répartition

La fonction de répartition d'une v.a. X à valeurs dans $\mathbb{R}$ (ou plus exactement de sa loi) est la fonction F_X, de $\mathbb{R}$ dans l'intervalle réel $[0, 1]$, qui à $x \in \mathbb{R}$ associe :

$$F_X = Pr(X \leq x) .$$

La fonction de répartition d'une variable aléatoire continue est la primitive de la densité qui s'annule en $-\infty$:

$$F_X(x) = Pr(X \leq x) = \int_{-\infty}^{x} f_X(t)dt .$$

C'est une fonction continue sur $\mathbb{R}$. En tout point x où f_X est continue, F_X est dérivable et :

$$\frac{dF_X(x)}{dx} = f_X(x) .$$

Le théorème suivant donne une interprétation du produit de convolution de deux fonctions de répartition.

Théorème 1.3.1. *[Fel50] Soient X et Y deux variables aléatoires indépendantes de fonctions de répartition F et G respectivement. Alors*

$$Pr(X + Y \leq t) = \int_{-\infty}^{+\infty} G(t - x)F\{dx\} \ .$$

1.3.3 Processus et chaînes de Markov

Pour décrire un ensemble de phénomènes où le hasard intervient indépendamment de notre volonté, nous introduisons la notion de processus stochastique.

Un processus stochastique est une famille de variables aléatoires X_t :

$$\{X_t, t \in T\}$$

où t représente souvent le temps. Les valeurs de T peuvent être continues ou discrètes.

Une classe particulière de ces processus a été définie dans les années 30 par le mathématicien russe Markov [Fel60].

Nous appellerons *état* une valeur du processus stochastique prise à un instant t, et *transition* le passage d'un état à un autre.

Un processus est markovien, si pour tout instant u, et pour toute valeur $X_u = x$ donnée, la probabilité pour que le processus prenne la valeur y à un instant quelconque t ultérieur $(t > u)$, ne dépend pas des valeurs prises par le processus avant l'instant u. Le processus est *sans mémoire.*

Les chaînes de Markov sont des processus markoviens qui évoluent dans le temps "discret" selon des transitions probabilistes. Elles vont donc décrire un ensemble de phénomènes aléatoires selon le principe des processus stochastiques.

Soit $(X_t)_{t\in\mathbb{N}}$ une suite de variables aléatoires à valeur dans $\mathbb{R}$. Nous disons que (X_t) est une chaîne de Markov si :

$$Pr(X_t + 1 = x_{t+1}|X_t = x_t, \ldots, X_0 = x_0) = Pr(X_{t+1} = x_{t+1}|X_t = x_t)$$

Ce qui signifie que X_{t+1} ne dépend que de X_t. L'indice est en général interprété comme le temps, ce qui nous précise que la probabilité sur la valeur de X à l'instant $t + 1$ ne dépend que de sa valeur à l'instant t : ceci est appelé la propriété de Markov. De plus la chaîne de Markov sera dite homogène si la probabilité $Pr(X_{t+1} = x_{t+1}|X_t = x_t)$ ne dépend pas de t.

Pour nos problèmes, X_t caractérise le système à l'instant t : cela peut être le graphe résiduel (sommets et arêtes non encore supprimés). L'état initial ($t = 0$) est le graphe tout entier.

Une chaîne de Markov est irréductible si, partant d'un état quelconque et considérant un état quelconque, nous avons une probabilité non nulle pour qu'un jour, le système passe par cet état final. Dans ce cas nous disons que tous les états communiquent.

1.4 Notion de la complexité algorithmique

Il existe un grand nombre d'algorithmes possibles pour la plupart des problèmes. Comment en choisir le meilleur ? Quels sont les différents degrés de complexité que l'on peut rencontrer ?

Ces questions sont souvent d'une grande importance pratique, et la théorie associée constitue un domaine d'étude très poussée de l'informatique (Pour un développement plus détaillé voir [Gra94]).

1.4.1 Coût d'un algorithme

Étant donné un algorithme et une donnée d'entrée x, nous pouvons mesurer le coût $c(x)$, aussi appelé la complexité, de la méthode en question. Ce qui nous intéresse le plus souvent est la complexité temporelle : l'exécution de chaque instruction primitive nécessite un certain temps, et le temps d'exécution de l'algorithme est le temps cumulatif de toutes les instructions exécutées l'une après l'autre. (Pour simplifier nous supposons parfois que les instructions primitives ont toutes le même coût, ainsi défini comme coût unitaire.) Très souvent le coût cumulatif est proportionnel au nombre d'itérations, et nous nous contentons de déterminer ce dernier. L'analyse de complexité consiste ainsi à étudier la fonction $c : x \to c(x)$ qui associe le coût $c(x)$ à chaque entrée possible x soumise à l'algorithme.

1.4.2 Coût moyen, dans le pire et dans le meilleur des cas

L'analyse fine que nous venons de décrire n'est souvent pas praticable. Dans un tel cas, nous préférons des informations plus globales, en regroupant les données d'une même « taille ». Typiquement les données d'entrée x admettent une notion de taille naturelle, que nous noterons $|x|$: pour une liste ou un vecteur c'est sa longueur, pour un nombre entier c'est la longueur de son développement binaire, pour un polynôme c'est son degré, etc.

Pour une taille n donnée, nous pouvons alors regarder l'ensemble $X_n = \{x/|x| = n\}$ de toutes les données de taille n. Nous définissons le coût moyen par :

$$c^m(n) = \frac{1}{|X_n|} \sum_{x \in X_n} c(n).$$

Ici nous sous-entendons que l'ensemble X_n est fini et que toutes les données sont équiprobables. (D'autres distributions de probabilités sont parfois mieux adaptées, selon le contexte.) Pour être prudent, nous regardons également le coût dans le pire des cas :

$$c^+(n) = \max_{x \in X_n} c(n).$$

Ce point de vue est indispensable si nous voulons garantir une certaine performance dans tous les cas, non seulement en moyenne. Il est parfois instructif de regarder le coût dans le meilleur des cas :

$$c^-(n) = \min_{x \in X_n} c(n).$$

1.4.3 Complexités asymptotiques

Souvent c'est le principe de l'algorithme que nous voulons juger, et non les détails de l'implémentation. Nous voulons donc abstraire de tous les facteurs constants ; de toute façon, ils changeront d'une implémentation à une autre, et d'une machine à une autre. Pour ne retenir que l'essentiel, nous regroupons les fonctions suivant leur « ordre de grandeur ».

Nous utilisons les notations suivantes (voir [SF96]). Soient f et g deux fonctions de $\mathbb{N}$ dans $\mathbb{R}^+$. Nous disons que :

- $f(n) = O(g(n))$ si et seulement si il existe une constante c et un entier n_0 tels que $\forall n > n_0,\ f(n) \leq cg(n)$.
- $f(n) = \Omega(g(n))$ si et seulement si il existe une constante c et un entier n_0 tels que $\forall n > n_0, f(n) \geq cg(n)$.
- $f(n) = \Theta(g(n))$ si et seulement si il existe deux constantes $c_1 < c_2$ et un entier n_0 tels que $\forall n > n_0, c_1 g(n) \leq f(n) \leq c_2 g(n)$.
- $f(n) = o(g(n))$ si $f(n)$ est négligeable devant $g(n)$, c'est-à-dire, $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.
- $f(n) \backsim g(n)$ si $f(n)$ est asymptotiquement équivalente à $g(n)$, c'est-à-dire, $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$.

1.4.4 Principales classes de la complexité

Dans la pratique, nous avons souvent affaire à des fonctions de complexité dont le comportement asymptotique est un des suivants :

- **Complexité constante,** $O(1)$: les instructions primitives en C++ sont de coût constant, par exemple tous les calculs effectués sur les variables de type int (dont la taille est fixée). De même pour accéder à l'élément $v[i]$ d'un vecteur v.
- **Complexité logarithmique,** $O(\ln n)$: un programme de complexité logarithmique devient seulement très légèrement plus lent quand n croît. Chaque fois que n est doublé, le coût n'augmente que par addition d'une constante.
- **Complexité linéaire,** $O(n)$: c'est le mieux que l'on puisse espérer pour un algorithme qui doit traiter n données une par une. Chaque fois que n est doublé, le coût double lui aussi.
- **Complexité presque linéaire,** $O(n \ln n)$: c'est la complexité typique pour les algorithmes de type « diviser pour régner ». Chaque fois que n est doublé, le coût est un peu plus que doublé (mais guère plus).
- **Complexité sous-quadratique,** $O(n^\alpha)$ **avec** $\alpha < 2$: la multiplication de deux entiers de longueur n en numération décimale nécessite un temps $O(n^{1.585})$. Cette complexité situe donc entre la complexité linéaire de l'addition et la complexité quadratique de la multiplication scolaire.
- **Complexité quadratique,** $O(n^2)$: les complexités quadratiques sont typiques pour les algorithmes traitant tous les couples parmi n données (éventuellement par deux boucles imbriquées).
- **Complexité polynomiale,** $O(n^k)$ **avec** $k > 1$: typiquement un algorithme qui traite tous les k-uplets parmi n données est de complexité $O(n^k)$. De tels algorithmes ne sont utilisables que pour des problèmes relativement petits.
- **Complexité exponentielle,** $O(e^{\alpha n})$ **avec** $\alpha > 0$**, voire** $O(e^{p(n)})$ **avec un polynôme** p : un algorithme de complexité exponentielle est pratiquement inutilisable, sauf peut-être pour les problèmes très petits.
- **complexité sur-exponentielle** : il existe aussi des fonctions de croissance sur-exponentielle, comme $\exp(\exp(n))$. Des algorithmes d'une telle complexité n'ont pas d'intérêt pratique ; ils peuvent néanmoins être très importants au niveau théorique, par exemple pour montrer l'existence d'une solution, aussi inefficace qu'elle soit.

Chapitre 2

Généralités sur les algorithmes distribués

Sommaire

Dans ce chapitre, nous introduisons quelques rappels sur les systèmes distribués, leur définition et les modèles de calcul couramment rencontrés dans la littérature. Puis, nous décrivons la motivation de l'utilisation d'un système distribué et nous exposons les différents types de topologie. Nous présentons, ensuite, les problèmes soulevés par de tels systèmes. Nous détaillons, avant de terminer ce chapitre, un panel non exhaustif des problèmes classiques de l'algorithmique distribuée. Enfin, nous présentons les algorithmes distribués probabilistes indiquant ce qui change par rapport au cas déterministe.

2.1 Système distribué

Dans [Tel94], Gerrard Tel a définit un système distribué comme étant "une collection de ressources autonomes de calcul inter-connectées". Chacune de ces entités est appelée site quel que soit sa nature (ordinateurs, processus, processeurs, ...).

L'auteur précise "que ces ressources doivent assurer leur propre contrôle", pour être qualifiées d'autonomes. La qualification d'inter-connectées signifie que les ressources peuvent s'échanger des informations (quelque soit le support de communication).

La communication entre les processus peut se faire soit par un moyen de mémoire partagée (modèle de mémoire partagée) ou par l'intermédiaire des liens qui relient les procédés directement les uns avec les autres (système de transmission de messages).

Un système distribué est, généralement, modélisé par un graphe où l'ensemble des sommets du graphe représente les processeurs (appelés nœuds) et l'ensemble des arêtes représente les liens de communications.

Les situations impliquant les systèmes distribués sont très variées, et visent des buts différents. En voici les principaux :

- *Les calculs intrinsèquement distribués* : Dans de nombreuses applications telles que le transfert de l'argent dans la banque, ou parvenir à un consensus entre les parties qui sont géographiquement éloignés, le calcul est intrinsèquement distribué.
- *Le partage des ressources* : Des ressources telles que des périphériques, des ensembles complets de données dans des bases de données, bibliothèques spécialisées, ainsi que des données (variables/fichiers) ne peuvent pas être entièrement répliqués sur tous les sites car il est souvent ni pratique ni rentable. En outre, ils ne peuvent être placés sur un seul site parce que l'accès à ce site pourrait s'avérer être un goulot d'étranglement. Par conséquent, ces ressources sont généralement distribués dans tout le système. Par exemple, les bases de données distribuées telles que DB2 partitionnent les ensembles de données sur plusieurs serveurs, en plus de les répliquer sur quelques sites d'accès rapide, ainsi que la fiabilité.
- *L'accès aux données et ressources géographiquement éloignées* : Dans de nombreux scénarios, les données ne peuvent pas être répliquées sur chaque site participant à l'exécution distribuée car il peut être trop grand ou trop sensible pour être répliqué. Par exemple, les données de paie dans une multinationale est à la fois trop grande et trop sensible pour être répliquées à chaque succursale/site. Il est donc conservé à un serveur central qui peut être interrogé par des succursales. De même, les ressources spéciales telles que super-ordinateurs n'existent que dans certains endroits, et d'accéder à ces super-ordinateurs, les utilisateurs doivent se connecter à distance. Progrès dans la conception des appareils mobiles à ressources limitées aussi bien comme dans la technologie sans fil avec laquelle ces appareils communiquent ont donné un nouvel élan à l'importance des protocoles distribués et middleware.

2.1.1 Exemples de grands systèmes distribués

Tous les services classiques d'Internet sont de bons exemples : WWW, DNS, SMTP,... Ils suivent une approche de type client/serveur, parfois avec des hiérarchies.

Des systèmes comme *BitTorrent* ont une architecture pair-à-pair où tout le monde peut être à la fois client et serveur, et il n'y a pas de véritable maître.

Les systèmes embarqués sont de plus souvent répartis. De nos jours, ces systèmes sont de plus en plus présents dans les machines qui nous entourent comme, les voitures, les avions et les trains.

Les réseaux de capteurs sans fils sont comme leur nom l'indique des réseaux ad hoc dont les nœuds sont des capteurs qui récoltent et transmettent les données récoltées. Un tel réseau doit savoir s'organiser tout afin de pouvoir être "semé" de façon imprécise.

Le réseau constitué de satellites GPS (*Global Positioning System*) et des récepteurs GPS est un autre exemple de système réparti. Ici les clients reçoivent des informations de plusieurs serveurs (les satellites) pour déterminer leur position.

2.1.2 Topologie des systèmes distribués

Les interconnexions entre les différents sites du système distribué ne respectent aucune règle particulière (c'est-à-dire que les systèmes distribués peuvent être structurés de manière quelconque) ou les systèmes peuvent être organisés en fonction d'un certain nombre de règles. Certains algorithmes ne fonctionnent qu'en exploitant la particularité topologique d'un système distribué. Nous détaillons ici les topologies les plus couramment utilisées pour la conception d'algorithmes.

- *Arbre* : Un arbre de n nœuds est un réseau connexe contenant exactement $n-1$ liens de communication. Un arbre a la particularité de ne pas contenir de cycle. Il n'existe alors entre deux sites qu'un unique chemin simple. Il est possible de construire un arbre couvrant sur tout réseau arbitraire connexe.
- *Grille* : Une grille de $n \times m$ nœuds est un réseau pour lequel chaque nœud est étiqueté (i, j) tel que $i < n, j < m$. Deux nœuds (i, j) et (i', j') sont voisins si :
  $((i = i') \wedge (j = j' + 1)) \vee ((i = i') \wedge (i = j' - 1)) \vee ((i = i' + 1) \wedge (j = j')) \vee ((i = i' - 1) \wedge (j = j'))$
- *Grille torique* : Une grille torique de $n \times m$ nœuds est un réseau pour lequel chaque nœud est étiqueté (i, j) tel que $i < n, j < m$. Deux nœuds (i, j) et (i, j) sont voisins si :
  $((i = i') \wedge (j = j' + 1 \mod n)) \vee ((i = i') \wedge (j = j - 1 \mod n)) \vee ((i = i' + 1 \mod n) \wedge (j = j')) \vee ((i = i' - 1 \mod n) \wedge (j = j'))$.
  Tous les sommets d'une grille torique ont 4 voisins.
- *Étoile* : Une étoile est un arbre où un nœud (appelé le centre) est voisin de tous les

autres sommets.
La distance maximale entre deux sites dans une étoile est de 2.

- *Anneau* : Un anneau de taille n est un réseau où les sites numérotés de 0 à $n-1$ sont tels qu'il n'existe un canal de communication qu'entre i et $i+1 \mod n$.
  Il existe exactement entre deux sites distincts quelconques deux chemins simples.
- *Clique* : Une clique, ou réseau complet, est un réseau où pour tout couple de sites (i, j), i est voisin de j.
  Dans ce type de topologie, tous les sites peuvent communiquer entre eux de manière directe et non à travers une séquence de sites. La mise en place d'une stratégie permettant la réduction de communications y est donc particulièrement aisée.

2.1.3 Formalisme et modèles d'exécutions

Un système distribué peut être modélisé par des systèmes de transitions. Nous utilisons les définitions telles qu'elles sont posées par Tel dans [Tel94]. Un système de transitions consiste en l'ensemble de tous les états possibles du système. Afin d'éviter toute confusion entre les états d'un seul processeur et les états du système (états globaux), nous appellerons configurations les états du système.

Définition 2.1.1. Un *système de transition* S est un triplet $\{C, \longrightarrow, I\}$ où :
- C est un ensemble de configurations.
- $\longrightarrow$ est une relation de transition binaire sur C.
- I est un sous-ensemble de C représentant les configurations initiales.

Définition 2.1.2. Une *exécution* est une séquence maximale $E = (\gamma_1, \gamma_2, \dots)$ où $\gamma_0 \in I$ et $\forall i, \gamma_i \longrightarrow \gamma_{i+1}$.

Définition 2.1.3. Une *configuration* $\gamma \in C$ est dite terminale s'il n'existe pas de configuration α telle que $\gamma \longrightarrow \alpha$.

Définition 2.1.4. Un *système distribué* peut être représenté par un graphe non-orienté connexe $G = (V, E)$, où V représente l'ensemble des processeurs exécutant un algorithme distribué donné et E représente l'ensemble des liens de communication bidirectionnels reliant les processeurs.

Définition 2.1.5. Un *algorithme* est dit *permanent* s'il n'existe pas de configuration terminale.

Un algorithme peut alors se définir via ses configurations et ses relations de transition.

2.1.4 Caractéristiques des systèmes distribués

Anonymat

Une première caractérisation est faite suivant l'anonymat des processeurs. Ainsi, un système distribué est dit :

- *Anonyme*, si tous les processeurs sont identiques et incapables de se distinguer des autres processeurs de même degré de façon déterministe. Dans ce cas, chaque processeur identifie ses liens de communications par une numérotation locale, ces numéros sont appelé numéros de port.
- *Non anonyme*, si les processeurs peuvent se distinguer (en utilisant des identifiants par exemple).

Uniformité

La deuxième caractérisation est faite par rapport au type d'algorithme distribué exécuté sur un processeur. Ainsi, un algorithme distribué est dit :

- *Uniforme*, si tous les processeurs exécutent le même algorithme.
- *Semi-uniforme*, si tous les processeurs, sauf un, exécutent le même algorithme (par exemple un nœud désigné comme racine exécute seulement une partie de l'algorithme).
- *Non-uniforme*, si au moins un processeur n'exécute pas le même algorithme.

Connaissance partielle

Du fait que les systèmes distribués possèdent deux niveaux d'abstraction bien distincts (celui des processus et celui du système tout entier), nous avons besoin de notions relatives à ces niveaux.

La notion d'*état* local est utilisée pour désigner l'état d'un processus. L'état global du système, appelé *configuration*, est la liste des états locaux des processus présents dans le système.

De même, le terme d'action est utilisé pour désigner un pas d'évolution d'un processus, et nous utilisons le terme de transition (correspondant à ce qu'un (ou plusieurs) processus effectuent (simultanément) une action) pour désigner un pas dévolution du système. Et ce pour caractériser l'évolution du système.

Synchronisme

La notion de temps dans les systèmes distribués est une notion complexe. Comme nous l'avons dit précédemment, les processeurs sont des unités indépendantes de calcul. Chaque unité possède donc sa propre horloge physique et sa vitesse de calcul qui peuvent être hétérogènes. Avoir une horloge globale est une problématique en soi. De plus, une synchronisation du système doit prendre en compte les temps de communication. Un système « idéal » serait un système *synchrone*, autrement dit tous les processeurs auraient une horloge identique, une vitesse d'exécution identique et les communications se feraient toujours à la même vitesse. Un tel système n'existe pas, et nous caractérisons souvent les systèmes distribués par l'absence de mémoire et d'horloge commune.

Les temps de communication varient souvent au sein d'un même système distribué, dans le meilleur des cas nous connaîtrons une borne sur ce temps (réseau faiblement synchrone). Les algorithmes distribués doivent donc faire abstraction de ces différences. Un système a un comportement asynchrone, lorsque tout message envoyé arrivera à son destinataire en un temps fini mais non borné. Cela n'a pas de sens pour le modèle de communication par registres puisqu'aucun message n'est échangé. En revanche, l'exécution faite par les unités est asynchrone, c'est-à-dire qu'aucune horloge globale n'est disponible.

Fiabilité

Un système distribué a le potentiel inhérent à assurer une fiabilité accrue en raison de la possibilité de reproduire les ressources et les exécutions, ainsi que la réalité que les ressources géographiquement distribuées ne sont pas susceptibles de tomber en panne/dysfonctionnement en même temps dans des circonstances normales. La fiabilité comporte plusieurs aspects :

- la *disponibilité*, c'est-à-dire, la ressource devrait être accessible en tout temps,
- l'*intégrité*, c'est-à-dire, la valeur/état de la ressource devrait être correcte, face à l'accès simultané de plusieurs processeurs, selon les sémantiques attendues par l'application,
- la *tolérance aux pannes*, c'est-à-dire, la capacité pour se remettre de défaillances du système, lorsque ces échecs peuvent être définis à se produire dans l'un des nombreux modèles de défaillance.

Augmentation du rapport performance/coût

Le rapport performance/coût est augmenté par le partage des ressources et l'accès aux données et ressources géographiquement éloignées. Bien que le débit plus élevé n'a

pas toujours été le principal objectif de l'utilisation d'un système distribué, néanmoins, une tâche peut être partagée entre les différents ordinateurs du système distribué. Une telle configuration offre un rapport performance/coût mieux que d'utiliser des machines parallèles spéciales.

Indépendamment de ses caractéristiques, une autre composante importante d'un système distribué est la façon dont les processeurs communiquent entre-eux. Il nous faut donc aussi modéliser la méthode de communication des processeurs.

2.1.5 Modes de communication

La nature des systèmes distribués, c'est-à-dire que chaque processeur n'a qu'une connaissance limitée, impose que chaque processeur communique avec les autres processeurs du système. Cela permet aux processeurs de compléter les connaissances dont ils disposent et de résoudre ainsi la tâche qui leurs a été confiée. Dans la littérature, deux modèles de communications sont couramment utilisés : la communication par *registres* et par *échange de messages*.

Communication par registres

Le modèle de communication par registres a été introduit par *Dijkstra* [Dij72]. Dans ce modèle, chaque processeur communique avec les processeurs voisins à l'aide d'emplacements mémoire partagées, appelés registres. Cela permet à chaque processeur de lire l'état de tous ses processeurs voisins. Plus précisément, un processeur P_i a accès à deux types de registres avec chacun de ses voisins P_j : un registre en lecture r_{ji} (P_i ne peut que lire la valeur de ce registre mais pas la modifier) et un registre en écriture w_{ji} (P_i peut lire et modifier la valeur de ce registre). Ainsi, chaque processeur P_i utilise les registres en écriture r_{ij} qu'il modifie pour communiquer les nouvelles valeurs de ses variables aux processeurs voisins P_j.

Communication par échange de messages

Le modèle de communication par passage de messages (ou simplement par messages) est le modèle couramment utilisé dans les systèmes distribués. Celui-ci est beaucoup plus proche de la réalité que le modèle de communication par registres. Dans ce modèle, un processeur communique avec les processeurs voisins par l'envoie et la réception de messages à travers les canaux (ou liens) de communication adjacents. Nous ferons l'hypothèse que les canaux de communication délivrent les messages suivant l'ordre d'émission de ceux-ci, c'est-à-dire qu'ils ont la propriété d'être FIFO (*First-In First-Out*). De plus, les canaux de

communication peuvent être unidirectionnels ou bidirectionnels.

Dans ce modèle, il existe plusieurs modes de communications. Nous citons que les deux modes suivants :

- *Point-à-point* : Chaque processeur ne peut communiquer directement qu'avec un certain nombre de processeurs, ses voisins. Nous modélisons de tels systèmes naturellement par un graphe connexe. Les sommets sont les processeurs, et les arêtes les liens directs de communication. Nous supposerons que le graphe est non-orienté et sans multi-arêtes.
- *Réseau de diffusion* (broadcast/radio networks). Chaque processeur peut communiquer le même message simultanément à un certain nombre de récepteurs, comme un émetteur radio le ferait. Là encore il y a de nombreuses variantes suivant que les réceptions multiples en un processeur créent ou non des collisions et si les collisions sont détectables en tant que telles.

2.1.6 Quelques problèmes soulevés par les systèmes distribués

Les systèmes distribués apportent d'une part des solution nouvelle ou parfois plus rapide à de nombreux problèmes et soulèvent d'autres par de nouvelles questions. Les systèmes distribués ont besoin de méthodes particulières pour être étudiés car ils sont relativement différents des systèmes classiques.

Tout d'abord, il est courant qu'en algorithmique distribuée ne connaisse qu'une partie de l'état du système, contrairement au cas classique où nous avons une connaissance global de celui ci. Cette hypothèse est tout à fait raisonnable. En effet, nous imaginons mal, dans un système comportant de nombreux processus, que chacun d'entre eux maintienne une connaissance globale du système. Cela demanderait une place mémoire trop importante avec beaucoup d'échange de messages. Dans le cas distribué, nous pouvons donc réussir à accomplir des tâche en n'ayant qu'une connaissance partielle du système.

Lorsque nous voulons dans le cas de l'exclusion mutuelle, s'assurer qu'un seul processus du réseau possède un certain privilège, la difficulté vient justement du fait qu'il est impossible, pour n'importe quel processus, de savoir si, à l'autre bout du réseau, un autre processus possède ou non un privilège.

Un autre point important est l'absence du temps global sur le système. Chaque processus autonome peut à priori évoluer selon une échelle de temps qui lui est propre. Ainsi on n'a pas vraiment de contrôle permettant de savoir dans quel ordre les processus vont agir. Certains événements peuvent même avoir lieu de manière simultanée.

Un autre problème majeur concerne le coût d'une communication. En effet, une solution

évidente à un problème tel que l'acheminement d'une information d'un site i vers un site j serait d'envoyer cette information à tous les sites voisins (mécanisme d'inondation). A la première réception de cette information, un site la renvoie à tous ses voisins excepté celui qui lui a transmis. Ainsi de proche en proche, le site j finit par recevoir cette information. Le coût de cette communication est donc $O(m)$ messages (m étant le nombre de liens de communication du réseau). Il est pourtant possible, en sélectionnant le plus court chemin entre i et j d'acheminer cette information avec un coût inférieur ou égal à $n-1$ messages (la longueur du plus grand chemin simple entre i et j, n étant le nombre de sites du réseau). Or dans un système distribué, les sites s'échangent en permanence des informations et l'utilisation d'une stratégie d'acheminement de messages inadaptée surchargera alors considérablement la bande passante du réseau et conduira inévitablement a une baisse importante des performances du système.

L'exécution d'un algorithme distribué est non-déterministe : l'ordre de réception des messages et la disparité des délais de communication sont sources d'aléas. Si deux sites i et j envoient au même moment un message au site k, l'ordre de réception de ces deux messages peut modifier le comportement du site k et donc l'exécution future de l'algorithme. Même si deux exécutions consécutives d'un algorithme sur un même réseau fournissent un résultat satisfaisant les spécifications du problème, elles peuvent donc fournir des résultats différents.

Enfin, l'ensemble des sites composant le système distribué n'ont pas la même perception du temps : il n'existe pas d'horloge globale dans un système distribué. Chaque site ne peut évaluer l'état d'avancement des autres sites que par le biais des communications qu'il a eu avec ces autres sites, communications qui sont asynchrones. Néanmoins, nombreuses solutions considèrent l'existence d'une horloge globale ou au moins une communication synchrone (c'est-à-dire que le délai de communication est borné).

L'étude des systèmes distribués consiste donc à étudier d'une part l'architecture de communication entre les différents sites et d'autre part à concevoir des algorithmes distribués et à analyser leurs performances.

2.2 Algorithmique distribuée

2.2.1 Quelques problématiques de l'algorithmique distribuée

Il apparaît que certains problèmes naturels sur les réseaux distribués sont difficiles à résoudre et n'admettent parfois aucune solution. Nous allons en fournir quelques exemples et voir comment leur "difficulté" peut être modulée. Il y a en particulier les problèmes de type

"rupture de symétrie" dont le problème d'élection est le plus significatif : *étant donné une certaine situation initiale, est-il possible de distinguer un nœud unique [IR90] [BSV+96] ?*

Pour lors, il apparaît que dans ce cas, si le réseau a de plus une topologie "symétrique" le problème n'a pas de solution (élection dans un anneau anonyme). Nous y reviendrons dans la section suivante. La difficulté de ces problèmes peut être modulée en fonction de connaissances supplémentaires sur la structure du réseau que peuvent avoir les processus.

Par ailleurs et dans le reste de ce document, nous sommes amenés à introduire et à analyser des algorithmes probabilistes et à analyser la probabilité d'obtention d'une certaine solution. Néanmoins, l'existence d'algorithmes permettant de résoudre certains problèmes dépend des hypothèses faites sur le réseau ou de la connaissance que chaque processus a sur le réseau, par exemple : réseau synchrone ou asynchrone, réseau anonyme ou non, connaissance de la topologie, de la taille ou une borne de celle-ci.

Symétrie et impossibilité de l'élection

Depuis le travail original d'Angluin [Ang80], il est connu que certains problèmes comme l'élection n'ont parfois aucune solution pour des raisons de symétrie "interne" au réseau. Un réseau G est "symétrique" s'il existe un autre graphe H, de taille inférieure, tel que G est en quelque sorte constitué de copies de H entrelacées de telle manière qu'un processus ne puisse, à l'aide des informations locales auxquelles il a accès, savoir si le réseau sous-jacent sur lequel il s'exécute est G ou H (il y a une application injective de G dans H). Dans cette configuration, nous disons que G est un revêtement de H (voir FIGURE 5). Un algorithme distribué étant constitué essentiellement d'échanges d'information entre un sommet et ses voisins, il ne peut rassembler que des informations locales, et par conséquent, son exécution sur H peut être similaire et indistinguable, sous un certain sens, à une exécution sur G. Malheureusement, la condition d'Angluin est nécessaire, mais pas suffisante. En effet, dans [BSV+96] les auteurs montrent qu'il existe des graphes qui n'admettent pas un recouvrement, mais pour lesquels il n'est pas possible de trouver un d'algorithme d'élection du chef.

D'ailleurs, si nous cherchons à élire, nous pouvons éventuellement obtenir un seul sommet élu sur H, mais G étant constitué de copies de H, il existe une exécution de l'algorithme sur G, similaire à celle sur H, qui s'achèvera avec de multiples sommets élus. L'algorithme ne pourra donc élire systématiquement sur G. Par conséquent, le problème de l'élection n'a pas de solution sur G.

Par exemple, un certain nombre de problèmes sont plus "faciles" à résoudre si l'on connaît le nombre de sommets dans le réseau.

Il s'avère que le genre d'asymétrie, qui peut être détecté d'une façon distribuée, dépend

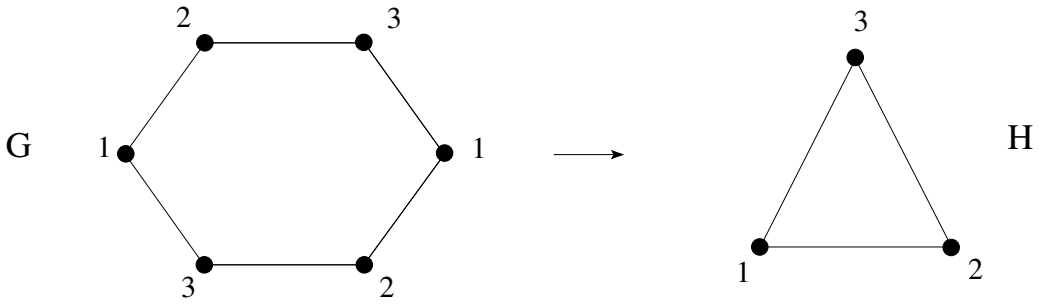


Figure 5 – Le graphe G est un revêtement simple de H

du modèle de calcul utilisé.

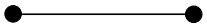


Figure 6 – Graphe K_2

Par exemple, considérons le graphe de la figure 6 (graphe très symétrique).

Initialement les deux sommets sont dans le même état et ils exécutent le même algorithme. Cependant, si l'un d'eux décide d'envoyer un message, forcément, la même décision sera prise par l'autre sommet. Ainsi, les deux sommets s'échangent mutuellement des messages, puis chacun d'eux calcule son nouvel état. Les nouveaux états sont identiques puisque le contenu du message reçu est similaire à celui envoyé au voisin. Par conséquent, les sommets évoluent de la même manière et en aucun temps, ils seront dans des états différents.

Détection de la terminaison

Une autre problématique qui se pose dans l'algorithmique distribuée concerne la détection de la terminaison. Dans le cas séquentiel, lorsqu'un algorithme s'exécute, il lui est facile de constater s'il a terminé sa tâche ou pas. Dans le cas distribué c'est plus délicat. En effet, même si, localement, un processus peut constater qu'il n'a aucune instruction à effectuer, du fait qu'il fonctionne en interaction avec le reste du réseau, il ne peut prédire si, dans le futur, il va recevoir, ou non, un message qui devra provoquer une réaction de sa part. Il ne peut pas *à priori* cesser d'attendre des messages éventuels provenant de ses voisins. Un observateur extérieur peut constater que, globalement, la tâche à réaliser est achevée, le calcul distribué est terminé : tous les sommets sont dans un état où il n'ont et n'auront plus rien à faire. Une telle terminaison est définie comme implicite : le calcul est effectivement terminé mais aucun sommet ne le *sait*. Nous pouvons également prescrire, cela paraît raisonnable et même indispensable dans certains cas, que la terminaison soit détectée localement par un

ou tous les sommets. Nous pouvons exiger la détection de la terminaison locale (le sommet sait qu'il a atteint son état final, qu'il n'effectuera plus aucune opération) ou la terminaison globale (un sommet sait que tous les autres sommets ont atteint leur état final ; il s'agit en général du sommet effectuant le dernier pas de calcul).

Tolérance aux pannes

La capacité d'un système à fonctionner malgré une défaillance d'une de ses composantes est appelée tolérance aux pannes (parfois nommée tolérance aux fautes, en anglais *fault tolerance*).

En présence de pannes, un problème tel que celui du consensus peut se résoudre tout simplement dans un système synchrone. Par contre, le résultat classique de Fischer, Lynch et Paterson [FLP85] montre que, dans un système asynchrone, il est impossible de le résoudre, même en se restreignant à des pannes définitives de processus.

Étant donné qu'il est impossible d'empêcher totalement les pannes, une solution consiste à ce que : lorsqu'une des ressources tombe en panne, les autres ressources prennent le relais afin de laisser le temps aux administrateurs du système de remédier à l'avarie.

2.2.2 Mesures de complexité

Pour résoudre un problème donné, il existe bien souvent plusieurs méthodes et algorithmes distribués, mais bien qu'ils résolvent le problème considéré ceux-ci peuvent avoir des performances radicalement différentes. Ainsi, afin de comparer l'efficacité de plusieurs algorithmes distribués résolvant le même problème, nous considérons le temps que met un algorithme pour résoudre le problème et l'espace mémoire nécessaire sur chaque nœud pour exécuter cet algorithme.

Complexité en messages

La principale mesure utilisée pour évaluer l'efficacité d'un algorithme dans un système distribué asynchrone utilisant le modèle de communication par messages est la complexité en messages. Dans ce type de systèmes, l'exécution des algorithmes est déterminée par les envoies et réceptions de messages. Ainsi, plus le nombre de message est important plus l'efficacité de l'algorithme est faible car celui-ci va utiliser une quantité importante de ressource (bande passante) du réseau. La complexité en messages d'un algorithme distribué est le nombre de messages échangés par l'algorithme jusqu'à la terminaison de celui-ci.

En général pour l'analyse de la complexité en messages d'un algorithme, nous faisons l'hypothèse qu'un message traverse un lien de communication en une unité de temps.

Complexité en temps

Dans les systèmes distribués synchrones utilisant le modèle de communication par registres, la principale mesure d'efficacité n'est pas la complexité en messages mais la complexité en temps. Dans un système synchrone, comme tous les processeurs s'exécutent en même temps par phases, il est donc essentiel de mesurer le nombre de phases nécessaire à un algorithme jusqu'à sa terminaison. De même, pour les systèmes distribués utilisant le modèle de communication par registres comme aucun message n'est échangé, l'efficacité d'un algorithme est le temps mis par celui-ci pour terminer son calcul.

Il existe deux unités pour mesurer la complexité en temps d'un algorithme. Il y a d'une part le nombre de « pas de calcul » et d'autre part le nombre de « round (ou tour) » nécessaire pour résoudre le problème considéré. Bien que le nombre de pas de calcul soit une unité plus fine que le nombre de rounds, bien souvent c'est le nombre de rounds qui est donné pour un algorithme. En effet, la première unité est plus complexe à établir suivant les algorithmes. Nous allons maintenant donner la définition de ces deux unités de mesure pour la complexité en temps d'un algorithme distribué. Nous disons qu'un système distribué effectue une transition lorsqu'un ou plusieurs processeurs réalisent une action (c'est-à-dire exécutent leur algorithme), suite à la réception d'un message ou à la lecture de registres.

Définition 2.2.1. (Pas de calcul). Un pas de calcul est une transition effectuée par le système distribué.

Définition 2.2.2. (Round (ou tour)). Un round (ou tour) est une portion de l'exécution d'un système distribué dans laquelle chaque processeur P_i a exécuté une actions au moins une fois. En d'autres termes, ou bien P_i a exécuté au moins une action, ou bien P_i ne peut exécuter d'actions au moins avant le prochain round.

Complexité en mémoire

La complexité en mémoire est la taille en nombre de bits nécessaires à stocker les variables d'un algorithme exécuté sur chaque processeur du système. Cela comprend aussi la taille des registres stockant les messages échangés.

Les complexités en temps et en mémoire dépendent de paramètres liés aux caractéristiques du système. Généralement, nous rencontrons les ordres de grandeur suivants :

- *Constante* : la complexité calculée ne dépend pas du système.

- *Degré* : la complexité calculée dépend de la taille maximale du voisinage (c'est-à-dire du degré maximum) d'un nœud dans le système.
- *Taille du système* : la complexité calculée dépend de la taille du système (le nombre de processeurs).

2.3 Algorithmes probabilistes

Les algorithmes probabilistes sont des algorithmes qui utilisent des tirages de type pile ou face ou des générateurs de nombres aléatoires.

À la différence des algorithmes *déterministes*, le hasard joue un rôle fondamental dans l'évolution de tels algorithmes.

Les algorithmes probabilistes sont utilisés pour "accroître" la rapidité de la résolution du problème posé ou encore, tout simplement, pour résoudre des problèmes n'admettant pas de solution déterministe.

Un des paradigmes de l'informatique distribuée est le bris de la symétrie des processus en octroyant à l'un d'entre eux un privilège ; être leur *chef*. Il est bien connu qu'une fois qu'un chef est trouvé, la plupart des autres propriétés du réseau puissent être calculées facilement. Par exemple, le chef peut initier le calcul d'un un arbre couvrant, calculer le nombre de sommets dans le réseau, assigner à chacun d'eux une identité unique, et. Réciproquement, si chaque processus a une identité unique, le processus dont la plus petite (ou plus grande) identité peut être désigné pour être le chef.

Sur le plan théorique, le problème d'élection dans un réseau synchrone ou asynchrone est d'ailleurs, par essence, sans solution exacte lorsque les identités des processus sont indiscernables ; il est alors impossible de briser la symétrie structurelle des processus du système. Le papier original dans le domaine est dû à Angluin [Ang80].

Ainsi, le problème ne peut être résolu par un algorithme simplement déterministe. La symétrie est impossible à briser et le problème est sans solution. Seuls des algorithmes probabilistes permettent de contourner cette difficulté en donnant une solution partielle très acceptable au sens probabiliste à ce genre de problèmes.

Élection dans les anneaux

L'élection sur des anneaux a déjà donné lieu à des quantités de recherches, tant dans le cas où les processus sont tous distingués par leurs identités (anneaux avec identités) que dans celui où ils sont sans identités, indiscernables (anneaux anonymes). Nous savons également différencier les classes d'anneaux selon des critères pertinents du point de vue

de la complexité en communication des algorithmes, c'est à dire anneaux synchrones ou asynchrones, uni- ou bidirectionnels, mais aussi anneaux avec ou sans information structurelle (sens général de la direction, connaissance exacte ou approchée de la taille de l'anneau, etc.) : tous types de connaissance à même d'améliorer considérablement la complexité en communication de bien des algorithmes distribués.

Nous avons le théorème suivant dû à Angluin [Ang80] et Itai et al [IR90] :

Théorème 2.3.1. *Sur un anneau anonyme de taille n, aucun protocole* **déterministe** *ne peut résoudre le problème de l'élection sans connaissance de la valeur de n par les processus.*

Afin de briser la symétrie dans un anneau anonyme dont la taille est connue, Itai et Rodeh [IR90] supposent que chaque processus a accès à une suite infinie de nombres réels représentant les probabilités. Évidemment, ceci est fait pour obtenir des preuves plus propres.

Théorème 2.3.2. *Étant donné un paramètre $\varepsilon > 0$ quelconque et quelque soit la taille n de l'anneau, il est possible de déterminer exactement n avec une probabilité d'erreur inférieure à ε grâce à un algorithme probabiliste se terminant par messages (sans détection de la terminaison).*

Caractérisations des algorithmes probabilistes

A cause du "hasard", les algorithmes probabilistes peuvent fournir des solutions qui ne sont pas nécessairement optimales, ou encore ne pas se terminer du tout. Par conséquent, si la probabilité que de tels événements se produisent est faible, il suffit d'exécuter l'algorithme plusieurs fois et nous sommes "presque" sûr d'obtenir le "bon" résultat en un temps "réduit". Ce qui amène à distinguer deux catégories d'algorithmes probabilistes :

- Algorithmes de type *Las Vegas* : ceux sont des algorithmes probabilistes qui
  - ▶ se terminent en un temps dont l'espérance mathématique, noté $\mathbb{E}$, est bornée et,
  - ▶ retournent une solution correcte.

  Formellement, soit $\mathcal{A}$ un algorithme qui traite un problème P et soit $c(\mathcal{A})$ à sa complexité. Nous cherchons en général à avoir $\mathbb{E}[c(\mathcal{A})] < \infty$ (et aussi à le minimiser).

- Algorithmes de type *Monte Carlo* : le nom de ces algorithmes fait allusion aux jeux de hasard pratiqués à Monte-Carlo. Ceux sont des algorithmes probabilistes qui
  - ▶ se terminent en un temps fini et,
  - ▶ retournent une solution correcte avec probabilité $\geq 1/3$.

  Le véritable développement des méthodes de Monte-Carlo s'est effectué, sous l'impulsion de John von Neumann et Ulam notamment, lors de la seconde guerre

mondiale au moment où des recherches sur la fabrication de la bombe atomique sont faites. Notamment, ces chercheurs ont dû utiliser ces méthodes probabilistes pour résoudre des équations aux dérivées partielles.

Nous pouvons noter que le résultat d'un algorithme distribué est un élément de l'ensemble S des solutions possibles ; ceci donne naturellement naissance à des questions du type : étant donné un élément e de S, calculer la probabilité pour que le résultat de l'algorithme distribué soit e. Par ailleurs, dans le chapitre qui suit nous donnerons un modèle général pour caractériser la répartition de probabilités d'être élu pour tous les sommets d'un arbre.

Deuxième partie

Algorithmes distribués probabilistes d'élection et de calcul d'arbre couvrant

Chapitre 3

Élection uniforme dans les graphes à grilles triangulaires

Sommaire

Dans ce chapitre, nous introduisons et analysons un algorithme probabiliste d'élection uniforme dans un réseau qui a pour topologie la structure d'un graphe à grilles triangulaires. Ainsi et dans un premier temps, nous exposons les règles qui permettent de générer de manière distribuée de telle famille de graphes. Nous présentons ensuite les différentes étapes de processus d'élection proposé. Les instructions de ce processus sont exécutées en parallèle par tout nœud "simplicial" du réseau. Dans un second temps, nous analysons analytiquement ce processus. Pour ce faire nous le modélisons par un processus markovien de mort en temps continu. Nous montrons par conséquent que notre algorithme est totalement équitable et ce dans la mesure où tous les sommets ont la même probabilité d'être élus.

Le travail présenté dans ce chapitre a fait l'objet de la publication [SHH13b] et a été présenté comme communication dans [HSH13][SHKH13][HSKD13][SHK12].

Ce chapitre est organisé comme suit : la section 3.1, présente un historique sur le problème d'élection depuis son apparition pour la première fois. Dans la section 3.2, nous rappelons quelques définitions nécessaires pour comprendre le reste du chapitre. Dans la section 3.3, nous donnons un ensemble de règles de construction de la classe de graphes à grilles

triangulaires. Dans la section 3.4, nous exposons notre algorithme distribué probabiliste d'élection uniforme dans cette classe de graphes. Finalement, la section 3.5, nous étudions l'équité de notre algorithme, en fournissant des outils nécessaires pour son analyse.

3.1 Introduction

Le problème de l'élection est de choisir exactement un élément dans l'ensemble de processeurs. Ainsi, à partir d'une configuration où tous les processeurs sont dans le même état, nous devons obtenir une configuration où exactement un processeur est dans l'état *chef* et tous les autres processeurs sont dans l'état *perdant*. Le chef peut être employé plus tard pour prendre des décisions ou pour centraliser des informations.

Ce problème a été posé la première fois par Le Lann [Lan77] et depuis, de nombreux algorithmes pour élire un sommet dans un graphe sont connus [Ray88, Tel00]. Il a été résolu sous plusieurs hypothèses différentes : le graphe peut être un anneau, un arbre, un graphe complet ou en général un graphe connexe. Le système peut être synchrone ou asynchrone. Le graphe peut être orienté ou non orienté. Les processeurs peuvent ou ne peuvent pas connaître la taille du graphe ou une borne de celle-ci. Les communications peuvent être synchrones ou asynchrones.

Il est, par ailleurs connu que certaines classes de graphes n'admettent pas d'algorithmes d'élection distribuée déterministes [God02]. Par conséquent, les algorithmes probabilistes sont apparus pour donner une solution exacte ou approchée à cette problématique.

Dans [MSDZ05], les auteurs ont proposé un algorithme complètement distribué permettant de réaliser une élection équitable dans les arbres : si l'arbre est de taille $n > 0$, alors tous ses sommets ont la même probabilité $\frac{1}{n}$ d'être élus. Par la suite les auteurs ont cherché s'il y a la possibilité d'adapter ledit algorithme à d'autres classes de graphes, ce qui a aboutit à un algorithme donnant la même chance d'être élu à tous les sommets d'un k-arbre Dans [HSdZ+04], après ils ont étendu l'algorithme au cas des polyominoïdes dans [HSZ05] et [HRSZ10]. En suite ils l'ont adapté pour que la probabilité, pour un sommet de l'arbre, d'être élu soit guidée par une loi de probabilité donnée. ce qui a fait l'objet au résultats présentés dans [MSDZ09].

La motivation principale derrière l'étude présentée dans ce chapitre est de présenter et d'analyser un algorithme distribué probabiliste d'élection uniforme dans les graphes à grilles triangulaires. L'algorithme enlève du graphe tout sommet dont la durée de vie est expirée (le graphe restant demeure un graphe à grilles triangulaires). L'analyse de l'algorithme révèle le fait surprenant que, quelque soit l'endroit où le sommet est placé dans le graphe, il a la même probabilité de survivre que les autres, comme dans le cas des arbres, des k-arbres

et des polyominoïdes.

Les réseaux étudiés ici sont asynchrones, les processeurs n'ont pas accès à une horloge globale et tout message envoyé par processeur à un de ses voisins arrive dans un délai fini. Ce délai sera négligeable dans la suite de l'étude afin de simplifier l'analyse. Des étiquettes sont attachées aux sommets et aux arêtes. Elles représentent leurs états.

Nous considérons des *calculs locaux cellulaires* qui, à chaque étape, peuvent modifier l'état (ou l'étiquette) d'un sommet. En effet, la nouvelle étiquette d'un sommet dépend de sa précédente étiquette (état) et des étiquettes de ses voisins. Notre approche se caractérise par l'utilisation d'un délai aléatoire pour l'étiquetage ; un sommet ne peut modifier son état que si le délai d'attente associé à ce sommet est fini. Ces délais sont des variables aléatoires exponentiellement distribuées, définies indépendamment, pour les sommets actifs. Le paramètre d'une variable aléatoire d'un sommet est égal au poids attribué à ce sommet. Le poids est calculé localement en fonction du poids initial et des poids collectés des voisins non élus. Le processus d'étiquetage continue jusqu'à ce que aucune transformation ne soit possible, c'est-à-dire qu'une configuration finale soit atteinte. Dans cette configuration, il y a uniquement un sommet qui a une étiquette différente des autres, ce sommet est considéré comme l'élu.

Le concept d'équité ou de l'équitabilité (en anglais *fairness*) est une propriété importante, voir [DIM95]. Nous pouvons par exemple utiliser cette propriété pour garantir que tous les processus (machines), qui collaborent pour faire un calcul complexe, ont la même quantité de données à traiter.

Le but principal de ce chapitre est de présenter les travaux précédents qui traitent le problème d'élection équitable dans les arbres [MSZ03b], les k-arbres [HSdZ+04] et les polyominoïdes [HRSZ10]. Ainsi, nous présentons des algorithmes probabilistes fondés sur l'élimination successive des sommets simpliciaux qui pour chaque type desdits graphes. Pour chaque type de ces graphes, le processus d'élimination supprime les sommets simpliciaux un par un jusqu'à ce qu'il ne reste qu'un seul. Ce dernier étant considéré comme l'élu. Du moment où ce processus est réitéré plusieurs fois, tout sommet aura la même chance d'être élu.

3.2 Définitions et notations

Un graphe à grilles triangulaires (GGT) est un graphe fini dont les sommets sont des points de $\mathcal{Z} = \mathbb{Z} \times \mathbb{Z}$ où $\mathbb{Z}$ dénote l'ensemble des entiers relatifs. Les sommets sont liés par une relation de voisinage, comme présenté dans [BDFK95].

Les arêtes sont des liens entre les paires de points, c'est-à-dire l'ensemble de paires de points de formes : $\{(x,y),(x,y+1)\}$ ou $\{(x,y),\ (x+1,y)\}$ ou bien $\{(x,y),(x+1,y-1)\}$ pour tout $x \in \mathbb{Z}$ et pour tout $y \in \mathbb{Z}$ (voir FIGURE 8).

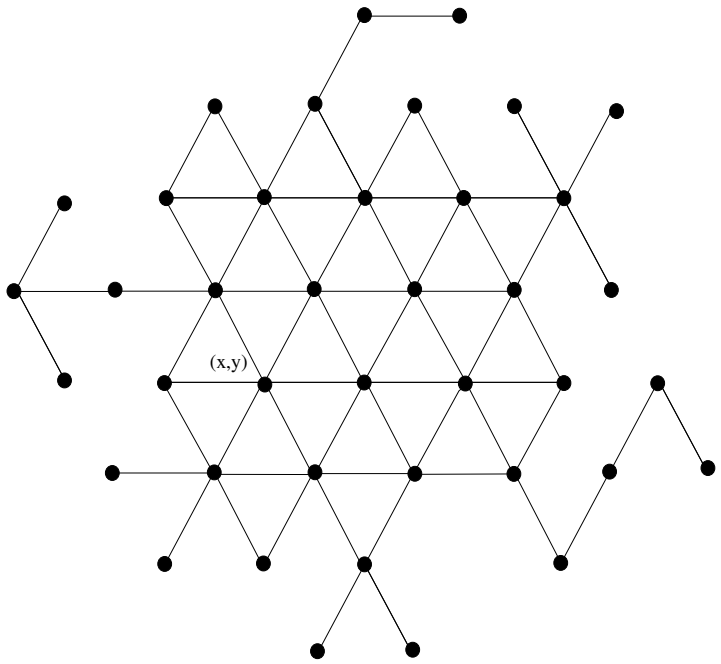


FIGURE 7 – Exemple de graphe à grilles triangulaires

Tout sommet d'une arête e est appelé *extrémité* de e. Pour chaque sommet v de coordonnées (x,y), nous utilisons les termes habituels tels que « up » pour désigner le voisin de coordonnées $(x,y+1)$, « down » pour le voisin $(x+1,y-1)$, « right » pour $(x+1,y)$ et « left » pour $(x-1,y)$. Pour le sommet (x,y) de la figure 7, la figure 8 illustre ses voisins.

Soit $\mathcal{S}_\mathcal{E}$ l'ensemble de toutes les arêtes dont les extremités sont voisins (selon la définition de voisinage ci-dessous) et soit $\mathcal{I}_\mathcal{G}$=$(\mathcal{Z},\mathcal{S}_\mathcal{E})$ le graphe infini constitué de l'ensemble des sommets $\mathcal{Z}$ et l'ensemble des arêtes $\mathcal{S}_\mathcal{E}$.

Une cellule est un sous-graphe de $\mathcal{S}_\mathcal{E}$, induite par un ensemble de trois paires des sommets voisins. Ces sommets voisins ne peuvent être que sous les formes suivantes :

- La forme $\triangle$, appelée aussi *cellule « up » de triangle*. Les sommets voisins ont pour cordonnées : (x,y), $(x+1,y)$ et $(x,y+1)$.
- La forme $\triangledown$, appelée aussi *cellule « down » de triangle*. Les sommets voisins ont pour cordonnées : (x,y), $(x+1,y)$ et $(x+1,y-1)$.

Nous rappelons qu'un chemin est une suite alternée finie $\sigma = v_0, e_1, v_1, \cdots, v_{k-1}, e_k, v_k$ de $k+1$ sommets et de k arêtes distinctes ($k \geq 0$), tel que chaque arête e_i a comme extrémités v_{i-1} et v_i.

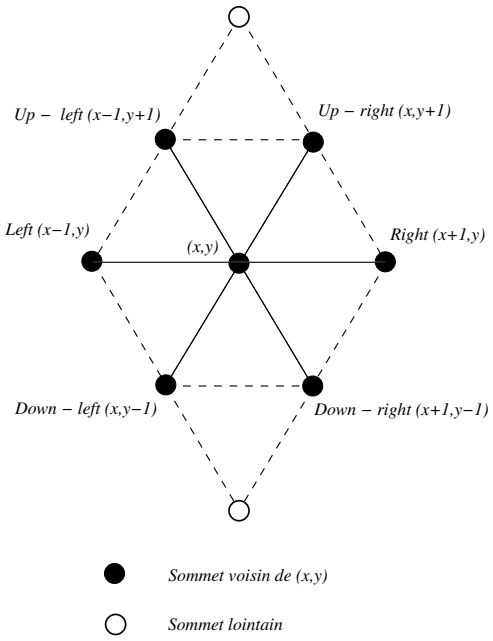


Figure 8 – Sommets voisins dans un graphe à grilles triangulaires

La longueur d'un chemin σ est le nombre de ses arêtes k. Il convient de noter qu'un chemin peut passer plusieurs fois par un sommet, alors qu'il ne peut emprunter qu'une seule fois une arête.

Dans un GGT, un *cycle* est un chemin de longueur $k \geq 4$ dans lequel le premier sommet v_0 et le dernier v_k coïncident.

Par ailleurs, étant donné un cycle γ dans un graphe, Robert Sedgewick dans [Sed92] explique comment définir ses sommets intérieurs. Nous empruntons le même concept et nous définissons les sommets intérieurs d'un cycle dans un GGT comme suit :

Définition 3.2.1. Soit γ=(x_0, y_0), (x_1, y_1), $\cdots$, (x_{k-1}, y_{k-1}), (x_0, y_0), $((x_k, y_k) = (x_0, y_0))$ un cycle dans un GGT, un sommet (x, y) est à l'intérieur du cycle γ si *card* $\{i \mid y = y_i$ *et* $y \neq y_{i+1}$ *et* $x \leq x_i\}$ est impair.

Selon cette définition, les sommets de γ sont à l'intérieur du cycle γ.

Pour illustrer cette définition, nous donnons l'exemple suivant. Étant donné le GGT présenté dans la figure 9. Considérons le cycle constitué par les sommets blancs et les arêtes pointillées. Dans ce cycle, nous avons l'ensemble constitué de sommets $(x + 1, y)$, $(x + 3, y)$ et $(x + 5, y)$ a un cardinal impair. Ainsi, le sommet (x, y) est à l'intérieur de ce cycle.

Définition 3.2.2. Un graphe à grilles triangulaires $G = (V, E)$ est un sous-graphe partiel de $\mathcal{I}_{\mathcal{G}}$ soumis aux conditions suivantes :

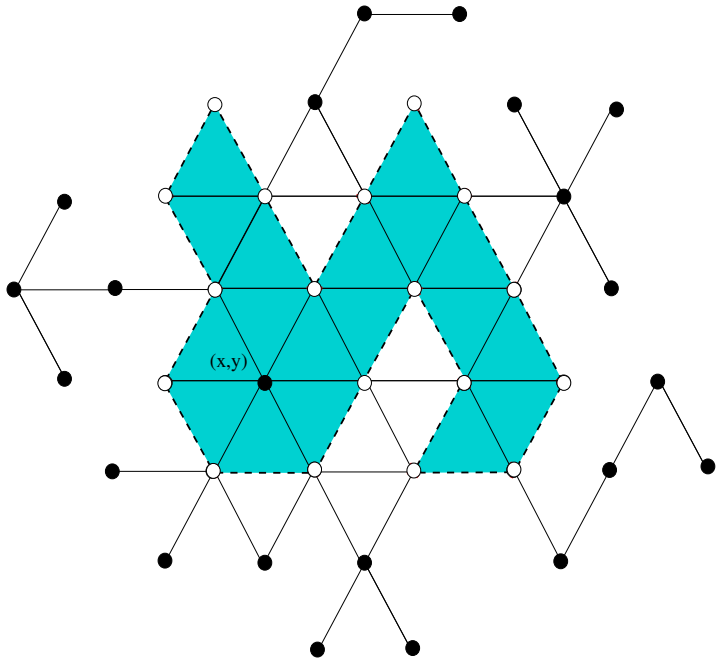


FIGURE 9 – Sommet à l'intérieur d'un cycle

1) V est fini,
2) G est connexe et
3) G ne contient pas de trous, c'est-à-dire pour tout cycle γ dans G, les sommets à l'intérieur de γ sont contenus dans V et si deux sommets voisins sont à l'intérieurs de γ, alors l'arête qui relie ces deux sommets est dans E.

La taille de $G = (V, E)$ est le cardinal de l'ensemble V.

Un pavage (tiling en anglais) [BNRR95] est une façon de remplir un espace à l'aide d'un motif répétitif, sans trou ni débordement. Par conséquent, il est facile de voir que la dernière propriété est équivalente à la propriété de pavage de G ; l'ensemble des sommets internes de γ et leurs arêtes de liaison constituent un sous-graphe d'une grille.

Définition 3.2.3. Un Graphe à grille triangulaire $G_s = (V_s, E_s)$ est appelé sous-graphe du graphe à grilles triangulaires $G = (V, E)$ si et seulement si $V_s \subseteq V$ et $E_s \subseteq E$ tel que $E_s = E \cap \{\{u, v\} |\ \exists! e = (u, v)$, pour tout $(u, v) \in V_s^2\}$.

3.3 Construction distribuée d'un GGT

La classe des GGTs peut être définie par récurrence sur le graphe $\mathcal{I}_{\mathcal{G}}$. En outre, la construction peut se faire d'une manière distribuée.

Soit $\mathcal{S}_{\mathcal{P}}$ l'ensemble des sous-graphes partiels de $\mathcal{I}_{\mathcal{G}}$ obtenu par les règles inductives suivantes :

a) Pour tout $(x, y) \in \mathcal{Z}$, le graphe $G = (\{v = (x, y)\}, \phi)$ est dans $\mathcal{S}_{\mathcal{P}}$.
b) Soit $G = (V, E) \in \mathcal{S}_{\mathcal{P}}$. Considérons deux sommets voisins v et v' tels que $v \in V$ et $v' \notin V$, alors $G' = (V \cup \{v'\}, E \cup \{\{v, v'\}\})$ est dans $\mathcal{S}_{\mathcal{P}}$.
c) Soit $G = (V, E) \in \mathcal{S}_{\mathcal{P}}$. Supposons que V contient trois sommets voisins, ou bien de cordonnées $v_1 = (x, y)$, $v_2 = (x + 1, y)$ et $v_3 = (x, y + 1)$, ou bien de cordonnées $v_1 = (x, y)$, $v_2 = (x + 1, y)$ et $v_3 = (x + 1, y - 1)$ se trouvant dans une cellule de $\mathcal{I}_{\mathcal{G}}$, tel que deux arêtes de cette cellule sont dans E et la troisième, appelée e, ne l'est pas, alors $G' = (V, E \cup \{e\})$ est dans $\mathcal{S}_{\mathcal{P}}$.

La construction est totalement distribuée et l'application des règles de réécriture introduites dans [LMS99] nécessite seulement la connaissance des secteurs de voisins qui sont dans une boule de rayon 2. Il s'agit des transformations modifiant l'étiquetage des sous-graphes de rayon 2. De ce fait, la construction locale peut s'exprimer en considérant des transformations affectant un sommet v et l'ensemble des sommets qui sont à une distance de 2 de ce sommet (c'est-à-dire ses voisins et les voisins de ses voisins). Ainsi, la famille des GGT peut être engendrée par une grammaire algébrique similaire à une grammaire hors-contexte.

À ce stade, il est difficile de prouver que l'ensemble $\mathcal{S}_{\mathcal{P}}$ est la classe de tous les GGT sur $\mathcal{I}_{\mathcal{G}}$. La proposition suivante montre l'équivalence des deux définitions.

Proposition 3.3.1. *Un sous-graphe partiel $G = (V, E)$ de $\mathcal{I}_{\mathcal{G}}$ est un GGT si et seulement s'il appartient à $\mathcal{S}_{\mathcal{P}}$.*

Preuve.

$\Longleftarrow$ Supposons que $G = (V, E) \in \mathcal{S}_{\mathcal{P}}$ et prouvons que G est un GGT. Il est claire que le graphe défini par la règle (a) ci-dessus est un GGT. Par conséquent, il suffit de prouver que les constructions données par les règles (b) et (c) préservent la structure des GGT. Supposons que G est un GGT et montrons que le sous-graphe G'_b obtenu par (b) et le sous-graphe $\mathcal{I}_{\mathcal{G}}$ obtenu par (c) sont aussi des GGT. Les propriétés de la connexité et de la finité sont évidentes. Il ne reste de montrer qu'aucun trou n'est créé lors de l'application de la règle (b) ou de la règle (c).

— En appliquant la règle (b), un nouveau sommet v' est ajouté à G. Comme v' est de degré 1, il n'y a aucun nouveau cycle dans G'_b et tous les sommets à l'intérieur d'un cycle dans G demeurent à l'intérieur du même cycle dans G'_b. Évidemment, le même fait est vérifié pour toute arête dont les extrémités sont dans G'_b.
— Soit $G'_c = (V, E'_c)$ est le graphe obtenu à partir du G par l'application de la règle (c). Soit v un sommet à l'intérieur d'un cycle γ dans G'_c. Si toutes les arêtes

sont dans E, alors v devrait être dans V. Autrement, γ emploie une arête d'une cellule constituée par l'ensemble de sommets $\mathbf{S} = \{v_1, v_2, v_3\}$. Soit $\{v_1, v_2\}$ cette arête. L'arête $\{v_1, v_2\}$ n'appartient pas à E, de plus nous avons $E'_c = E \cup \{v_1, v_2\}$. Dans ce cas, il est possible de transformer γ en un autre cycle γ' inclus dans E en évitant v_1 et ceci en empruntant d'autres sommets de $\mathbf{S}$.

$\Longrightarrow$ Soit $G = (V, E)$ un GGT. Montrons par une récurrence sur le cardinal de l'ensemble V que $G \in \mathcal{S}_{\mathcal{P}}$.

Si V est de cardinal 1, alors $G \in \mathcal{S}_{\mathcal{P}}$. Supposons maintenant que tout graphe G de taille[1] $n \geq 2$ est un GGT, alors $G \in \mathcal{S}_{\mathcal{P}}$. Soit $G' = (V', E')$ un GGT de taille $n + 1$. Si G' possède un sommet v de degré 1, lors sa suppression et de et de celle de son arête incidente, G' est transformé en G. Il est claire que G préserve les propriétés (1)-(3) et par conséquent, selon l'hypothèse de récurrence, il appartient à $\mathcal{S}_{\mathcal{P}}$. Une application de la règle (b) permet d'affirmer que G' est aussi dans $\mathcal{S}_{\mathcal{P}}$.

Supposons maintenant que tout sommet du $G = (V, E)$ est de degré supérieur ou égal à 2. Nous avons $|E| - |V| \geq 0$, sinon, G est un arbre et admet un sommet de degré 1. Nous utilisons maintenant une deuxième récurrence sur $|E| - |V|$. Il est évident de voir que G admet au moins un cycle. Soit γ un *cycle maximal*[2] dans G. Il est facile de voir que si nous supprimons une arête du cycle γ de G, le graphe résiduel obtenu, noté $\mathcal{R}$, préserve les propriétés (1)-(3). Ainsi, selon l'hypothèse de récurrence sur $|E| - |V|$, nous avons : $\mathcal{R} \in \mathcal{S}_{\mathcal{P}}$. Une application de la règle (c) sur $\mathcal{R}$ permet de reconstruire G en tant que membre de $\mathcal{S}_{\mathcal{P}}$.

□

3.4 Algorithme d'élection uniforme dans les graphes à grilles triangulaires

L'algorithme d'élection asynchrone, présenté dans cette section, est conçu pour des réseaux anonymes ayant une topologie de GGT. Chaque sommet connaît seulement les orientations des arêtes qui le relient à ses voisins, mais ne connaît ni la taille du GGT ni ses propres coordonnées dans le plan. La solution dans le cas général consiste à calculer premièrement un arbre couvrant et de lancer deuxièmement l'élection par chaque sommet feuille dans cet arbre.

En utilisant les propriétés des GGT, nous allons concevoir un algorithme distribué qui

1. nombre de sommets
2. cycle qui n'est pas à l'intérieur d'un autre.

permet de choisir uniformément un sommet dans un GGT.

3.4.1 Élection distribuée

L'algorithme d'élection est décrit dans cette sous-section par un système de réécriture de graphe. Les systèmes de réécriture de graphe, ou plus généralement les calculs locaux dans les graphes, sont des modèles puissants qui fournissent des outils généraux pour coder les algorithmes distribués, pour comprendre leur puissance et prouver leur validité. Le lecteur pourra se référer aux références suivantes pour plus de détails [LMS95][LMS99][Sel04][CMZ04].

Notre algorithme distribué est basé sur le système de réécriture présenté dans [LMZ95]. Chaque sommet (resp. arête) possède une étiquette qui représente son état. Rappelons qu'une règle de calcul est définie par un graphe connexe et deux étiquetages de ce graphe, donc elle est appliquée localement. Soit la règle de réécriture $\mathcal{R}$ donnée par un graphe connexe H et deux fonctions de marquages de H [LMZ95] : un étiquetage initial λ et un étiquetage final λ'. Soit G un graphe étiqueté. Si G contient un sous-graphe étiqueté isomorphe avec (H, λ), alors en appliquant la règle $\mathcal{R}$, nous changeons l'étiquetage de ce sous-graphe en λ'.

Nous pouvons alors coder notre algorithme distribué par des règles de réécritures locales. En effet, des étiquettes attachées aux sommets et aux arêtes sont localement modifiées, c'est à dire que les sommets et les arêtes appartiennent à un sous-graphe de rayon fixe 2 dans lequel nous pouvons appliquer une de nos règles. La réécriture est exécutée jusqu'à ce que nous obtenons un graphe irréductible où aucune règle n'est applicable. Initialement, tous les sommets du graphe G ont la même étiquette. Nous cherchons un graphe noethérien [Sel04] réécrivant le système tel que lorsque nous obtenons un graphe étiqueté irréductible, après un certain nombre d'étapes de réécritures, il existe une étiquette spéciale attachée à exactement un seul sommet ; ce sommet sera considéré comme l'élu.

Dans la suite, nous utilisons un système de réécriture de graphe enrichi par des délais aléatoires. Dans tel système, une règle ne peut être appliquée que si le délai correspondant est expiré. Nous définissons des calculs locaux dans un GGT dont les réécritures ne sont qu'une illustration. Ces calculs sont formalisés comme des modifications des étiquettes (ou des états) associées aux sommets et aux arêtes du graphe. Toute modification est réalisée selon les connaissances locales : la nouvelle étiquette attachée à un sommet v (resp. à une arête e) ne dépend que du sous-graphe induit par les sommets qui sont dans la boule de rayon 2. Le système de réécriture de graphes appliqué ici intègre aussi des contextes interdits [LMS92][God02][Sel04]. L'idée est d'empêcher l'application d'une règle de réécriture toutes fois les occurrences correspondantes sont « inclues » dans certaines

configurations spéciales, appelées contextes. Ainsi, une règle peut être appliquée que s'elle ne se produit pas dans un contexte interdit donné et que le délai associé est expiré.

Une *règle de réécriture avec contextes* interdits est un quadruplet $\mathcal{R} = (G_{\mathcal{R}},\lambda_{\mathcal{R}},\lambda'_{\mathcal{R}},\mathcal{F}_{\mathcal{R}})$ tel que $(G_R,\lambda_{\mathcal{R}},\lambda'_{\mathcal{R}})$ est une règle de réécriture et $\mathcal{F}_{\mathcal{R}}$ est un ensemble fini de contextes de $(G_{\mathcal{R}},\lambda_{\mathcal{R}})$. Une règle de réécriture peut être appliquée dans un sous-graphe si et seulement si ce sous-graphe n'est pas inclus dans une occurrence de l'un de ses contextes interdits [LMZ95] [LMS99].

$$\mathcal{R} : \{\mathcal{F}_{\mathcal{R}}; (G_{\mathcal{R}}, \lambda_{\mathcal{R}}) \longrightarrow (G_{\mathcal{R}}, \lambda_{\mathcal{R}'})\}\,.$$

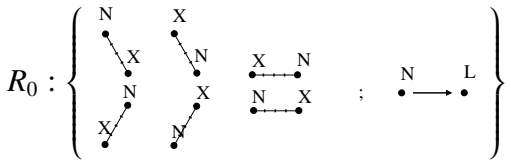
Soient $G = (V, E)$ un graphe à grilles triangulaires et $\mathcal{S}_{\mathcal{L}} = \{N, A, B, L\}$ l'ensemble des étiquettes utilisées. L'étiquette N encode l'état neutre, A encode l'état actif, B encode l'état battu (ou perdant) et L encode l'état élu (leader). Initialement, tout sommet possède un poids $w = 1$ et une étiquette N, nous disons qu'il est $N-$étiqueté. Nous notons par X tout état différent de B, c'est à dire $X \in \mathcal{S}_{\mathcal{L}} \setminus \{B\}$.

L'algorithme s'exécute sur un graphe à grilles triangulaires G de la manière distribuée suivante : tout sommet v de poids $w = 1$, $N-$étiqueté, décide localement s'il est actif ou non selon les règles d'activation ci-dessous.

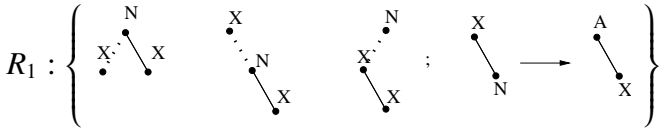
Nous représentons les règles d'activation des sommets par R_i et les règles de transmission de poids par R'_i. Ces règles permettent de changer l'étiquetage d'un sommet de $N-$étiqueté à $A-$étiqueté.

Règles d'activation

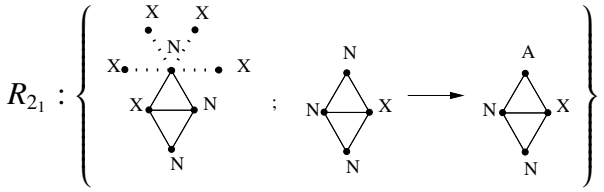
- R_0 : Si le degré de v est nul ($deg(v) = 0$), alors l'élection est terminée et v est le sommet élu. Il est important de noter que ce sommet est considéré comme un sommet *actif*.



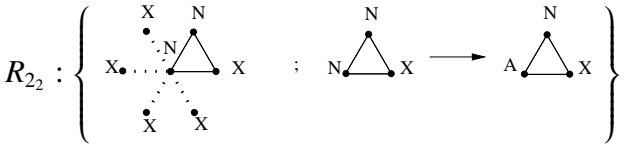
- R_1 : Si le degré de v est 1, alors v devient actif et génère une durée de vie qui est une v.a. exponentielle de paramètre égal à son poids. Une fois sa durée de vie expirée, il disparaît avec l'arête incidente et son voisin récupère son poids.

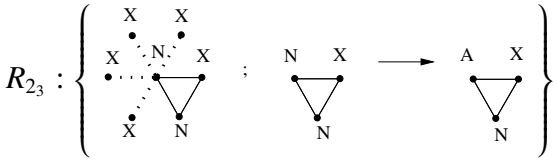


- R_2 : Si le degré de v est 2 alors, il peut devenir actif ou non, rt ce selon sa position. Nous distinguons cinq sous-règles pour que v devient actif.
  - La sous-règle R_{2_1} montre que si $v = (x, y+1)$ est au dessus de la cellule $\{(x,y);(x+1,y);(x,y+1)\}$ en cas de présence d'une autre cellule $\{(x,y);(x+1,y);(x+1,y-1)\}$ en bas de triangle, alors v devient actif.

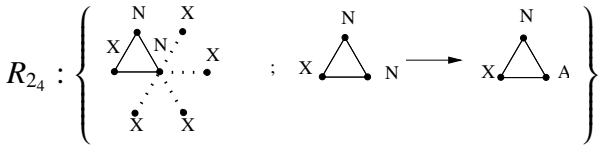


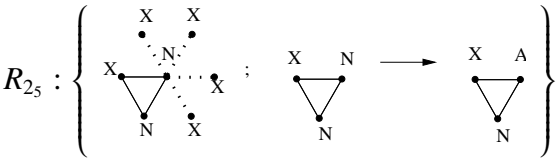
  - La sous-règle R_{2_2} (resp. R_{2_3}) exprime que si $v = (x,y)$ est à gauche d'une cellule de type "up" (resp. "down"), alors v devient actif.





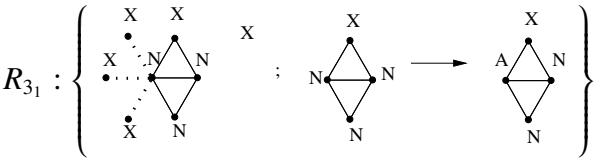
  - La sous-règle R_{2_4} (resp. R_{2_5}) explique que si $v = (x+1,y)$ est à droite d'une cellule de type "up" (resp. "down"), alors v devient actif.





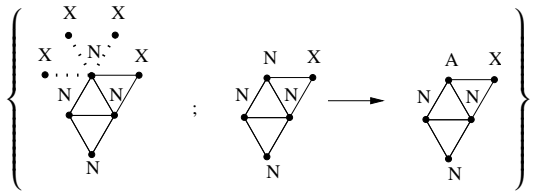
- R_3 : Si $deg(v) = 3$, alors il y a deux cas à étudier :
  - ► La sous-règle R_{3_1} décrit le fait que si v est à gauche des deux cellules de type "up" et "down" qui partagent l'arête horizontale, alors v devient actif.

R_{3_1} :



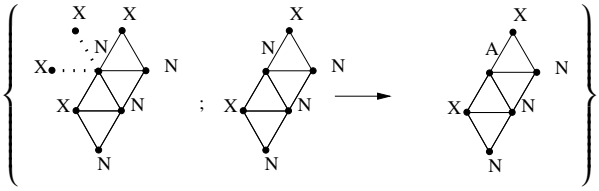
  - ► La sous-règle R_{3_2} explique que si $v = (x, y + 1)$ est situé sur en haut de la cellule ("up") $\{(x, y), (x, y + 1), (x + 1, y)\}$ et à gauche de la cellule ("down") $\{(x, y + 1), (x + 1, y), (x + 1, y + 1)\}$ avec la présence la cellule ("down") $\{(x, y), (x + 1, y), (x + 1, y - 1)\}$, alors v devient actif.

R_{3_2} :



- R_4 : Si $deg(v) = 4$, alors il peut devenir actif si et seulement si $v = (x, y + 1)$ appartient à trois cellules $\{(x, y), (x, y+1), (x+1, y)\}$, $\{(x, y+1), (x, y+2), (x+1, y+1)\}$, et $\{(x, y+1), (x+1, y+1), (x+1, y)\}$ et ce en présence de la cellule $\{(x, y), (x+1, y), (x+1, y-1)\}$ ("down").

R_4 :



Chaque fois qu'un sommet v de poids w_v devient actif, il génère sa durée de vie $L_t(v)$ qui est une variable aléatoire exponentiellement distribuée de paramètre égale au poids.

$$Pr(L_t(v) \geq t) = e^{-w_v(t)}.$$

Cette variable aléatoire est d'espérance $\frac{1}{w_v}$.

Règles de transmission de poids

Une fois que la durée de vie d'un sommet est expirée, ce sommet ne sera plus $A-$étiqueté. Nous disons qu'il est battu ou $B-$étiqueté, et toutes ses arêtes incidentes sont supprimées. Le poids d'un sommet battu sera transmis au voisin dicté par les règles R'_i. Dans ces règles,

d indique la durée de vie du sommet et lorsqu'un sommet devient $B-$étiqueté, toutes les arêtes incidentes sont supprimées. Les arêtes pointillées sont les arêtes à travers lesquelles les poids sont transmis.

Nous allons montrer par la suite que l'ensemble de ces arêtes constituera un arbre couvrant du graphe G.

$$R'_0 : \left\{ \begin{array}{c} \text{A} \\ \bullet \end{array} \xrightarrow{\text{d = 0}} \begin{array}{c} \text{L} \\ \bullet \end{array} \right\}$$

- R'_0 : L'élection est terminée, le sommet restant est considéré comme l'élu.
- R'_1 : Le sommet voisin u de v, ayant récupéré le poids de ce dernier, est soit dans l'état actif, soit dans l'état neutre. Dans ces deux cas, nous avons les deux sous règles suivantes :
  - ► Si u est neutre alors au moment où il récupère le poids de v, il décide localement s'il devient actif dans le graphe résiduel $G' = (V \setminus \{v\}, E \setminus \{\{v,u\},\ u \in V\})$.

$$R'_{1_1} : \left\{ ; \quad \begin{array}{c} A^{(d,w)} \\ \diagdown \\ \text{N}^{(\infty,w')} \end{array} \xrightarrow{\text{d = 0}} \begin{array}{c} \text{B} \\ \vdots \\ \text{N}^{(\infty,w+w')} \end{array} \right\}$$

  - ► Sinon, u est actif au moment de la suppression de v alors u devient l'élu, nous nous retrouvons partiellement dans le cas de la règle R_0.

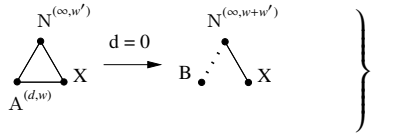
$$R'_{1_2} : \left\{ ; \quad \begin{array}{c} \text{A}^{(d,w)} \\ \diagdown \\ \text{A}^{(d',w')} \end{array} \xrightarrow{\text{d < d'}} \begin{array}{c} \text{B} \\ \vdots \\ \text{L} \end{array} \right\}$$

- R'_2 : Si $deg(v) = 2$, alors il est actif et une fois que sa durée de vie est expirée, il est supprimé avec ses arêtes incidentes. Quatre situations peuvent se présenter :
  - ► Si le sommet actif $v = (x, y+1)$ est situé au dessus de la cellule $T_u = \{(x,y),(x,y+1),(x+1,y)\}$ à condition de l'existence de $T_d = \{(x,y),(x+1,y),(x+1,y-1)\}$, alors le voisin $u = (x+1, y)$ récupère son poids.

$$R'_{2_1} : \left\{ \begin{array}{c} \text{A}^{(d,w)} \\ \text{X} \quad \text{N}^{(\infty,w')} \end{array} \xrightarrow{\text{d = 0}} \begin{array}{c} \text{B}^{(\infty,w+w')} \\ \text{X} \quad \text{N} \end{array} \right\}$$

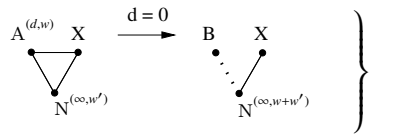
- Si le sommet actif $v = (x, y)$ est le sommet de droite en bas du triangle de type "up" $T_u = \{(x, y), (x, y+1), (x+1, y)\}$, alors son voisin $u = (x, y-1)$ récupère son poids lorsque v est supprimé.

R'_{2_2} :

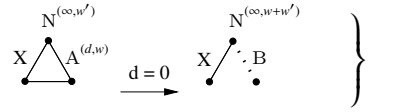


- Concernant R'_{2_3} et R'_{2_4}, la transmission du poids se fait à travers l'arête incidente diagonale du sommet supprimé (pareilles aux règles R'_{2_1} et R'_{2_2}).

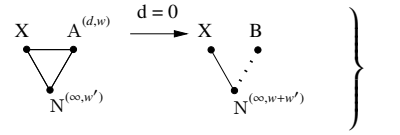
R'_{2_3} :



R'_{2_4} :

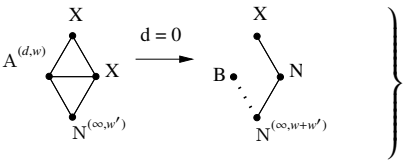


R'_{2_5} :



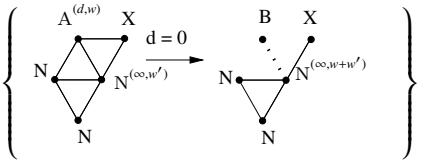
- R'_3 : Si $deg(v) = 3$, alors v est enlevé et son voisin de droite en bas de la cellule collecte son poids. Dans cette règle nous distinguons deux cas :
  - Si le sommet actif $v = (x, y)$ appartient à ces deux paires de cellules $\{(x, y), (x, y+1), (x+1, y)\}$ et $\{(x, y), (x+1, y), (x+1, y-1)\}$, alors v est supprimé quand sa durée de vie est achevée, et son voisin $u = (x+1, y-1)$ récupère son poids.

R'_{3_1} :



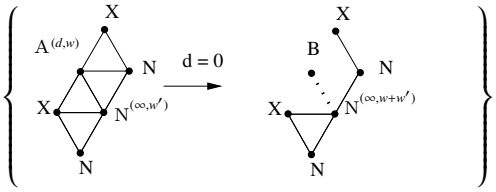
► Si le sommet actif $v = (x, y)$ appartient à ces deux paires de cellules $\{(x, y), (x + 1, y), (x+1, y-1)\}$ et $\{(x, y), (x+1, y-1), (x, y-1)\}$, alors son voisin $u = (x+1, y-1)$ reçoit sont poids, quand sa durée de vie est terminée.

R'_{3_2} :



- R'_4 : Si le sommet actif $v = (x, y)$ appartient aux cellules $(\{(x, y), (x, y + 1), (x + 1, y)\}, \{(x, y), (x + 1, y), (x + 1, y - 1)\}, \{(x, y), (x + 1, y - 1), (x, y - 1)\})$, alors son voisin $u = (x + 1, y - 1)$ récupère son poids, après sa disparition.

R'_4 :



Lemme 3.4.1. *Le système de réécritures ci-dessus est noethérien.*

Validation : La validation de ce système est obtenue par le lemme suivant :

Lemme 3.4.2. *Soit (G, λ) un graphe simple connexe tel que chaque sommet est étiqueté par son identité (un entier unique de $[1, n]$ où n est la taille du graphe), et chaque arêtes est étiquetée 0. Soit (G, λ') un graphe étiqueté tel que : $(G, \lambda) \xrightarrow[\mathcal{R}]{*} (G, \lambda')$. Le graphe (G, λ') satisfait :*

1. *Toutes les arêtes incidentes aux nœuds étiquetés i ont une étiquette α inférieure ou égale à i.*
2. *Tous les sommets étiquetés n sont connectés par des arêtes étiquetées n.*
3. *Le sous-graphe induit par les arêtes étiquetées i n'a pas de cycle $(i > 0)$.*
4. *Si (G, λ') est un graphe irréductible obtenu à partir de (G, λ), alors tous les sommets sont étiquetés n.*

Preuve.

1. Initialement, la propriété est vraie puisque tous les sommets ont leur propre identité comme état et toutes les arêtes sont étiquetées 0. Nous supposons que nous avons appliqué la règle k fois et que la propriété reste vraie. Si nous appliquons la règle

pour la $(k+1)^{ième}$ étape, seul un nœud v étiqueté j et une seule arête e, étiquetée α, vont changer leurs étiquette à i. Puisque par induction, toutes les arêtes incidentes au nœud v ont une étiquette inférieure ou égale à j, et comme $i > j$, alors tous les arêtes, excepté e, ont une étiquette strictement inférieure à i. La nouvelle étiquette de l'arête e est i, qui est égale à l'étiquette des sommets incidents. Ainsi, la propriété est vraie après l'étape de réécriture.

2. La preuve est par induction. Initialement, la propriété est juste car il n'existe qu'un seul sommet étiqueté n. Nous supposons qu'après k applications de la règle la propriété reste vraie. Il existe un sous-graphe H connectant toutes les arêtes et les sommets étiquetés n. Supposons maintenant que nous appliquons la règle pour la $(k+1)^{ième}$ étape sur un sommet v. Si la nouvelle étiquette de v est n, il est évident que dans ce cas, le nœud v a au moins un voisin étiqueté n qui appartient à H et que l'arête les liant sera étiquetée n. Puisque, par induction, tous les sommets de H qui sont étiquetés n sont connectés par des arêtes étiquetées n, le nœud v, avec le nouvel état n, sera connecté à H par une arête étiquetée n. Ainsi, la propriété est vraie.
3. Au début, toutes les arêtes sont étiquetées 0. Nous supposons que la propriété est vraie après k applications de la règle, nous allons montrer qu'elle est toujours vraie après la $(k+1)^{ième}$ étape. Quand nous appliquons la règle pour changer l'étiquette d'un nœud v de j à i, nous changeons l'étiquette de v et de l'arête incidente à i. Pour pouvoir appliquer la règle, l'étiquette de v doit être strictement inférieure à i et par la propriété 1 toutes les étiquettes des arêtes adjacentes à v sont inférieures à i donc aucun cycle n'est créé.
4. Nous supposons qu'il existe un nœud avec l'étiquette p, $p < n$ dans (G, λ). Il existe au moins un sommet avec l'étiquette n. Comme le graphe est connexe, il existe un chemin reliant le sommet étiqueté n au sommet étiqueté p. Ce chemin contient au moins une arête liant un nœud étiqueté n avec un autre nœud étiqueté $k < n$. Dans ce cas, la règle peut être appliquée. Ce qui contredit le fait que (G, λ') est irréductible.

□

Dans la suite, nous avons besoin de la définition suivante.

Définition 3.4.1. Un sommet qui appartient à un cycle maximal dans G est appelé *sommet périphérique.*

Lemme 3.4.3. *Soit v un sommet actif de degré 2, 3 ou 4 dans un graphe à grilles triangulaires G. Alors v est un sommet périphérique.*

Preuve. Soient $G = (V, E)$ un graphe à grilles triangulaires et $v = (x, y)$, $v \in V$, un sommet actif de degré 2, 3 ou 4. Par définition, v est situé dans une cellule. Soit γ un cycle maximal dans G contenant v à son intérieur. Si $v \in \gamma$, alors la preuve est complète. Sinon, il existe un sommet plus proche $u = (x', y)$ de γ tel que $x' < x$. Or, G est un GGT et tout cycle γ

dans G ne contient pas de trous ; les arêtes du segment $[(x', y), (x, y)]$ sont dans E. Ceci ne peut pas être vrai puisque v n'admet aucune arête à sa gauche. □

3.4.2 Propriétés invariantes

L'algorithme d'élection proposé ci-dessus enlève un sommet actif une fois sa vie expirée. Ainsi, la question que nous pouvons nous poser est : "*comment assurer la continuité du processus de suppression ?*"

Pour répondre à cette question nous devons prouver que le graphe résiduel préserve les propriétés propres aux graphes à grilles triangulaires.

Proposition 3.4.1. *Soit $G = (V, E)$ un graphe à grilles triangulaires de taille ≥ 2 et soit v un sommet actif dans G. Le graphe $G' = (V \setminus \{v\}, E \setminus \{\{v, u\}, \forall\ u \in V\})$ est un graphe à grilles triangulaires.*

Preuve. Soient $G = (V, E)$ un graphe à grilles triangulaires de taille ≥ 2, v un sommet actif dans G et le graphe $G' = (V \setminus \{v\}, E \setminus \{\{v, u\}, \forall\ u \in V\})$. Pour montrer la proposition, nous devons montrer que G' est un graphe connexe.

- Si $deg(v) = 1$, alors la suppression de v et son arête incidente dans G n'introduit ni la déconnexion de G' ni la création d'un trou dans G'.

- Si $deg(v) = 2$ ou $deg(v) = 3$ alors soient v, v_1, v_2 trois sommets du G tel que v est un sommet actif dont la durée de vie vient d'expirer (nous sommes dans le cas des règles R_k et R'_k ,$k = 1, 2, 3$). Considérons un sommet $u \in V \setminus \{v, v_1, v_2\}$. Alors, si le sommet u est accessible aux sommets v_1, v_2, v_3, à travers un chemin qui passe par v, alors lorsque v sera supprimé, u restera accessible à v_i par un autre chemin empruntant les sommets $v_{j\neq i,\ j=1,2}$. Par conséquent, selon le lemme 3.4.3, v est un sommet périphérique et sa suppression ne crée aucun trou.

□

3.4.3 Arbre couvrant standard

Soit $G = (V, E)$ un graphe à grilles triangulaires. Le graphe $T = (V, F)$ constitué par les arêtes sur lesquelles les poids sont transmis, pourra aussi être généré d'une façon distribuée

comme suit :

— Si $e = \{(x, y), (x + 1, y - 1)\}$ est une arête dans E alors e appartient à F, c'est-à-dire que toute arête horizontale dans E appartient à F.
— Si $e = \{(x + 1, y - 1), (x + 1, y)\} \in E$ et appartient au cycle $\lambda = (x, y), (x, y + 1), (x + 1, y), (x + 1, y - 1)$, alors $e \in F$.

Le graphe $T = (V, F)$ relie tous les sommets de G est connexe et sans cycle. Il est donc un arbre.

Proposition 3.4.2. *Le graphe $T = (V, F)$ décrit ci-dessus est un arbre couvrant du G.*

Preuve. Nous pouvons prouver cette proposition par une construction inductive de T sur G :

1. Si $G = (\{(x, y)\}, \emptyset)$, le graphe à grilles triangulaires se compose de seulement un sommet, alors la proposition est affirmée ($T = (\{(x, y)\}, \emptyset)$).
2. Soit $G = (V, E)$ un graphe à grilles triangulaires et soit $T = (V, F \subseteq E)$ l'arbre couvrant de G obtenu par les règles ci-dessus. Considérons deux sommets voisins v et v' tels que $v \in V$ et $v' \notin V$. Conformément à la règle inductive vue dans la section 3.3, le graphe G'=$(V \cup \{v'\}, E \cup \{\{v, v'\}\})$ est un GGT. À présent, il reste à prouver que l'arbre $T' = (V \cup \{v'\}, F \cup \{\{v, v'\}\})$ est un arbre couvrant de G'.

   Aucun cycle n'est créé lorsque la nouvelle arête $\{v, v'\}$ est ajoutée. Ainsi, le graphe joignant l'arbre couvrant de G et l'arbre $(V_A = \{v, v'\}, E_A = \{\{v, v'\}\})$ est un arbre couvrant du G' (voir FIGURE 10).
3. Soit $G = (V, E)$ un graphe à grilles triangulaires et soit $T = (V, F \subseteq E)$ son arbre couvrant. Supposons maintenant que V contienne 3 sommets voisins de forme $v_1 = (x, y)$, $v_2 = (x + 1, y)$, $v_3 = (x, y + 1)$ ou bien $v_1 = (x, y)$ et $v_2 = (x + 1, y)$ et $v_4 = (x + 1, y - 1)$ tel que les deux arêtes de la cellule sont dans E et la troisième, appelée e, ne l'est pas. Alors conformément à règle inductive de section 3.3, le graphe résiduel $G' = (V, E \cup \{e\})$, après l'insertion de la nouvelle arête, e est un GGT (voir FIGURE 10). Néanmoins, il reste à prouver que la transmission des poids s'effectue à travers l'arbre couvrant $T' = (V, F')$ de G'.

   Soit $C_1 = \{v_1, v_2, v_3\}$ et $C_2 = \{v_1, v_2, v_4\}$ deux cellules de $G' = (V, E \cup \{e\})$ et soit $e_1 = \{v_1, v_2\}, e_2 = \{v_2, v_3\}, e_3 = \{v_2, v_4\}, e_4 = \{v_1, v_3\}$ et $e_5 = \{v_1, v_4\}$ les arêtes sur lesquelles les poids des sommets supprimés sont transmis à leurs successeurs dans G'. Alors, nous avons :
   - Si $e \in \{e_1, e_2, e_3\}$ alors $F' = F$. (Dans ce cas l'arbre couvrant ne change pas). Voir FIGURE 11.
   - Si $e = e_2$ alors $F' = F \backslash \{e_4\} \cup \{e_2\}$ (voir FIGURE 12).
   - Si $e = e_5$ alors $F' = F \backslash \{e_4\} \cup \{e_5\}$.

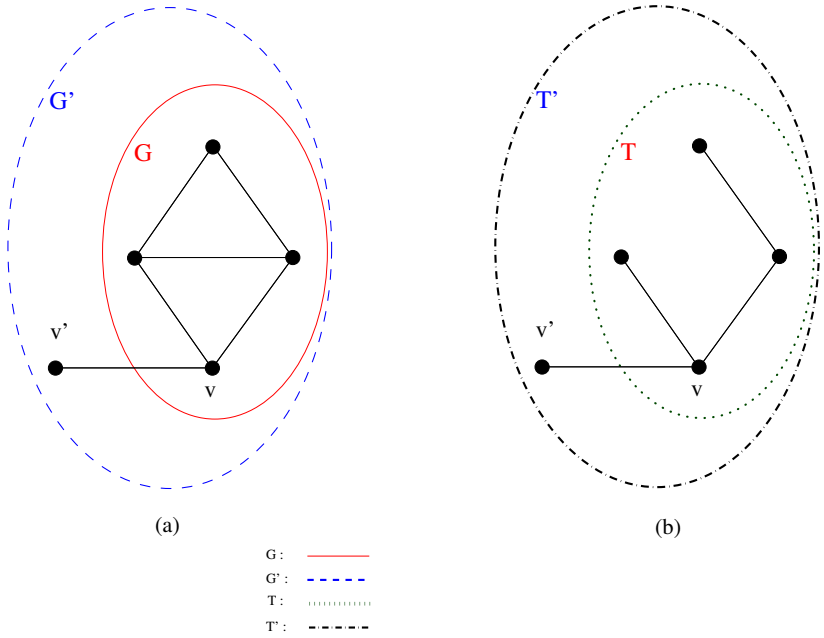


Figure 10 – Arbre couvrant du graphe G après l'insertion d'un sommet v'

Nous pouvons facilement voir que le graphe T' est un graphe connexe et, d'ailleurs, aucun cycle n'est engendré avec ces instructions et il contient tous les sommets de G. Par conséquent, T' est un arbre couvrant de G.

□

Remarque 3.4.1. L'arbre couvrant construit par les règles R_i et R'_i est unique.

Définition 3.4.2. L'arbre couvrant $T = (V, F)$ est appelé arbre couvrant standard du graphe à grilles triangulaires G.

La figure 14 donne l'arbre couvrant standard du graphe à grilles triangulaires donné dans la figure 7.

Proposition 3.4.3. *Soient $G = (V, E)$ un graphe à grilles triangulaires et $T = (V, F)$ son arbre couvrant standard. Alors, un sommet $v \in V$ est actif dans G si et seulement s'il est une feuille*[3] *dans T.*

Preuve.

⇒ Soient six sommets d'un graphe à grilles triangulaires, $G = (V, E)$ définis comme suit :

- $v_1 = (x, y)$
- $v_2 = (x, y + 1)$
- $v_3 = (x + 1, y)$
- $v_4 = (x + 1, y - 1)$
- $v_5 = (x + 1, y + 1)$
- $v_6 = (x, y + 2)$,

3. Un sommet feuille est un sommet de degré 1.

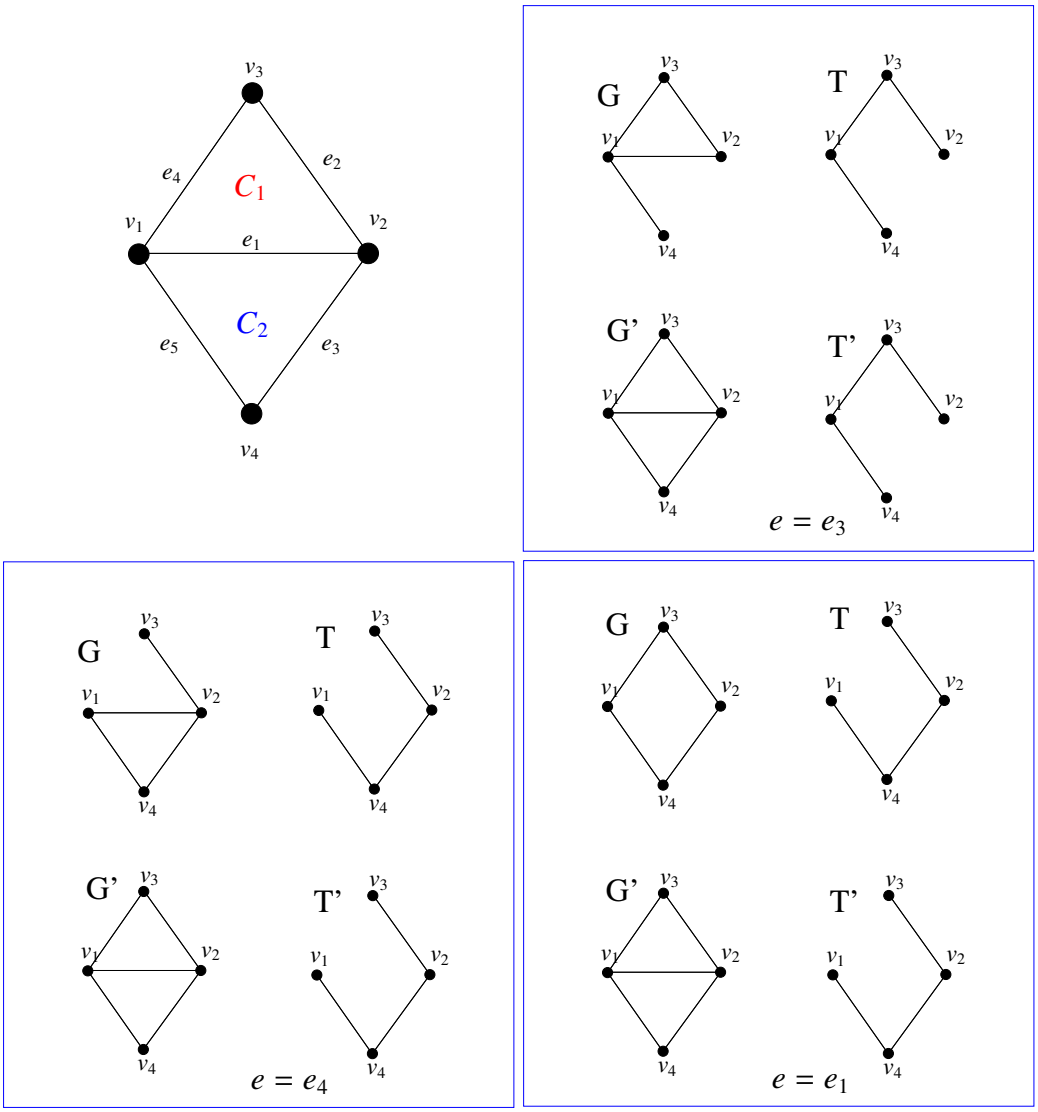


FIGURE 11 – Arbre couvrant du graphe G au cas où $e \in \{e_1, e_2, e_3\}$

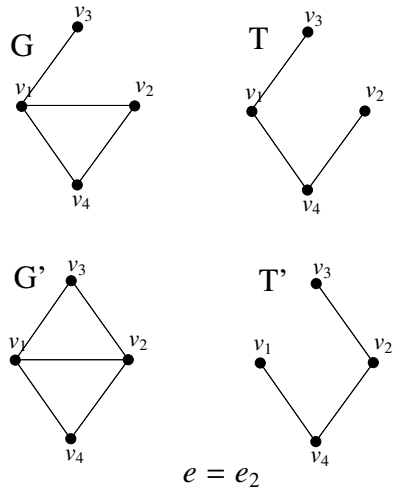


Figure 12 – Arbre couvrant du graphe G au cas où $e = e_2$

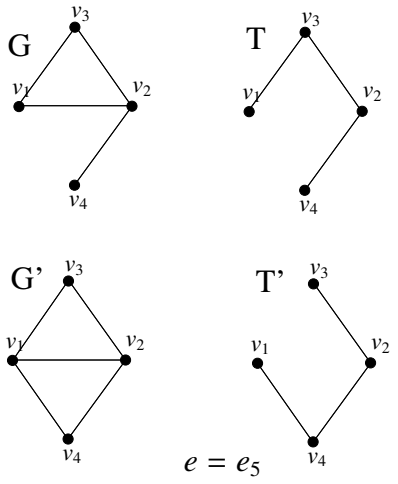


Figure 13 – Arbre couvrant du graphe G au cas où $e = e_5$

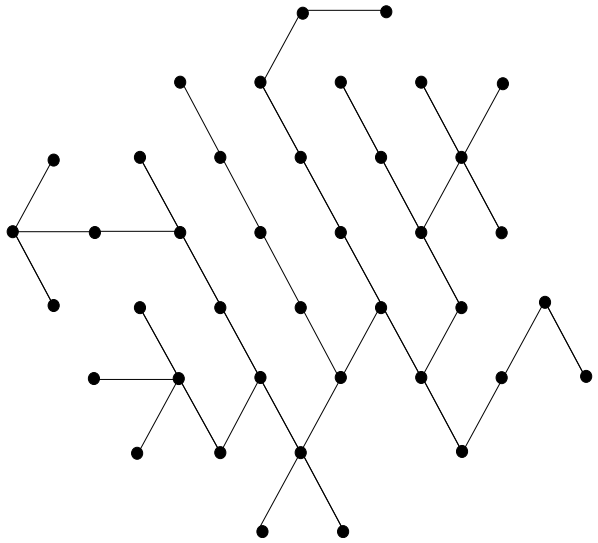


FIGURE 14 – Arbre couvrant standard du graphe donné dans la figure 7

et soient $C_1 = \{v_1, v_2, v_3\}$, $C_2 = \{v_1, v_3, v_4\}$, $C_3 = \{v_2, v_3, v_5\}$ et $C_4 = \{v_2, v_5, v_6\}$ quatre cellules (voir FIGURE 15).

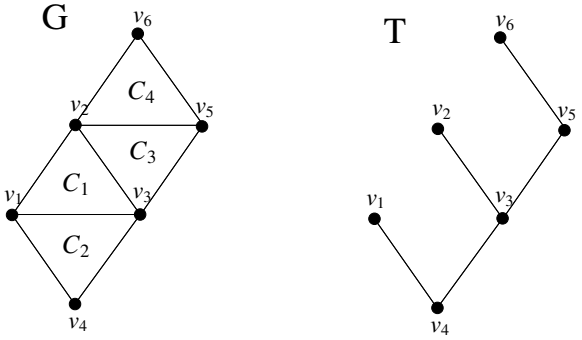


FIGURE 15 – Un graphe G contenant quatre cellules C_1, C_2, C_3 et C_4, avec son arbre couvrant T

- Si $deg(v) = 1$, alors certainement, v est une feuille dans T.
- Si $deg(v) = 2$, alors nous distinguons les cas suivants :
  (i) Si v appartient à la cellule C_1, et C_2 n'existe pas dans G, alors v est une extrémité de l'arête l'horizontale $\{v_1, v_3\}$, et vu que la transmission du poids ne passe pas par de cette arête, donc v est de degré 1 dans T (c'est à dire une feuille).
  (ii) Si v appartient à la cellule C_2, alors v est une extrémité de l'arête horizontale $\{v_1, v_3\}$, et puisque la transmission de poids ne passe pas par cette arête, donc v est une feuille dans T.

(iii) Si $v = v_2$ et les cellules C_1 et C_2 existent dans G, alors v est une des extrémités de l'arête $\{v_2, v_3\}$. Vu que la transmission de poids ne passe pas à travers l'arête $\{v_1, v_2\}$, alors v devient une feuille dans T.

- Si $deg(v) = 3$ et v appartient à la fois à C_1 et à C_2, alors nous avons deux cas :
  (i) Si $v = v_1$, alors la transmission de poids ne passe pas par l'arête $\{v_1, v_2\}$ ni par l'arête $\{v_1, v_3\}$. Donc v devient une feuille dans T.
  (ii) Si $v = v_2$, alors la transmission de poids ne passe pas par les arêtes $\{v_1, v_2\}$ et $\{v_2, v_5\}$. Donc devient une feuille dans T.
- Si $deg(v) = 4$, et v appartient à trois cellules C_1, C_3 et C_4 de G, alors la transmission de poids passe seulement par l'arête $\{v_2, v_3\}$. Donc v devient une feuille dans T.

$\Leftarrow$ Supposons maintenant que v est une feuille dans $T = (V, F)$ et prouvons que v est un sommet actif dans G.

- Si $deg(v) = 1$, alors il est claire que v est un sommet actif dans G.
- Si $deg(v) = 2$, alors les deux arêtes incidentes à v dans G ne peuvent pas être dans la même ligne, par ailleurs v n'est pas une feuille dans T. Au cas où ces arêtes font partie de la même cellule, seulement l'arête $\{v_1, v_4\}$ ou bien $\{v_2, v_3\}$ est dans F, en outre, v est dans le contexte de la règle R_2, alors v est un sommet actif.
- Si $deg(v) = 3$, alors avec un raisonnement similaire au cas précédent, seul une des arêtes incidente de v est dans F. Donc, v est dans le contexte de la règle R_3, du coup, v est actif.
- Si $deg(v) = 4$, et les cellules C_1, C_3 et C_4 sont dans G, alors v est dans le contexte de la règle R_4. Par conséquent, seulement l'arête v_2, v_3 est dans F, et selon les règles de construction de F, le sommet v est actif.

□

Proposition 3.4.4. *Soient G un graphe à grilles triangulaires de taille ≥ 2, T son arbre couvrant standard et v est un sommet actif dans G. Aussi, soient G' le graphe résiduel lorsque v et ses arêtes incidentes sont supprimés et T' son arbre couvrant standard. Alors, T' ne peut être obtenu à partir de T que par l'élimination de la feuille v et de son arête incidente.*

Preuve. Soient G un graphe à grilles triangulaires de taille ≥ 2 et T son arbre couvrant standard. Il est évident de voir que l'arbre résiduel T', après la suppression de la feuille v et de son arête incidente dans T, est un arbre couvrant du G'. G' est le graphe à grilles triangulaires obtenu à partir de G une fois que v et ses arêtes incidentes sont disparus de G. Maintenant, il reste à prouver que T' est l'arbre couvrant standard de G' :

— Manifestement, les arêtes diagonales de type $(Up - right, Right)$, comme $\{(x, y), (x +$

$1, y)\}$ (voir FIGURE 8) de G' sont dans T'.

— Les arêtes diagonales de type $(Right, Down - right)$ de G' qui ne sont pas situées sur le côté $(Up - left, Left)$ d'une cellule de G', satisfont la même condition dans G. Elles sont alors dans T. En outre, elles sont dans l'arbre couvrant de G'.

□

D'après ce qui précède, nous pouvons résumer l'algorithme d'élection probabiliste distribué sur un graphe à grilles triangulaires G comme suit :

Algorithme 1 Élection probabiliste dans un graphe à grilles triangulaires.

1: **Début**
2: **Tant que** G n'est pas réduit à un sommet unique **Faire**
3: ▶ Tout sommet qui est actif ou devient actif (règles $R_0 - R_3$) génère sa durée de vie selon son poids ;
4: ▶ Une fois que la durée de vie d'un sommet actif est expirée, il est enlevé avec toutes ses arêtes incidentes et son voisin dans l'arbre couvrant standard collecte son poids ;
5: **Fin Tant que**
6: **Fin**

3.5 Analyse de l'algorithme

L'algorithme d'élection dans un graphe à grilles triangulaires est vu comme un algorithme d'élection dans son arbre couvrant standard : comme nous venons de le voir dans la proposition 3.4.2, chaque sommet actif dans un graphe à grilles triangulaires est une feuille dans son arbre couvrant standard et les poids de ce sommet dans les deux configurations sont égaux (revêtement).

Soit $\mathbf{G}=(V, E)$ un graphe à grilles triangulaires. Initialement, tous les sommets ont le même *poids* 1 : $w(v) = 1, \forall v \in V$. Selon les règles vues dans la section 3.4.1, lorsqu'un sommet actif disparaît, son successeur collecte son poids et l'additionne à son poids courant. A l'instant où un sommet v devient actif dans le graphe à grilles triangulaires résiduel G', son poids est le nombre de sommets disparus de ses côtés [4]. La durée de vie $L(v)$ du sommet v est une variable aléatoire ayant une distribution exponentielle de paramètre égal à $w(v)$:

$$Pr(L(v) > t) = e^{-w(v)t}, \quad \forall t \geq 0.$$

4. les sommets des sous arbres dont ses voisins sont les racines.

Nous disons que la mort du sommet actif v se produit selon un processus markovien de paramètre égal à son poids $w(v)$. Cette propriété est équivalente à : la probabilité de disparition de v dans l'intervalle de temps $[t, t+h]$ est $w(v)h + o(h)$, lorsque $h \to 0$ à tout instant t, et cela indépendamment de ce qui se passe ailleurs et de ce qui s'est produit dans le passé (la loi exponentielle est une loi sans mémoire). Le processus aléatoire est une variante du processus de mort pure qui est, à son tour, un exemple spécial du processus de Markov en temps continu [Fel50].

Théorème 3.5.1. *La stratégie décrite ci-dessous mène à une élection probabiliste totalement équitable : dans un graphe à grilles triangulaires, tous les sommets ont la même probabilité d'être élus.*

Nous présentons, dans la sous-section suivante, quelques résultats préliminaires aidant à faire la preuve du théorème.

3.5.1 Uniformité de l'élection

L'élection probabiliste peut être modélisées mathématiquement par un processus de Markov en temps continu. L'état initial du processus est $G = (V, E)$ (le graphe à grilles triangulaires en entier). Soit $\mathcal{E}_G$ l'ensemble des états de tous les sous-graphe à grilles triangulaires $Q = (U, F)$ de G qui vérifient la propriété suivante : toutes les fois que deux sommets diagonaux de la forme $((x, y)$ et $(x+1, y-1))$ ou $((x, y)$ et $(x+1, y+1))$ sont dans Q, alors le sommet de droite $(x+1, y)$ est dans U et ce s'il est dans V.

Exemple 3.5.1. Le graphe à grilles triangulaires de la figure 17 a 3 sous graphes à grilles triangulaires, chacun est obtenu avec une probabilité de $\frac{1}{3}$ comme le montre la figure 19.

La proposition suivante montre que $\mathcal{E}_G$ est l'ensemble de tous les sous-graphe à grilles triangulaires qui peuvent être atteint à partir G par un ordre d'élimination des sommets actifs (rappelons que lorsque un sommet actif est supprimé, toutes ses arêtes incidentes sont aussi supprimées).

Proposition 3.5.1. *Un sous-graphe à grilles triangulaires Q d'un graphe à grilles triangulaires G peut être atteint avec une probabilité positive si et seulement si Q est dans $\mathcal{E}_G$.*

Preuve.

⇒ Soit Q un sous-graphe à grilles triangulaires atteint à partir de $G = (V, E)$ avec une probabilité positive et prouvons que $Q \in \mathcal{E}_G$. Selon la définition du processus de transition, Q doit être obtenu à partir de G par $k-$ordre d'élimination de sommets

actifs ($0 \leq k < n$) où $|V| = n$. Pour $k = 0$, la proposition est vraie. Supposons maintenant qu'elle est vraie pour k et prouvons qu'elle reste vraie pour $k+1$. Ainsi, soit Q un graphe à grilles triangulaires obtenu à partir d'un certain graphe à grilles triangulaires R de G par la suppression d'un sommet actif v et de ses arêtes incidentes. Par hypothèse de récurrence, R est dans $\mathcal{E}_G$ et puisque aucun sommet de degré ≥ 2, se trouvant du côté droit d'une cellule de R, n'est actif, le sous-graphe à grilles triangulaires résiduel Q satisfait aussi la condition d'être dans $\mathcal{E}_G$.

$\Leftarrow$ Soit maintenant $Q = (U, F)$ un graphe à grilles triangulaires dans $\mathcal{E}_G$. Nous prouvons par une récurrence décroissante sur $m = |U|$ que G peut être atteint par $n - m$ transitions avec une probabilité positive. Pour $m = n$, Q est égal à G et par conséquent, $Q \in \mathcal{E}_G$. Supposons que $m < n$, nous devons donc montrer qu'il existe un sommet $v \in V \setminus U$, tel que son ajout à U avec un certain nombre d'arêtes qui le relient aux sommets de Q, produit un nouveau graphe à grilles triangulaires R appartenant à $\mathcal{E}_G$. Or, puisque $m < n$, il existe un sommet $u \in V \setminus U$. Considérons alors un chemin qui relie u à un sommet $s \in U$, soit $v \notin U$ le dernier sommet de ce chemin, alors v a des sommets voisins dans U, (voir FIGURE 16).

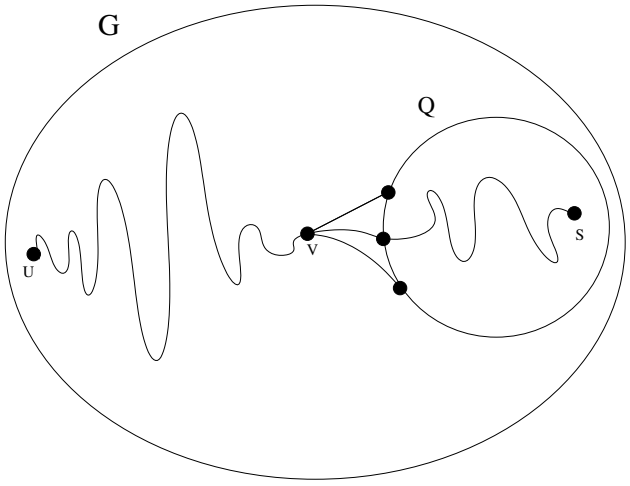


FIGURE 16 – Accessibilité de G à partir d'un sous-graphe à grilles triangulaires $Q \in \mathcal{E}_G$

— Si v a un seul sommet voisin dans U, alors v est un sommet actif dans R (son degré est 1 dans R). Ainsi, R est dans $\mathcal{E}_G$.
— Sinon, v a deux ou trois sommets voisins dans U. Dans ce cas, soit (x, y) les coordonnées du sommet v. Selon l'hypothèse que Q est dans $\mathcal{E}_\mathbf{G}$, il n'y a pas de sommets voisins de v dans U de la forme $((x, y+1)$ et $(x-1, y))$ ou $((x-1, y)$ et

$(x, y-1)$), de telle sorte que v soit un sommet situé au côté droit d'une cellule de R composée par ces sommets. Par conséquent, v ne peut être qu'au côté gauche d'une ou de deux cellules de R. En outre, v est un sommet actif dans R. Par conséquent, $R \in \mathcal{E}_G$.

□

Soit Q l'état du système à l'instant t. Selon la structure aléatoire distribuée de l'algorithme, tout sommet actif v de Q a une durée de vie exponentiellement distribuée de paramètre égal à son poids.

Ce fait est équivalent au fait que dans l'intervalle de temps $[t, t+\Delta t]$, v peut disparaître avec toutes ses arêtes incidentes avec la probabilité $w(v)\Delta t + o(h)$, quand $h \to 0$, et ceci indépendamment de ce qui se passe ailleurs dans le reste du graphe à grilles triangulaires et aussi indépendamment de ce qu'il s'est produit dans le passé.

Nous pouvons, ainsi, montrer que la probabilité de passage du Q au graphe à grilles triangulaires R, obtenu par la suppression du sommet actif v et de toutes ses arêtes incidentes, est :

$$G_{(\mathbf{Q},\mathbf{R})} = \frac{w(v)}{\sum_{u \ actif \ dans \ \mathbf{Q}} w(u)}. \tag{1}$$

à condition que Q ne soit pas réduit à un sommet. Les états absorbants (voir [Fel50]) sont les graphes à grilles triangulaires réduits à un sommet (sommet élu).

Les propriétés ci-dessous caractérisent le processus d'élimination dans un graphe à grilles triangulaires.

- *Le taux de mort du GGT* $\mathbf{G}$ *est :* $\lambda(G) = w(G) = \sum_{u \ actif \ dans \ G} w(u)$.
- *La durée de vie de* G *:* $L(G) = min_u\{L(u),\ u \ actif \ dans \ G\}$ *a la fonction de distribution :*

$$Pr(L(G) \leq x) = 1 - Pr(L(G) \geq x) = 1 - e^{-x\lambda(G)},\ \forall x \in \mathbb{R}^+.$$

Proposition 3.5.2. *Soit* Q *un GGT dans* $\mathcal{E}_G$ *et soit* $G_Q(t)$ *la probabilité que l'état de l'élection au temps* t *est* Q. *Nous avons :*

(i) $\dfrac{dP_G(t)}{dt} = -w(G)P_G(t),$

(ii) pour tout sous-graphe à grilles triangulaires $Q \neq G$ *de taille au moins* 2 *et qui appartient à* $\mathcal{E}_G$*,*

$$\frac{dP_{\mathbf{Q}}(t)}{dt} = -w(\mathbf{Q})P_{\mathbf{Q}}(t) + \sum_v w(v)P_{\mathbf{R}}(t),$$

avec $\mathbf{R} = \mathbf{Q} \cup (\{v\}, \{\{v,u\}, u$ *adjacent à* v *dans* $T\})$*, (rappelons que* T *est l'arbre couvrant standard de* G*)*

où la sommation est effectuée sur tous les sommets v *adjacent à* Q *dans* T *n'appartenant pas à* Q*, et*

(iii) $$\frac{dP_{(\{v\},\emptyset)}(t)}{dt} = \sum_{u \text{ adjacent à } v \text{ dans } G} w(u)P_{(\{v,u\},\{\{v,u\}\})}(t),$$

avec la condition initiale $P_G(0) = 1$*.*

Preuve. Soit Q un sous-graphe à grilles triangulaires de G et considérons l'évolution du processus dans l'intervalle $[t, t+h]$. Calculons la probabilité de se trouver dans l'état Q à l'instant $t+h$.

— Pour $Q \neq G$ et non réduit à une feuille, nous avons :

$$P_Q(t+h) = \sum_v P_R(t)\pi_{R,Q}(h) + P_Q(t)\pi_{Q,Q}(h) + o(h),$$

où $R = Q \cup \{v\}$, et $\pi_{R,Q}(h)$ est la probabilité d'une transition (directe) de R à Q dans un intervalle de temps de longueur h ; la sommation est effectuée à tous les sommets v adjacents à Q et qui ne lui appartiennent pas. Nous obtenons :

$$P_Q(t+h) = h\sum_v \lambda(v)P_R(t) + P_Q(t)[1 - \lambda(Q)] + o(h).$$

Par conséquent,

$$\frac{P_Q(t+h) - P_Q(t)}{h} = -\lambda(Q)P_Q(t) + \sum_v \lambda(v)P_R(t) + \frac{o(h)}{h}.$$

Ceci prouve **(ii)**.

— Pour prouver **(i)**, nous remarquons que dans le cas de $P_T(t+h)$, la somme ci-dessus ($\sum_v \ldots$) du côté droit disparaît, puisque T n'a aucun arbre facteur qui le précède. Ceci établit **(i)**.

— Pour montrer **(iii)**, il suffit de remarquer que l'état $(\{v\},\emptyset)$ ou simplement $\{v\}$, est absorbant et, ainsi, $\pi_{\{v\},\{v\}}(h) = 1$. Donc, dans $P_{(\{v\},\emptyset)}(t+h)$, le terme négative disparaît. Un calcul simple donne **(iii)**.

Les propositions 3.4.2-3.5.1 permettent de confirmer que tout ordre de transition sur $\mathcal{E}_G$ peut être simulé, avec la même probabilité, par un ordre de transition sur l'ensemble d'arbres facteurs [5] de l'arbre couvrant standard de G. Ainsi, l'étude du processus d'élection dans un graphe à grilles triangulaires est traduite par une étude d'élection sur son arbre couvrant standard [MSZ03a].

□

La solution du système d'équations différentielles de la proposition 3.5.2 donne une description mathématique de la probabilité d'être dans l'état Q à l'instant t. Elle caractérise *en principe* la probabilité de distribution des états à un instant donné t. En particulier, elle devrait nous permettre de calculer les probabilités d'absorption [Fel50]. Malheureusement, il n'existe pas une solution explicite de ce système pour tout graphe à grilles triangulaires. Des solutions sur des cas particuliers existent. Dans la suite nous allons présenter un cas d'études.

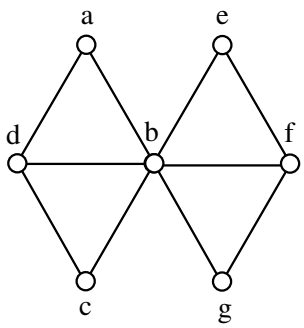


Figure 17 – Exemple d'étude

Exemple 3.5.2. Considérons le graphe à grilles triangulaires de la figure 17. Le graphe de transitions du processus d'élection est donné dans la figure 19. Dans ce graphe, les indices en dessous des noms des sommets représentent leurs poids courants. La séquence des graphes de la figure 18 présente un ordre d'élimination pour élire a. Pour ce sommet, il existe 7 ordres d'élimination.

Les étapes du processus d'élimination peuvent aussi être vues comme une construction d'un arbre couvrant, où la racine de cet arbre est le sommet élu.

5. Arbres obtenus par élimination de sommets feuilles.

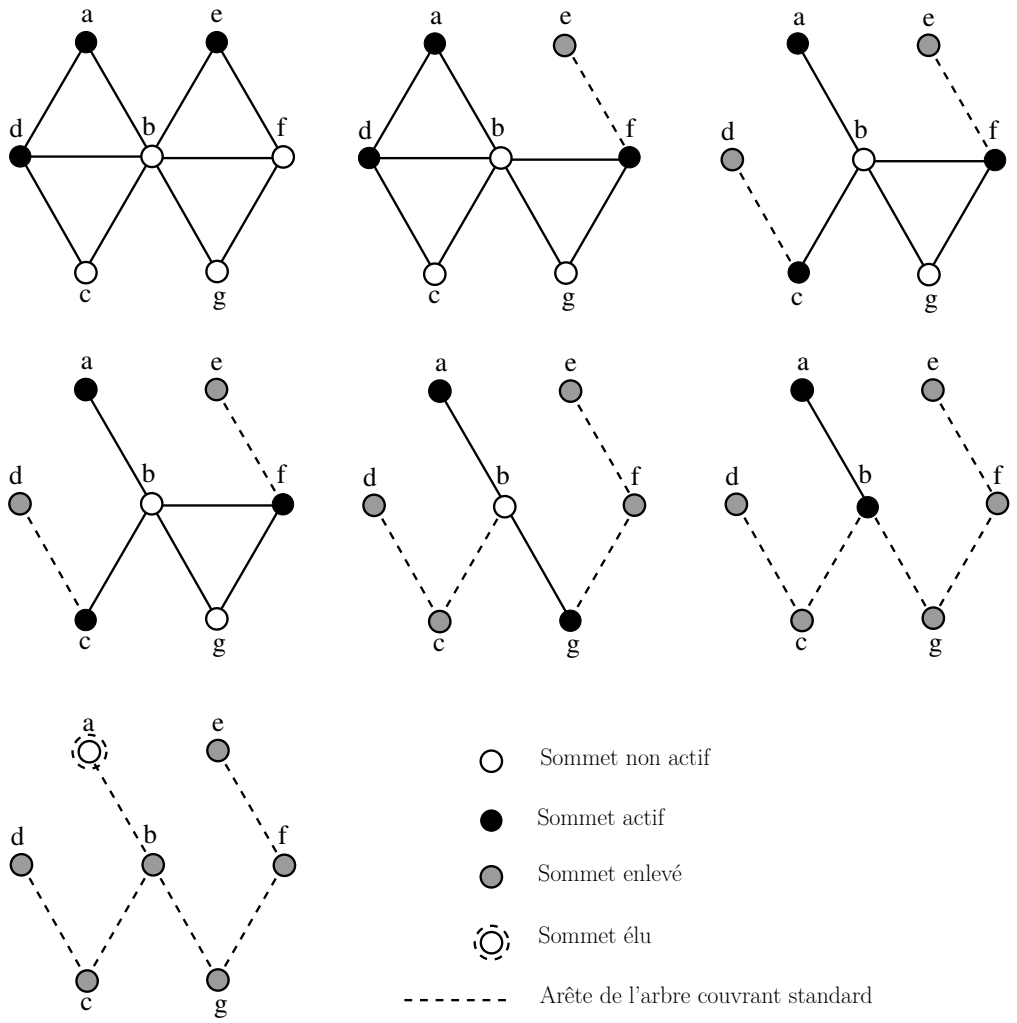


FIGURE 18 – Exemple d'élection dans un graphe à grilles triangulaires

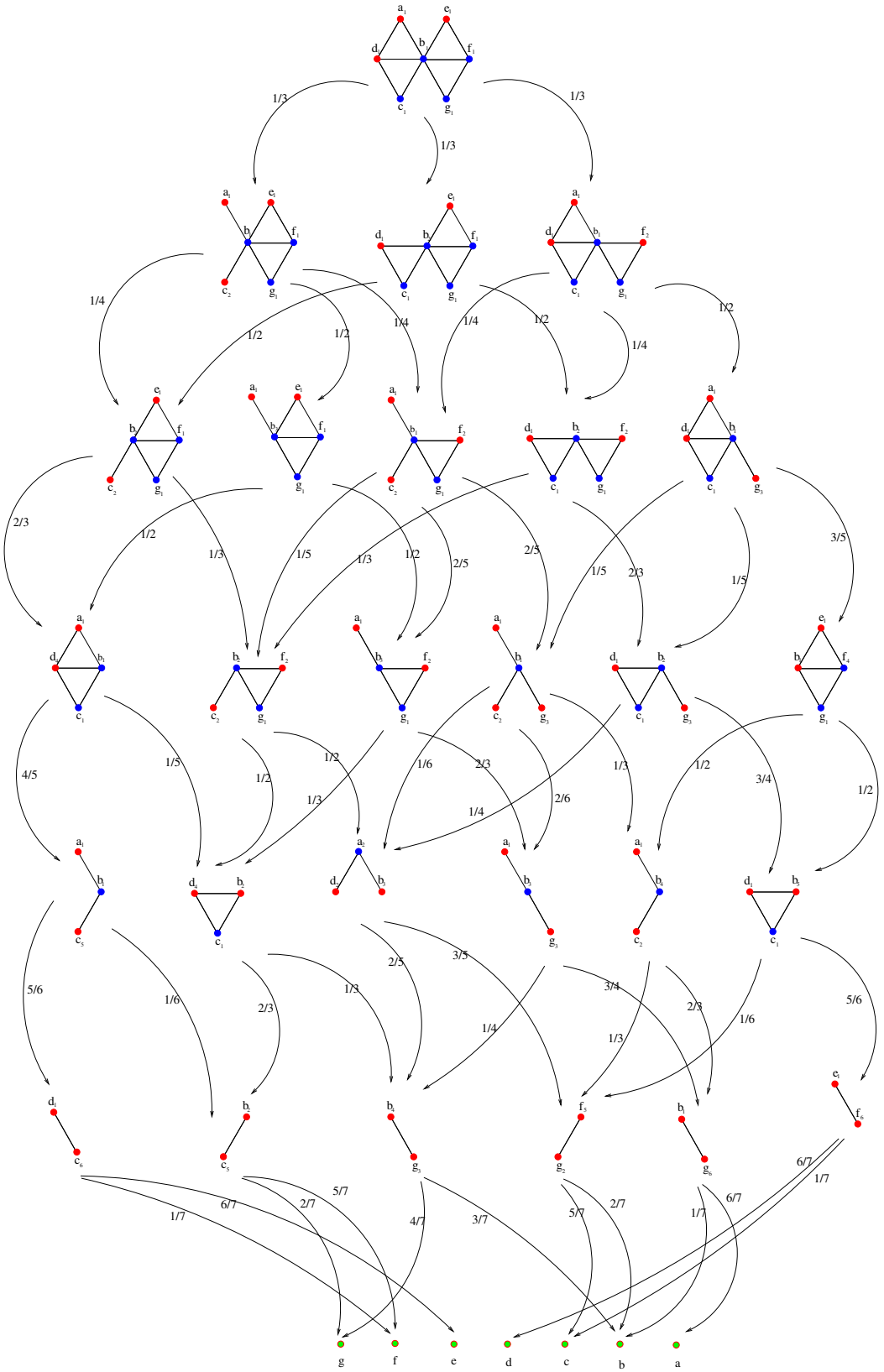


Figure 19 – Exemple graphe de transitions

Initialement, tout sommet actif commence son propre calcul en générant sa durée de vie. Une fois que la durée de vie d'un sommet actif est expirée, il est enlevé avec toutes ses arêtes incidentes. Dans la figure 18, les arêtes sur lesquelles les poids sont transmis sont représentées par les arêtes pointillées, tandis que les autres sont enlevées du dessin qui représente le graphe. A une étape intermédiaire du processus d'élimination, les arêtes pointillées constituent des arbres couvrant partiels. Par conséquent, nous nous sommes assurés qu'à la fin tous les arbres couvrants partiels seront fusionnés en un seul arbre couvrant. Or, cet arbre est unique, alors il sera considéré comme l'arbre couvrant standard du GGT. Ainsi, l'étude de l'algorithme d'élection dans un GGT est projetée à une étude d'élection dans son arbre couvrant standard.

Nous avons vu dans le modèle de l'arbre qu'initialement tous les sommets ont le même poids 1 et que chaque feuille a une durée de vie qui est une variable aléatoire exponentiellement distribuée avec un paramètre égal à son poids. Au moment où la durée de vie d'une feuille est expirée, elle est enlevée avec son arête incidente et son poids est récupéré par son père. Ce processus de suppression continue jusqu'à ce que l'arbre soit réduit à un seul sommet. Le sommet restant est considéré comme l'élu. Par conséquent, la probabilité d'élire un sommet v dans un GGT G est identique à la probabilité d'élection dans son arbre couvrant standard. Celle-ci s'est avérée être égale à $\frac{1}{n}$, où n est la taille de G.

Dans la suite, nous supposons que T est l'arbre couvrant standard d'un GGT G de taille n. Les feuilles de T sont enlevées suivant un processus aléatoire comme décrit ci-dessus jusqu'à ce que T soit réduit à un sommet unique. Dans la suite, nous allons prouver l'uniformité de l'élection pour tous les sommets de T.

Tout d'abord, nous introduisons une légère modification sur le modèle d'élimination des feuilles dans T. Ensuite, nous traduisons le modèle en une variante sur les arbres orientés. En effet, pour un sommet v donné, l'unique arbre enraciné en v peut être construit. Ces arbres enracinés peuvent être employés d'une manière simple pour calculer les probabilités d'absorption.

Nous considérons une forêt d'arbres enracinés. Soit F une forêt d'arbres enracinés, nous présentons un processus de mort sur F comme suit. Chaque feuille v a une durée de vie exponentiellement distribuée de paramètre égal à son poids ; au commencement, tous les sommets de F ont un poids de égal à 1. À tout intervalle de temps $[t, t + \Delta t]$, si la durée de vie d'une feuille est expirée, la feuille est enlevée avec son unique arête incidente. Si la feuille disparue possède un père, alors son père prend son poids, et l'ajoute à son poids. Le processus d'élimination de feuilles continue sur la forêt réduite jusqu'à l'extinction totale de la forêt.

Preuve. [Preuve du théorème principal 3.5.1]

Soient F_1 et F_2 deux forêts de taille respectivement n_1 et n_2. La preuve peut s'effectuer en s'appuyant sur le fait qu'il existe une similitude d'exécution entre le processus d'élection sur G et celui sur son arbre couvrant standard T_G. Par conséquent, pour tout sommet v du G, la probabilité d'élire v dans G est égale à la probabilité d'élire ce sommet dans T_G.

□

3.6 Conclusion

Dans ce chapitre, nous avons présenté et analysé un algorithme permettant de donner la même chance d'être élu à tous les sommets d'un graphe à grilles triangulaires.

Il est alors intéressant d'étudier la faisabilité de l'algorithme dans des classes plus générales.

Des études ont été faites pour les arbres [MSDZ05], les $k-$arbres [HSdZ+04] et les polyominoïdes [HRSZ10], mais d'autres classes semblent intéressantes à considérer : la classe des graphes polyominoïdes en $\mathbb{Z}^3$ semble être un bon cas à étudier.

Or, la difficulté semble résider dans : (1) la possibilité d'un sommet de se décider s'il est simplicial et ceci sans connaissance globale sur le graphe, (2) trouver un modèle pour assurer la transmission des poids des sommets supprimés et garantir l'uniformité à la fin.

Chapitre 4

Construction distribuée d'un arbre couvrant minimal

Sommaire

Dans ce chapitre, nous présentons et analysons deux algorithmes distribués probabilistes pour la construction d'arbre couvrant minimal. Ces algorithmes sont basés en premier lieu sur l'algorithme de rendez-vous où chaque processus p du réseau génère pour chaque voisin q un nombre aléatoire, choisi uniformément dans l'intervalle réel $[0, 1]$. Ce nombre représente la durée d'attente nécessaire pour obtenir un rendez-vous avec ce voisin. L'algorithme de rendez-vous permet ainsi de produire k sous-arbres où k est la taille du couplage maximal produit. L'étape suivante consiste à fusionner ces sous-arbres en un, en joignant aussi les sommets isolés.

Pour l'algorithme $\mathcal{B}$, les arêtes complémentaires de jonction pour réaliser cette fusion sont choisies de telle sorte que leurs poids soient minimaux.

Nous prouvons par la suite que le graphe résiduel est acyclique, contient tous les sommets du graphe initial et utilise les arêtes de poids minimal.

Une étude détaillée sur le nombre de message échangés sera présenté à la fin de ce chapitre. Cette étude montre que l'algorithme que nous proposons présente une meilleure performance que ceux qui nous sommes connus en terme d'échange de messages. Nous montrons également que l'arbre couvrant obtenu est minimal et nous caractérisons le nombre prévu de phases nécessaires pour réduire le nombre de forme de l'arbre couvrant de n à 1. Nous prouvons que ce nombre attendu est délimitée entre $O(\log_2(n))$ et $O(n)$.

Le travail présenté dans ce chapitre a été couronné par la publication des articles [HSHD14][SBK15] et a été présenté comme communication dans [SHH13a][SHHK14][SBK14].

Le présent chapitre est organisé comme suit. La section 4.1 présente quelques algorithme classiques de construction d'arbre couvrant minimal ainsi qu'une petite description de chaqu'un d'eux. La section 4.2 expose certaines notations et définitions nécessaires pour la compréhension du reste du chapitre. Elle présente également le modèle et les hypothèses de notre d'étude. Dans la section 4.4 nous présentons notre premier algorithme distribué probabiliste pour la construction d'un arbre couvrant, noté $\mathcal{A}$. En fin, dans la section 4.5 nous donnons une description formelle et détaillée du deuxième algorithme noté $\mathcal{B}$, et nous présentons une analyse de ce dernier dans la sous-section 4.5.1.

4.1 Introduction

Le problème d'arbre couvrant minimal est un problème d'optimisation combinatoire très connu, il consiste à relier tous les sommets d'un graphe connexe non orienté, sans créer aucun cycle tout en choisissant les arêtes de poids minimal. En informatique les différentes méthodes pour chercher l'arbre couvrant minimal ont joué un rôle central dans l'analyse et la conception des algorithmes de construction de tels arbres [MKI11].

Il existe de nombreux algorithmes de recherche d'un arbre couvrant de poids minimal. Nous citerons, entre autres, l'algorithme de Borůvka qui est le premier algorithme de recherche d'arbre couvrant minimal découvert, publié par Otakar Borůvka en 1926 [BVP11]. Son principe est de réduire un graphe G en contractant des arêtes : nous choisissons peu à peu les arêtes qui seront dans l'arbre, et à chaque fois que l'on en choisit une, nous fusionnons les nœuds que cette arête relie. Ainsi, il ne reste plus de sommets non liés.

L'algorithme de Prim est un algorithme glouton [1] qui permet de trouver un arbre couvrant minimal dans un graphe connexe, valué et non orienté.

Il a été conçu en 1957 par Robert C. Prim [Mar02]. Cet algorithme consiste à choisir arbitrairement un sommet et à faire croître un arbre à partir de ce sommet. Chaque augmentation se fait de la manière la plus économique possible.

En d'autres termes, cet algorithme trouve un sous-ensemble d'arêtes formant un arbre sur l'ensemble des sommets du graphe initial, et tel que la somme des poids de ces arêtes soit minimale. Si le graphe n'est pas connexe, alors l'algorithme ne déterminera l'arbre couvrant minimal que d'une composante connexe du graphe.

L'algorithme de Kruskal est un algorithme de recherche d'arbre recouvrant de poids minimum ou arbre couvrant minimum dans un graphe connexe, valué et non-orienté. Il a été conçu en 1956 par Joseph Kruskal [NL12]. L'algorithme consiste à d'abord ranger par ordre de poids croissant les arêtes d'un graphe, puis à retirer une à une les arêtes selon cet ordre et à les ajouter à l'arbre couvrant minimum cherché tant que cet ajout ne fait pas apparaître un cycle dans l'arbre couvrant minimum.

Les algorithmes présentés ci-dessous sont pour l'ensemble séquentiels et se basent sur une connaissance globale pour réaliser le calcul. Dans notre étude, nous introduisons et analysons deux algorithmes distribués probabilistes pour la construction d'arbre couvrant minimal, ces algorithme sont basés sur l'algorithme de rendez-vous présenté dans [HMR+06], qui est un algorithme distribué probabiliste pour trouver un couplage maximal.

Nous considérons que toutes les arêtes dans lesquelles des rendez-vous ont eu lieu, sont automatiquement dans l'arbre couvrant minimal. Ces arêtes constituent la base de notre algorithme pour construire l'arbre couvrant minimal d'une manière distribuée. Par la suite, nous fusionnons ces arêtes en ajoutant les arêtes de poids minimal, tout en s'assurant que l'ajout d'une nouvelle arête ne crée pas de cycle.

4.2 Définitions et modèle

4.2.1 Définitions et notations

Soit $T = (V, E)$ un arbre. Les définitions suivantes sont équivalentes et définissent un arbre :

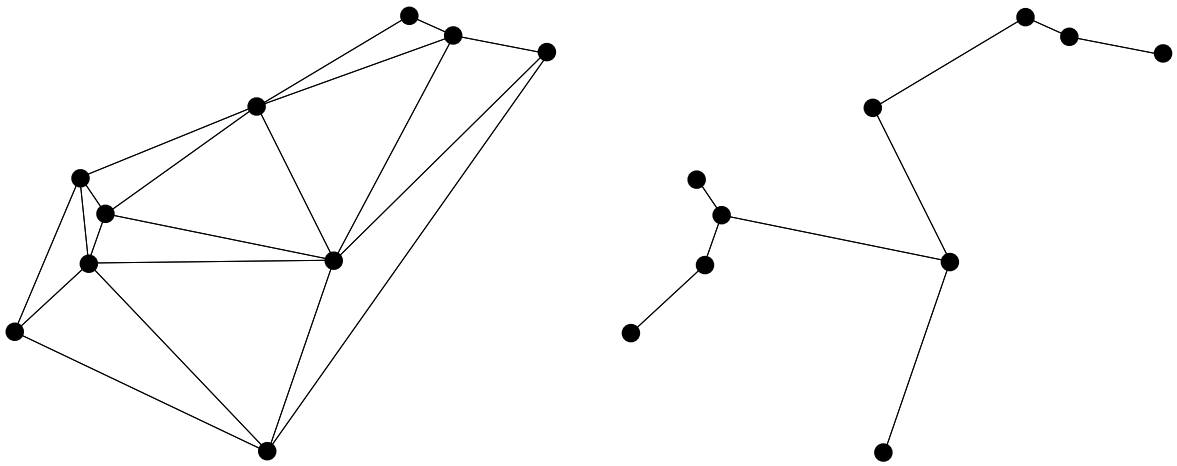
- T est connexe et sans cycle.

1. Un algorithme glouton (greedy algorithm en anglais) est un algorithme qui suit le principe de faire, étape par étape, un choix optimum local, dans l'espoir d'obtenir un résultat optimum global

- Il existe un seul et unique chemin entre toute paire de sommets.
- T n'est plus connexe si une arête de T est supprimée.
- T est sans cycle, un cycle sera créé suite à l'ajout d'une arête entre deux sommets non-adjacents dans T.
- T est connexe et contient $n-1$ arêtes.
- T est un sous-graphe acyclique et contient $n-1$ arêtes.

Définition 4.2.1. Soit $G = (V, E)$ un graphe et $T = (V_T, E_T)$ un arbre. T est un *arbre couvrant* si et seulement si $V = V_{T'}$ et $E_{T'} \subseteq E_T$.

Un exemple d'arbre couvrant est donné à la figure 20 (b) en considérant le graphe de la figure 20 (a).



(a) Exemple de graphe (b) Exemple d'arbre couvrant

FIGURE 20 – Arbre couvrant d'un graphe

Définition 4.2.2. Soit $T = (V_T, E_T)$ et $T' = (V_{T'}, E_{T'})$ deux arbres. T' est *sous-arbre* de T si et seulement si $V_{T'} \subseteq V_T$ et $E_{T'} \subseteq E_T$.

Le problème du rendez-vous dans un réseau distribué coïncide avec celui du couplage dans un graphe [Die00]. Rappelons la définition d'un couplage.

Définition 4.2.3. Un *couplage* dans un graphe simple non orienté $G = (V, E)$ est un sous-ensemble d'arêtes $\mathcal{M}$ de E, tel que $\forall (e_1, e_2) \in \mathcal{M}^2, e_1 \cap e_2 = \emptyset$.

Définition 4.2.4. Un couplage de G est *maximum* si aucune arête ne peut être ajoutée sans violer la contrainte de la définition 4.2.3.

Un *couplage à cardinalité maximale* est un couplage maximal ; l'inverse n'est pas vrai. La cardinalité maximale s'appelle *nombre de couplage*.

La figure 21 montre un exemple de *couplage maximal* (arêtes plus épaisses) dans le graphe. D'autre part, un *arbre couvrant* T d'un graphe connexe et non orienté G est un

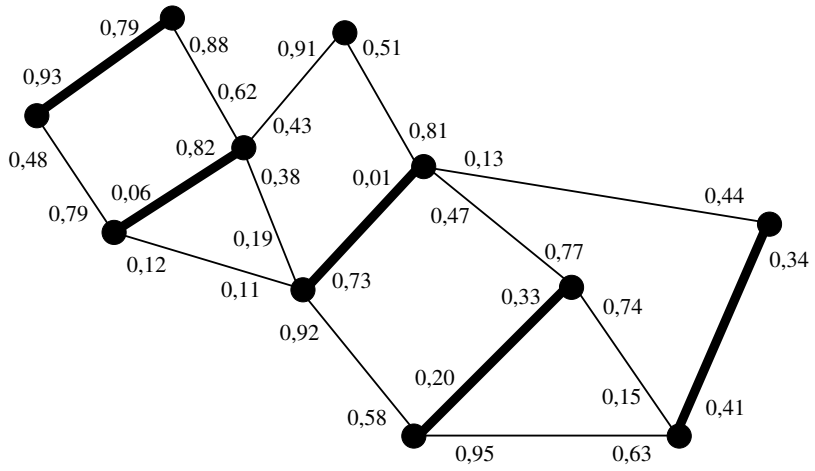


Figure 21 – Exemple de couplage maximal où les extrémités des aretes plus épaisses sont saturées

arbre constitué de tous les sommets de G et un sous-ensemble de l'ensemble d'arêtes de E.

Un *arbre couvrant minimal* de G est une sélection d'arêtes de G qui forment un arbre couvrant de poids minimal [Gol04]. C'est à dire que tous les sommets appartiennent à G et ils sont reliés sans former un cycle. La figure suivante montre un exemple de l'arbre couvrant minimal obtenu à partir du graphe de la figure 22.

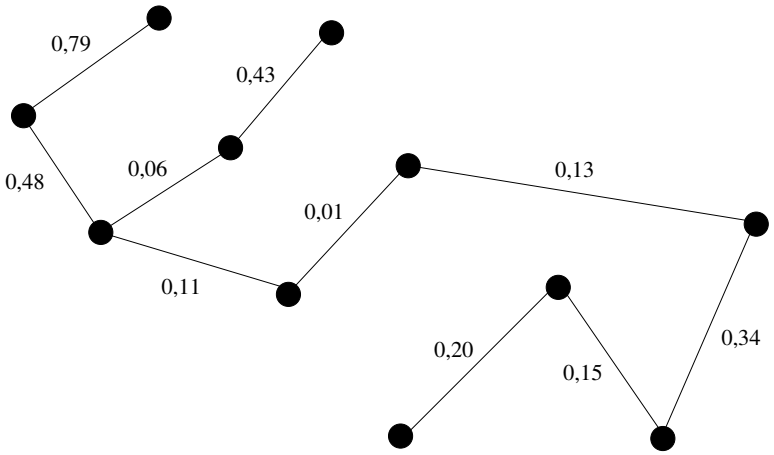


Figure 22 – Arbre couvrant minimal du graphe de la figure 21

4.2.2 Modèle et hypothèses

Dans ce chapitre, nous considérons le modèle standard de réseaux de communication point à point [Tel00][Lav95b]. Un réseau est décrit par un graphe connexe non orienté $G = (V, E)$ où V est l'ensemble de tous les sommets de G et E est l'ensemble des arêtes reliant ces sommets. Les sommets représentent les nœuds de réseau ou des processus et les arêtes représentent les liens de communication entre les processus.

Nous supposons que chaque processus dispose d'une mémoire partagée ou non local, d'une capacité limitée et d'au moins un processeur. Il ne peut communiquer directement qu'avec ses voisins par échange de messages. En outre, il est capable de distinguer entre ses ports (tout port à partir duquel un message est reçu ou envoyé). Les identités (adresses IP sur Internet) des autres processus, en particulier ceux de ses voisins, sont inconnus (connaissance locale).

Pour notre étude, les processus ne communiquent dans le réseau que par échange de messages. L'échange de messages se fait sans perte ou altération ou modification. La façon dont les échanges entre les opérations sont effectuées détermine le type de réseau.

L'horloge est globale et commune à tous les processus, et nous admettons que tout événement, qui a lieu à un certain temps, est de durée nulle. Nous supposons aussi que les communications ne prennent aucun temps, c'est-à-dire que le délai entre la décision d'un sommet et la notification à ses voisins est égal à 0.

4.3 Algorithmes classiques de construction d'arbre couvrant minimal

4.3.1 Algorithme de Borůvka-Sollin

Otakar Borůvka (né le 10 mai 1899 à Uhersky Ostroh – mort le 22 juillet 1995 à Brno en Tchécoslovaquie) était ingénieur en télécommunication. En 1926, il écrit un premier article "O jistém problému minimálním" dans lequel il décrit un algorithme permettant de trouver un arbre couvrant de poids minimum dans un réseau de distribution électrique. Son premier article, jamais publié, est redécouvert par Gustave Choquet en 1938 puis par George Sollin en 1961.

L'algorithme de Borůvka s'applique sur des graphes valués dont les poids des arêtes sont tous différents.
L'algorithme est initié par un sommet quelconque et il le joint au sommet le plus proche.

L'algorithme passe de ce sous-réseau de 2 sommets à un sous-réseau à 3 sommets en lui adjoignant une nouvelle arête qui a le poids le plus minimal des arêtes adjacentes. Il s'agit donc d'un simple critère d'optimisation pour passer d'un sous réseau de $n-1$ sommets à un sous-réseau de n sommets par adjonction extérieure au sous-réseau initial.

Algorithme 2 Borůvka.

Entrée : $G = (V, E, w)$. Un graphe orienté et valué ;
Sortie : T un arbre couvrant minimal ;
début
    $T \longleftarrow \emptyset$;
    répéter
        $F \longleftarrow a$ forêt consistant en arête incidente la plus petite pour chaque sommet dans G ;
        $G \longleftarrow G\ F$;
        $T \longleftarrow T \cup F$;
    jusqu'à $|T| = n - 1$;
fin

Complexité

Chaque étape de Borůvka réduit le nombre de sommets par un facteur d'au moins deux. Par conséquent, la boucle sera exécutée au plus $O(\log n)$ fois. Dans chaque itération, toute la contraction peut être faite en $O(m)$ fois. Au total, l'algorithme de Borůvka a un temps d'exécution de $O(m \log n)$.

4.3.2 Algorithme de Prim

Algorithme de Prim a été conçu par un scientifique en informatique Robert Prim en 1957.
Il commence à partir d'un sommet arbitraire, et se construit sur un seul arbre couvrant minimal partiel, à chaque étape en ajoutant une arête reliant le sommet le plus proche mais pas encore dans l'arbre couvrant minimum partiel actuel. Il croît jusqu'à ce que l'arbre s'étend à tous les sommets du graphe d'entrée. Cette stratégie est gloutonne dans le sens où, à chaque étape l'arbre couvrant partiel est augmenté avec une arête qui est la plus petite parmi toutes les arêtes voisines possibles.

Algorithme 3 Prim.

1: **Entrée** : $G = (V, E, w)$. Un graphe orienté et valué ;
2: **Sortie** : T un arbre couvrant minimal ;
3: **début**
4: $T \longleftarrow \emptyset$;
5: Soit r un sommet choisi arbitrairement de V ;
6: $U \longleftarrow r$;
7: **répéter**
8: Trouver $u \in U$ et $v \in V - U$ tel que l'arête (u, v) est la plus petite arête entre U et $V - U$;
9: $T \longleftarrow T \cup \{(u, v)\}$;
10: $U \longleftarrow U \cup \{v\}$;
11: **jusqu'à** $|U| = n$;
12: **fin**

Complexité

La complexité de l'algorithme dépend fortement de la manière dont est implémenté le choix de l'arête/sommet à ajouter dans l'ensemble à chaque étape. Avec une représentation naïve, comme une simple liste d'adjacence et des recherches dans celle-ci, nous obtenons une complexité en $O(m.n)$ avec m le nombre d'arêtes et n le nombre de sommets. Si nous utilisons un tas min binaire, la complexité devient alors $O(m. \log n)$. En utilisant un tas de Fibonacci, nous pouvons encore descendre à $O(m + n. \log n)$.

4.3.3 Algorithme de Kruskal

Le principe est le suivant : nous organisons un parcours des arêtes dans l'ordre des valeurs croissantes. Initialement, le graphe partiel A est vide. Nous ajoutons l'arête courante à A si et seulement si elle ne crée pas de cycle dans A, et nous nous arrêtons en fin de parcours, ou dès que nous avons obtenu $n - 1$ arêtes dans A.

À l'issue de l'algorithme, nous avons bien un arbre partiel de poids minimum, en effet, on a construit un graphe partiel sans cycle, ayant au moins $n - 1$ arêtes (par hypothèse, le graphe G est connexe), et d'autre part, toute arête u non retenue forme un cycle avec des arêtes situées avant elle dans la liste ordonnée des arêtes.

Pour faciliter le test : u ne crée pas de cycle, il suffit de gérer les composantes connexes de A au fur et à mesure de sa construction. Une arête pourra être ajoutée si et seulement si ses deux extrémités n'appartiennent pas à la même composante. Les composantes des deux

extrémités sont alors fusionnées en une seule.

Algorithme 4 Kruskal.

Entrée : $G = (V, E)$. Un graphe orienté et valué ;
Sortie : T un arbre couvrant minimal de G ;
début
    Soit P une partition des sommets de G, où chaque sommet est dans un ensemble séparé ;
    Soit Q une liste avec priorités gardant en mémoire les arêtes de G, ordonnées selon leur poids ;
    Tant que Q n'est pas vide **faire**
        $(u, v) \longleftarrow Q.eneverMin()$;
        Si $P.trouver(u)! = P.trouver(v)$ **alors** ;
            $A jouter(u, v)T$;
            $P.union(u, v)$;
        retourner T ;
    Fin Tant que ;
fin

Complexité

Lorsque le parcours est organisé, nous effectuons au plus $n - 1$ pas d'itérations. L'algorithme est donc en $O(n)$+ complexité maximale du tri des arêtes. Ce dernier terme dépend de la technique utilisée :

- tri préalable (les meilleurs algorithmes connus sont en $O(m \log m)$),
- recherche séquentielle dans la liste des arêtes, à chaque pas d'itération (complexité totale de $\Sigma_{k=1}^{n-1} O(m - k)$ puisqu'à l'étape k il reste $m - k$ arêtes),
- organisation "en tas" de la liste des arêtes et réorganisation à chaque étape :
  - $O(m \log m)$ pour l'organisation initiale,
  - $O(log(m - k))$ pour la réorganisation à l'étape k (nombre d'arêtes restant = $m - k$) d'où, globalement : $O(m \log m) + \Sigma_{k=1}^{n-1} O(\log m)$ étant prépondérant, le terme $O(m \log m)$ puisque $m \geqslant n - 1$ (graphe connexe).

4.4 Algorithme $\mathcal{A}$: Construction distribuée d'un arbre couvrant

Dans cette section, nous introduisons et analysons un algorithme distribué probabiliste, noté $\mathcal{B}$, pour la construction d'arbre couvrant. Cet algorithme est basé dans un premier temps sur l'algorithme de *HS* [HMR+06] qui permet de produire un couplage maximal. Dans un second temps notre algorithme utilise les arêtes du couplage maximal produit et les relie de manière distribuée sans former de cycle. dans la suite, nous définissons les règles qui permettent d'assurer cette dernière opération de jointure.

Tout processus q génère une $t_q(r) = C_v$: une variable aléatoire (v.a) choisie uniformément dans l'intervalle réel $[0, 1]$ pour tous les voisins r, et qui représente en même temps une couleur C_v selon la quelle deux sous-arbre (deux patates) peuvent établir un lien (une seule arête) de communication.

Chaque sommet v communique avec ses voisins par envoi et réception de messages.

Tous sommet v a les paramètres suivants :

- $t_v(r)$ représente une variable aléatoire choisie uniformément par le sommet v dans l'intervalle réel $[0, 1]$ pour chaque voisin r.
- P_v est une variable booléenne qui représente l'existence du sommet v dans le graphe résiduel (i.e. P_v égale à *true* si v appartient à l'arbre couvrant, et *false* autrement)
- d_v est un paramètre qui compte le nombre de zéros reçus par le sommet v (si d_v est égal au degré de v alors v est isolé).
- S_v est l'ensemble des valeurs générées par v pour tout voisin r de v, S_v contient $t_v(r)$, et soit $S_{v,r} = S_v \cup S_r \backslash \{t_v(r), t_r(v)\}$.

Soit V_v un ensemble des voisins de v tels que $\forall r \in V_v$, nous avons $p_v = true$ (initialement v peut choisir n'importe quel voisin) et $V_{v,r} = V_v \cup V_r$.

À chaque fois que nous avons un rendez-vous entre v et r, ces sommets partagent l'ensemble $S_{v,r}$ (Voir ALGORITHME 5).

L'algorithme 5 résume la première étape pour construire un arbre couvrant.

4.4.1 Analyse de l'algorithme

L'algorithme commence par générer uniformément $2|E|$ variables aléatoires réelles indépendantes (v.a) dans l'intervalle réel $[0, 1]$: une variable aléatoire pour chaque demi arête. Nous supposons que les $(2|E|)!$ permutations sur l'ensemble de ces nombres réels ont la même probabilité.

Algorithme 5 Construction distribuée d'un arbre couvrant (Première partie).

1: Tout sommet v attend jusqu'à ce que l'un des événements suivants se produit :
2: **Début**
3: **Tant que** $(t \leq 1)$ **Faire**
4: **Si** v reçoit 0 de son voisin r **alors**
5: v augmente d_v de 1 ;
6: **Sinon Si** v reçoit 2 de son voisin r **alors**
7: $P_{v,r}$ devient *true* et v envoie $false$ à tous les autres voisins ;
8: **Sinon Si** v reçoit $false$ de son voisin r **alors**
9: $V_v = V_v \backslash P_{v,r}$;
10: **Sinon Si** v reçoit 3 de son voisin r **alors**
11: $P_{v,r}$ devient *true* telle que $min(S_v) = t_v(b) \in S_v (b \neq r$ et $b \in V_v)$;
12: **Sinon Si** v reçoit 1, S_r et V_r de son voisin r et $V_{v,r} \neq \emptyset$ **alors**
13: **Si** $min(S_{v,r}) = t_v(b) \in S_v$ **alors**
v envoie 2 à r et 0 à tous les autres voisins (* il y a un rendez-vous entre v et r, et les arêtes (v, r) et (v, b) feront partie de l'arbre couvrant *) ;
14: **Sinon** $min(S_{v,r}) = t_r(b) \in S_r (b \neq v)$ **alors** $p_{v,r}$ devient *true* et v envoie 3 à r, et 0 aux autres voisins (* v est une feuille dans l'arbre couvrant *) ;
15: **Finsi**
16: **Sinon Si** $t = t_v(r)$ et v n'a reçu aucun message de son voisin r **alors**
17: v envoie 1, S_v et V_v à r et 0 aux autres voisins (* il y a un rendez-vous entre v et r *) ;
18: **Sinon Si** $t = 1$ **alors**
19: **Si** $\exists r$ $p_{v,r} = true$ **alors**
v appartient à l'arbre couvrant (* v est une feuille de l'arbre couvrant *)
20: **Sinon** $p_{v,r} = false$, $\forall r \in V_r$ (* $t > 1$ *) ;
21: **Finsi**
22: **Sinon** le tour se termine sans la participation de v à un rendez-vous, donc c'est un sommet isolé avec $d_v = deg(v)$ (* le nombre des zéros reçus par v égale au degré du sommet v *)
23: **Finsi**
24: **Fin tant que**
25: **Fin**

En effet, pour maintenir cette hypothèse, nous devons tout simplement postuler que, pour chaque arête $e = \{u, v\} \in G$, l'algorithme produit deux v.a. continues $t_{v,u}$ et $t_{u,v}$, supposons que ces v.a. associées aux arêtes soient indépendantes. La probabilité de générer les mêmes valeurs tend vers zéro.

Le premier rendez-vous a lieu sur une arête $e = \{u, v\}$, si au moins une des deux v.a. associées, $t_{v,u}$ ou $t_{u,v}$, est minimale dans le graphe entier. Ainsi, pour le premier rendez-vous sur G, toute arête a la même chance $1 \setminus |E|$ d'être choisie.

L'attribution du premier rendez-vous aux sommets u et v sur l'arête $\{u, v\}$, implique que ces deux sommets et leurs arêtes d'incidentes sont enlevées du graphe. L'algorithme continue à s'exécuter sur le graphe résiduel (préservant les générations aléatoires des arêtes restantes) jusqu'à ce qu'aucune arête ne demeure dans l'ensemble des arêtes du graphe.

Le nombre de rendez-vous dans un tour est tout simplement le nombre total d'arêtes auxquelles des rendez-vous sont assignés. Nous notons par $N(G)$ ce nombre. C'est une variable aléatoire entière. Elle prend la valeur 0 avec probabilité 1 si $E = \emptyset$, la valeur 1 avec probabilité 1 si $|E| = 1$. En général, elle prend une valeur de l'ensemble de toutes les cardinalités des couplages maximaux dans G, avec une certaine probabilité.

Proposition 4.4.1. *Soit $G = (V, E)$ un graphe avec $|V| = n$ et $|E| = m$, la probabilité d'avoir initialement k sous-arbres couvrants est :*

$$Pr(N(G) = k) = \frac{1}{m} \sum Pr(N(G_e) = k - 1), \quad \forall k \geq 1. \tag{2}$$

Où G_e est le graphe résiduel qui contient toutes les arêtes de G, sauf l'arête e et ses arêtes incidentes à ses extrémités.

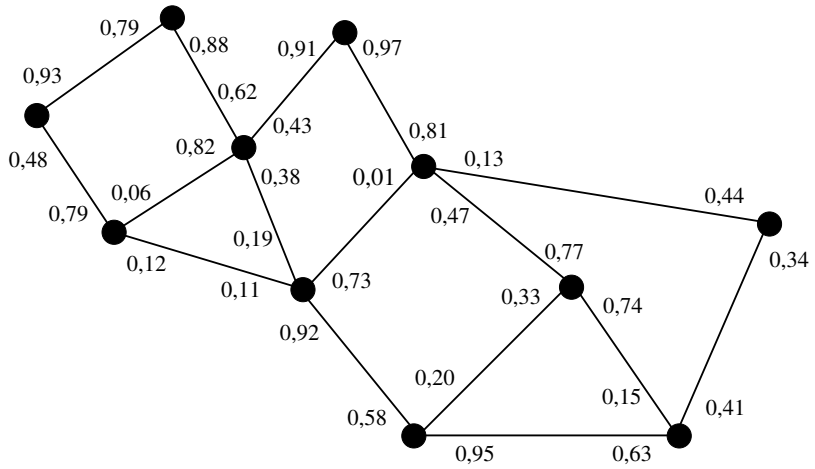


FIGURE 23 – Exemple étudié

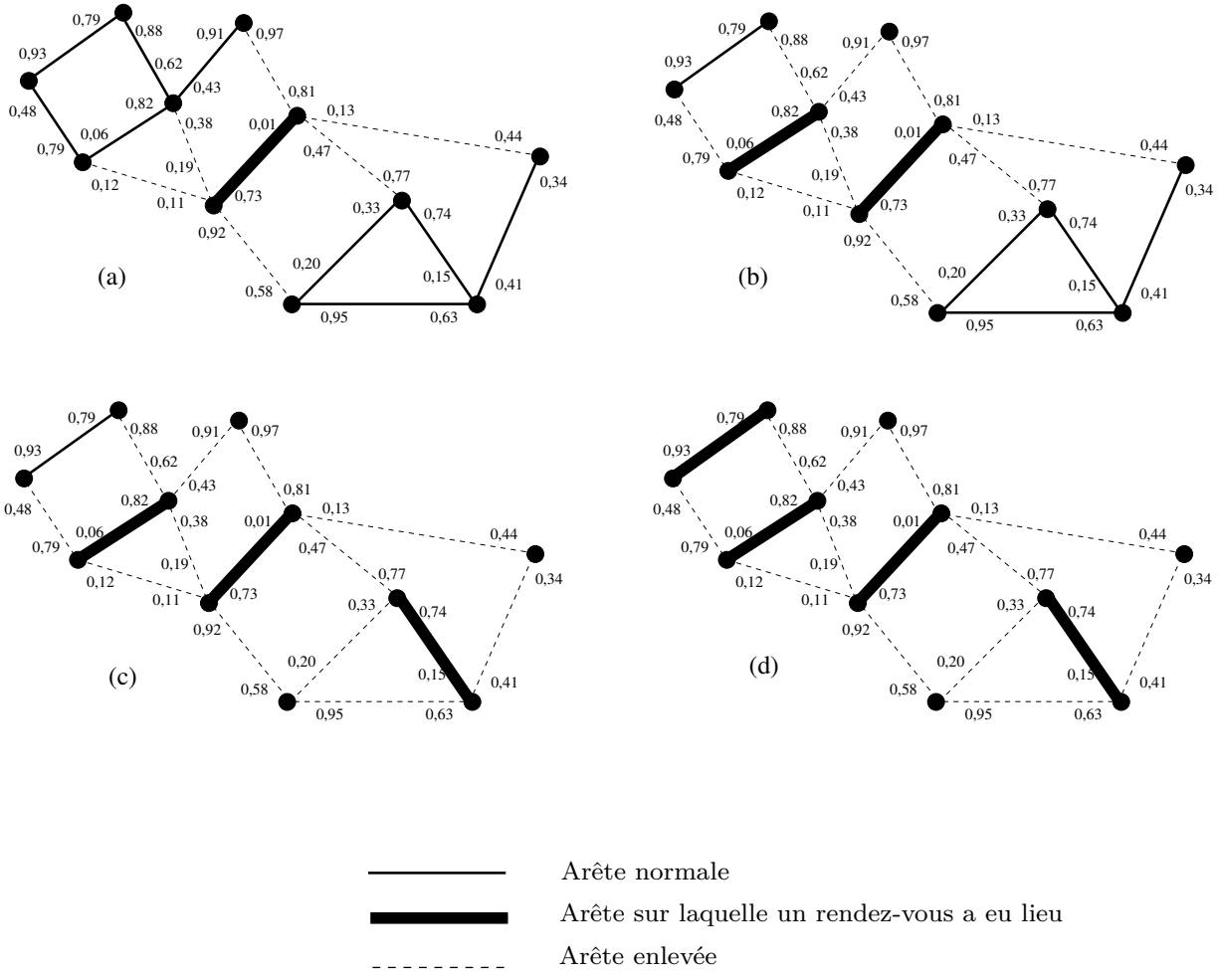


Figure 24 – Exemple d'éxecution de la première partie de l'algorithme

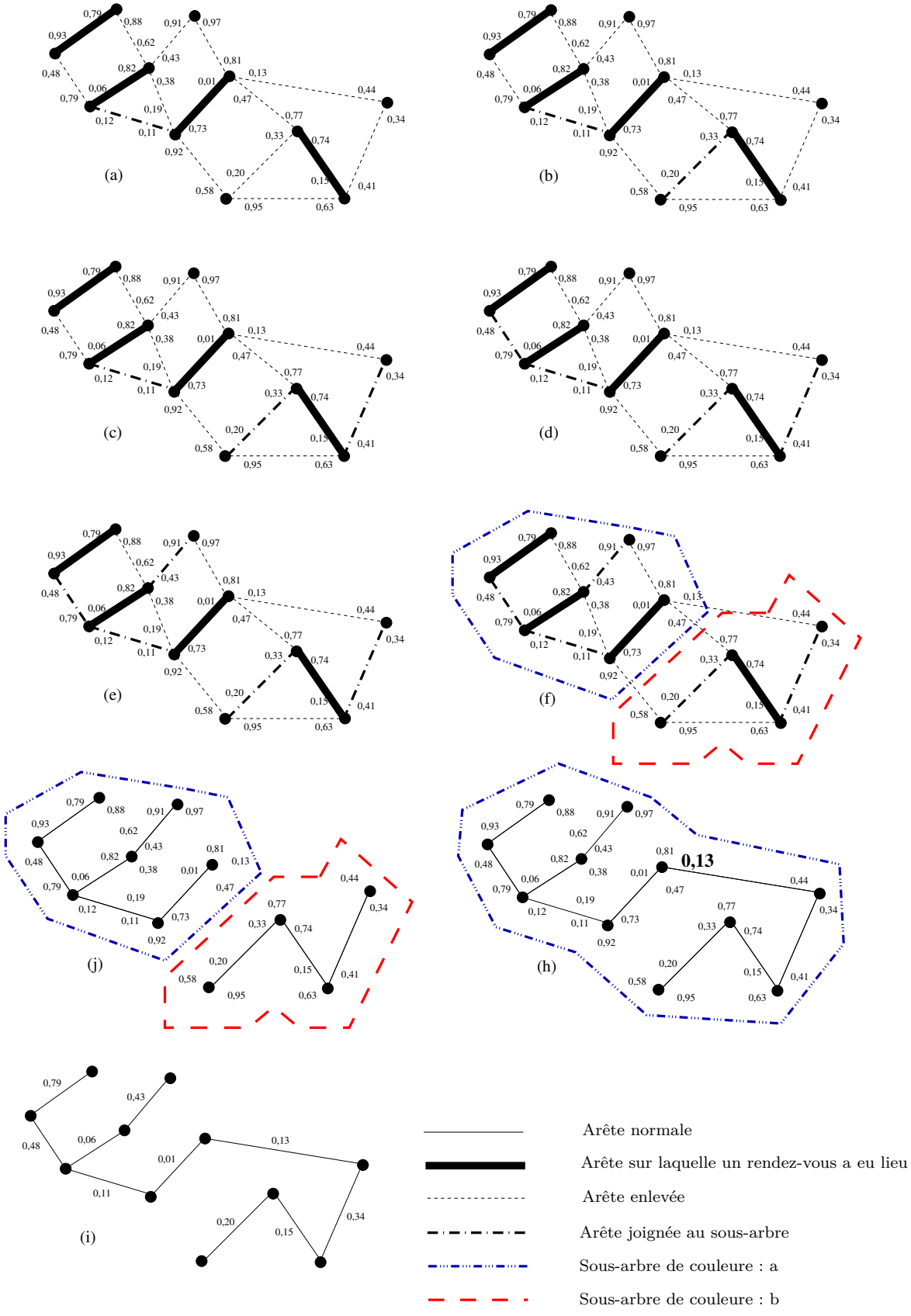


FIGURE 25 – Exemple d'éxecution de la deuxième partie de l'algorithme

Exemple 4.4.1. Soit le graphe de la figure 23. est le graphe sur lequel nous allons exécuter notre algorithme de construction d'arbre couvrant. Le graphe de transitions issus de processus de construction d'arbre couvrant est donné dans la figure 24. Cette dernière représente la première partie de l'algorithme qui consiste à établir tous les rendez-vous possible dans le graphe, c'est à dire établir un couplage maximal.Ceci est possible par l'application de l'algorithme de *HS*. La seconde partie est lancée pour relier toutes les arêtes sur lesquelles des rendez-vous on eu lieu, ainsi que les sommet isolés. La figure 25 donne une trace de l'exécution de la seconde partie de l'algorithme, appliquée au graphe résiduel de la première partie. Elle présente les différentes étapes de construction d'arbre couvrant en se basant sur le couplage maximal produit par la première partie.

Comme nous l'avons vu au début de cette section, initialement chaque processus (sommet) génère une variable aléatoire choisie uniformément dans l'intervalle réel $[0, 1]$ pour tous ses voisins, chaque variable représente une couleur différente, cette couleur nous sera utile dans la deuxième étape de cet algorithme. L'algorithme de rendez-vous (*HS*) continue son exécution jusqu'à ce nous obtenons un couplage maximal dans le graphe, et c'est là où se termine la première étape. Une fois le couplage maximal est atteint, l'algorithme déclenche la seconde partie en reliant toutes les arêtes auxquelles les rendez-vous ont été assignés (les sous arbres) et les sommets isolés pour former un graphe acyclique. Pour ce faire, le sous-arbre qui contient la plus petite valeur, sélectionne l'arête incidente sur laquelle cette valeur a été générée en infectant son sou-arbre voisin par sa couleur, et ce pour éviter toute création de cycle (voir Figure 24 (h)), ainsi l'algorithme continue son exécution jusqu'à ce que nous obtenons un sous-graphe T de G portant une seule couleur, qui est la couleur représentée par la plus petite valeur dans G (Voir Figure 24 (i)).

Proposition 4.4.2. *La stratégie décrite ci-dessous mène à la construction d'un sous graphe T de G.*

- *T est est connexe,*
- *T est acyclique,*
- *T relie tous les sommets de G.*
  Donc T est un arbre couvrant de G.

Dans cette section, nous avons présenté un algorithme distribué probabiliste pour la construction d'un arbre. Nous avons prouvé que l'arbre construit par ledit algorithme est un arbre couvrant du graphe G, en revanche, nous n'avons pas encore prouvé que l'arbre construit par notre algorithme est minimal.

Une deuxième piste de recherche consiste à trouver l'arbre couvrant *minimal* toujours en se basant sur l'algorithme de *HS*. Dans ce qui suit nous allons présenter une variante de notre algorithme $\mathcal{A}$, qui permet de calculer un arbre couvrant minimal.

4.5 Algorithme $\mathcal{B}$: Construction d'un arbre couvrant minimal

Dans cette section, nous proposons et analysons un algorithme distribué probabiliste, noté $\mathcal{B}$ pour construire l'arbre couvrant minimal basé sur l'algorithme rendez-vous *HS* [HMR+06].

De nombreuses recherches ont été menées sur le problème couplage maximal, comme algorithme *MSZ* [MSZ03c], algorithme de *Kuhn* [KOT14], *Heuberger algorithme* [HW11] et algorithme *HS* [HMR+06]. Dans algorithme *MSZ* les auteurs ont proposé et analysé un algorithme aléatoire pour obtenir des rendez-vous entre voisins dans un graphe anonyme, ils ont montré également l'efficacité de leur algorithme dans plusieurs types de graphes.

Dans algorithme *HS* [HMR+06], les auteurs ont introduit et étudié un algorithme distribué probabiliste pour trouver le couplage maximal dans un graphe. Cet algorithme est basé sur les délais aléatoires qui sont générés de manière uniforme dans l'intervalle réel $[0, 1]$. Un rendez-vous aura lieu entre une paire de processeurs voisins si les deux processeurs sont libres au moment prévu.

Dans l'algorithme que nous proposons, nous utilisons l'algorithme de *HS* [HMR+06] parce que le nombre prévu de rendez-vous trouvés par cet algorithme est considérablement supérieure à celui obtenu dans les autres algorithmes distribués probabilistes connus.

Pour trouver l'arbre couvrant minimal, notre algorithme fusionne deux algorithmes. Le premier, permet d'assigner des rendez-vous entres les paires de sommets, tandis que le deuxième s'exécute pour relier les arêtes dans lesquels les ont eu lieu et les sommets isolés afin d'obtenir un graphe acyclique qui contient tous les sommets du graphe initial. Cet algorithme peut être considéré comme une version distribué de l'algorithme de *Borůvka* [BVP11].

Cependant, nous enrichissons l'algorithme de *HS* par quelques instructions pour construire un graphe acyclique qui relie toutes les aretes sur lesquelles des rendez-vous on eu lieu et les sommets isolés. Comme prouvé dans [HMR+06], l'algorithme de *HS* offre un nombre maximal de rendez-vous.

La description informelle de notre algorithme est comme suit :

- Pour un graphe donné G, chaque sommet $v \in G$ génère uniformément une variable aléatoire pour chaque voisin. Ces variables aléatoires générées par un sommet pour tous ses voisins appartiennent à l'intervalle réel $[0, 1]$. Elles représentent un temps d'attente pour obtenir un rendez-vous avec ses voisins. Le nombre de ces variables aléatoires générées par tous les sommets est deux fois le nombre d'arêtes (c'est à dire une variable aléatoire pour chaque demi-arête). *L'agenda* d'un sommet est l'ensemble de toutes les variables aléatoires (temps d'attente) proposées par ce sommet à ses

voisins.
- Lorsque l'horloge atteint le temps plus petit de tous les temps généré, un rendez-vous se produit entre le sommet qui génère ce temps d'attente et le voisin pour lequel ce temps est généré. Ainsi, les voisins retirent de leurs agendas, les valeurs générées pour les deux sommets au rendez-vous.
- L'algorithme continue de s'exécuter par les autres sommets.

La description formelle de notre algorithme est donnée dans 6.

Pour chaque sommet $v \in G$, nous définissons les paramètres suivants :
- $t_{v,u}$ est le temps d'attente d'un sommet v pour obtenir un rendez-vous avec un voisin u. Il s'agit d'une variable aléatoire choisie uniformément par le sommet v dans l'intervalle réel $[0, 1]$ pour un voisin u.
- $\mathcal{A}_v$ les agendas de v. C'est l'ensemble de toutes les valeurs générées par v pour ses voisins.
- $\mathcal{A}_{v,u} = \mathcal{A}_v \cup \mathcal{A}_u \setminus \{t_{v,u}, t_{u,v}\}$. C'est l'ensemble des agendas partagés entre deux voisins v et u.
- t_{min} est le temps d'attente minimal du sommet v. Cette valeur est égale à : $t_{min} = min(\mathcal{A}_v)$.
- T_{min} est le temps d'attente minimal de toutes les valeurs générées par les deux sommets voisins u, v. Cette valeur est égale à : $T_{min} = min(\mathcal{A}_{v,u})$.
- $S_{v,u}$ est l'état de l'arête $\{v, u\}$ qui est une variable booléenne initialisée à $false$ (c'est à dire l'arête $\{v, u\}$ au départ n'est pas dans l'arbre couvrant minimal). Quand elle devient à $true$, l'arête $\{v, u\}$ fera partie de l'arbre couvrant minimal.
- S_v est l'état du sommet v. Si l'état de v est $true$, alors le sommet v est dans l'arbre couvrant minimal.
- S^v_{min} est l'ensemble des durées d'attente minimales générées par le sommets v et ses voisins. Cet ensemble est utilisé pour s'assurer qu'aucun cycle n'est créé.
- $msg < valeur, ensemble1, ensemble2 >$ est le format des messages échangés pas les sommets.

Algorithme 6 Construction distribuée d'un arbre couvrant minimal.

1: Tout sommet v exécute en parallèle les instructions suivantes :
2: **Début**
3: $\mathcal{A}'_v := \mathcal{A}_v$;
4: $S^v_{min} := \emptyset$;
5: $t_{min} := min(\mathcal{A}_v)$;
6: **Tant que** $((t < t_{min}))$ ou $(t \neq 1)$)
7: Recevoir $msg < n, \mathcal{A}_u, S^u_{min} >$ De u ;
8: $\mathcal{A}_{u,v} := \mathcal{A}_u \cup \mathcal{A}_v \setminus \{t_{u,v}, t_{v,u}\}$;
9: **Si** $(\mathcal{A}_{v,u} \neq \emptyset)$ **alors**
10: $T_{min} := min(\mathcal{A}_{u,v})$;
11: **Selon** (n)
12: **Cas** (0)
13: $\mathcal{A}_v := \mathcal{A}_v \setminus \{t_{v,u}\}$;
14: **Cas** (1)
15: $S_v := true$;
16: $S_{v,u} := true$;
17: **Si** $(T_{min} \in \mathcal{A}_v)$ **alors**
18: Envoyer $msg < 2, \emptyset, S^v_{min} >$ à u' tel que $t_{v,u'} := T_{min}$;
19: **Sinon**
20: Envoyer $msg < 3, \emptyset, S^v_{min} >$ à u
21: Envoyer $msg < 0, \emptyset, \emptyset >$ aux autres ;
22: **Fin Si**
23: **Cas** (2)
24: **Si** $(S^v_{min} \cap S^u_{min} = \emptyset)$ **alors**
25: Envoyer $msg < 4, \emptyset, \emptyset >$ à u
26: $S_{v,u} = true$;
27: **Fin Si**
28: **Cas** (3)
29: $S^v_{min} := S^v_{min} \cup S^u_{min}$;
30: Envoyer $msg < 2, \emptyset, S^v_{min} >$ à u' tel que $t_{v,u'} = T_{min}$;
31: **Cas** (4)
32: $S_{v,u} = true$;
33: **Fin Si**
34: **Fin Tant que**
35: **Si** $(t = t_{min})$ et $(S_v \neq true)$ **alors**
36: $S_v := true$;
37: $S_{v,u} := true$;
38: Envoyer $msg < 1, \mathcal{A}_v, S^v_{min} >$ à u tel que $t_{v,u} := t_{min}$;
39: Envoyer $msg < 0, \emptyset, \emptyset >$ aux autres ;

Algorithme 6 Construction d'arbre couvrant minimal (suite)

Sinon
($t = 1$) Sélectionner u tel que $t_{v,u} = min(\mathcal{A}'_v)$
Envoyer $msg < 2, \emptyset, \emptyset >$ à u ;
$S_v := true$;
$S_{v,u} := true$;
Fin Si
Fin

Notre algorithme fonctionne comme suit :
Chaque sommet v attend jusqu'à ce que l'un des événements suivants se produit :

- Si v reçoit le message $msg < 0, \emptyset, \emptyset >$ de son voisin u, alors il supprime le temps $t_{v,u}$ de son ensemble $\mathcal{A}_v$.
- Si v reçoit un message de forme $msg < 1, \mathcal{A}_v, S^u_{min} >$ de son voisin u alors il change l'état de S_v et $S_{v,u}$ à *true*. Il y a un rendez-vous entre v et u. Par conséquent, l'arête $\{v, u\}$ appartient à l'arbre couvrant minimal. Dans ce cas, l'un des deux événements suivants se réalise :
  1. Si $T_{min} \in \mathcal{A}_v$ lorsque $T_{min} := min(\mathcal{A}_{u,v})$, alors l'arête $\{v, u'\}$ appartient à l'arbre couvrant minimal. Le message $msg < 2, \emptyset, S^v_{min} >$ sera envoyé à u' pour informer que l'arête $\{v, u'\}$ appartient à l'arbre couvrant minimal. Donc le temps $t_{v,u'}$ sera affecté à T_{min}.
  2. Dans le cas contraire, v envoie le message $msg < 3, \emptyset, S^v_{min} >$ à u pour lui donner la possibilité de choisir l'arête qui sera connectée à l'arête $\{v, u\}$ dans l'arbre couvrant minimal. En même temps, v envoie le message $msg < 0, \emptyset, \emptyset >$ aux autres voisins.
- Si v reçoit le message $msg < 2, \emptyset, S^u_{min} >$ du voisin u, alors si $S^v_{min} \cap S^u_{min} = \emptyset$, ce qui veut dire que l'arête $\{u, v\}$ ne crée aucun cycle, v envoie le message $msg < 4, \emptyset, \emptyset >$ à u et change son état $S_{v,u}$ à *true*.
- Si v reçoit le message $msg < 3, \emptyset, S^u_{min} >$ du voisin u, alors v ajoute l'ensemble S^u_{min} à S^v_{min} et envoie le message $msg < 2, \emptyset, S^v_{min} >$ à son voisin u' qui vérifie $t_{v,u'} = T_{min}$.
- Si v reçoit le message $msg < 4, \emptyset, \emptyset >$ du voisin u, alors v change son état $S_{v,u}$ à *true*. Cependant, ce test est pour garantir qu'aucun cycle ne sera créé en ajoutant l'arête $\{v, u\}$ à l'arbre couvrant minimal.
- Si $t = t_{v,u}$ et $S_v \neq true$, alors v change les états $S_{v,u}$ et S_v à *true* et envoie le message $msg < 1, \mathcal{A}_v, S^v_{min} >$ à u de telle sorte que $t_{v,u} = t_{min}$.
- Sinon (* $t = 1$ ce qui signifie que v est un sommet isolé *), v envoie le message $msg < 2, \emptyset, \emptyset >$ à u tel que $t_{v,u} = min(\mathcal{A}'_v)$. Par conséquent, v change son état $S_{v,u}$ et S_v à *true*.

4.5.1 Analyse de l'algorithme

Espérance mathématique du nombre de sous-arbres couvrants

En général, dans un algorithme distribué, un paramètre important qui mesure l'efficacité, c'est le nombre attendu d'événements qui peuvent avoir lieu dans une unité de temps. En particulier, dans un algorithme distribué probabiliste, un paramètre important qui mesure l'efficacité est l'espérance du nombre d'événements qui peuvent se produire pendant un tour de l'algorithme, voir [MSZ03c]. Dans ce qui suit, nous étudierons ce paramètre pour notre algorithme. Donc, soit $\mathbb{E}(G)$ l'espérance mathématique du nombre de sous-arbres couvrants $N(G)$ et soit $M_e(G)$ une variable aléatoire qui vaut 1 si l'arête $e = \{v, u\}$ est dans l'arbre couvrant minimal, et 0 dans le cas contraire, pour l'arête e dans le graphe $G = (V, E)$.

Proposition 4.5.1. *Initialement, le nombre des sous-arbres couvrant peut être écrit comme suit :*

$$N(G) = \sum_{e \in E} M_e(G). \tag{3}$$

Proposition 4.5.2. *L'espérance mathématique du nombre de sous-arbres couvrant, notée* $\mathbb{E}(N(G))$*, est donnée par la formule suivante :*

$$\begin{aligned} \mathbb{E}(N(G)) &= \sum_{k} kPr(N(G) = k) \\ &= \sum_{e \in E} \mathbb{E}(M_e(G)). \end{aligned}$$

Il est clair que la construction de l'arbre couvrant du graphe G se fait par fusion de plusieurs sous-arbres. Les sous-arbres sont produits essentiellement par les arêtes sur lesquelles les rendez-vous ont eu lieu. Initialement, le nombre de sous-arbres égal au nombre d'arêtes du couplage maximal. Au fil du temps, l'algorithme ajoute à ces sous-arbres couvrants, les sommets isolés et parfois fusionne certains entre eux. A la fin de l'algorithme, tous les sous-arbres couvrants ainsi que les sommets isolés sont fusionnés en construisant un seul arbre couvrant, qui représente *l'arbre couvrant minimal*.

4.5.2 Preuve de minimalité

Il est simple de constater que l'arbre couvrant produit est minimal. En effet, l'algorithme *HS* sélectionne les arêtes comme des sous-arbres couvrants, qui ont les poids minimaux

dans le graphe entier, et de même façon, la deuxième étape de notre algorithme, utilise les arêtes qui ont les valeurs minimal dans graphe pour fusionner les sous-arbres couvrants ainsi que les sommets isolés produits par l'algorithme *HS*. Par conséquent, l'arbre couvrant construit est *minimal*.

4.5.3 Efficacité de l'algorithme $\mathcal{B}$

L'un des paramètres de mesures de complexité, est le nombre de messages échangés. Vu l'importance de ce paramètre, nous nous sommes intéresses à calculer le nombre maximal de messages échangés durant la construction de l'arbre couvrant minimal. Le nombre de messages échangés dépend de la topologie du réseau. Ainsi, nous allons dans la suite caractériser ce nombre pour les topologies les plus utilisées.

1) *Graphes en étoile* : En théorie des graphes, un graphe en étoile S_k est un graphe complet biparti avec un nœud interne et $k-1$ feuilles.

   Soit le graphe $G = (V, E)$ un graphe en étoile, tel que V représente l'ensemble de ses sommets et E représente l'ensemble de ses arêtes. Dans ce cas, le couplage maximal égal à 1, quelque soit le nombre de sommets (pair ou impair).
   - Pour un graphe en étoile avec un nombre de sommets impair, soient $|V| = n$ et $|E| = n-1$ tel que $n \in \mathbb{N}$. Le cardinal de couplages maximale est égal à 1. Il nous faut donc exactement $n-2$ arêtes pour construire l'arbre couvrant minimal.
   - Pour un graphe en étoile avec un nombre de sommets pair, soient $|V| = n$ et $|E| = n-1$ tel que $n \in \mathbb{N}$. Nous avons besoin d'exactement $n-1$ arêtes pour construire l'arbre couvrant minimal puisque le cardinal de couplages maximal est égal à 1.

   La figure 26 montre un exemple de couplages maximal dans un graphe en étoile avec $|V| = 9$.

   Proposition 4.5.3. *La borne supérieure du nombre de messages échangés dans un graphe en étoile pour construire un arbre couvrant minimal, notée $N_{msg}(n)$, est est exprimée par la récurrence suivante :*

$$\begin{cases} N_{msg}(1) = 0 \\ N_{msg}(2) = 2 \\ N_{msg}(n) \leq 2n-1, \qquad \forall n \geq 3. \end{cases} \tag{4}$$

   Où n est la taille du graphe ($|V| = n$).

   Preuve. Soit le graphe $G = (V, E)$ un graphe en étoile ($|V| = n, |E| = m$) et soit $(U_n)_{n\geq 1}$ la suite arithmétique qui vérifie : $U_3 = 5$, et $U_{n-1} + 2$.

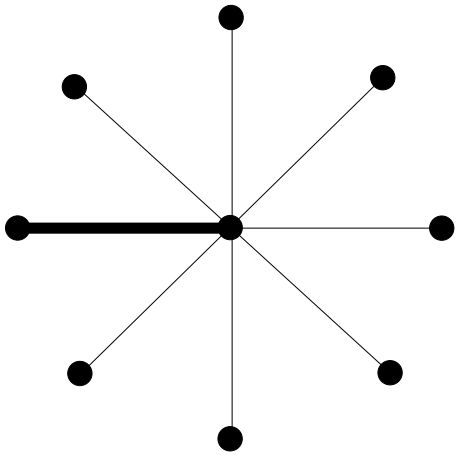


FIGURE 26 – Couplage maximal dans un graphe en étoile

Pour $n = 3$, il est clair que le nombre maximal de messages échangés pour $|V| = 3$ est inférieur ou égal à 5 ($N_{msg}(3) \leq 5$). En ajoutant un nouveau sommet, ce nombre augmentera de 2. Cependant pour $n = 4$, le nombre maximal de messages échangés ($N_{msg}(4)$) sera égal à 7. Supposons maintenant que $N_{msg}(n+1) \leq 2n - 1$ et pouvons que $N_{msg}(n+1) \leq 2n + 1$. Dans le pire des cas en ajoutant un nouveau sommet à G, le nombre maximal de messages échangés augmentera de 2. Donc, $N_{msg}(n+1) \leq 2n - 1 + 2 = 2n + 1$ ce qui prouve la proposition 4.5.3. □

2) *Graphes en chaîne* : Un graphe en chaîne est une suite d'arêtes qui relient un ensemble de sommets distincts les uns aux autres et ils ne forment pas de cycles.

Soit $G = (V, E)$ une chaîne tels que V représente l'ensemble de ses sommets et E représente l'ensemble de ses arêtes. Pour construire un arbre couvrant, plusieurs cas peuvent se présenter :

- Pour une chaîne dont le nombre de sommets est impair, soit $|V| = n$ et $|E| = n - 1$, $n \in \mathbb{N}$. Le cardinal du couplage maximal est égale à $\lfloor \frac{n}{2} \rfloor$. Donc, il nous faut exactement $\lfloor \frac{n}{2} \rfloor$ arêtes pour reconstruire l'arbre couvrant minimal.
- Pour une chaîne dont le nombre de sommets est impair, soit $|V| = n$ et $|E| = n - 1$, $n \in \mathbb{N}$.
  - Dans le meilleur des cas, le cardinal de couplage maximal est égal à $\lfloor \frac{n}{2} \rfloor$. Donc il nous faut exactement $\lfloor \frac{n}{2} - 1 \rfloor$ arêtes pour reconstruire l'arbre couvrant minimal.
  - Dans le pire des cas, il nous faut exactement $\lfloor \frac{n}{2} \rfloor$ arêtes pour reconstruire l'arbre couvrant minimal.

La figure 27 présente un exemple de couplage maximal dans une chaîne avec $|V| = 7$.

Proposition 4.5.4. *La borne supérieure de nombre de messages échangés dans une*

Figure 27 – Couplage maximal dans une chaîne

chaîne, notée $N_{msg}(n)$, est caractérisée par :

$$\begin{cases} N_{msg}(1) = 0 \\ N_{msg}(n) \leq \ 3n-4, \qquad \forall n \geq 2. \end{cases} \tag{5}$$

Où n est la taille de la chaîne ($|V| = n$).
Preuve. Soit $G = (V, E)$ une chaîne ($|V| = n, |E| = n-1$), et soit $(U_n)_{n\geq 1}$ la suite arithmétique qui vérifie : $U_2 = 2$, et $U_n = 3n-4, \forall n \geq 2$.

Pour $n = 2$, il est clair que le nombre maximal de messages échangés est inférieur ou égal à 2 ($N_{msg}(2) \leq 2$). En ajoutant un nouveau sommet, ce nombre s'incrémente de 3. Cependant pour $n = 3$, le nombre maximal de messages échangés ($N_{msg}(3)$) est égale à 5. Supposons maintenant que $N_{msg}(n) \leq 3n-4$ et pouvons que $N_{msg}(n+1) \leq 2n-2$. Dans le pire des cas en ajoutant un nouveau sommet à G, le nombre maximal de messages échangés augmentera de 2. Donc, $N_{msg}(n+1) \leq 3n-4+2 = 2n-2$ ce qui prouve la proposition 4.5.4. □

3) *Graphe en anneau* : Un graphe est dit en anneau lorsqu'il est connexe et que l'ensemble de sommets forment un cycle unique. Un anneau ayant n sommets possède n arêtes.
   - Pour un anneau dont le nombre de sommets est impair, soit $|V| = |E| = n$, $n \in \mathbb{N}$. Le cardinal du couplage maximal est égal à $\lfloor \frac{n}{2} \rfloor$. Donc, il nous faut exactement $\lfloor \frac{n}{2} \rfloor$ arêtes pour reconstruire l'arbre couvrant minimal.
   - Pour un anneau dont le nombre de sommets est pair, soit $|V| = |E| = n$, $n \in \mathbb{N}$. Dans le meilleur des cas, le cardinal du couplage maximal est égal à $\lfloor \frac{n}{2} \rfloor$. Il nous faut exactement $\lfloor \frac{n}{2} - 1 \rfloor$ arêtes pour reconstruire l'arbre couvrant minimal.

   La figure 28 présente un exemple de couplage maximal dans un anneau avec $|V| = 5$.
   Proposition 4.5.5. *La borne supérieure de nombre de messages échangés dans un graphe en anneau, notée $N_{msg}(n)$, est donnée par la formule suivante :*

$$\begin{cases} N_{msg}(1) = 0 \\ N_{msg}(n) \leq \ 4n-6, \qquad \forall n \geq 2. \end{cases} \tag{6}$$

   Où n est la taille de l'anneau ($|V| = n$).

4) *Graphe en double-étoile* : Soit $G = (V, E)$ un graphe en double-étoile tels que $|V| = n$ et $|E| = n-1$. Dans ce cas le couplage maximal est égal à $\lfloor \frac{n}{2} \rfloor$. Pour construire l'arbre

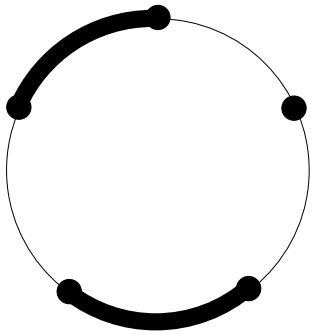


FIGURE 28 – Couplage maximal dans un anneau

couvrant minimal, il nous faut exactement $\lfloor \frac{n}{2} \rfloor$.
La figure 29 présente un exemple de couplage maximal dans une double étoile avec $|V| = 17$.

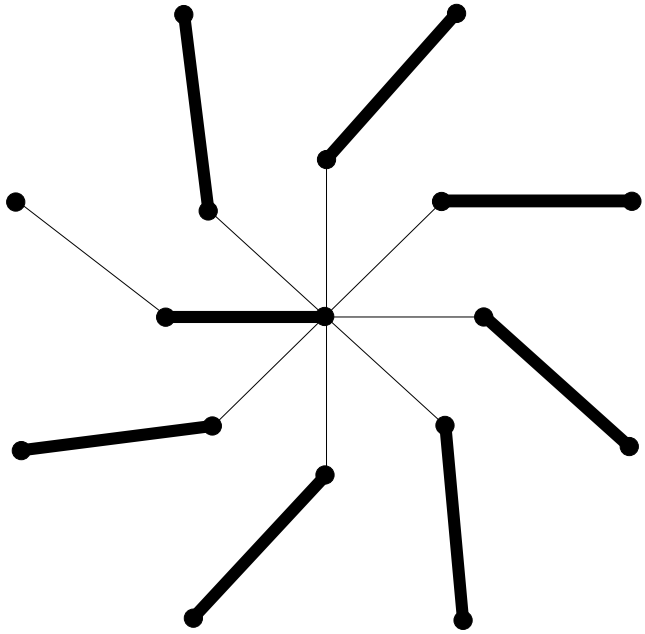


FIGURE 29 – Couplage maximal dans un graphe en double-étoile

Proposition 4.5.6. *La borne supérieure du nombre de messages échangés dans un graphe en double-étoile, noté $N_{msg}(n)$, est donné par l'expression suivante :*

$$\begin{cases} N_{msg}(1) = 0 \\ N_{msg}(n) \leq 3n - 4, \quad \forall n \geq 2. \end{cases} \tag{7}$$

Où n est la taille de graphe.

4.5.4 Nombre moyen de phases

Dans un graphe $G_s = (V_s, E_s)$ des sous-arbres, un processus de construction d'arbre couvrant minimal peut être vu simplement comme un processus de fusionnement successives de sous arbres jusqu'à l'obtention d'un seul arbre.

Le nombre moyen de phase prévu pour réduire le nombre de sous-arbres pour n_s à un est caractérisé dans la proposition suivante.

Proposition 4.5.7. *Soit $G_s = (V_s, E_s)$ un graphe de sous-arbres tel que $|V_s| = n_s$, et soit $\mathcal{X}(n_s)$ le nombre moyen de phases prévues pour obtenir un arbre couvrant. Nous avons :*

$$\begin{cases} \mathcal{X}(n_s) = 1 + \sum_{k=k_{min}}^{k_{max}} Pr(\mathcal{M}(G) = k)\mathcal{X}(n_s - k) \\ \mathcal{X}(1) = 0. \end{cases}$$

où :

- *k_{min} est la cardinalité minimale du couplage maximal,*
- *k_{max} est la cardinalité maximale du couplage maximal.*

Proposition 4.5.8. *Soit $G_s = (V_s, E_s)$ un graphe de sous-arbres de n_s sous-arbres. $\mathcal{X}(n_s)$ Le nombre moyen d'itérations nécessaires pour calculer un arbre couvrant en utilisant notre algorithme de construction est minoré par $\lceil \log_2(n_s) \rceil$ et il est majoré par $n_s - 1$.*

Preuve. Pour évaluer la *borne inférieure*, considérons le cas d'un graphe complet ou un graphique bipartite de taille n_s. Pour ces deux types de graphes, la procédure de Rendez-vous [HMR+06] produit $\frac{n_s}{2}$ Rendez-vous dans la première exécution. Ce qui crée $\frac{n_s}{2}$ fusions : toute arête assignée par un Rendez-vous, il y aura fusion de ses extrémités.

Le graphe résiduel est maintenant avec $\frac{n_s}{2}$ sous-arbres si n pair ou $\frac{n_s}{2} + 1$ si n est impair.

Pour ce graphe, un nouveau appel de la procédure de Rendez-vous [HMR+06] produira $\frac{n_s}{2^2}$ Rendez-vous, et par conséquent $\frac{n_s}{4}$ fusion.

L'appel de l'algorithme de *HS* [HMR+06] continuera tant que le nombre de sous-arbre dans la graphe résiduel est supérieur à 1, c'est-à-dire lorsque $\frac{n_s}{2^h} = 1$. D'où $h = \log_2(n_s)$.

Pour le cas de la *borne supérieure*, il suffit de prendre le graphe G sous forme d'une étoile. Pour ce graphe l'algorithme de *HS* [HMR+06] produit à chaque appel un seul Rendez-vous ; c'est-à-dire fusion de deux sous-arbres à chaque itération. Par conséquent, $n_s - 1$ itérations pour avoir un seul sous-arbres. □

4.5.5 Étude comparative

Le tableau 1 montre une étude comparative entre notre algorithme et d'autres algorithmes connus. Elle présente majeurs critères de comparaison des algorithmes étudiés.

Pour l'ensemble des caractéristiques énumérées par cette table, nous nous sommes assurés que notre algorithme a des bonnes performances en terme de calcul d'arbre couvrant minimal.

- n(resp. m) est le nombre de sommets (resp. *artes*).
- k est le nombre des agents, $k > 2$.
- $\alpha = \alpha(m, n)$ est l'inverse de la fonction d'Ackermann's.
- $\beta = \beta(m, n)$ est le nombre de log-itérations nécessaires pour passer de n à un nombre inférieur à m/n.

4.6 Conclusion

Dans ce chapitre, nous avons présenté et analysé deux algorithmes distribués de construction d'arbre couvrant en se basant sur l'algorithme de *HS* [HMR+06]. Le premier algorithme $\mathcal{A}$ permet de produire un arbre couvrant en général. Cet algorithme ne garanti pas que l'arbre généré est minimal. Ainsi et pour assurer la propriété de minimalité de l'arbre couvrant calculé, le second algorithme $\mathcal{B}$ améliore le résultat de $\mathcal{A}$ et ne sélectionne que l'arbre couvrant minimal.

Par ailleurs et pour mesurer l'efficacité de notre algorithme, nous avons caractérisé le nombre de messages échangés pour construire l'arbre couvrant. Nous avons calculé pour certaines familles de graphes une borne supérieure pour ce nombre.

Pour l'algorithme $\mathcal{B}$, nous avons évalué le nombre d'itérations nécessaires pour construire un arbre couvrant minimal dans le cas général. Nous avons montré que ce nombre est borné par $O(\log_2(n))$ et $O(n)$.

Caractéristiques / Algorithme	**Classique**	**Déterministe**	**Distribué**	**Agents mobiles**	**Probabiliste**	**Randomisé en temps linéaire**	**Complexité**
Borůvka	✓						$O(m\log(n))$
Prim	✓						$O(m+n\log(n))$
Fredman et Tarjan		✓					$O(m\log(\beta))$ graphe non orienté
Gabow et al.		✓					$O(m\beta)$ graphe non orienté
Karger et al.						✓	Temps linéaire prévu $O(m)$
Fredickson							$O(\sqrt{m})$
Chazelle		✓					$O(m\alpha)$
S. Abbas			✓	✓			borné entre n et $O(n\log(k))$ graphe complet
Notre algorithme			✓		✓		borné entre $O(\log_2(n))$ et $O(n)$

Table 1 – Étude comparative entre algorithmes de construction d'arbre couvrant

Chapitre 5

Conception et réalisation d'une plate-forme de simulation d'algorithmes distribués probabilistes

Sommaire

Dans ce chapitre, nous présentons une plate-forme de simulation d'algorithmes distribués. Dans un premier temps, nous traitons des notions relatives aux domaines de la simulation en introduisant les concepts de bases de la simulation. Nous évoquons ensuite, les problématiques de la simulation distribuée ainsi que les différents standards et architectures créés pour mettre en place de telles simulations. Enfin, nous présentons une plate forme de simulation des algorithmes distribués permettant d'implémenter, tester et vérifier les algorithmes distribués codés sous forme de calculs locaux et spécialement ceux décrits sous forme de règles de réécriture.

Le travail présenté dans ce chapitre a donné naissance aux communications nationales et internationales [BSNH15][SBN+15][SBF15], il est en cours de publication (soumis au journal : Computer and System Sciences [BSNH16]).

5.1 Briques de base de la simulation

5.1.1 Généralités et objectifs

Le terme *simulation* est utilisé pour qualifier un ensemble de techniques et de principes permettant l'étude de systèmes réels à partir d'un modèle aussi fidèle que possible, qui reflète au mieux les aspects du système réel que nous voulons étudier, compte tenu du besoin et des contraintes de moyens et de délais de développement [Can01].

La simulation est appliquée dans de nombreux domaines (militaire, aéronautique, spatial, ...) dans le but d'atteindre des objectifs divers tels que le prototypage, l'évaluation de performance, l'aide à la prise de décision ou encore l'entraînement.

Les techniques de simulation proposent des avantages indéniables sur de vraies expériences pour des systèmes réels. Les expériences sur le système réel peuvent être coûteuses, dangereuses voire impossibles. Par exemple, le système réel est inexistant ou encore lorsque l'échelle de temps de la dynamique de son évolution est incompatible avec l'expérimentateur humain (soit beaucoup trop grande ou beaucoup trop petite). De plus, la manipulation facile des modèles (modification de paramètres) et la suppression des effets indésirables (perturbations sur le modèle réel) permettent d'avoir une vision plus complète du système étudié.

En pratique, une simulation requiert au moins les éléments suivants (voir FIGURE 30) :

- Le modèle du système simulé, bien entendu (ex. : un système distribué).

- Les données paramétrant le modèle (accélération maximale, armement, etc.).
- Le modèle décrivant l'environnement du système (bande passante, terrain, météo, émissions électromagnétiques...)
- Les données d'environnement (ex. temps, voisinage, nombre de sommets, ...).
- Le moteur de simulation, qui va gérer le temps et les évolutions du systèmes.
- Le scénario, qui décrit la situation initiale du système (par exemple les sommets simpliciaux) et les événements prévisibles qui surviendront au cours du temps (par exemple l'arrivée d'une formation ennemie à tel endroit, à telle date).

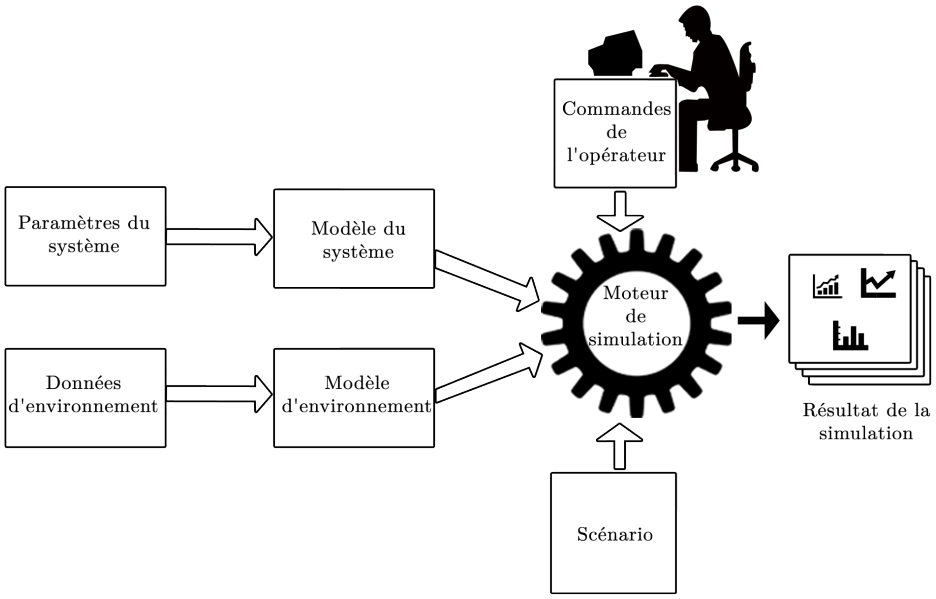


Figure 30 – Schéma des composants d'une simulation

5.1.2 Formalisation de la théorie de la simulation

La formalisation de la théorie de la simulation a été introduite pour la première fois par Bernard P. Zeigler [ZKP00]. Dans ses travaux, il propose un cadre formel regroupant quatre notions essentielles pour le fondement de la simulation : le système source, le cadre expérimental, le modèle et enfin le simulateur (voir Figure 31). Ces différentes notions sont reliées par les relations de modélisation et de simulation.

- Le **système source** est l'objet que nous cherchons à étudier. Il peut être vu comme une source de données observables (phénomène physique ou chimique que nous souhaitons étudier ou un véhicule à concevoir).

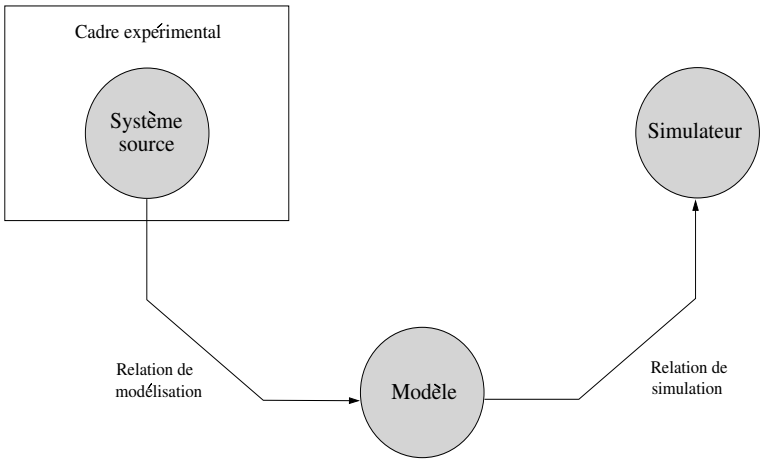


FIGURE 31 – Notions essentielles dans la théorie de la modélisation et la simulation

- Le **cadre expérimental** est une spécification des conditions dans lesquelles le système source est observé. Le cadre expérimental spécifie un contexte particulier correspondant à la situation envisagée pour l'étude. Ce cadre d'étude doit être délimité de façon adéquate afin d'assurer la pertinence du modèle et la précision des résultats obtenus par la simulation.

- Le **modèle** est l'objet sur lequel nous allons travailler afin de réaliser la simulation. Nous le définissons comme l'abstraction du système source vue dans le contexte du cadre expérimental. Par conséquent, il est indispensable d'avoir un modèle cohérent avec ledit système pour obtenir une simulation pertinente. Le choix d'un modèle s'effectue donc en fonction de nombreux critères tels que les objectifs de l'étude ainsi que l'infrastructure (matérielle et logicielle) choisie (ou imposée) pour réaliser la simulation. Les modèles et surtout leurs comportements peuvent être exprimés dans une variété de formalismes reposant sur des concepts et des propriétés mathématiques selon deux dimensions : la dimension temporelle et la dimension spatiale [LK97]. La dimension temporelle peut être continue ou discrète et la dimension spatiale peut être décrite par un ensemble de valeurs fini ou infini. Les différents formalismes pour représenter un système résultent des choix pour les dimensions temporelles et spatiales. Par exemple, le formalisme des équations différentielles est adapté aux modèles à temps continu et à espace d'état infini.

- Le **simulateur** est l'entité capable de modéliser l'évolution du système en fonction des instructions du modèle. C'est un système de calcul qui permet de produire le comportement d'un modèle à partir des règles qui régissent celui-ci. Selon les besoins

du modèle, le simulateur peut être un dispositif technique (matériel et logiciel) plus ou moins évolué. Dans notre cas, c'est un dispositif logiciel.

- La **relation de modélisation** définit les parties du système source qu'il faut représenter et la manière dont elles le seront. La modélisation est donc l'opération par laquelle le modèle est construit à partir du système source et de son cadre expérimental.

- La **relation de simulation** assure que le simulateur reproduit le comportement spécifié en respectant les instructions définies dans le modèle.

5.2 Problème du temps

Le temps est une variable essentielle dans une simulation. Il détermine en effet l'évolution de nombreuses variables d'état (par exemple l'état d'un nœud en fonction de son voisinage et du temps).

Réellement, le temps est une variable continue. Il sera nécessaire de le discrétiser afin d'être traité par ordinateur. Cette discrétisation, artificielle et arbitraire, peut être source d'imprécision et de problèmes de causalité.

Dans le contexte de la simulation, il peut y avoir confusion entre différentes sortes de temps. En effet, cette notion de temps dans une simulation ne repose pas sur un concept unique. Dans la littérature, nous trouvons plusieurs définitions de temps pour une simulation, chacune décrivant un aspect temporel bien spécifique. Richard Fujimoto [Fuj90] propose de différencier ces notions en classifiant trois définitions distinctes du temps :

- *Le temps physique* est le temps de référence du système physique.
- *Le temps simulé* est une représentation du temps physique pendant la simulation. C'est un ensemble de valeurs ordonnées représentant différents instants pour le système physique modélisé.
- *Le temps absolu* ou temps d'horloge est le temps qui s'écoule pendant l'exécution de la simulation. Il peut être mesuré par le simulateur à partir d'une horloge matérielle telle que l'horloge processeur.

La progression de l'évolution de la simulation dans le temps discret peut se faire selon deux approches [Fuj90] : l'approche dirigée par le temps ou synchrone (voir Section 5.3.2) et événementielle ou asynchrone (voir Section 5.3.3).

5.3 Simulation distribuée

5.3.1 Contexte distribué

Les simulations séquentielles étaient fortement limitées par les performances de la machine supportant l'exécution globale de la simulation. La nécessité d'utiliser des modèles (ainsi que leurs comportements) de plus en plus complexes a amené l'émergence de l'utilisation des technologies de distribution pour la simulation. Ainsi, la simulation distribuée consiste à répartir l'exécution de la simulation sur plusieurs nœuds (processeurs ou ensemble de processeurs) à travers un réseau informatique. L'ensemble des simulateurs distribués participe à une simulation globale et peut être vue comme un unique simulateur.

Le support d'exécution de la simulation est donc constitué d'un ensemble de processeurs asynchrones qui s'échangent de l'information par l'intermédiaire de messages. Cette solution permet ainsi de fédérer plus de ressources de calcul afin de simuler plus rapidement ou encore à réaliser des simulations qu'il était impossible de mettre en œuvre de façon séquentielle [RW].

La simulation distribuée consiste donc à éclater le modèle global de simulation en sous-modèles et à simuler chacun de ces sous-modèles (ou processus de simulation) sur un nœud du système distribué. Il faut alors résoudre trois problèmes afin de réaliser une simulation sur un support distribué [Cha12] :

1. **Le placement des tâches dans un système distribué** : Ce problème est souvent ramené à un problème d'optimisation pour lequel nous tentons d'optimiser la charge des nœuds (due à l'exécution des simulateurs atomiques sur les processeurs) ainsi que la communication réseau (due à la taille des messages et à leur fréquence). Nous utilisons alors des techniques telles que le recuit simulé [KG94] qui fut utilisé en premier pour le placement de tâches sur des machines multiprocesseurs [BM91]. D'autres techniques ont été envisagées telles que celles des algorithmes génétiques [Hol92], [HAR94].
2. **L'interopérabilité des simulateurs** : l'interconnexion de plusieurs simulateurs hétérogènes participant à une seule et même exécution implique de mettre en place une homogénéisation entre ces différents simulateurs pour définir des entités standards ainsi que des moyens de communication. Ce support d'interopérabilité est fourni par un standard de simulation distribuée. L'exécution de la simulation est assurée par une architecture de simulation distribuée qui assure le déroulement de la simulation globale avec des simulateurs en concordance avec le standard choisi.
3. **La construction d'un schéma d'exécution temporel** : Ce problème revient à trouver un schéma qui garantisse la progression du temps simulé et assure que les

processus du modèle (les simulateurs) sont exécutés correctement par rapport à ce temps simulé (la relation de causalité est bien respectée). Nous affectons alors une horloge logique du temps simulé à chaque simulateur et nous synchronisons les évolutions de chacune de ces horloges afin d'assurer un écoulement cohérent. Les solutions envisagées sont alors différentes si l'on se place dans le cadre de la simulation distribuée dirigée par le temps ou de la simulation distribuée dirigée par les événements [Mis86].

5.3.2 Simulation distribuée d'un système synchrone

Dans ce cas, de la même façon que pour la simulation séquentielle, toutes les horloges logiques (de temps simulé) progressent de concert. Lorsqu'elles ont la valeur t, les simulateurs exécutent les événements correspondants avant de progresser jusqu'à la valeur $t+T$ (T étant le pas de temps simulé de la simulation globale). L'horloge globale de ce temps simulé est alors répartie par duplication dans chacun des processus distribués. La solution la plus courante est d'utiliser le concept de synchroniseur introduite par Baruch Awerbuch [Awe85]. Plusieurs synchroniseurs ont été envisagés [LT88] [CCGZ90] et se distinguent par leur complexité en temps et en nombre de messages ainsi que sur la technique de distribution (ils réalisent un contrôle plus ou moins centralisé). Généralement, l'évolution des processus est cadencée par une horloge globale dont les cycles sont appelés les pulsations. L'hypothèse de base repose sur l'idée que les délais de transferts et de traitements des messages sont inférieurs à cette pulsation. Ainsi un message émis au début d'une pulsation sera reçu et traité avant le début de la pulsation suivante. Sous ces hypothèses, les synchroniseurs permettent de maintenir une horloge globale dans chaque simulateur local sur une architecture distribuée.

5.3.3 Simulation distribuée d'un système asynchrone

Approches pessimistes

Afin de préserver une cohérence temporelle dans l'entrelacement des événements, l'approche pessimiste (ou encore l'approche conservative) maintient en permanence un ordre correct dans l'exécution des événements. Dans ce cas le système de simulation n'évoluera que si il considère qu'il n'y a aucun problème d'incohérence temporelle. C'est-à-dire que chaque simulateur ne traitera le message que lorsqu'il est sûr de ne pas recevoir un autre message avec une date de traitement inférieure. Cette approche peut conduire à des situations d'inter-blocage entre les processus de simulation particulièrement lorsqu'il existe des boucles de communications entre les simulateurs. Cela nécessite donc de mettre en place des algorithmes afin de prévenir ces inter-blocages (et donc les éviter) ou bien de les détecter et

les résoudre afin de poursuivre normalement l'exécution ensuite.

Approches optimistes

L'approche optimiste (ou approche spéculative) est en contraste avec l'approche précédente. En effet, aucune prévention des fautes temporelles n'est effectuée et le système de simulation évolue jusqu'à ce qu'une incohérence temporelle survienne. Par exemple, un message peut parvenir à un simulateur avec une date de traitement (estampillé sur le temps logique) qui est inférieure à la date de simulation atteinte par ce processus. Dans ce cas, nous effectuons des retours en arrière pour revenir à un instant cohérent et nous reprenons la simulation à ce point. Le principal mécanisme optimiste, nommé le Time Warp, a été proposé par Jefferson [Jef85].

5.4 Générateur des nombres aléatoires

La simulation du *hasard* est essentielle dans le monde de la simulation distribuée. Dans ce cas, le hasard est modélisé à l'aide d'un générateur aléatoire, suivant certains algorithmes. En fait, il s'agit d'un générateur de *nombres pseudo aléatoires*, car ils suivent une série dont les éléments successifs sont liés par des relations mathématiques, de sorte que le comportement des simulations basées sur des modèles aléatoires peut souvent être reproduit. En effet, il est difficile de trouver une définition parfaite d'un « nombre aléatoire ». Nous dirons ici qu'une séquence est aléatoire si chaque terme est imprévisible à celui qui ne connaît pas l'algorithme, et que la répartition des termes soit uniforme et résiste à tous les tests statistiques traditionnels.

La plupart des générateurs aléatoires actuels utilisent un *Générateur à Congruence Linéaire* (GCL), basé sur une formule de ce type :

$$X_{i+1} = (aX_i + c) \bmod m$$

a, c et m sont des constantes entières positives déterminées.

X_0 est appelée le germe (*seed* en anglais). Ce germe va déterminer toute la séquence, de sorte qu'il sera possible de rejouer plusieurs fois une même situation. Les bibliothèques de génération de nombres aléatoires ont généralement une fonction qui permet d'initialiser ce germe à une valeur donnée.

a et c sont respectivement le multiplicateur et l'incrément.

m est le module. Pour des raisons de performances de calcul, nous aurons tendance à choisir une puissance de 2 pour m, par exemple 2^{32}.

Si $c = 0$, nous parlons de GCL multiplicatif (GCLM), qui peut être intéressant pour améliorer les performances, bien que ce soit rarement nécessaire avec la puissance des ordinateurs modernes.

5.4.1 Aspect aléatoire

L'exécution d'un algorithme distribué est non-déterministe : l'ordre de réception des messages et la disparité des délais de communication sont sources d'aléas. Si deux nœuds i et j envoient au même moment un message au nœud k, l'ordre de réception de ces deux messages peut modifier le comportement du nœud k et donc l'exécution future de l'algorithme. Même si deux exécutions consécutives d'un algorithme sur un même réseau fournissent un résultat satisfaisant les spécifications du problème, elles peuvent donc fournir des résultats différents.

Il est très important de noter que nous n'avons pas utilisé la classe `Random` pour générer des nombres aléatoires sinon nous risquons de ne pas générer les mêmes séquences de nombres, ce qui simplifiera le teste de nos algorithmes.... Parmi les exigences des algorithmes distribués, que nous avons présenté, est que les nombres aléatoires générés par les différents processus du réseau devrait être non corrélatifs. Présentement, nous n'avons pas des bonnes méthodes pour produire d'une manière distribuée des nombres pseudo-aléatoires non-corrélatifs.

Pour plus de détail sur la génération des nombres aléatoires, le lecteur pourra consulter l'ouvrage suivant [PTVF07] chapitre 7. Une version électronique est aussi disponible sur le site Numerical Receipes [1].

La classe `generator` contient un générateur uniforme et une graine globale à tous les générateurs. Cette classe est ensuite dérivée afin d'obtenir le générateur voulu à partir de la variable uniforme.

5.5 Les outils existants

Plusieurs plates-formes sont disponibles pour simuler les algorithmes distribués déterministes. Les plus répondus sont : ViSiDiA [MS], LYDIAN [KPT00], VADE [MPT98] et la liste n'est pas exhaustive. La particularité de l'outil que nous présentons par rapport

1. http ://www.nrbook.com/a/bookcpdf.html

à ceux déjà existants est sa disposition d'un environnement adéquat pour simuler des algorithmes distribués probabilistes, et aussi son architecture.

5.5.1 ViSiDiA

ViSiDiA (Visualization and Simulation of Distributed Algorithms) développé par les membres du Laboratoire Bordelais de Recherche en Informatique (LaBRI) de l'Université Bordeaux 1 [2]. Il permet d'implémenter des algorithmes distribués et d'animer leurs exécutions. Son interface graphique permet à l'utilisateur de créer un réseau et de prototyper les algorithmes distribués. Cet outil comprend déjà une bibliothèque d'algorithmes distribués. L'ajout d'un algorithme peut se faire de deux manières : soit l'implémenter à la main en utilisant les fonctionnalités de ViSiDiA, soit le dessiner en utilisant une interface qui permet de créer des règles de réécriture simples avec contextes interdits.

5.5.2 LYDIAN

LYDIAN (An Extensible Educational Animation Environment for Distributed Algorithms) est un environnement pour la simulation et la visualisation des algorithmes distribués qui permet aux utilisateurs d'avoir un environnement expérimental pour tester et visualiser le comportement des algorithmes distribués [3].

LYDIAN possède un créateur de réseau et une autre fenêtre pour l'animation. Celle-ci est basée sur la trace de l'exécution des algorithmes distribués. L'environnement distribué utilise une horloge globale.

5.5.3 VADE

VADE (Visualisation of Algorithms in Distributed Environnements) est un outil développé au Weizmann Institute of Science. VADE permet de visualiser les algorithmes distribués dans un système asynchrone. Il permet, entre autre, à l'utilisateur de visualiser l'exécution d'un algorithme dans une page Web tandis que l'exécution réelle se déroule sur un serveur distant. Ceci permet son utilisation à distance sans avoir besoin de l'installer. Le système distribué utilisé par VADE est le suivant :

- le réseau est fiable ;
- les messages envoyés par un processeur arrivent dans le même ordre de leur envoi ;

2. http ://visidia.labri.fr/
3. http ://www.cse.chalmers.se/research/group/lydian/index.html

- le réseau est asynchrone, chaque processeur possède sa propre horloge et il n'y a pas d'horloge globale.

Pour synchroniser l'animation et l'exécution de l'algorithme, des événements sont envoyés pour modifier les états des composants et ce par l'utilisation du *send synchronization* et du *receive synchronization*.

5.5.4 PARADE

PARADE (PARallel program Animation Development Environnement) est un environnement pour la visualisation et pour déboguer les algorithmes parallèles et distribués. Il se compose de trois parties :

- programme de surveillance et de trace ;
- système de visualisation ;
- utilisation de la trace pour visualiser les actions du programme.

5.6 Architecture et conception de notre plate-forme

Le but cette plate-forme est de simplifier l'implémentation, et de tester les algorithmes distribués à la fois probabilistes et déterministes codés sous forme de calculs locaux et spécialement ceux décrits sous forme de règles de réécriture. L'environnement de ladite plate-forme est décrit par un système asynchrone (pas d'horloge globale, chaque processeur a sa propre horloge) et anonyme (les processeurs n'ont pas d'identité). En plus, il doit être ouvert, c'est à dire, si une information est requise, comme la taille du réseau ou l'identité, les processus peuvent accéder à l'information demandée.

Le simulateur est donc le lien entre l'interface graphique et les algorithmes distribués comme l'illustre la figure 32. Il modélise un réseau des processeurs asynchrones. Chaque processeur communique seulement avec ses voisins par échange de message et il est implémenté par un *Thread Java*. Le simulateur contrôle les échanges de messages entre les *Threads*.

5.6.1 Diagramme de cas d'utilisation

La figure 33 expose le diagramme de cas d'utilisation qui représente la structure des grandes fonctionnalités nécessaires aux utilisateurs de notre plate-forme. En effet les acteurs

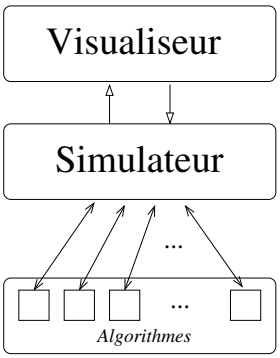


FIGURE 32 – Architecture de notre plate-forme

principaux de notre système qui sont :

- L'utilisateur de l'application qui manipule des différentes fonctionnalités offertes par notre plate forme (dessiner un graphe, choisir un algorithme, exécuter un algorithme...).
- Le développeur d'algorithmes qui a comme tâche de développer de nouveaux algorithmes qui peuvent êtres exécutables dans notre plate-forme en respectant certaines conditions.

5.6.2 Diagramme de classes

La figure 34 présente le diagramme de classes de notre plate-forme de simulation, qui regroupe les différentes classes qui interviennent dans ladite plate-forme. Ce diagramme contient l'ensemble des classes qui composent notre plate-forme. Nous offrons, dans la section 5.7, une description détaillée de quelques classes importantes.

5.7 Réalisation et implémentation

Un algorithme distribué est un algorithme qui s'exécute sur tout nœuds du système distribué. Contrairement à un algorithme centralisé, un algorithme distribué réagit à des événements extérieurs, messages qui lui sont envoyés ou informations lues dans des registres. De la même façon, il génère des événements pour les autres processus (soit par envoie de messages ou par une écriture dans des registres). Un algorithme distribué est donc un algorithme placé sur chaque nœud du réseau qui « réagit » et « génère » des événements. Cette vision locale sur chaque processeur génère la principale difficulté liée à l'algorithmique distribuée, à savoir l'absence d'une vison globale du réseau pour pouvoir prendre des décisions locales.

Figure 33 – Diagramme de cas d'utilisation

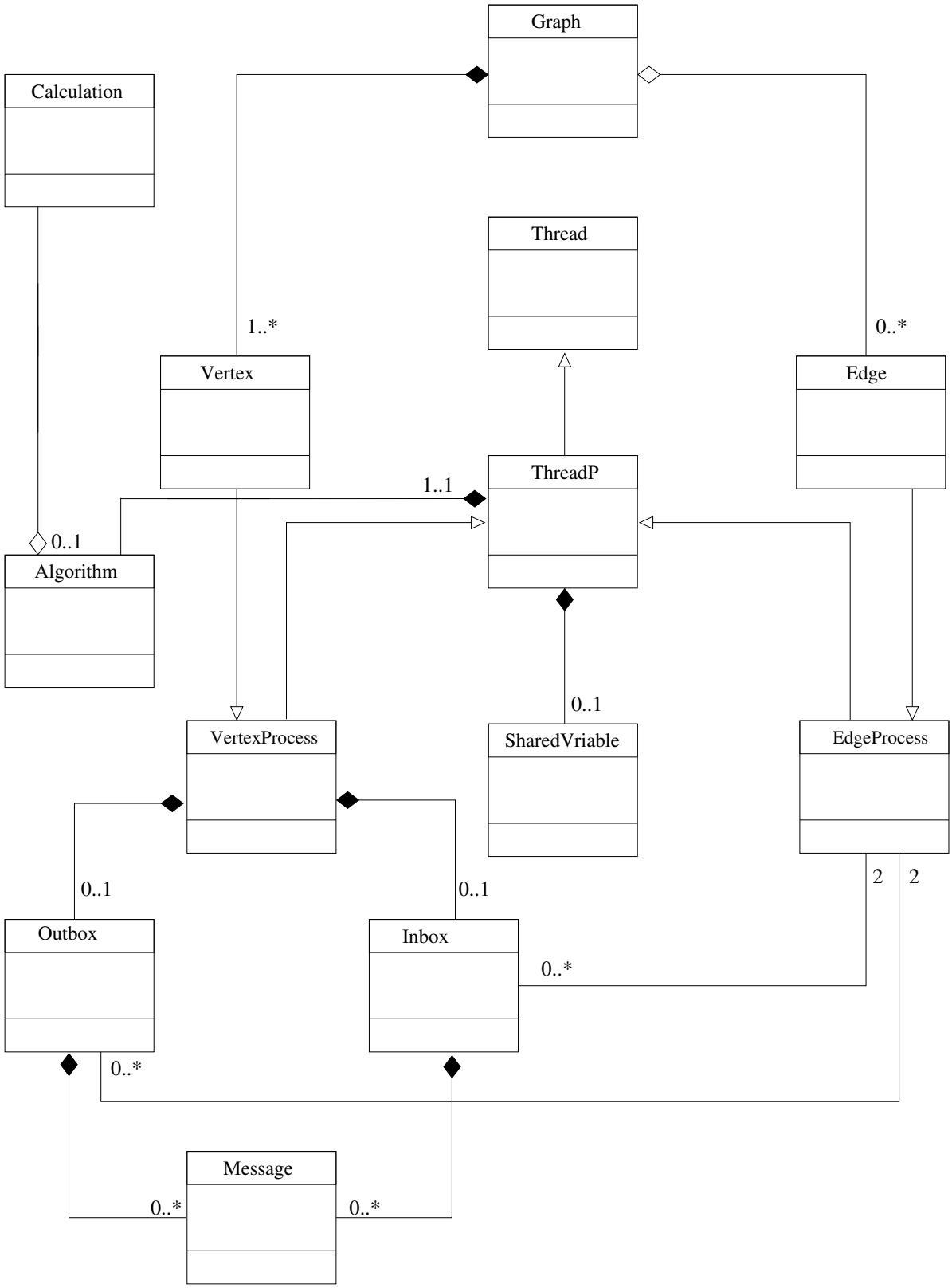


FIGURE 34 – Diagramme de classes

5.7.1 Implémentation

Nous rappelons qu'un tel système est formalisé par un graphe dont les sommets représentent les processeurs et les liens de communication entre deux processeurs sont représentés par les arêtes.

Un algorithme est implémenté en Java et sera dupliqué à chaque sommet du graphe. Nous associons pour chaque sommet v_i un processus java ps_v_i et pour chaque arête u un processus java ps_e_i. Un sommet est implémenté par une classe `Vertex` qui contient les primitifs nécessaires comme son état interne et ses boites aux lettres.

Puisque les communications entre les processus sont basées sur des échanges de messages. Un message est implémenté par une classe `Message` qui comporte les attributs suivants :

- l'attribut `IdMsg` étant l'identificateur du message. Il est unique,
- l'attribut `source` étant l'identifiant du sommet source du message,
- l'attribut `destination` étant l'identifiant du sommet destination du message,
- l'attribut `body` étant le corps de message.

Pour s'assurer que deux nœuds du réseaux n'ont pas des références sur le même objet message, ce dernier est dupliqué (clone) avant d'être mis dans la boite aux lettres du destinataire.

La classe `Message` peut être manipulée par les méthodes suivantes :

- `send(message)` : une méthode qui une fois appelée par un processus sommet, le message à envoyer sera déposé dans la boîte d'envoi.
- `receiveMessage()` : une méthode qui rend la liste des messages reçus par un sommet.

Dans la figure 35 la boite d'envoi du sommet v_1 est partagée entre le processus sommet de v_1, nommé ps_v_1 et les trois processus arêtes associés aux arêtes e_1, e_2 et e_3, nommés respectivement ps_v_1, ps_v_2 et ps_v_3. En effet, le processus sommet accède à cette ressource via sa méthode `send` pour l'envoi des message alors que les processus arêtes accèdent aussi pour chercher s'il y a des messages à acheminer.

5.7.2 Description de l'interface de simulateur

La figure 36 présente l'interface du premier démarrage de notre plate-forme. C'est la partie la plus intéressante de l'interface graphique. De nombreuses tâches peuvent être effectuées dans cette partie parmi lesquelles :

- dessiner un graphe,

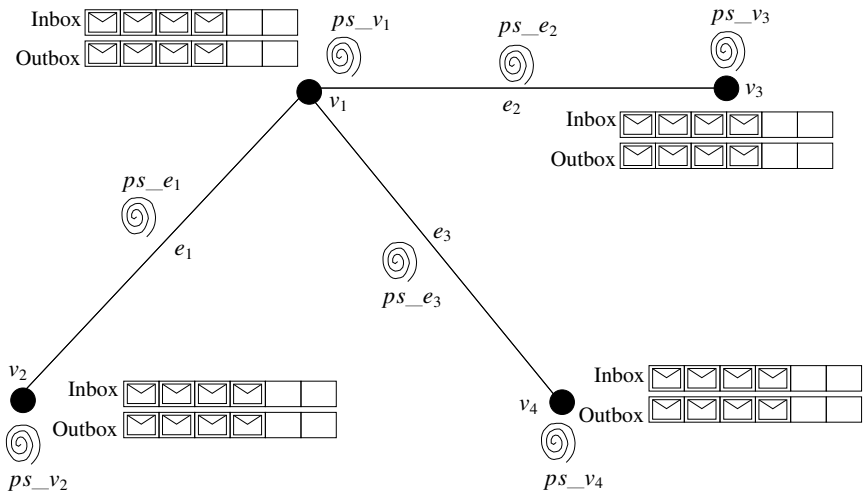


FIGURE 35 – Exemple d'accès aux boites aux lettres

- importer / exporter un graphe (en format GML),
- charger un algorithme,
- lancer la simulation, la mettre en pause et l'arrêter,
- voir en temps réel les messages s'échanger sur les arêtes,
- éditer les étiquettes des sommets...

Une fois l'utilisateur termine l'ajout des sommets, des arêtes et des poids, des processus seront associés à chaque sommet. Le graphe est prêt et la procédure de simulation peut être démarrée. L'étape suivante consiste à choisir un algorithme compilé parmi ceux inclus dans la liste déroulante comme le montre la figure 38.

Ensuite vient l'étape de la simulation proprement dite. Les processus démarrent l'exécution de l'algorithme en invoquant la méthode `run()`. Lesdits processus se déplacent, marquent des arêtes, se répliques ou se suicident conformément à ou aux algorithmes l'on vient de programmer. Ces actions sont représenté graphiquement dans la fenêtre de simulation. Des statistiques paramétrables peuvent également être établies. Pour cela l'utilisateur doit avoir préalablement programmé une classe nommée `Calculation`.

5.8 Conclusion

Dans ce chapitre, nous avons proposé un outil de simulation des algorithmes distribués, ce dernier a une architecture totalement distribuée et ne supposant aucune hypothèse sur le réseau. Plusieurs processus peuvent initialiser une requête de débogage simultanément et n'ont connaissance que de la taille du réseau. Le modèle considéré est le modèle distribué

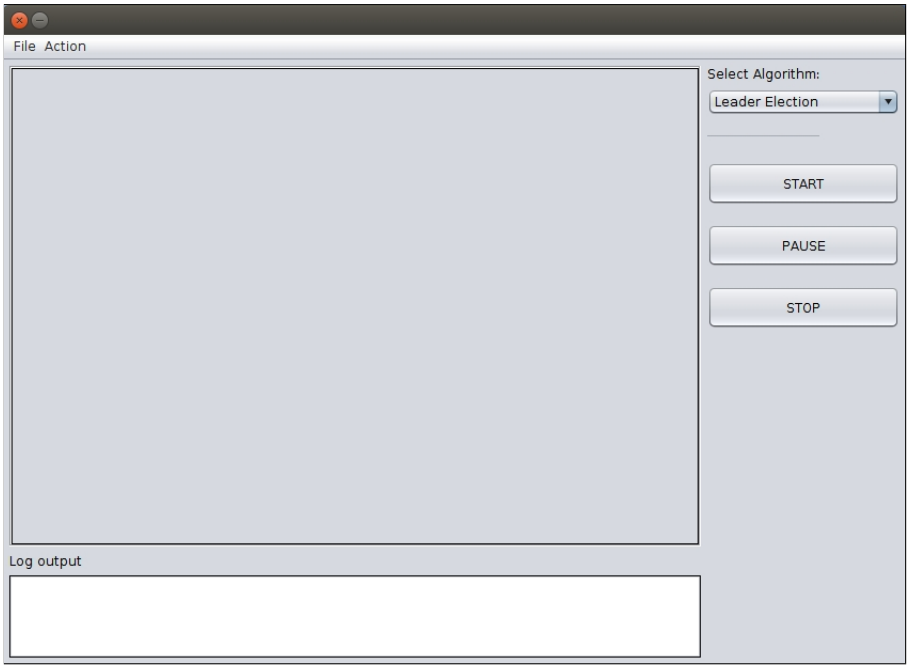


Figure 36 – Interface d'accueil

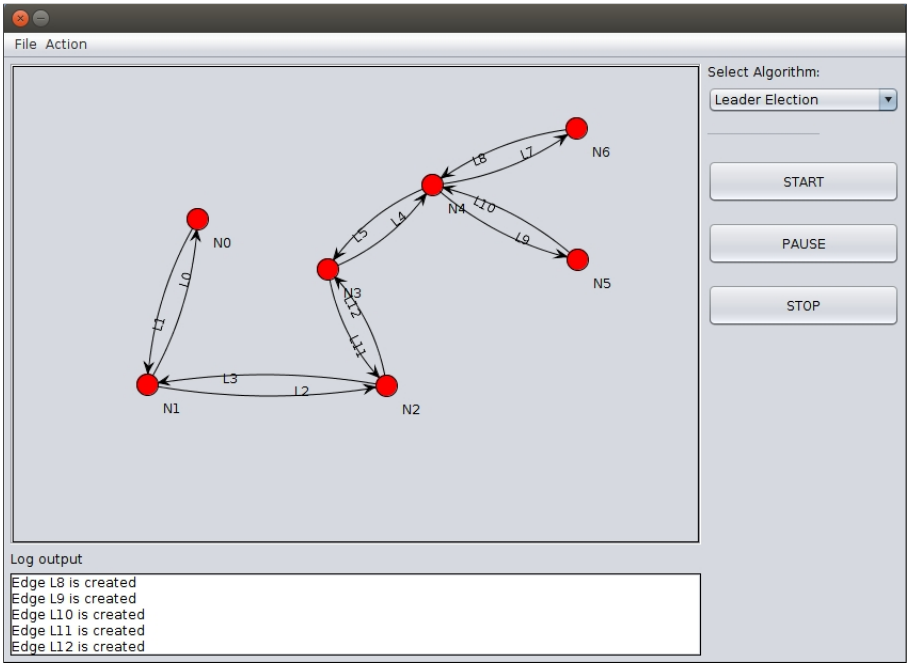


Figure 37 – Dessin de graphes

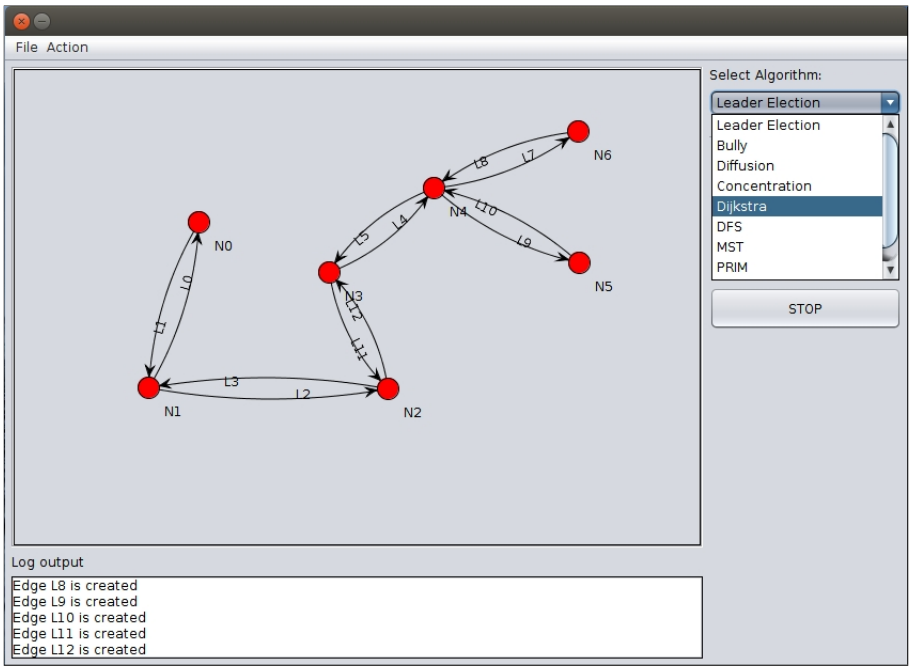


FIGURE 38 – Exécution de l'algorithme

anonyme à base de passage de messages présenté dans ce chapitre. Ainsi, nous avons montré que notre outil dispose d'une architecture unique pour tester, comparer et valoriser des nouveaux algorithmes. De plus, l'interopérabilité des technologies utilisées permet à un utilisateur et/ou développeur de rapidement intégrer cette outil dans son environnement et ainsi d'implémenter et tester les algorithmes distribués.

Grâce à cet outil, notre équipe de recherche, dispose dorénavant d'une plate forme puissante pour implémenter, développer, tester et simuler toute sorte d'algorithmes distribués.

Conclusion et perspectives

Les travaux décrits dans cette thèse s'articulent autour des algorithmes distribués, plus particulièrement la conception et l'analyse de ceux qui sont probabilistes. Dans ce qui suit nous exposons les principales contributions réalisées durant la période de thèse. Ensuite, nous présentons quelques pistes de recherche qui méritent d'être explorées.

En ce qui concerne nos contributions, dans un premier temps nous avons introduit et analysé un algorithme distribué probabiliste d'élection uniforme dans les graphes à grilles triangulaires. Dans un second temps, nous avons proposé et analysé deux algorithmes distribués qui permettent de calculer un arbre couvrant minimal. Pour la validation des algorithmes proposés, nous avons développé une plate-forme de simulation.

Comme première contribution, nous avons présenté un algorithmes distribué probabiliste d'élection uniforme dans la famille des *graphes à grilles triangulaires*. Ainsi, nous avons exposé les règles qui permettent de générer et de manière distribuée de telle famille de graphes. En suite, nous avons mis en place les différentes règles du processus d'éliminations qui constituent la base de notre algorithme. Ces règles sont exécutées en parallèle par tout nœud simplicial du réseau. En fin, nous avons analysé le processus d'élection en le modélisant par un processus markovien de mort en temps continu. Comme résultat principal de cette étude, c'est la preuve que nous avons donné la même chance d'être élu à tous les sommets d'un graphe à grilles triangulaires, c'est à dire, si le graphe est de taille $n > 0$, alors tous ses sommets ont la même probabilité $\frac{1}{n}$ d'être élus.

La deuxième contribution consiste à concevoir et analyser deux algorithmes distribués probabilistes pour la construction d'arbre couvrant minimal. Ces algorithmes sont basés en premier lieu sur l'algorithme de rendez-vous où chaque processus p du réseau génère pour chaque voisin q un nombre aléatoire, choisi uniformément dans l'intervalle réel $[0, 1]$. L'algorithme de rendez-vous permet ainsi de produire k sous-arbres où k est la taille du couplage maximal produit. Chaque arêtes ou un rendez-vous a eu lieu est considérée comme un sous-arbre couvrant. À chaque tour de l'exécution de notre algorithme, les sous-arbres couvrant sont fusionnés. L'exécution continue jusqu'à ce que tous les sous-arbres couvrant sont fusionnés en un seul. Nous avons prouvé par la suite que le graphe résiduel est acyclique, contient tous les sommets du graphe initial et utilise les arêtes de poids minimal. En fin, nous avons présenté une étude détaillée sur le nombre de messages échangés. Cette étude montre que l'algorithme que nous avons proposé présente une meilleure performance que ceux qui nous sommes connus en terme d'échange de messages. Nous avons caractérisé le nombre prévu de phases nécessaires pour fusionner les sous-arbres couvrants de n à 1. Nous

avons prouvé que ce nombre attendu est délimitée entre $O(\log_2(n))$ et $O(n)$.

Enfin et au delà de ces aspects théoriques, nous avons présenté une plate-forme de simulation des algorithmes distribués permettant d'implémenter, tester et vérifier les algorithmes distribués codés sous forme de calcul local et ceux décrits sous forme de règles de réécriture. Nous espérons que cette plate-forme sera utilisée de plus en plus massivement par toute personne qui s'intéresse à l'algorithmique distribuée, étudiant ou chercheur soit-elle. Pour cela, nous poursuivons nos développements, nos améliorations et nos collaborations avec d'autres chercheurs dans le but de proposer un outil de référence.

Par ailleurs, plusieurs pistes nous paraissent pour l'heure intéressantes à explorer notamment le problème d'élection. Il s'agira tout d'abord, de proposer une nouvelle méthode qui consiste à ne pas affecter le même poids initial à tous les sommets du graphe mais un poids quelconque. Ce qui permettrait de guider la distribution de la loi de probabilité d'élection pour tout sommet v.

D'autres directions de recherche nous semblent intéressantes dont l'une concerne l'étude d'un algorithmes distribué probabiliste d'élection dans les graphes aléatoires et l'autre pourrait être la conception et l'analyse d'un algorithmes d'élection uniforme dans la famille de graphes triangulés puisque celle-ci est plus large que celle des graphes à grilles triangulaires.

Nos observations montrent aussi qu'il reste encore un long travail de formalisation rigoureuse pour arriver à proposer une méthode générale et automatique pour la description d'algorithmes distribués avec des systèmes de réétiquetages. Nous pensons que nos travaux futurs dans ce cadre peuvent donner lieu à des résultats aussi puissants qu'inattendus.

Bibliographie

[Ang80] D. Angluin. Local and global properties in networks of processors. In *Proceedings of the 12th Symposium on theory of computing*, pages 82–93, 1980.

[Awe85] Baruch Awerbuch. Complexity of network synchronization. *J. ACM*, 32(4) :804–823, October 1985.

[BDFK95] M. Benantar, U. Dogrusoz, J. Flaherty, and N. S. Krishnamoorthy. Triangle graphs. *Applied Numerical Mathematics*, 17 :85–96, 1995.

[Ber85] C. Berge. *Graphs and Hypergraphs*. North-Holland, 1985.

[BM91] S. W. Bollinger and S. F. Midkiff. Heuristic technique for processor and link assignment in multicomputers. *IEEE Transactions on Computer*, 40, 1991.

[BNRR95] D. Beauquier, M. Nivat, E. Remila, and J.M. Robson. Tiling figures of the plane with two bars, a horizontal and a vertical one. *Computational Geometry*, 5 :1–25, 1995.

[BSNH15] S. Boudakkou, E.M. Stouti, K. Nassiri, and A. El Hibaoui. A simulation platform for distributed algorithms. In *the 17th Journeys of research (December 2015) Abdelmalek Essaadi University.*, 2015.

[BSNH16] S. Boudakkou, E.M. Stouti, K. Nassiri, and A. El Hibaoui. A simulation platform for distributed algorithms. 2016.

[BSV+96] P. Boldi, S. Shammah, S. Vigna, B. Codenotti, P. Gemmell, and J. Simon. Symmetry breaking in anonymous networks : Characterizations. In *Israel Symposium on Theory of Computing Systems*, pages 16–26, 1996.

[BVP11] Gustavo Bergantiños and Juan Vidal-Puga. The folk solution and borüvka's algorithm in minimum cost spanning tree problems. *Discrete Applied Mathematics*, 159(12) :1279 – 1283, 2011.

[Can01] Pascal Cantot. *Introduction à la simulation*. Septembre 2001.

[CCGZ90] Ching-tsun Chou, Israel Cidon, Inder S. Gopal, and Shmuel Zaks. Synchronizing asynchronous bounded delay networks. *IEEE Transactions on Communications*, pages 144–147, 1990.

[Cha12] J.B. Chaudron. *Architecture de simulation distribuée temps réel*. PhD thesis, Université de Toulouse, 2012.

[CMZ04] J. Chalopin, Y. Métivier, and W. Zielonka. Election, naming and cellular edge local computations. In *Proc. of International conference on graph transformation*, volume 3256, pages 242–256, ICGT'04, LNCS, 2004.

[DFJ+98] L. Devroye, A.M. Frieze, M. Jerrum, C. McDiarmid, M. Molloy, R. Motwani, P. Raghavan, B. Reed, and D. Welsh. Probabilistic methods for algorithmic discrete mathematics. Springer, 1998.

[Die00] R. Diestel. *Graph Theory*. Springer, Berlin, second edition, electronic edition, February 2000.

[Dij72] E.W. Dijkstra. Hierarchical ordering of sequential processes. *Operating Systems Techniques, Academic Press,*, pages 72–93, 1972.

[Dij74] E.W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11) :643–644, 1974.

[DIM95] S. Dolev, A. Israeli, and S. Moran. Analyzing expected time by scheduler-luck games. *IEEE Transactions on Software Engineering*, 21(5) :429–439, May 1995.

[Fel50] W. Feller. *An Introduction to Probability Theory and Its Application, Volume I.* John Wiley and Sons, 1950.

[Fel60] W. Feller. *An Introduction to Probability Theory and Its Application, Volume I.* John Wiley and Sons, 1960.

[Fel66] W. Feller. *An Introduction to Probability Theory and Its Application, Volume II.* John Wiley and Sons, 1966.

[FHR72] J.-R. Fiksel, A. Holliger, and P. Rosenstiehl. Intelligent graphs. In R. Read, editor, *Graph theory and computing*, pages 219–265. Academic Press (New York), 1972.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2) :374–382, 1985.

[Fuj90] Richard M. Fujimoto. Parallel discrete event simulation. *Commun. ACM*, 33(10) :30–53, October 1990.

[God02] E. Godard. *Réécritures de graphes et algorithmique distribuée*. PhD thesis, Université Bordeaux I, 2002.

[Gol04] Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*. Elsevier, 2 edition, 2004.

[Gra94] Knuth Donald Ervin Patashnik Oren Graham, Ronald L. *Concrete mathematics : a foundation for computer science*. Addison-Wesley, 1994.

[GSB94] R. Gupta, S.A. Smolka, and S. Bhaskar. On randomization in sequential and distributed algorithms. *ACM Comput. Surv.*, 26(1) :7–86, Mar 1994.

[HAR94] E. S. H. Hou, N. Ansari, and H. Ren. A genetic algorithm for multiprocessor scheduling. *IEEE Trans. Parallel Distrib. Syst.*, 5(2) :113–120, February 1994.

[HMR+06] A. El Hibaoui, Y. Métivier, J.M. Robson, N. Saheb-Djahromi, and A. Zemmari. Analysis of a randomized dynamic timetable handshake algorithm. Technical Report 1402-06, LaBRI, Université Bordeaux 1, May 2006.

[Hol92] John H. Holland. *Adaptation in Natural and Artificial Systems : An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.

[HRSZ10] A. El Hibaoui, John Michael Robson, Nasser Saheb-Djahromi, and Akka Zemmari. Uniform election in trees and polyominoids. *Discrete Applied Mathematics*, 158(9) :981–987, 2010.

[HSdZ+04] A. El Hibaoui, N. Saheb-djahromi, A. Zemmari, A. El Hibaoui, Nasser Saheb-djahromi, Akka Zemmari, and Universite Bordeaux I. A uniform probabilistic election algorithm in k-trees, 2004.

[HSH13] I. Hind, E.M. Stouti, and A. El Hibaoui. Probabilistic distributed algorithm for uniform election in triangular grid graphs. In *the second Meeting of young researchers (May 2013) Faculty of Science, Tetuan.*, 2013.

[HSHD14] I. Hind, E.M. Stouti, A. El Hibaoui, and A. Dahmani. Probabilistic distributed algorithm for minimum spanning tree construction. *Journal of Theoretical and Applied Information Technology (JATIT)*, 67(3), 2014.

[HSKD13] I. Hind, E.M. Stouti, K.E. El Kadiri, and A. Dahmani. Probabilistic distributed algorithm for uniform election in polyo-triangular graphs. In *Journées Doctorales en Théchnologies de l'Information et de la Communication (JDTIC'13), Faculté des Sciences Kénitra, Maroc.*, 2013.

[HSZ05] A. El Hibaoui, N. Saheb, and A. Zemmari. Polyominoids and uniform election. *FPSAC : 17th International Conference on Formal Power Series and Algebraic Combinatorics*, June 2005.

[HW11] Clemens Heuberger and Stephan Wagner. The number of maximum matchings in a tree. *Discrete Mathematics*, 311(21) :2512 – 2542, 2011.

[IJ93] A. Israeli and M. Jalfon. Uniform self-stabilizing ring orientation. *Information and Computation*, 104(2) :175–196, 1993.

[IR90] A. Itai and M. Rodeh. Symmetry breaking in distributed networks. *Information and Computation*, 88(1) :60–87, 1990.

[Jef85] David R. Jefferson. Virtual time. *ACM Trans. Program. Lang. Syst.*, 7(3) :404–425, July 1985.

[KG94] Hermann Kopetz and Günter Grünsteidl. Ttp-a protocol for fault-tolerant real-time systems. *Computer*, 27(1) :14–23, January 1994.

[KOT14] Daniela Kühn, Deryk Osthus, and Timothy Townsend. Fractional and integer matchings in uniform hypergraphs. *European Journal of Combinatorics*, 38(0) :83 – 96, 2014.

[KPT00] B. Koldehofe, M. Papatriantafilou, and Ph. Tsigas. Lydian : An extensible educational animation environment for distributed algorithms. In *5th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education (ITiCSE'2000)*, page 189. ACM press, 2000.

[Lan77] G. Le Lann. Distributed systems – toward a formal approach. In *Proceedings of the IFIP Congress 77*, pages 155–160, 1977.

[Lav95a] C. Lavault. *Évaluation des algorithmes distribués*. HERMES, 1995.

[Lav95b] Christian Lavault. *Évaluation des algorithmes distribués : analyse, complexité, méthodes*. Collection Informatique. Hermès, 1995 (53-Mayenne, Paris, 1995.

[LK97] Averill M. Law and W. David Kelton. *Simulation Modeling and Analysis*. McGraw-Hill Higher Education, 2nd edition, 1997.

[LMS92] I. Litovsky, Y. Métevier, and E. Sopena. Definition and comparison of local computations on graphs and networks. In *MFCS'92*, volume 629 of *Lecture Notes in Comput. Sci.*, pages 364–373, 1992.

[LMS95] I. Litovsky, Y. Métivier, and E. Sopena. Different local controls for graph relabelling systems. *Math. Syst. Theory*, 28 :41–65, 1995.

[LMS99] I. Litovsky, Y. Métivier, and E. Sopena. Graph relabelling systems and distributed algorithms. In H. Ehrig, H.J. Kreowski, U. Montanari, and G. Rozenberg, editors, *Handbook of graph grammars and computing by graph transformation*, volume 3, pages 1–56. World Scientific, 1999.

[LMZ95] I. Litovsky, Y. Métivier, and W. Zielonka. On the recognition of families of graphs with local computations. *Inform. and Comput.*, 118 :110–119, 1995.

[LR81] D. Lehmann and M.O. Rabin. On the advantage of free choice : A symmetric and fully distributed solution to the dining philosophers problem. *Proc. of POPL*, pages 133–138, 1981.

[LR94] D. Lehmann and M.O. Rabin. On the advantages of free choice : A symmetric and fully distributed solution to the dining philosophers problem. Number chapter 20, pages 333–352, Prentice Hall, 1994.

[LT88] K. B. Lakshmanan and K. Thulasiraman. On the use of synchronizers for asynchronous communication networks. In *Proceedings of the 2Nd International Workshop on Distributed Algorithms*, pages 257–277, London, UK, UK, 1988. Springer-Verlag.

[Lyn96] N.A. Lynch. *Distributed algorithms*. Morgan Kaufman, 1996.

[Mar02] Chip Martel. The expected complexity of prim's minimum spanning tree algorithm. *Information Processing Letters*, 81(4) :197 – 201, 2002.

[Mis86] Jayadev Misra. Distributed discrete-event simulation. *ACM Comput. Surv.*, 18(1) :39–65, March 1986.

[MKI11] Yusuke Matsumoto, Naoyuki Kamiyama, and Keiko Imai. An approximation algorithm dependent on edge-coloring number for minimum maximal matching problem. *Information Processing Letters*, 111(10) :465 – 468, 2011.

[MPT98] Y. Moses, Z. Polunsky, and A. Tal. Algorithm visualization for distributed environments. In *Proceedings IEEE Symposium on Information Visualization*, pages 71–78, 1998.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[MS] M. Mosbah and A. Sellami. Visidia : A tool for the VIsualization and SImulation of DIstributed Algorithms. *http ://www.labri.fr/visidia/*.

[MSDZ05] Yves Métivier, Nasser Saheb-Djahromi, and Akka Zemmari. Locally guided randomized elections in trees : The totally fair case. *Inf. Comput.*, 198(1) :40–55, 2005.

[MSDZ09] Jean-François Marckert, Nasser Saheb-Djahromi, and Akka Zemmari. Election algorithms with random delays in trees. *DMTCS Proceedings*, 0(01), 2009.

[MSZ03a] Y. Métivier, N. Saheb, and A. Zemmari. A uniform randomized election in trees (extended abstract). In *Proceedings of The 10th International Colloquium on Structural Information and Communication Complexity (SIROCCO 10)*, pages 259–274. Carleton university press, 2003.

[MSZ03b] Yves Métivier, Nasser Saheb-Djahromi, and Akka Zemmari. A uniform randomized election in trees. In *SIROCCO 10 : Proceedings of the 10th Internaltional Colloquium on Structural Information Complexity, June 18-20, 2003, Umeå Sweden*, pages 259–274, 2003.

[MSZ03c] Yves Métivier, Nasser Saheb, and Akka Zemmari. Analysis of a randomized rendezvous algorithm. *Inf. Comput*, 184(1) :109–128, 2003.

[NL12] Zhan Ning and Wu Longshu. The complexity and algorithm for minimum expense spanning trees. *Procedia Engineering*, 29(0) :118 – 122, 2012. 2012 International Workshop on Information and Electronics Engineering.

[PTVF07] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition : The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 3 edition, 2007.

[Rab82] M. Rabin. N-process mutual exclusion with bounded waiting by 4 log 2 n - valued shared variable. *Journal of Computer and System Sciences*, 25(1) :66–75, August 1982.

[Ray88] M. Raynal. *Networks and distributed computation : concepts, tools, and algorithms*. MIT Press, Cambridge, 1988.

[Ros00] K.H. Rosen. *Handbook of discrete and combinatorial mathematics*. CRC Press, 2000.

[RW] R. Righter and J.C. Walrand. Distributed simulation of discrete event systems. In *Proceedings of the IEEE*.

[SBF15] E.M. Stouti, S. Boudakkou, and Ahmed Faize. Simulation of distributed algorithms. In *First Scientific Edition, (16 and 17 December 2015) , Mohamed First University, Polydisciplinary Faculty of Nador*, 2015.

[SBK14] E.M. Stouti, J. El Bakkali, and K.E. El Kadiri. Probabilistic distributed algorithm for a spanning tree construction based on the handshake algorithm. In *The 2th International Workshop on Software Engineering and Systems Architecture, December 13, 2014.*, 2014.

[SBK15] E.M. Stouti, J. El Bakkali, and K.E. El Kadiri. Probabilistic distributed algorithm for the spanning tree construction based on a handshake algorithm. *Global Journal of Engineering Science and Researches (GJESR)*, 2(3), 2015.

[SBN+15] E.M. Stouti, S. Boudakkou, K. Nassiri, A. El Hibaoui, and K.E. El Kadiri. Design and implementation of distributed probabilistic algorithm simulator. In *Colloque International sur le développement de l'Entrepreneuriat Innovant en milieu Universitaire : Innovation au profit du Développement Régional*, 2015.

[Sed92] R. Sedgewick. *Algorithms in C++*. Addison-Wesley Co., 1st edition, 1992.

[Sel04] A. Sellami. *Des Calculs Lacaux aux Algorithmes Distribués*. PhD thesis, Université Bordeaux I, 2004.

[SF96] Robert Sedgewick and Philippe Flajolet. *An introduction to the analysis of algorithms*. Addison-Wesley-Longman, 1996.

[SHH13a] E.M. Stouti, I. Hind, and A. El Hibaoui. Probabilistic distributed algorithm for spanning tree construction based on the handshake algorithm. In *the 15th Journeys of research (December 2013) Abdelmalek Essaadi University.*, 2013.

[SHH13b] E.M. Stouti, I. Hind, and A. El Hibaoui. Probabilistic Distributed Algorithm for Uniform Election in Triangular Grid Graphs. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 4(6), 2013.

[SHHK14] E.M. Stouti, I. Hind, A. El Hibaoui, and K.E. El Kadiri. Analyse d'un algorithme distribué pour la construction d'arbre courant minimal. In *the 16th Journeys of research (December 2014) Abdelmalek Essaadi University.*, 2014.

[SHK12] E.M. Stouti, A. El Hibaoui, and K.E. El Kadiri. Polyominoïdes and uniform election. In *The first meeting of young researchers RENASSARS-1, Faculty of Science, Tetuan.*, 2012.

[SHKH13] E.M. Stouti, I. Hind, K.E. El Kadiri, and A. El Hibaoui. Algorithme d'election uniforme dans les graphes à grilles triangulaires. In *the second Meeting of young researchers (May 2013) Faculty of Science, Tetuan.*, 2013.

[Tel91] G. Tel. Topics in distributed algorithms. 1991.

[Tel94] G. Tel. Introduction to distributed algorithms. 1994.

[Tel00] G. Tel. *Introduction to distributed algorithms.* Cambridge University Press, 2000.

[Wil91] David Williams. *Probability with Martingales.* Cambridge mathematical textbooks. Cambridge University Press, 1991.

[YK96] M. Yamashita and T. Kameda. Computing on anonymous networks : Part i - characterizing the solvable cases. *IEEE Transactions on parallel and distributed systems*, 7(1) :69–89, 1996.

[YK99] M. Yamashita and T. Kameda. Leader election problem on networks in which processor identity numbers are not distinct. *IEEE Transaction on Parallel and Distributed Systems*, 9(10) :878–887, 1999.

[Zem09] A. Zemmari. *Présentation et Analyse de quelques Algorithmes Distribués Probabilistes.* PhD thesis, Université Bordeaux I, 2009. HdR Habilitation.

[ZKP00] Bernard P. Zeigler, Tag Gon Kim, and Herbert Praehofer. *Theory of Modeling and Simulation.* Academic Press, Inc., Orlando, FL, USA, 2nd edition, 2000.