Intelligence en essaim pour la distribution de simulations dans un écosystème computationnel

Guilhelm Savin

UNIVERSITÉ DU HAVRE

UFR Sciences et Techniques

THÈSE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DU HAVRE

Discipline : Informatique

Guilhelm SAVIN

7 janvier 2014

INTELLIGENCE EN ESSAIM POUR LA DISTRIBUTION DE SIMULATIONS DANS UN ÉCOSYSTÈME COMPUTATIONNEL

Directeur :

Damien OLIVIER — Professeur à l'université du Havre

Co-directeur :

Antoine DUTOT — Maître de conférence à l'université du Havre

Rapporteurs :

Pascal BOUVRY — Professeur à l'université du Luxembourg

Cyril FONLUPT — Professeur à l'université du Littoral Côte d'Opale

Examinateurs :

Paul BOURGINE — Directeur de recherche à l'école Polytechnique de Paris

Nicolas MONMARCHÉ — Maître de conférence à l'université de Tours

Remerciements

Me voici donc à rédiger ces quelques mots qui clôturent la rédaction de ce manuscrit, et qui suscitent en moi à la fois impatience et circonspection ; l'impatience de pouvoir remercier toutes ces personnes avec lesquelles j'ai eu l'occasion d'interagir au cours de mon doctorat. Cependant, je l'écris avec toute la réserve possible de crainte d'en oublier certains. S'il y a bien une partie qui vient du cœur, je pense qu'il s'agit de celle-ci, vous m'excuserez donc, je l'espère, de ne pas l'aborder avec la rigueur d'un quelconque conformisme.

J'ai découvert le monde de la recherche durant mes premières années de fac. C'était pour moi une nouveauté à laquelle mes enseignants de cette époque, pour la plupart cités dans ces remerciements, ont su me donner goût. Par la suite, il m'a été donnée l'occasion de faire un pas de plus dans cet univers et pour cela, je n'en remercierai jamais assez Damien. Merci de m'avoir donné ma chance et d'avoir cru en moi malgré les passes difficiles. Toutes les rencontres sont enrichissantes dans une vie, mais certaines se démarquent des autres. Tu fais, sans conteste, partie de ces personnes à part, qui m'ont permis d'évoluer et d'élargir ma vision du monde. Si je devais retenir une seule des choses que j'ai pu apprendre en te côtoyant, ce serait que rien n'est plus enrichissant que d'être soi-même une entité qui participe, avec d'autres entités au travers d'une collaboration dans la joie et la bonne humeur, à la création d'un tout plus grand que nous. Cette thèse est aussi un travail collectif, il s'agit d'une structure qui émerge de nombreuses interactions et je suis très honoré d'avoir pu faire partie de son noyau à tes côtés. J'espère que les moutons remplacent désormais les avions.

Cette thèse n'aurait sûrement pas pu voir le jour sans Antoine, autre particule du noyau, avec qui je suis resté en contact durant mon année bordelaise et qui m'a motivé pour me lancer dans ce projet. Un énorme merci d'avoir accepté de la co-encadrer. Merci pour toutes les re^{n}lectures dont je t'ai abreuvé en flux continu. Tu fais aussi partie des enseignants qui ont participé à forger mon attirance pour la recherche et qui m'ont permis d'être là aujourd'hui. Plus important encore, merci de m'avoir fait découvrir ces petites choses formidables que sont le maki et le sushi.

La spatialisation, quelque peu inhabituelle, dans laquelle j'ai évolué ces dernières années, m'a apporté une dimension humaine particulière qui a complété l'apport scientifique de ces travaux. Je tiens à exprimer toute mon affection à ma famille d'adoption havraise, Nath et Ced, pour m'avoir fait comprendre que de se sentir chez soi ne dépendait pas tant du lieu, que des personnes qui s'y trouvent. Vous êtes tous les deux de superbes personnes, qui tiennent désormais une place particulière dans mon cœur. Je souhaite que la vie vous

apporte ce que vous désirez.

L'aboutissement de cette thèse n'est possible que grâce aux personnes qui ont accepté de la rapporter et de participer au jury. Un grand merci à Pascal BOUVRY et Cyril FONLUPT pour m'avoir fait l'honneur d'être rapporteurs de cette thèse. C'est un moment quelque peu angoissant d'avoir terminé la rédaction de son manuscrit et d'être en attente des rapports qui vont évaluer les travaux qui y sont présentés et qui correspondent à plusieurs années de travail. Mais quel profond soulagement de découvrir, à leur réception, que cette thèse a été jugée suffisamment digne d'intérêt pour mériter d'être soutenue. Je remercie très sincèrement Paul BOURGINE et Nicolas MONMARCHÉ pour m'avoir eux aussi fait l'honneur d'accepter d'examiner ce manuscrit. Vous faites tous partie des personnes jouant un rôle déterminant dans la conclusion de ces travaux, et je vous en suis profondément reconnaissant.

Merci à Stefan qui, non satisfait de m'avoir appris à équilibrer les arbres de manière stendhaleuse, m'a fait découvrir la réaction hors du commun se produisant lorsqu'un échantillon de Bg est plongé dans une solution aqueuse dans des conditions de température faible. Et plus généralement, merci pour toutes les aventures de la même-pas-cap team. Un grand merci pour ta relecture très constructive de ce manuscrit.

Merci à Yoann pour les nombreux moments partagés. Et il y en a ! D'abord en tant qu'enseignant, puis en tant que thésard collègue de bureau, et désormais en tant qu'ami. Je pense que notre invention du frigo-luge, sport méconnu du grand public, restera dans les méandres oubliés de l'histoire, faute d'avoir prévu un podium adapté.

La plupart des thésards présents au début de ma thèse sont désormais partis. Seuls deux restent, perdus dans les abîmes d'une rédaction qui semblait ne pas vouloir se terminer. Merci à Haïfa pour avoir formé le groupe de vétérans des thésards de la B213 ! Ainsi que pour ta bonne humeur quasi permanente. Je n'oublie pas les différents thésards qui se sont peu à peu éloignés vers de nouveaux horizons : Nizar, Michel, Gaétan, Karim, Franzo, Cédric, Djam, Fahem, et, plus au sud, Frédéric.

Quand tout semble se dérober sous nos pieds, il reste toujours une base solide sur laquelle s'appuyer. Je tiens donc à remercier ma famille pour son soutien tout au long de ces années. En particulier ma mère, Catherine, pour sa chasse aux fautes d'orthographe et ma grand-mère, Christiane, pour m'avoir fréquemment offert le gîte et régalé de raie au beurre noir. Mais aussi mon père Jean-Noël, Stéphanie, et bien sûr, mes sœurettes Laureline et Maïté et mon frère Thibault. J'ai aussi une pensée affectueuse pour mes grand-parents jurassiens, Danièle et Michel, que je ne vois pas aussi souvent que je le souhaiterai. Ainsi que pour mon grand-père, Bernard, qui s'est éteint avant d'avoir pu voir cette thèse terminée. Merci à ma tante Valérie de m'avoir permis de me ressourcer régulièrement là où la terre prend fin. Merci à tous les membres de ma famille que je n'ai pas la place de citer ici, mais qui n'en sont pas moins la cible de mes remerciements. Merci aussi à ma belle-famille, Antoine, Candice, Dominique, Nadine, pour m'avoir souvent accueilli et nourri ; ainsi que pour votre soutien.

Merci aux différents membres de l'équipe RI2C, et plus généralement du LITIS, voire du LMAH ! En particulier, Véronique, toujours souriante et à l'écoute ; Frédéric, avec qui il est vraiment plaisant d'interagir. Mais aussi Éric, et, plus récemment Rodolphe. Sans oublier les irréductibles Cyrille et Aziz, qui ne doivent avoir un badge que pour entrer dans

le bâtiment. J'ai une pensée chaleureuse pour Claire ; ainsi que pour Laurent, Dominique, Jean-Luc, Guillemette, Pierrick, Guillaume, et Arnaud.

Merci aux amis avec qui j'ai pu passé régulièrement de bons moments. En particulier le clan des amis de toujours, Cédric, Jérémy, François, Mathieu, Antony, Aline, Élise ; Cédric et Aline ont aussi eu à subir mes squattes réguliers, au détriment de certaines madeleines dont mes papilles en remercient encore la cuisinière. Le clan Chateau-Corson, Yann, Rachel, Antoine, Paula, Marie, Philippe, Catherine. Le clan voileux, dont la première rencontre se fit au milieu de délicieuses effluves de fromage, sous la supervision intransigeante de Nath ; merci donc à Clem, Tanguy, Benj, Romain, Julie. Sans oublier Vesela, Raphaël, Fabienne, Gaby, Rafika, Caroline, Maya. Et le clan d'Oc, Xavier, Manu, Carine, Benjamin, Mélanie, Christine, Gérard, Corinne, Isabelle. Je m'excuse par avance auprès de ceux que j'ai oublié, car il y en a très certainement !

Je remercie également ceux qui suivent les chemins d'Iwama. J'ai pu, grâce à vous, trouver une paix intérieure qui a contribué à me donné la force de terminer. Merci donc à Jean-Louis, pour tes discussions enrichissantes, et pour m'avoir poussé à finaliser ce manuscrit. Merci à Alexis, Virginie et Christelle, mais aussi à tout les autres. Namasté.

Merci à la SNCF, pour m'avoir gracieusement offert des heures gratuites à bord de ses trains, même si j'aurais aimé pouvoir en contrôler le moment pour en profiter. Merci au Trappiste, à la Petite Rade, au Miyaki et aux kebabs.

Enfin, une place spéciale pour quelqu'un de spécial, ma dulcinée, Solenne, qui me fait la joie de partager ma vie depuis bientôt huit années. Désolé pour les absences, et l'humeur parfois (souvent ?) irascible. Mais surtout, merci pour ta présence, ton soutien, et ton amour. Il va être temps pour nous de prendre de la hauteur pour continuer de découvrir de nouveaux horizons, peuplés d'étoiles encore inconnues. Avec tout mon amour.

Merci à vous, qui parcourez cette page et peut être les suivantes, car, ce faisant, vous contribuez à leur donner un sens.

À ma puce Daelys, qui a vu le jour au début de cette thèse,
et qui m'a accompagné toutes ces années.

Tous ceux là, ce sont des individus, qui ne sont pas des dieux bien sûr, ce sont des hommes, mais ces hommes, ils communiquent les uns avec les autres, ils mettent en commun, ils créent quelque chose qui est supérieur à eux, par leurs liens, par leur mise en commun, par leurs échanges, ils font plus qu'eux même et au fond c'est toute la réalité de notre univers, que chaque fois que l'on assemble, on fait apparaître autre chose, on fait apparaitre de l'inattendu ; et bien quand on assemble des hommes, quelque chose de nouveau apparait.

Albert Jacquard, Noms de dieux, 1994

Table des matières

Table des figures

Liste des Algorithmes

Première partie

Introduction générale

Préambule

Ce manuscrit présente nos travaux sur la distribution de simulations de système complexe. À une autre échelle, nous considérons l'ensemble des entités de la simulation, et l'infrastructure matérielle utilisée pour la distribution, comme un système complexe à part entière. Nous effectuons le parallèle avec les écosystèmes naturels afin de mettre en évidence que nous sommes en présence d'un environnement (le système distribué) qui fournit des ressources (calcul, mémoire, *etc*...) à un ensemble d'organismes (les entités) afin de leur permettre de se développer, de s'exécuter mais aussi d'interagir. Ces éléments nous amènent à penser en terme d'écosystème computationnel désignant l'ensemble de notre système distribué dédié aux simulations de système complexe. Au travers de cette vision nous cherchons à dégager des propriétés souvent propres aux systèmes naturels que l'on souhaiterait obtenir, sur le long terme, pour ce type de système distribué : robustesse, adaptabilité, résilience.

Notre travail n'est qu'un pas dans cette direction. Nous proposons de centrer ce dernier autour des interactions et, à cette fin, de modéliser les simulations sous la forme d'un graphe dynamique les représentant. Cela nous amène à réfléchir sur les différentes façons d'aborder la dynamique, mais aussi la lecture de ce modèle. En effet, nous cherchons dans ce graphe des motifs, c'est à dire des organisations, qui vont nous permettre d'obtenir un plus haut niveau de représentation du graphe et, éventuellement, de gouverner le système de façon à en optimiser les propriétés qui lui permettront de résister et de s'adapter aux perturbations. Ces éléments sont complétés par le développement d'une bibliothèque de graphe dynamique permettant à la fois la modélisation, mais également des mesures locales, globales ou structurelles. À cela s'ajoute un intergiciel offrant aux simulations la possibilité d'utiliser les informations collectées au niveau du modèle et la possibilité de migrer les entités composant ces simulations afin d'aller vers une distribution efficace.

Nous avons choisi de diviser la présentation de ces travaux en quatre parties. La première partie, nommée « Introduction générale », nous permet de développer le cadre global des travaux qui seront présentés dans les parties suivantes. Ainsi, un premier chapitre « Contexte » aborde les systèmes complexes et les réseaux d'interactions ; puis nous détaillons la notion de système distribué, suivi d'une analogie entre un système distribué dédié aux simulations de système complexe et un écosystème naturel. Nous présentons ensuite, dans un second chapitre, notre « Problèmatique » qui introduit les contraintes liées à la modélisation des interactions et à la dynamique. D'une part la dynamique du réseau d'interactions, et d'autre part celle de l'environnement de distribution.

Dans la seconde partie, « Positionnement », nous présentons l'état de l'art des différents problèmes abordés. Ainsi le chapitre « Réseaux d'interactions » présente la modélisation d'un réseau d'interactions sous forme de graphe dynamique. Différentes approches de la modélisation de la dynamique sont abordées. Puis nous nous intéressons aux structures qui apparaissent dans ce graphe, ainsi qu'aux processus qui vont participer à la dynamique du graphe. Le chapitre suivant aborde la « Répartition de charge » sous l'angle du partitionnement de graphes, d'abord de manière générale, puis en s'appuyant sur les différentes approches statiques et dynamiques. Enfin, le chapitre « Calcul distribué » expose l'architecture générale de l'infrastructure d'un système distribué. Ce chapitre nous permet de décrire différents types de systèmes distribués tels que les grappes de calcul, les grilles, le cloud computing, pour terminer sur les écosystèmes computationnels ; et d'exposer une première approche sur la manière d'envisager l'algorithmique dans un tel système.

La troisième partie, « Modèle », est dédiée aux modèles que nous proposons pour représenter les interactions entre les entités et les organisations. Nous consacrons le premier chapitre de cette partie à la « Modélisation des interactions », en décrivant leur importance au sein du système, leur rôle dans l'émergence d'organisation, ainsi que certaines de leurs propriétés. Nous nous intéressons à la « Modélisation des organisations » dans le chapitre suivant, et plus particulièrement aux différentes échelles de représentation du système. Puis, nous verrons des « Détection des centroïdes » que nous pouvons appliquer à nos modèles afin d'augmenter la robustesse du système. Nous décrivons, en particulier, une mesure de centralité, basée sur une approche à base de fourmis numériques, dont le but est de détecter les centroïdes des organisations. Enfin, nous terminerons cette partie par une présentation de notre « Outil de modélisation », GraphStream, dont le but est de permettre la manipulation de la dynamique d'un graphe.

Enfin, la dernière partie « Distribution » expose notre contribution à la répartition dynamique et décentralisée de charge, ainsi qu'à la modélisation de l'environnement de distribution. Le chapitre « Répartition de charges », quant à lui, montre les enjeux de la répartition dynamique de charge. Il développe l'algorithme AntCo^2 qui est notre approche à base de colonie de fourmis utilisée pour détecter des organisations dans le graphe dynamique modélisant les entités. Ces organisations nous permettent de répartir les entités de manière à équilibrer la charge et à réduire la bande passante du réseau. Cet algorithme peut cependant conduire à l'oscillation de certaines entités entre plusieurs organisations, ce qui conduit à des migrations intempestives venant dégrader les performances globales du système. Nous proposons des méthodes de lissage pour permettre de réduire ces oscillations, suivies de résultats. Nous présentons dans un dernier chapitre « Intergiciel » notre contribution à la modélisation de l'intergiciel d'un écosystème computationnel. Ce modèle propose une architecture décentralisée, où les entités peuvent communiquer de manière asynchrone.

Nous terminons sur une discussion qui nous permet de conclure ce manuscrit et de présenter les perspectives qui pourront définir une suite future à ces travaux.

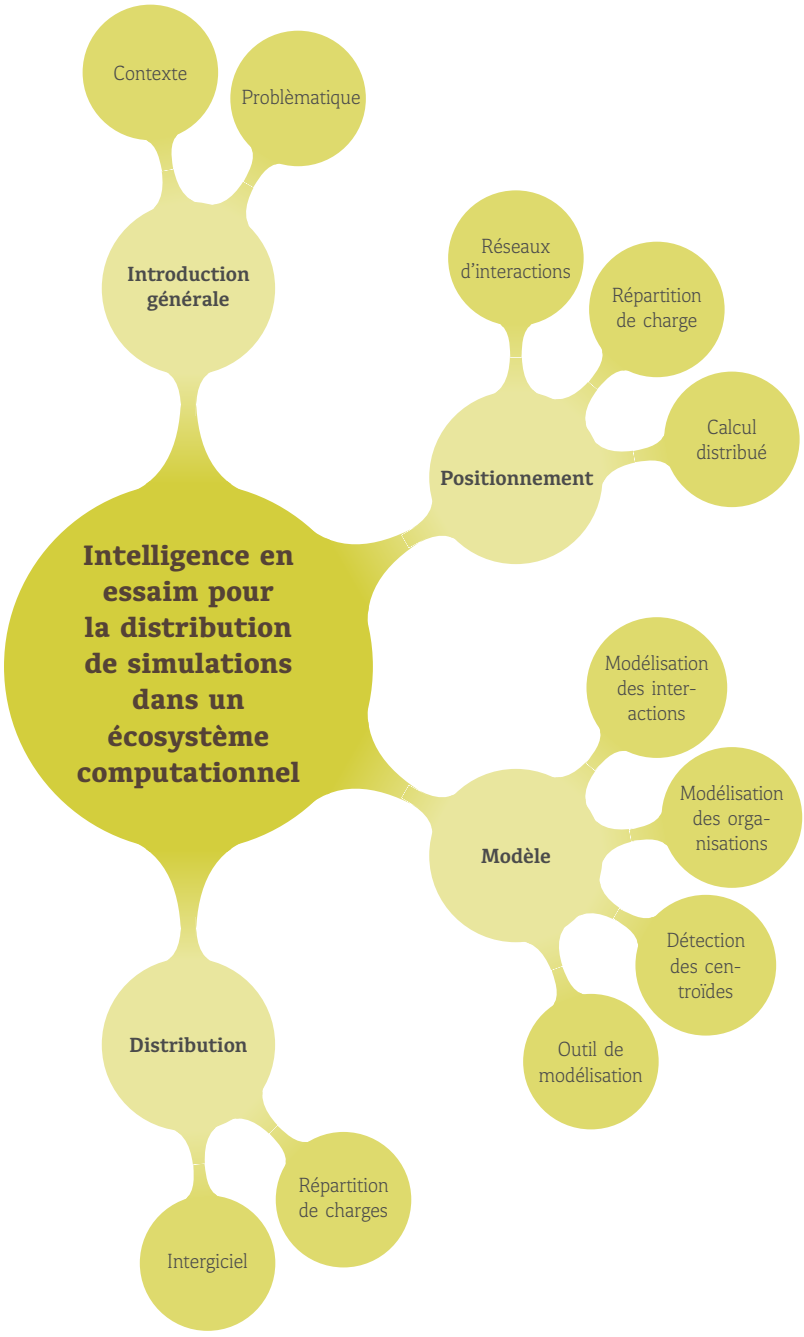


FIGURE 0.1 : Yet Another Table of Contents

CHAPITRE 1

CONTEXTE

Nous nous plaçons dans le cadre d'étude des systèmes complexes qui n'est pas sans poser de problème étant un domaine dynamique et foisonnant. En effet, il n'offre pas de définition consensuelle : certains abordent le problème à l'aide d'exemples tels que le vol en essaim (cf. figure 1.3), le trafic routier et ses embouteillages, ou encore les sociétés d'insectes (cf. figures 1.1 et 1.2)... D'autres l'abordent à l'aide de formulations généralistes telles que « *le tout est plus que la somme des parties* » qui bien qu'intéressantes ne peuvent constituer un cadre formel.

Il n'est pas dans nos prétentions dans cette introduction de proposer une définition qui lèverait tous ces manques et comblerait toutes les ambiguïtés. Notre propos se situe dans le contexte systémique où les interactions s'expriment et définissent des propriétés locales et globales à différents niveaux d'échelle éventuels.

1.1 Systèmes complexes

Les systèmes complexes sont un des composants de bases de nos travaux et pourtant, il est difficile d'en fournir une définition formelle et consensuelle, qui serait valable pour toutes

FIGURE 1.1 : Fourmis mangeant un morceau de pomme. Image sous licence Creative Commons 3.0 par Zainichi Gaikokujin

les personnes travaillant sur le sujet, du fait que le terme même de complexité est polysémique. La difficulté réside d'une part dans les multiples disciplines utilisant le paradigme des systèmes complexes qui rendent nécessaire l'utilisation d'un vocabulaire commun, et d'autre part dans la compréhension que nous avons du terme complexité. En tant qu'informaticiens, nous sommes habitués à manipuler des complexités qui expriment le temps de calcul nécessaire à un algorithme pour s'exécuter, la quantité de mémoire nécessaire à son exécution, ou encore la complexité aléatoire de Kolmogorov [KOLMOGOROV 1965] qui définit la taille du plus petit algorithme (en nombre de caractères) nécessaire à la production d'une donnée de sortie à partir de données d'entrée. Les systèmes complexes rentrent dans un champ multidisciplinaire ce qui implique que la compréhension de la complexité ne peut se restreindre aux définitions du domaine de l'informatique seul qui se limitent à la théorie de l'information et ne capturent pas des dimensions essentielles. Nous allons donc tenter dans la suite de fixer les notions de systèmes et de leur complexité tels que nous les appréhendons.

D'une manière générale, un système est un environnement, délimité du milieu extérieur

FIGURE 1.2 : Essaim d'abeilles sur un arbre. Image sous licence CC BY-NC par Firooo2/Flagstaffotos

par une membrane, dans lequel un ensemble d'éléments va avoir la possibilité d'interagir et éventuellement de s'organiser. Cet environnement et ces éléments sont régis par un certain nombre de contraintes qui vont permettre de caractériser le système. Les interactions peuvent être fixes comme dans le cas d'un moteur constituant un système électrique, ou libres, mais dans ce cas soumises à des contraintes comme par exemple un écosystème naturel. Nous verrons par la suite que cette membrane qui isole l'environnement peut être ouverte ou fermée en fonction du fait qu'elle permette ou non des échanges avec le milieu extérieur. Les éléments, que nous nommerons par la suite *entités*, ont la possibilité d'interagir entre eux mais aussi avec l'environnement. Celui-ci peut d'ailleurs servir d'intermédiaire à une forme d'interaction indirecte comme par exemple la *stigmergie* chez les termites constructeurs [GRASSÉ 1959]. La stigmergie est un mode de communication qui consiste à déposer un message dans l'environnement qui déclenchera ensuite une réaction de la part

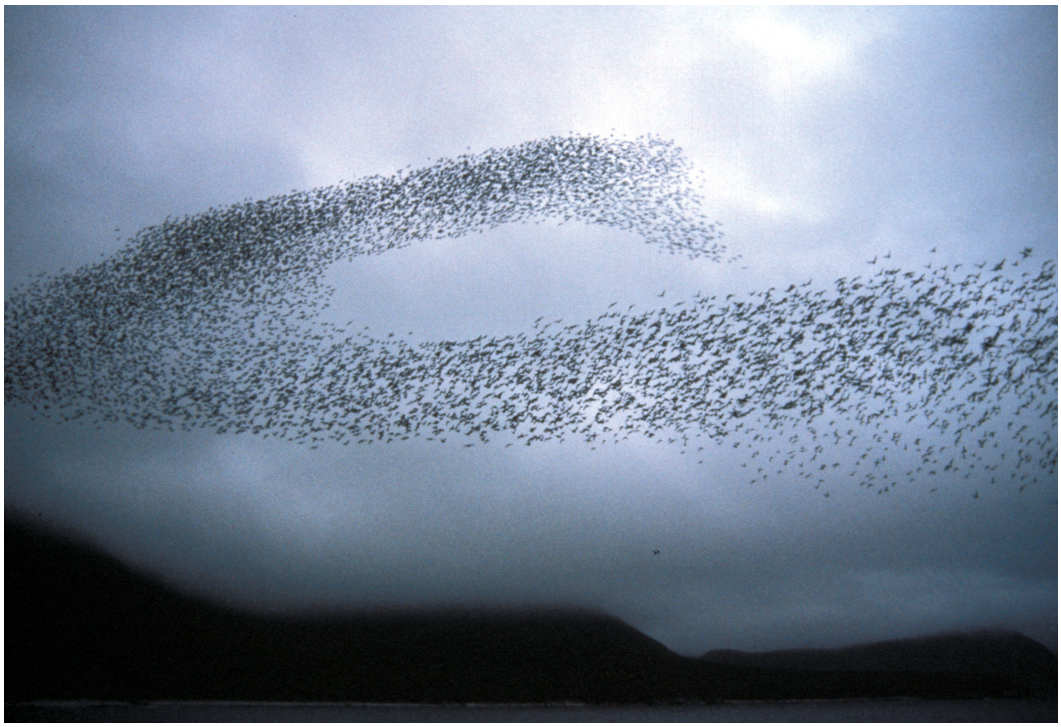


Figure 1.3 : Nuée d'oiseaux marins aux îles Shumagin. Cette image est dans le domaine publique

des entités qui en prendront connaissance. Des interactions entre les entités peut se dégager de l'organisation entre les entités, et de ce fait, on ne peut limiter le système à la somme de ses parties. Le système est au contraire formé par l'environnement, les entités, leurs interactions ainsi que des propriétés qui émergent du comportement et des interactions des entités.

On retrouve cette notion d'organisation formant un *tout* dans la définition de système donnée dans [Saussure 1931] :

> « Totalité organisée, faite d'éléments solidaires ne pouvant être définis que les uns par rapport aux autres en fonction de leur place dans cette totalité. »

Un système peut être *ouvert* ou *fermé*. Lorsqu'il est ouvert, la membrane qui délimite l'environnement du système est poreuse et laisse donc passer des flux de l'extérieur du système qui viennent perturber le système. Le système peut émettre en retour des flux vers l'extérieur. Ces flux qui *traversent* le système et sur lesquels le système n'a ni contrôle ni possibilité de prédiction absolue cassent le déterminisme du système et vont être responsables de phénomènes d'auto-organisation qui peuvent apparaître. Le système suit alors une trajectoire qui est *non-déterministe*. La définition d'un système fermé découle naturellement de la précédente comme étant son contraire : la membrane du système est hermétique et les flux traversant l'environnement se cantonnent donc à ceux générés par le système lui

même. Un système totalement fermé relève du concept. En effet, la nature ne possède pas d'exemple de tels systèmes et leur observation en serait d'ailleurs impossible : comment observer quelque chose qui ne se laisse pas traverser par la moindre particule et qui n'en émet aucune ? C'est pourquoi dans ces travaux nous considérerons que le système est ouvert afin d'être au plus proche de la réalité des systèmes naturels dont nous nous inspirons.

Le système que nous considérons peut donc être assimilé aux systèmes dissipatifs tels que formulés par Ilya Prigogine et Isabelle Stengers dans [Prigogine et Stengers 1996]. Ces systèmes ouverts sont traversés par des flux d'énergie et de matière provenant de l'environnement extérieur et convertissent une partie de ce flux en informations afin de créer de l'ordre à l'intérieur du système. Cette absorption d'énergie conduit à l'augmentation de l'entropie globale du système, qui est dissipée par des phénomènes d'auto-organisations.

La complexité repose sur plusieurs choses. Tout d'abord sur l'appréhension que l'on peut avoir sur les entités et leurs interactions. En effet, la quantité de ces éléments peut être telle qu'il devient difficile voire impossible, que ce soit pour un observateur humain ou même pour une machine, de considérer la totalité des éléments du système et de ce fait de pouvoir étudier le système par des approches globales. Elle est aussi liée au fait que le système soit *plongé dans le temps* [Coquillard et Hill 1997]. Ce temps va définir une succession d'états du système. Il peut être continu, c'est à dire que peu importe deux temps réels t_1 et t_2 auxquels correspond un état du système, on peut trouver un état au temps t_3 tel que $t_1 < t_3 < t_2$. Au contraire, le temps peut être discret : l'état du système n'est défini que pour certaines valeurs de temps qui sont séparées par un intervalle constant ou non. Dans ces travaux, nous nous focaliserons sur une approche par *événements discrets* qui consiste à considérer les modifications qui surviennent dans le système plutôt que le temps lui même [Zeigler, Praehofer et Kim 2000]. Cette temporalité, qu'elle soit événementielle ou non, mène donc au fait que l'on observe une dynamique des différents composants du système, que ce soit l'apparition ou la disparition d'entités ou d'interactions entre les entités, ou encore la dynamique des propriétés du système. Le Moigne dans [Le Moigne 1994] intègre cette notion d'évolution dans sa description du *Système Général* :

> « un objet qui, dans un environnement, doté de finalités, exerce une activité et voit sa structure interne évoluer au fil du temps, sans qu'il perde pourtant son identité unique. »

Du fait des interactions entre les entités et de la temporalité, on observe des *phénomènes* qui créent de l'organisation au sein du système et qui peuvent intervenir sur différentes échelles, spatiales ou temporelles. Il est donc nécessaire d'être capable d'effectuer un changement d'échelle, grâce à la détection des organisations, pour pouvoir observer à un plus haut niveau d'organisation. Ces phénomènes peuvent survenir à l'échelle microscopique sur une période de temps suffisamment courte pour pouvoir considérer le phénomène comme *instantané*. Ils oscillent autour d'un point d'équilibre. Ces phénomènes, que l'on qualifiera de *synchroniques*, bien qu'agissant à l'échelle microscopique ont des conséquences sur l'échelle macroscopique par leur répétition qui renforce d'autres phénomènes, qualifiés de *diachroniques*, qui agissent à une échelle macroscopique. L'action de ces derniers s'effectue sur une

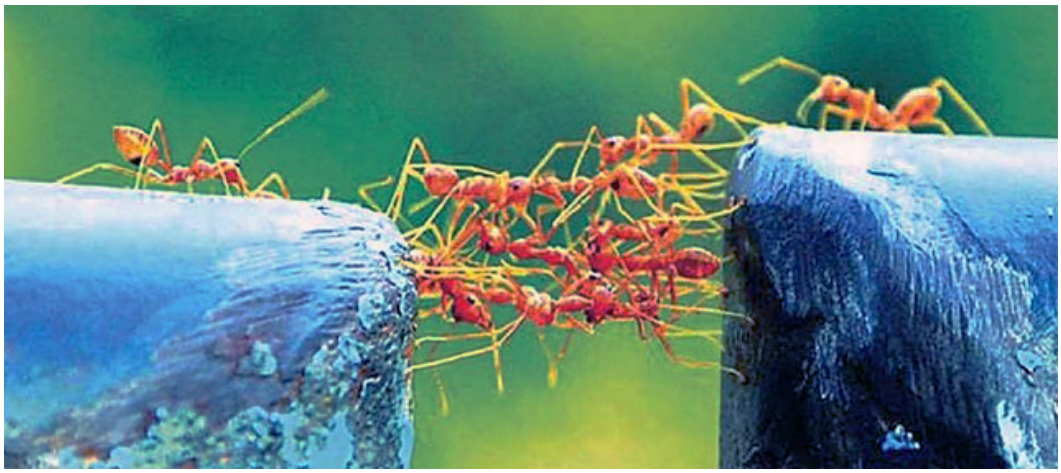


FIGURE 1.4 : Organisation entre fourmis pour traverser un trou. L'auteur de cette image est inconnu

fenêtre de temps beaucoup plus large avec des conséquences non-réversibles sur l'environnement qui entraînent des conséquences sur les phénomènes synchroniques. On peut citer comme exemple l'infiltration d'une goutte d'eau au travers d'une faille (phénomène synchronique), la répétition de ce phénomène sur des décennies voire des siècles aboutit à la création de concrétions, les stalactites ou stalagmites (phénomène diachronique).

Les interactions dans le système peuvent être de type *rétroactif* ou de type *proactif*. Dans le cas d'interactions rétroactives, l'entité va réagir à un stimulus indirectement provoqué par une entité de plus haut niveau, que ce soit une organisation, l'environnement ou le système lui-même. L'entité n'a ni contrôle ni but dans cette réaction. Le rôle de la rétroaction va être d'amplifier (rétroaction positive) ou de diminuer (rétroaction négative) l'intensité d'un phénomène. On retrouve des exemples de ce type d'interactions dans le système endocrinien, en particulier dans l'homéostasie de la thyroïde [DIETRICH, LANDGRAFE et FOTIADOU 2012] : un faible niveau d'hormones thyroïdiennes T3/T4 va entraîner de la part de l'hypothalamus la libération d'hormone thyréotrope (TRH) qui à son tour va stimuler l'hypophyse ; cette stimulation va aboutir à la production de thyréostimuline par l'hypophyse qui va stimuler la production d'hormone thyroïdienne jusqu'à un retour à un niveau normal dans le sang (voir la figure 1.5).

Les interactions proactives impliquent non seulement un contrôle de l'entité mais également la recherche d'un objectif au travers de l'interaction. L'entité cherche alors à agir à un plus haut niveau sur une organisation, l'environnement ou le système entier en le transformant, et en provoquant éventuellement les rétroactions d'autres entités. Cette action résulte de l'anticipation d'un possible événement à venir qui peut être prédit par l'analyse de ce que l'entité perçoit du système. Une interaction proactive peut être le résultat de la répétition d'une même interaction réactive qui, grâce à un mécanisme de mémoire et d'apprentissage, permet à l'entité de détecter le contexte précédant une certaine interaction, ce qui lui permet

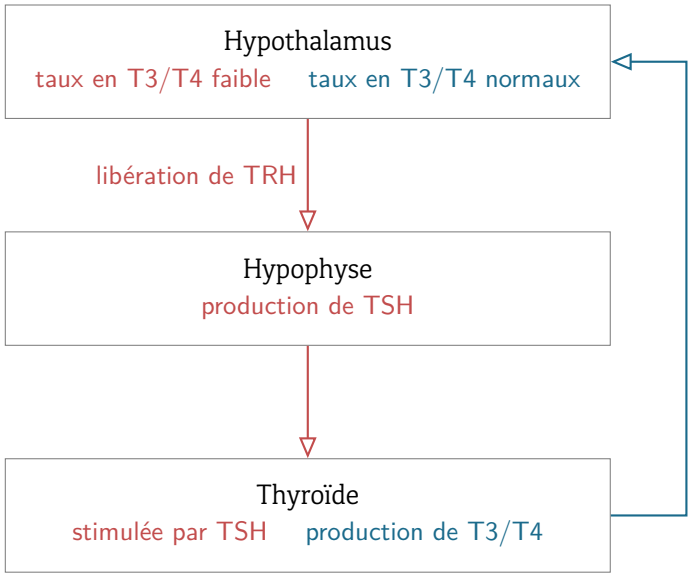


FIGURE 1.5 : Homéostasie du système thyroïdien.

d'anticiper l'interaction. On peut considérer comme exemple le système de régulation thermique d'un bâtiment dont le rôle serait d'y maintenir une certaine température. Ce système peut être équipé d'une sonde qui mesure la température extérieure et grâce à l'analyse de l'évolution de cette température, la sonde peut anticiper l'impact de la température de l'extérieur sur l'intérieur et ainsi agir de manière proactive en ajustant le réglage des appareils thermiques pour maintenir la température intérieure constante.

Un système complexe, bien que composé d'entités en interactions, est avant tout le fruit de ces interactions. On peut d'ailleurs considérer que le système est composé d'interactions plutôt que d'entités. Ceci implique qu'on ne peut limiter le système à ses seules entités et qu'on ne peut limiter son analyse à celle d'une partie de ses entités. Il s'agit d'un système *non-linéaire* dans lequel les interactions locales entre entités impliquent des réactions globales du système.

1.2 Écosystème computationnel

> « An ecosystem is a community of living organisms (plants, animals and microbes) in conjunction with the nonliving components of their environment (things like air, water and mineral soil), interacting as a system. »

Cette citation, extraite de WIKIPEDIA [1], reprend un vocabulaire qui nous est familier et qui nous permet de faire le lien avec la notion de système complexe dont il était question

1. https://en.wikipedia.org/wiki/Ecosystem, consultée le 4 août 2013

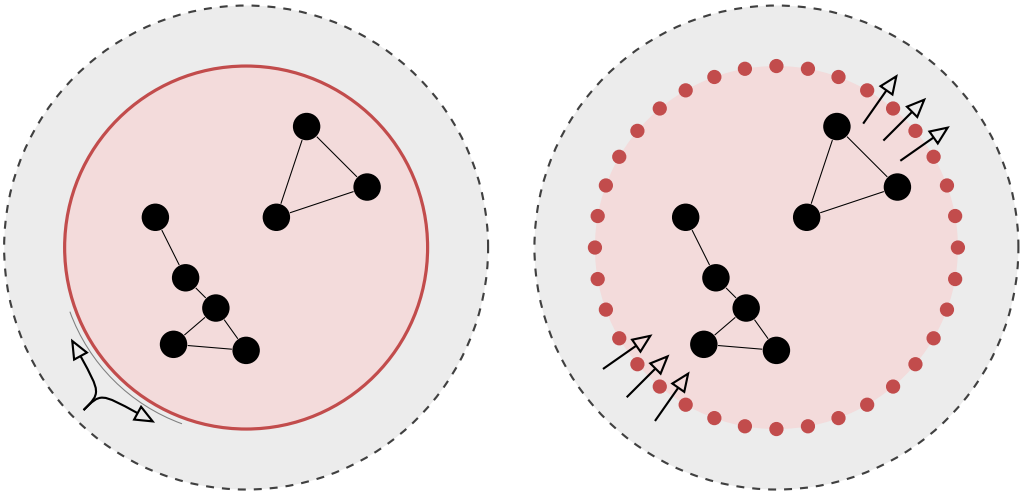


FIGURE 1.6 : À gauche, un système fermé dans lequel les perturbations extérieures ne peuvent franchir la membrane du système. Au contraire, à droite, un flux est généré par les perturbations qui traversent le système.

précédemment : il est question d'entités (les organismes) qui interagissent dans un environnement en formant un système. Il est aussi question d'éléments « *sans vie* » que l'on peut désigner comme étant les *ressources* de l'environnement. Nous allons maintenant définir et approfondir ce que nous entendons par *écosystème computationnel*.

1.2.1 Système distribué

Nous évoluons dans un monde qui est de plus en plus peuplé d'appareils divers et variés. Grâce à l'essor de la technologie, ces appareils se sont vus dotés de mobilité, mais plus important encore, leur capacité de communiquer et d'interagir entre eux n'a de cesse d'augmenter. De l'ordinateur de bureau à la maison elle même, en passant par le téléphone, la voiture…Tous ces appareils forment un ensemble de sous-réseaux qui peuvent s'interconnecter.

Ces appareils vont former un *système distribué* qui va permettre l'exécution de programmes dont la fonction se réduit au calcul dans le but de générer et d'échanger de l'information. Là où par le passé un programme se réduisait à une séquence d'instructions qui s'exécutaient sur un système isolé, il s'agit désormais d'ensembles de programmes qui s'organisent afin de fournir des services de plus haut niveau. Grâce aux connections existantes entre les appareils, les programmes peuvent être dispersés sur un ensemble de systèmes.

Il existe différents types de systèmes distribués, liés par exemple aux types d'unité de calcul ou encore à la topologie du réseaux, ce qui conduit à différentes définitions de ces systèmes. L. LAMPORT en 1987 donne la définition suivante d'un système distribué dans [LAMPORT 1987] :

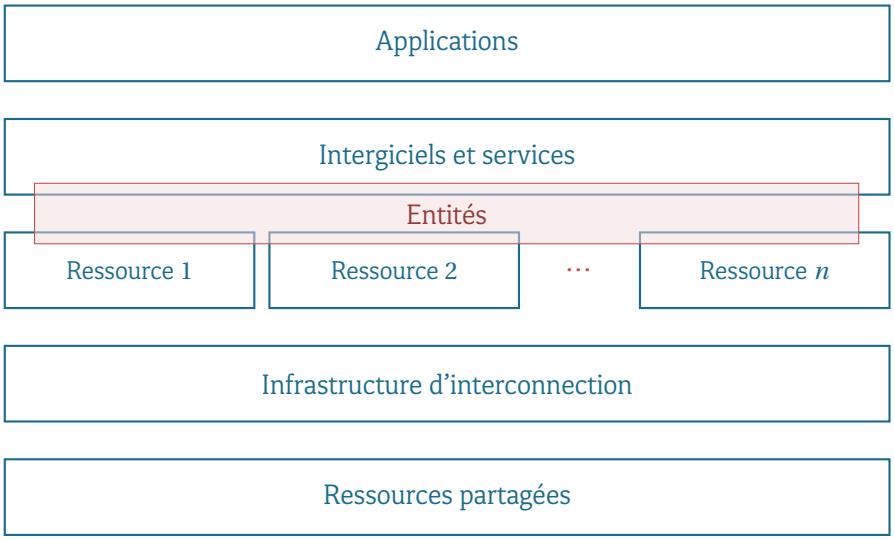


FIGURE 1.7 : Les différentes couches d'un système distribué

> « A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable. »

Cette définition considère donc un système distribué comme un ensemble de machines qui sont dépendantes les unes des autres à tel point que la perte inopinée d'une de ces machines peut entraîner un gel complet du système le rendant ainsi inutilisable. Elle transmet donc l'idée d'une infrastructure statique monolithique. Plus récemment en 2000 dans [EMMERICH 2000], W. EMMERICH nous donne la définition suivante :

> « A distributed system consists of a collection of autonomous hosts that are connected through a computer network. Each host executes components and operates a distribution middleware, which enables the components to coordinate their activities in such a way that users perceive the system as a single, integrated computing facility. »

L'auteur utilise ici le terme plus général d'*hôte* pour désigner les machines faisant partie du système et ici ces hôtes sont *autonomes* : nous avons donc une infrastructure modulaire dont les parties sont communicantes et indépendantes. L'ensemble agit de concert par l'intermédiaire d'un *intergiciel* qui permet aux différents composants s'exécutant dans le système de s'abstraire de leur environnement d'exécution en le considérant comme un unique environnement dans lequel ils peuvent évoluer.

Bien que cette dernière définition implique une certaine tolérance aux pannes dans le système, elle n'inclut pas toute la dynamique qui peut régir certains systèmes. Cela peut suffire pour des systèmes distribués purement dédiés au calcul, comme les grilles ou les grappes, qui ont une dynamique faible : celle-ci se limite aux pannes de machines et à l'ajout ou la suppression de ressources qui surviennent à une faible fréquence. On peut cependant considérer

des systèmes où les machines ont une fréquence d'apparition/disparition plus importante : par exemple, si le réseau comporte des appareils mobiles tels que des smartphones, du fait de leur mobilité et d'une connexion sans-fil au réseau sujette aux perturbations, ces périphériques peuvent se retrouver facilement déconnectés du réseau pour le ré-intégrer par la suite.

1.2.2 Vers une écologie computationnelle

Le terme « écologie » a été emprunté à la langue allemande *Ökologie* qui est une combinaison par le zoologiste et biologiste allemand E. H. Haeckel des termes du grec ancien « *οἶκος* » (oîkos) signifiant « maison » et du suffixe utilisé pour la construction du nom des sciences « *λογία* » (logia) signifiant « discours » [2]. L'écologie peut donc être considérée comme *la science de la maison*, mais comment peut-on définir *maison* dans ce contexte ? Avec « écologie » vient la notion d'*écosystème* dont l'étymologie peut nous permettre de le voir comme *le système formé par la maison*. On peut donc considérer cette maison comme un système composé d'un ensemble d'habitants (la biocénose), et d'un milieu (la bâtiment, le biotope) proposant un certain nombre de ressources à ses habitants (nourriture, couchage, sanitaires, eau, électricité, ...). Les ressources arrivent de l'extérieur du système et les habitants peuvent entrer et sortir : le système formé par la maison est donc ouvert.

Dans notre cas, la *maison* est un système distribué ouvert dont les habitants sont des entités logicielles. Tout comme la maison dispose de ressources permettant à ses habitants d'y vivre, notre système dispose de ressources (capacité de calcul, mémoire, ...) qui permettent à ses entités logicielles de calculer et d'échanger de l'information.

Les différentes machines d'un système distribué vont former le *biotope* d'un *écosystème computationnel* dans lequel nous allons trouver des ressources (capacité de calcul, mémoire, ...). Ce biotope va permettre d'accueillir une *biocénose* formée par les entités logicielles qui vont pouvoir évoluer, interagir et s'organiser.

Le terme d'écosystème computationnel a été principalement introduit par [Kephart, Hogg et Huberman 1989 ; Huberman 1988]. S'inspirer du naturel n'est pas un fait nouveau, et ceci à différentes échelles : physique, chimique, biologique et sociale [Zambonelli et Viroli 2011]. On trouve par exemple la métaheuristique du *recuit simulé* s'inspirant des systèmes thermodynamiques [Kirkpatrick, Vecchi et Gelatt 1983 ; Černý 1985], l'algorithme d'optimisation PSO [3] qui utilise des essaims de particules [Kennedy et Eberhart 1995] ou encore les algorithmes fourmis s'inspirant des colonies de fourmis ainsi que de leur mode de communication indirecte [Dorigo, Maniezzo et Colorni 1996]. On retrouve aussi cette inspiration dans les algorithmes évolutionnistes tels que les programmes évolutionnistes [Fogel 1962], les algorithmes génétiques [Holland 1975] ou encore la programmation génétique [Cramer 1985]. Il s'agit de méta-heuristiques s'inspirant de la théorie de l'évolution afin de faire évoluer, à l'aide de mécanismes de croisement et de mutation, une

2. Source : Trésor de la Langue Française informatisé, http://www.cnrtl.fr/etymologie/écologie
3. Particule Swarm Optimisation

population initiale de solutions à un problème donné afin de produire une nouvelle génération de meilleures solutions.

Si la nature est une source d'inspiration, c'est qu'elle sait faire preuve d'une *adaptabilité* que l'on souhaiterait pouvoir reproduire dans nos modèles. Au travers de la métaphore bio-inspirée des écosystèmes computationnels, nous voyons le système distribué comme étant lui même un *système complexe* afin d'en obtenir les avantages tout en devant en accepter les difficultés. Les entités logicielles sont alors les entités de ce système complexe qui interagissent dans le milieu formé par l'ensemble des ressources de calcul . L'avantage majeur que l'on souhaite obtenir de cet environnement de distribution vu comme un système complexe est la capacité d'adaptation dont sont capables de faire preuve de tels systèmes. Cette adaptabilité émerge autant de mécanismes de collaboration que de mécanismes de compétition qui permettent tout deux l'auto-organisation.

Nous pouvons clore ce chapitre par une citation de J. Lovelock dans [Lovelock 1993] qui présente un système comme un objet adaptatif, contrôlé par des boucles de rétroaction et capable d'utiliser de l'information afin d'évoluer :

> « En définitive, que nous considérions un simple four électrique, une chaîne de magasins de détail gérée par ordinateur, un chat endormi, un écosystème, ou Gaïa elle-même tant que nous nous intéressons à quelque chose qui est adaptatif, capable de récolter de l'information et d'emmagasiner expérience et savoir, son étude est une question de cybernétique et l'objet étudié peut être nommé un "système". »

Références

Černý, V. (1985). "Thermodynamical approach to the traveling salesman problem : An efficient simulation algorithm". In : *Journal of Optimization Theory and Applications* 45, p. 41–51.

Coquillard, P. et D. R. C. Hill (1997). *Modélisation et Simulation des Écosystèmes*. Masson.

Cramer, Nichael Lynn (1985). "A Representation for the Adaptive Generation of Simple Sequential Programs". In : *ICGA*, p. 183–187.

Dietrich, Johannes, Gabi Landgrafe et Elisavet H Fotiadou (2012). "TSH and Thyrotropic Agonists : Key Actors in Thyroid Homeostasis". In : *Journal of thyroid research*.

Dorigo, Marco, Vittorio Maniezzo et Alberto Colorni (1996). "The Ant System : Optimization by a colony of cooperating agents". In : *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS-PART B* 26 (1), p. 29–41.

Emmerich, Wolfgang (2000). *Engineering distributed objects*. Wiley.com.

Fogel, Lawrence J. (1962). "Autonomous automata". In : *Industrial Research* 4 (2), p. 14–19.

Grassé, Pierre-Paul (1959). “La Reconstruction du nid et les coordinations interindividuelles chez Bellicositermes natalensis et Cubitermes sp. La théorie de la stigmergie : essai d’interprétation du comportement des Termites constructeurs”. In : *Insectes sociaux*.

Holland, John H. (1975). *Adaptation in Natural and Artificial Systems*. University Michigan Press.

Huberman, Bernado A (1988). *The ecology of computation*. Elsevier Science Inc.

Kennedy, J et R Eberhart (1995). “Particle Swarm Optimization”. In : *IEEE International of first Conference on Neural Networks*.

Kephart, Jeffrey O, Tad Hogg et Bernado A Huberman (1989). “Dynamics of computational ecosystems”. In : *Physical Review A*.

Kirkpatrick, Scott, Mario P Vecchi et D. Gelatt (1983). “Optimization by simulated annealing”. In : *science* 220 (4598).

Kolmogorov, Andreï (1965). “Three Approaches for Defining the Concept of Information Quantity”. In : *Problems of information transmission* 1 (1), p. 1–7.

Lamport, Leslie (18 mai 1987). *distribution*. English. E-mail. URL : https://research.microsoft.com/en-us/um/people/lamport/pubs/distributed-system.txt (visité le 08/08/2013).

Le Moigne, Jean-Louis (1994). *La théorie du système général, théorie de la modélisation*. Les classiques du réseau intelligence de la complexité.

Lovelock, James (1993). *La terre est un être vivant : l’hypothèse Gaïa*. Champs sciences. Flammarion. ISBN : 978-2-0812-4481-8.

Prigogine, Ilya et Isabelle Stengers (1996). *La fin des certitudes : temps, chaos et les lois de la nature*. Odile Jacob. ISBN : 2738103308.

Saussure, Ferdinand de (1931). *Cours de linguistique générale*. Payot. Geneva.

Zambonelli, Franco et Mirko Viroli (2011). “A survey on nature-inspired metaphors for pervasive service ecosystems”. In : *International Journal of Pervasive Computing and Communications* 7 (3), p. 186–204.

Zeigler, Bernard P., Herbert Praehofer et Tag Gon Kim (2000). *Theory of Modeling and Simulation [Second Edition] : Integrating Discrete Event and Continuous Complex Dynamic Systems*.

CHAPITRE 2

PROBLÈMATIQUE

Nous avons présenté dans le chapitre précédent notre vision d'un système distribué comme un écosystème computationnel composé de son biotope (un ensemble de ressources de calcul interconnectées) et de sa biocénose (les entités logicielles qui vont s'exécuter). La mise en pratique de ce modèle n'est pas sans soulever certains problèmes, liés d'une part à l'environnement et d'autre part aux entités, que nous allons présenter dans ce chapitre.

La figure 2.1 montre les différents éléments qui vont intervenir dans le système distribué. Nous avons à la base une simulation, que l'on souhaite répartir sur un ensemble de machines, qui va produire un réseau d'interactions. Nous allons modéliser ce réseau d'interactions grâce à un graphe dynamique que nous allons pouvoir manipuler et analyser. Ce graphe va évoluer dans un environnement modélisant l'ensemble des machines qui sont à disposition. De cette analyse, nous allons pouvoir fournir des informations concernant la distribution de la simulation sur l'ensemble des machines.

D'un point de vue écologique, nous avons vu qu'un écosystème est composé d'un biotope et d'une biocénose. Le biotope représente un environnement spatialisé qui va fournir un habitat ainsi que des ressources à la biocénose pour lui permettre de vivre. Cette biocénose comporte l'ensemble des êtres vivants de l'écosystème (flore, faune, micro-organismes). Un ensemble de biotopes caractérisés par un *climat* forme un biotope de plus haut niveau appelé *biome* [1]. Ce dernier peut être terrestre ou aquatique. Enfin, l'ensemble des biomes va former la *biosphère*.

Dans le cadre des systèmes distribués, nous avons une infrastructure composée de machines et d'un réseau qui permet l'exécution d'entités logicielles. Cette infrastructure va pro-

1. Terme introduit pour la première fois par Frederic Edward CLEMENTS en 1916

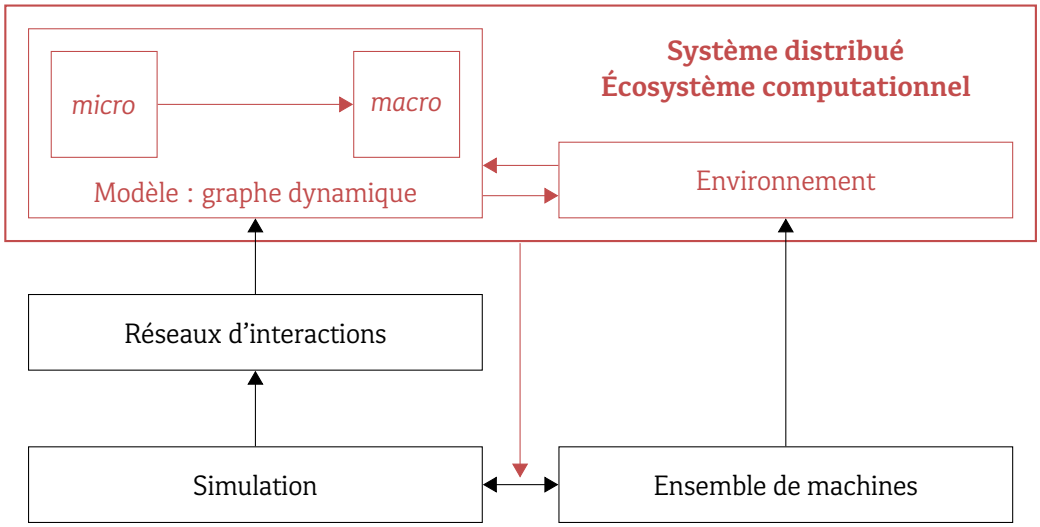


Figure 2.1 : Description du contexte. La partie colorée désigne ce à quoi nous nous intéressons c'est à dire d'une part le modèle d'un réseau d'interactions créé par une simulation et d'autre part un environnement de distribution modélisant les ressources disponibles.

poser des ressources (capacité de calcul, mémoires, accès à diverses sondes ou autres périphériques, etc...) aux entités logicielles afin de leur permettre de remplir leurs fonctions.

La table 2.1 récapitule la comparaison entre un écosystème d'un point de vue écologique et l'écosystème formé par un système distribué.

Nous manipulons et analysons la syntaxe de la simulation dont la sémantique nous est inaccessible. Tout au plus, nous approchons la sémiologie de la simulation. Cette syntaxe nous permet de mettre en relation l'ensemble des éléments intervenant dans le système. Afin de pouvoir l'analyser, il est nécessaire de définir un modèle qui nous fournisse les moyens de la manipuler.

Tout d'abord se pose le problème de la modélisation de la biocénose logicielle. Il s'agit d'entités logicielles qui ont la possibilité d'interagir entre elles. Deux problèmes se posent alors :

1. comment modéliser le réseau d'interactions formé par les entités en interaction ?
2. quel est le modèle d'interaction qui va nous permettre de définir de quelle manière des entités interagissent entre elles ?

La modélisation du réseau d'interaction peut être réalisée grâce à un graphe dans lequel chaque entité représente un nœud et où une arête représente une interaction entre deux entités. Il y a cependant un premier problème qui se pose : du fait que les entités sont réparties sur un ensemble de machines, chaque machine ne peut construire qu'un sous-graphe local, représentant les entités qu'elle héberge. On considère en effet comme contrainte que le nombre potentiel d'entités est trop important pour pouvoir réifier sur une seule machine une

	Écologie	Computationnel
Milieu	aquatique, terrestre	ensemble de machines connectées via un réseau
Ressources	eau, air, énergie	capacité calcul, mémoire, sondes, périphériques
Population	flore, faune	entités logicielles
Dynamique du milieu	climatique (saisons, météo), géologique (tectonique des plaques, érosion)	ajout/suppression de ressources, pannes, connexion/déconnexion de ressources
Dynamique de la population	cycle naissance/mort, reproduction	début et éventuellement fin de programmes
Interactions	liées aux sens/à la perception de l'autre	liées aux communications
	contraintes liées à l'environnement	

TABLE 2.1 : Comparaison entre la vision d'un écosystème d'un point de vue écologique et d'un point de vue computationnel.

structure de données dont la taille serait proportionnelle aux nombres d'entités ou d'interactions. Il est donc impossible de construire un graphe global qui représenterait l'ensemble des entités, ce graphe est donc un *graphe distribué* formé par l'ensemble des sous-graphes locaux de chaque machine.

De plus, afin de pouvoir modéliser la dynamique liée à la fois aux entités et aux interactions, le graphe devra lui même être dynamique. Nous avons besoin d'un modèle de dynamique qui soit adapté à la distribution du graphe et à l'absence de réification de la structure globale.

Nous avons vu que le système distribué que nous cherchons à modéliser est en soi un système complexe. Le contrôle de tels systèmes est un défi majeur. Plusieurs verrous en sont les causes, l'aspect multi-échelles temporelles et spatiales de la dynamique constitue sans doute l'élément essentiel. Une des méthodes consiste à identifier et modéliser des motifs de dynamique qui capturent plus ou moins grossièrement temporellement et spatialement la trajectoire du système. Cette démarche entreprise, il est alors envisageable à l'aide de ces informations d'influencer la trajectoire du système afin de le faire se diriger vers un objectif recherché. On tente de *gouverner* le système plutôt que de tenter d'obtenir un contrôle

optimal que nous sommes incapables d'exercer du fait que cela impliquerait de contrôler l'ensemble des variables qui, dans le cadre de tels systèmes, est considérable.

Nous complétons cette démarche en essayant également de découvrir des solutions sur la façon dont il est possible de gouverner un tel système afin d'en améliorer la *robustesse* et la *résilience*. La robustesse permet de quantifier la capacité du système à maintenir sa trajectoire lorsqu'il est soumis à des aléas qui créent des perturbations souvent majeures et en chaîne. La notion de résilience peut être définie de deux manières [C S. HOLLING 1996] : d'une part la résilience d'*ingénierie* définie dans [PIMM 1984], et d'autre part la résilience *écologique* telle que décrite dans [C. S HOLLING 1973]. La résilience d'ingénierie permet de mesurer la stabilité autour d'un état d'équilibre en s'appuyant sur la capacité du système à résister aux perturbations et sa vitesse à revenir à l'état d'équilibre. Cette définition est cependant insuffisante car elle ne prend pas en compte la possibilité de multiples états d'équilibre pour le système. La résilience écologique quant à elle permet de mesurer, pour un état d'équilibre stable, l'amplitude de la perturbation nécessaire pour modifier la trajectoire du système en le plaçant dans un autre domaine de stabilité. D'une manière plus générale, on peut considérer la résilience comme la capacité d'un système à s'adapter aux perturbations en ré-organisant sa structure interne afin de lui permettre de conserver ses caractéristiques globales [WALKER et al. 2004] :

> « Resilience is the capacity of a system to absorb disturbance and reorganize while undergoing change so as to still retain essentially the same function, structure, identity, and feedbacks. »

Dans le cadre des écosystèmes computationnels, cette dernière définition reflète ce que l'on souhaite obtenir du système distribué : peu importe les perturbations venant agir sur le système (dynamique des ressources de calcul, dynamique des entités, ...), on souhaite que ce système s'adapte et continue à remplir sa fonction.

Afin d'illustrer notre propos, nous pouvons considérer que nous avons besoin de calculer et de maintenir un plus court chemin entre deux nœuds du modèle, pour optimiser le routage des communications par exemple. Nous calculons donc une solution initiale puis, du fait de la dynamique, des perturbations viennent remettre en question la validité de notre solution. Il peut s'agir en l'occurrence de la suppression de certains nœuds ou arêtes faisant partie de la solution ou l'ajout de nouveaux éléments dans le graphe qui vont contribuer à l'apparition d'une nouvelle solution significativement meilleure. Nous attendons de notre système que malgré ces perturbations il soit capable de s'adapter en mettant à jour la solution initiale de façon à ce qu'elle continue à être valable. C'est cela que nous entendons par résilience.

Cette résilience du système repose sur l'auto-organisation qui émerge des interactions entre les entités et plus particulièrement de mécanismes de collaboration mais aussi de compétition. On trouvera principalement de la collaboration à l'intérieur de l'organisation, c'est à dire entre ses membres, tandis que la compétition résulte d'interactions entre des entités situées aux limites des organisations. La gouvernance du système, dans l'optique d'optimiser sa résilience, nécessite dans un premier temps la détection des organisations de manière

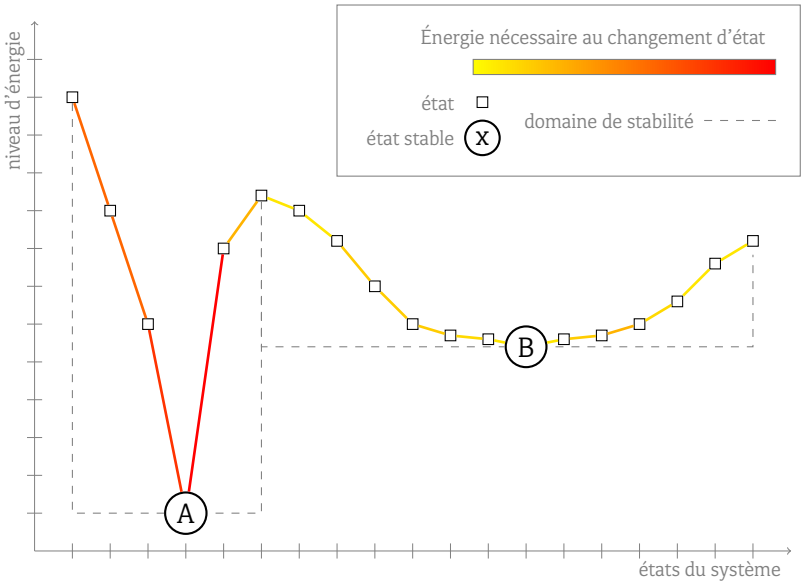


FIGURE 2.2 : Résilience d'ingénierie et résilience écologie. On constate qu'il faut plus d'énergie au système pour quitter l'état d'équilibre A que pour quitter l'état B : la résilience d'ingénierie est plus forte en A que en B. En revanche, le domaine de stabilité associé à l'état B est plus important que celui associé à l'état A : la résilience écologie de B est plus forte que celle de A.

à maximiser la robustesse de ces organisations. Pour cela, il est important de comprendre et de définir ce qui caractérise une organisation.

Une organisation est une structure du graphe associée à une sémantique qui va donner un sens et un rôle à l'organisation. Cette structure possède une *forme* qui va influencer la robustesse de l'organisation face aux perturbations. Par exemple, si la structure de l'organisation forme un sous-graphe complet, sa robustesse sera maximale car très difficile à casser. Au contraire, si la structure forme un chemin, la suppression d'un seul élément peut casser l'organisation.

La difficulté liée à la dynamique réside dans l'entretien de la structure de l'organisation afin de maintenir cette dernière dans le temps. En effet, une entité qui disparaît va modifier la structure de l'organisation dont elle fait partie. De même que l'ajout d'une nouvelle entité va nécessiter de déterminer si elle fait partie d'une nouvelle organisation en formation ou si elle peut être intégrée à une organisation existante ; dans ce dernier cas, il est nécessaire de mettre à jour la structure de cette organisation.

Afin de mieux analyser les organisations, on peut considérer que trois parties peuvent potentiellement composer une organisation (voir la figure 2.4) :

— le noyau qui, s'il existe, correspond à la partie intérieure stable de l'organisation ;
— une membrane regroupant l'ensemble des entités de l'organisation qui interagissent

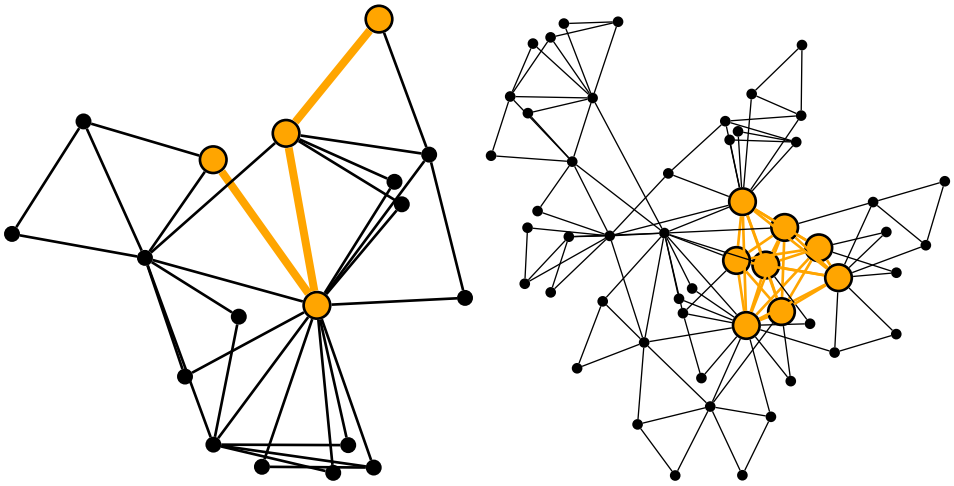


FIGURE 2.3 : Exemples de formes d'organisation. Les éléments colorés forment l'organisation.

avec au moins une entité en dehors de l'organisation ;

— une partie intérieure intermédiaire.

Le noyau reflète l'idée d'un point d'équilibre de l'organisation et offre donc des éléments d'informations nous permettant d'affiner la détection des organisations : si l'on est capable de détecter la présence de multiples points centraux à l'intérieur d'une même organisation, l'organisation a une forte probabilité d'être instable. L'affinage consiste alors à diviser l'organisation afin d'obtenir de nouvelles organisations plus petites mais plus robustes. Cela implique de disposer d'un outil adapté à notre contexte et permettant de caractériser ces points d'équilibres. Nous verrons par la suite une présentation de mesures de centralité qui permettent de détecter les points centraux d'un graphe.

Références

HOLLING, C. S (1973). "Resilience and stability of ecological systems". In : *Annual review of ecology and systematics* 4, p. 1–23.

HOLLING, C S. (1996). "Engineering resilience vs. ecological resilience". In : *Engineering Within Ecological Constraints*. The National Academies Press, p. 31–44. ISBN : 9780309051989.

PIMM, Stuart L (1984). "The complexity and stability of ecosystems". In : *Nature* 307 (5949), p. 321–326.

WALKER, Brian et al. (2004). "Resilience, Adaptability and Transformability in Social– ecological Systems". In : *Ecology and Society* 9 (2).

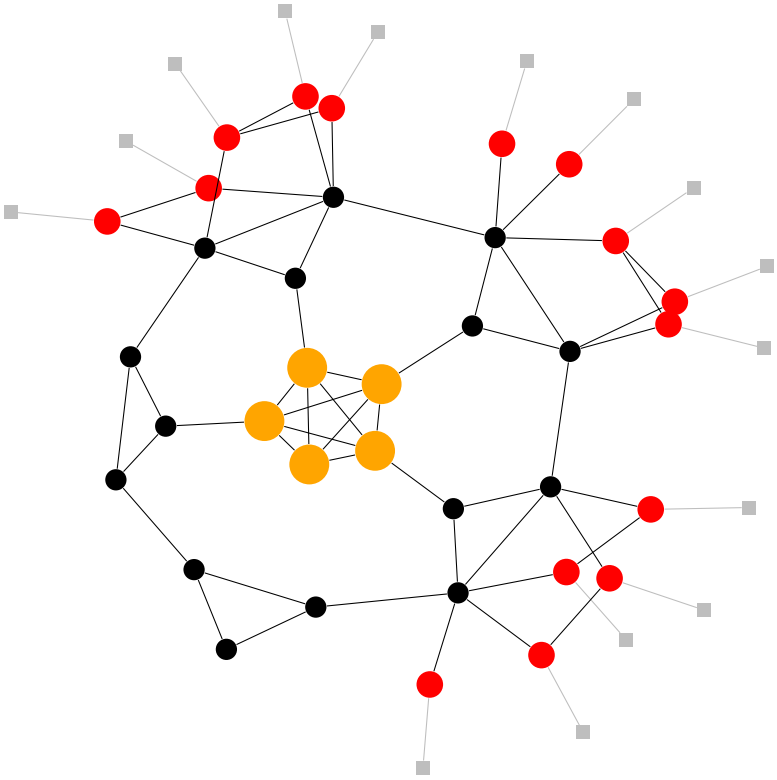


FIGURE 2.4 : Décomposition d'une organisation possédant les trois composantes : en orange le noyau, en noir la partie intermédiaire et en rouge la membrane. Les nœuds gris représentent les nœuds ne faisant pas partie de l'organisation.

Deuxième partie

Positionnement

CHAPITRE 3

RÉSEAUX D'INTERACTIONS

Les réseaux d'interactions sont partout dans le monde qui nous entoure. On les retrouve à tous niveaux : des interactions chimiques entre des molécules aux interactions sociales entre des êtres vivants. Ce chapitre est consacré à présenter la modélisation de réseaux d'interactions et en particulier celle de la dynamique présente dans ces réseaux.

De manière formelle, un réseau est un système dans lequel un ensemble d'entités sont connectées entre elles [M. E. J. NEWMAN 2003]. Ces connections résultent d'une interaction entre les entités concernées, c'est pourquoi on parlera de *réseau d'interactions*. Lorsque ce réseau est plongé dans le temps, nous allons pouvoir observer une dynamique au niveau des

entités, de nouvelles pouvant apparaître, d'autres existantes pouvant disparaître, ainsi qu'au niveau des interactions entre les entités. On observe aussi une dynamique des propriétés qui composent ces différents éléments. Cette dynamique, qui est en soi une composante du réseau, va être la pierre angulaire de nos travaux.

Parmi l'ensemble des réseaux, une certaine catégorie se distingue en regroupant des réseaux qui partagent certaines des caractéristiques suivantes :

— une structure complexe de par son irrégularité ou un nombre d'entités et d'interactions suffisamment important, rendant la visualisation et la compréhension du réseau trop compliquées pour l'œil humain ;
— une dynamique portant à la fois sur la structure (évolution des entités ainsi que des interactions) mais aussi sur les informations caractérisant les éléments du réseaux (entités et interactions) ;
— un comportement non-prédictible ;
— l'*émergence* de nouvelles propriétés du système, comme par exemple d'*auto-organisation* dont nous reparlerons par la suite ;
— des mécanismes de collaboration/compétition.

Ces réseaux sont associés aux *systèmes complexes* et vont être l'objet de notre attention. Les mondes réel et virtuel regorgent d'exemples de tels réseaux : les interactions protéines-protéines, les échanges entre individus sur Internet, les collaborations scientifiques, les communications entre des machines, *etc*... Certains exemples sont présentés sur la figure 3.1.

Nous avons vu dans ce qui précède que nous pouvons distinguer trois composants des réseaux d'interactions :

— tout d'abord, un ensemble d'entités ;
— puis des interactions entre ces entités ;
— et enfin, une dynamique omniprésente.

Nous utilisons le terme *entité* afin de définir de manière générique les *processus autonomes* qui composent ces systèmes : il s'agira selon le contexte de molécules, de cellules, d'individus ou encore d'ordinateurs voire de processus. On peut trouver ces objets sous d'autres noms dans la littérature (agents, nœuds, sommets). Les entités peuvent correspondre à des objets localisables dans notre espace physique lorsque le réseau est spatialisé, ou au contraire les entités peuvent évoluer dans un espace virtuel tel que le web.

Une interaction représente l'action réciproque d'une entité sur une autre. Cette action peut être directe lorsque une entité demande explicitement à une autre d'effectuer une certaine action, ou au contraire l'interaction peut se faire indirectement lorsque l'action d'une entité est le résultat du comportement d'autres entités. Les interactions indirectes peuvent par exemple être liées à des mécanismes de rétro-action, ou des mécanismes de dépôt de message. L'entité qui est à l'origine de l'interaction peut procéder de deux manières :

— soit elle rentre dans un état bloquant dont elle ne sortira qu'à la réception de la réponse de l'entité cible, une telle interaction peut alors être qualifiée de *synchrone* ;
— soit elle continue à effectuer d'autres actions tant que celles-ci ne dépendent pas du résultat de l'action initiale, puis lorsque ce résultat est disponible, l'entité peut alors

s'occuper de son traitement, on parlera alors d'interaction *asynchrone* ;

L'action que représente une interaction est elle aussi dépendante du contexte : réaction chimique, échange de courrier, connexion d'un câble sont autant d'exemples de ce que peut être cette action. La figure 3.1 illustre certaines de ces possibilités.

Afin d'être capables d'étudier et de manipuler ces réseaux, nous avons besoin d'une structure capable des les modéliser. La modélisation d'un ensemble d'éléments et d'un ensemble de paires de ces mêmes éléments est possible grâce aux graphes. Cependant, les graphes tels quels ne nous permettent pas de modéliser l'élément moteur qu'est la dynamique globale du réseau, c'est pourquoi nous allons nous intéresser au domaine des *graphes dynamiques*.

3.1 Graphes dynamiques

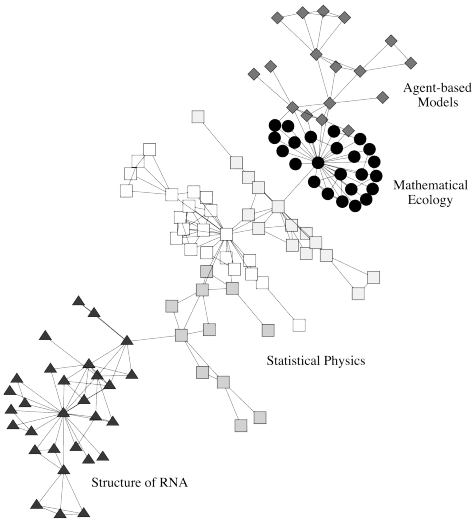
3.1.1 Une première approche

Le graphe est un objet à la fois informatique et mathématique permettant la représentation de problèmes composés d'un ensemble d'éléments pouvant être reliés entre eux. Dans la littérature, le graphe trouve son origine il y a quelques siècles, dans le *problème des sept ponts de Königsberg* de L. Euler[Euler 1741].

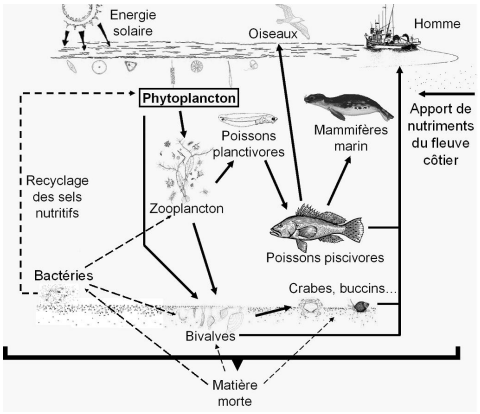
Les éléments représentés par le graphe sont appelés *nœuds*, et les relations existantes entre ces nœuds sont appelés respectivement *arcs* ou *arêtes* selon que la relation est ordonnée ou non. Dans le cas où les relations sont ordonnées, on parlera de *graphe orienté* ou *di-graphe*. Un graphe G est alors noté $G = (V, E)$ où V représente l'ensemble des nœuds du graphe et E l'ensemble des arêtes (des arcs dans le cas d'un graphe orienté).

Certains problèmes nécessitent que les nœuds et/ou les arêtes du graphe possèdent un poids. C'est le cas, par exemple, du problème de l'arbre couvrant de poids minimal [Borůvka 1926] où l'on cherche à extraire un sous-graphe dont l'ensemble des nœuds soit le même que celui du graphe d'origine et dont l'ensemble de arêtes soit un sous ensemble qui permette de connecter tous les nœuds sans qu'il y ait de cycle et de telle sorte que la somme des poids des arêtes de ce sous ensemble soit minimale. C'est aussi le cas de certains algorithmes de plus court chemin tel que Dijkstra [Dijkstra 1959] dans lesquels on cherche à minimiser la somme des poids de arêtes composant le chemin, ou encore le problème de flot maximum comme l'algorithme de Ford-Fulkerson [Ford 1956]. Lorsque les éléments du graphe disposent de tels poids, on parle alors de *graphe pondéré* en précisant éventuellement si la pondération porte sur les nœuds, les arêtes voire même les deux. Un graphe pondéré alors noté $G = (V, E, p_V, p_E)$ ou p_V est une fonction de $V \to \mathbb{R}$ et p_E une fonction de $E \to \mathbb{R}$ représentant respectivement les poids des nœuds et ceux des arêtes.

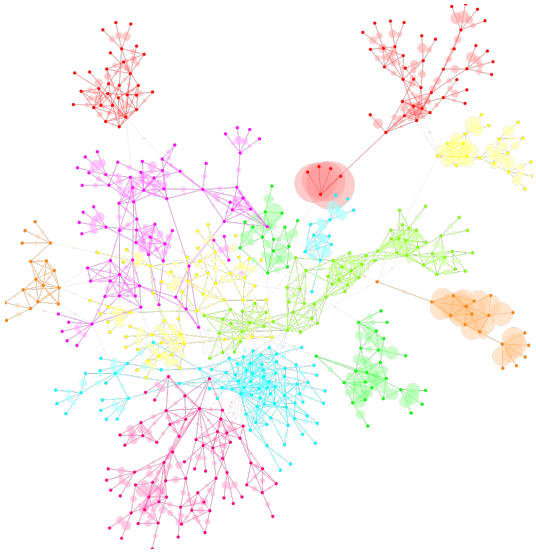
De manière plus générale, on peut souhaiter associer plus d'une valeur aux éléments du graphe. Ces valeurs, que nous allons désigner sous le terme *d'attributs*, vont pouvoir être utilisées pour caractériser les éléments. Il s'agit alors d'une association entre un mot, la *clef*, et la valeur correspondante. Si les ensembles K_V et K_E représentent respectivement l'ensemble



(a) La plus grande composante du réseau de collaboration de l'institut Santa Fe, avec les divisions principales indiquées par différentes formes de nœuds [GIRVAN et M. E. J. NEWMAN 2002]



(b) Le réseau trophique (ici d'un milieu côtier) décrit les liens existant entre les différents organismes d'un écosystème en terme de nutrition. Il se base sur les chaînes alimentaires et permet de visualiser les flux d'énergie et de matière existant au sein de l'écosystème. Source : https://fr.wikipedia.org/wiki/Réseau_trophique, consultée le 30 octobre 2013.



(c) Réseau d'interactions entre un échantillon de livres vendus sur le site Amazon. Les nœuds représentent des livres. Une interaction entre deux livres signifie que ces livres ont été achetés ensemble.

FIGURE 3.1 : Quelques exemples de réseaux

des clefs possibles pour les nœuds et les arêtes, on peut alors généraliser les fonctions p_V et p_E précédentes :

$$\begin{array}{rcccl} p_V & : & V \times K_V & \rightarrow & \mathbb{R} \\ p_E & : & E \times K_E & \rightarrow & \mathbb{R} \end{array} \tag{3.1}$$

Nous choisissons ici l'ensemble $\mathbb{R}$ pour désigner l'ensemble dont font partie les valeurs des attributs. Ces valeurs peuvent cependant être autre chose que des réels, il peut s'agir d'entiers, de couleurs, *etc*... L'ensemble $\mathbb{R}$ permet cependant d'englober toutes ces valeurs et de fournir ainsi une manière générique de décrire les attributs.

Un graphe dynamique permet de modéliser les modifications pouvant survenir au niveau de la structure du graphe (modification de l'ensemble des nœuds $V(G)$ ou de celui des arêtes $E(G)$) mais aussi sur l'évolution des attributs associés aux éléments (nœuds, arêtes).

Dans la littérature, la notion de *graphe dynamique* est référencée sous différents termes qui dépendent du contexte et du domaine dans lesquels le graphe est exploité. Par exemple la notion de *réseaux complexes* est particulièrement utilisée dans l'étude de réseaux extraits du monde réel comme c'est le cas de [Boccaletti et al. 2006] dans une étude du graphe du net. La particularité de ces réseaux est d'être constitués d'un ensemble massif et dynamique d'entités en forte interaction. La notion de réseaux complexes englobe entre autres celle de *réseaux sans échelle* ainsi que celle de *réseaux petit-monde* [Strogatz et Watts 1998] [1]. Les réseaux sans échelle introduits par A.-L. Barabasi et E. Bonabeau [Barabasi et Bonabeau 2003] sont des réseaux dans lesquels la distribution des degrés suit une loi de puissance, ainsi quelques nœuds sont fortement connectés tandis que la grande majorité ne le sont que très faiblement. Cette répartition des degrés fait que le réseau est invariant d'échelle, c'est à dire que l'on observe les mêmes propriétés peu importe l'échelle considérée (cf. la figure 3.2). Les réseaux petit-monde quant à eux possèdent une grande partie de nœuds faiblement connectés mais de telle sorte que la longueur moyenne du plus court chemin entre deux nœuds du réseau soit faible. Afin de montrer que le réseau que nous formons, en tant qu'être humain connecté à un certain nombre d'autres humains que nous connaissons, est de type petit-monde, le sociologue Stanley Milgram mena une expérience en 1967 dans laquelle il tente de faire transiter 60 lettres remises à des personnes d'une ville du Nebraska et devant atteindre une ville du Massachusetts en échangeant ces lettres uniquement en mains propres avec des connaissances personnelles. Sur ces 60 lettres, seulement 3 arrivèrent... mais en quatre jours (les deux villes étant séparées d'environ 2250 km).

Notre objectif n'est pas de nous consacrer ici à la description de l'ensemble des termes utilisés pour désigner un graphe dynamique. On trouvera dans la thèse de Yoann Pigné [Pigné 2008] une partie permettant d'approfondir la description de plusieurs des termes que nous avons vus. Pour notre part, nous retiendrons le terme *réseau d'interactions* pour désigner la structure réelle ou virtuelle émergeant d'interactions entre des éléments que nous nommons entités. Le terme *graphe dynamique* désigne l'objet qui va nous permettre de modéliser ce réseau. La figure 3.3 permet de visualiser ces différentes couches :

1. *scale-free networks* et *small-world networks*

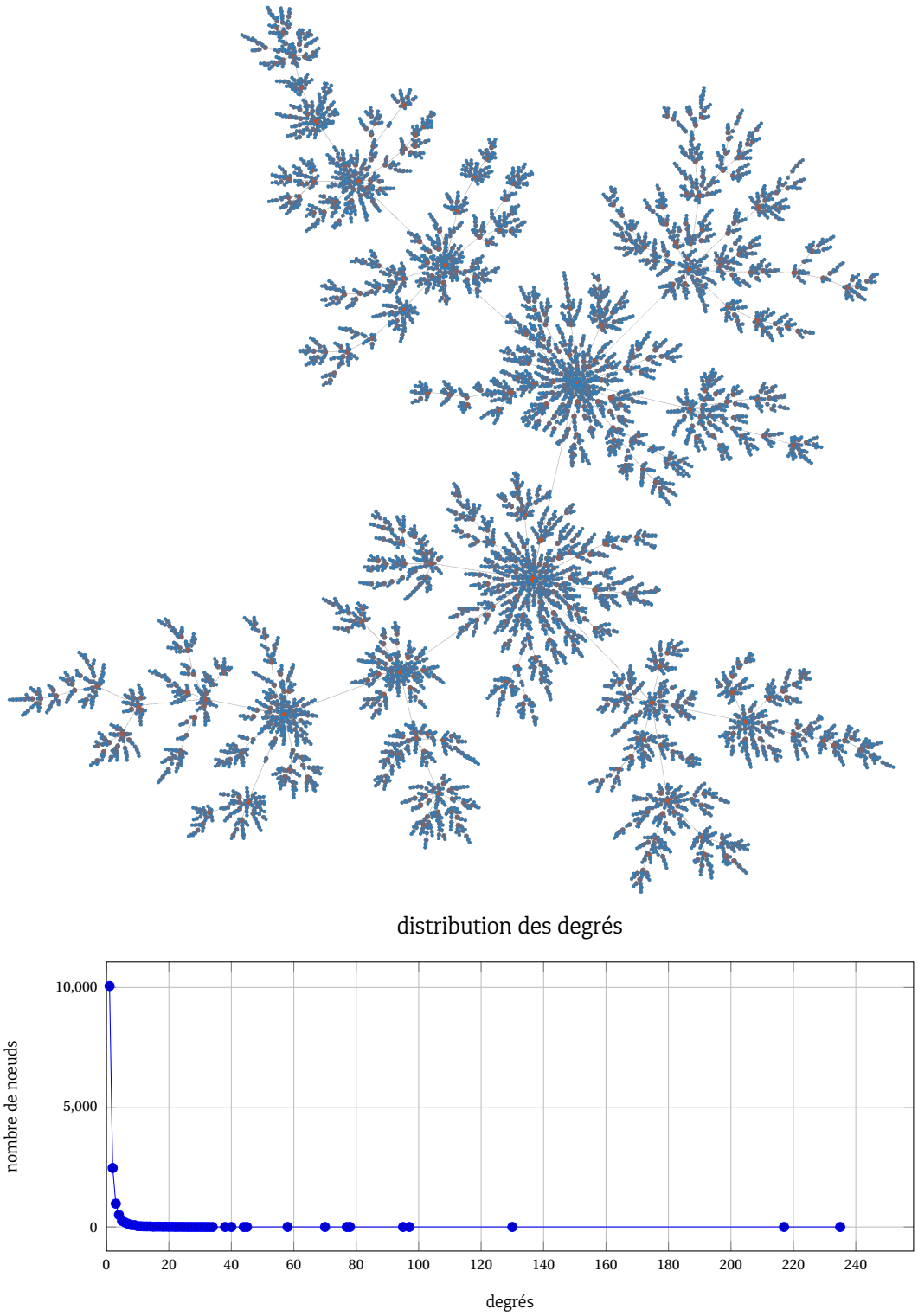


 FIGURE 3.2 : Réseau sans-échelle généré avec GRAPHSTREAM par attachement préférentiel.

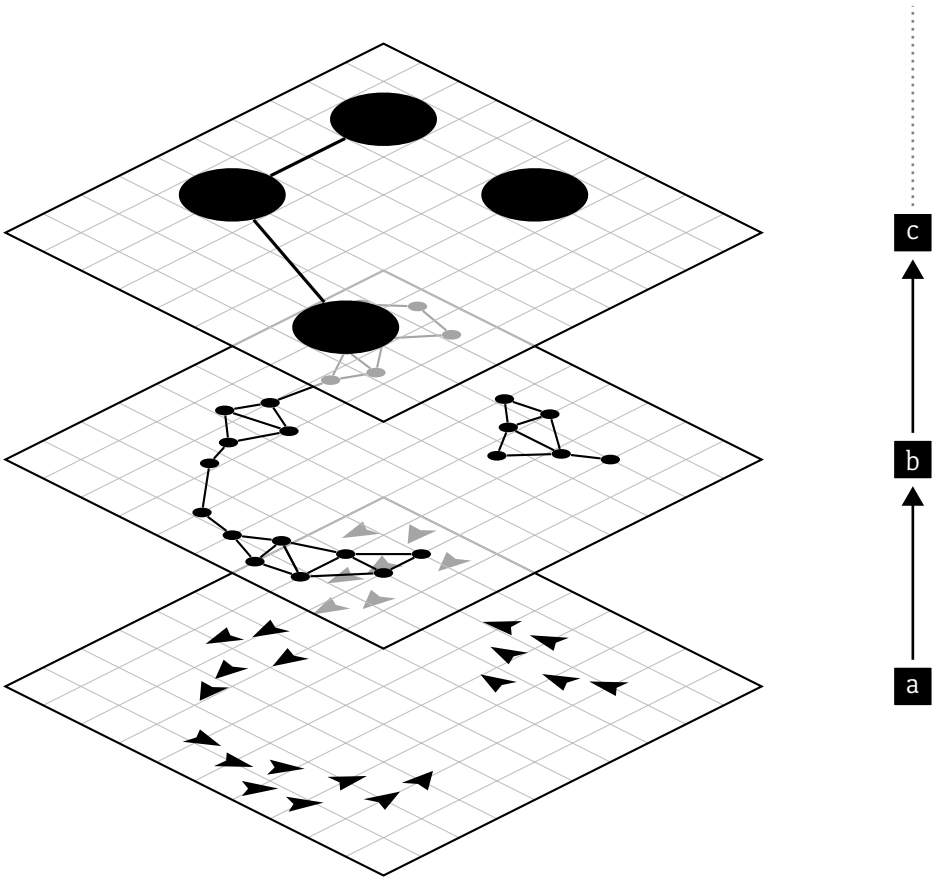


FIGURE 3.3 : Présentation des différentes couches en se basant sur une simulation de boids : (a) le réseau d'interactions, (b) le graphe dynamique initial modélisant le réseau, (c) une vue macroscopique du graphe dynamique

a. un ensemble d'agents nommés *boids* est simulé [REYNOLDS 1987] [2], leur but est de reproduire le comportement de groupe d'organismes vivants tels que certaines espèces de poissons ou encore certaines espèces d'oiseaux ; on considère qu'il y a une interaction entre deux boids lorsque la distance qui les sépare est suffisamment faible pour que l'un influence le comportement de l'autre ;
b. le graphe dynamique modélisant le réseau d'interactions : chaque *boid* est représenté par un nœud, les interactions sont représentées par une arête entre les nœuds correspondants ; il s'agit là d'une vue du graphe à l'échelle microscopique ;
c. le même graphe mais vu d'une échelle macroscopique : les boids ont été regroupés afin de former des structures.

Une vue macroscopique du graphe signifie que l'on va désormais observer des structures

2. le modèle des boids est présenté en détail dans l'annexe A

composées de nœuds plutôt que les nœuds eux mêmes. Ces structures seront principalement des organisations que nous allons présenter par la suite (cf .3.2.1). Ce type de vue implique de procéder à un changement d'échelle qui passera par une phase de détection des structures qui nous intéressent, puis par la création d'un nouveau graphe donc les nœuds représenteront les structures détectées à l'étape précédente.

3.1.2 Modélisation de la dynamique

Si les nœuds et les arêtes du graphe sont des éléments dont la représentation ne pose pas de problème majeur, la représentation de la dynamique qui va opérer à plusieurs niveaux sur le graphe est bien moins intuitive.

Nous allons nous intéresser aux différentes manières de modéliser la dynamique dans un graphe. Cette dynamique est composée d'un ensemble de modifications qui peuvent survenir dans le graphe. Dans [F. HARARY et GUPTA 1997], F. HARARY et G. GUPTA proposent différents types de dynamiques de graphe qui vont permettre de définir que les ensembles d'éléments du graphe (nœuds et arêtes) vont pouvoir évoluer. C'est à dire qu'il est possible que le graphe se voie ajouter ou retirer des nœuds ou des arêtes au fil du temps. Les auteurs définissent alors un type de graphe où seul l'ensemble des nœuds évolue, puis de manière similaire un graphe où seul l'ensemble des arêtes évolue. On notera que la dynamique ne peut difficilement porter que sur les nœuds du fait que la suppression d'un nœud implique la suppression de l'ensemble des arêtes adjacentes à ce nœud. Les auteurs définissent aussi des types de dynamiques permettant de faire évoluer le poids des nœuds et/ou des arêtes.

Cette première approche de la dynamique nous permet d'énumérer les modifications qui vont survenir sur un graphe dynamique. Tout d'abord des modifications sur les ensembles de nœuds et d'arêtes, il y a alors la possibilité d'ajouter et/ou de supprimer des éléments, et ensuite, des modifications sur les attributs associés aux éléments.

Nous allons voir dans ce qui suit différentes façons de modéliser ces modifications et par conséquent la dynamique du graphe.

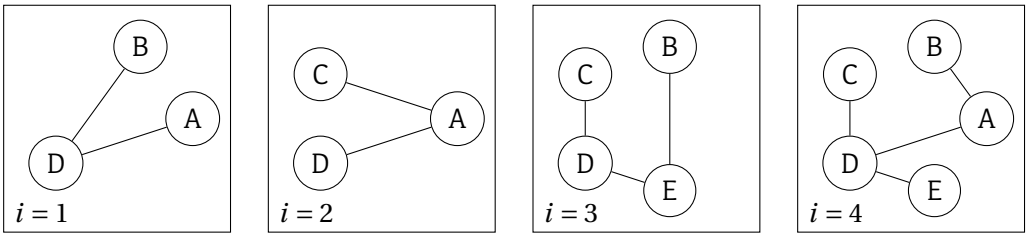
Dynamique séquentielle

Une première approche de la dynamique d'un graphe G consiste à le modéliser comme une séquence de graphes statiques G_i. i symbolise alors un temps discret et G_i représente l'état du graphe au temps i. Le graphe G est alors un triplet $(V(G), E(G), \{G_i\})$ où $V(G)$ est l'ensemble des nœuds présents dans au moins un des graphes G_i, de même pour $E(G)$ qui représente l'ensemble des arêtes présentes dans au moins un des graphes G_i.

On retrouve alors la définition de *graphe évolutif* proposé dans [Afonso FERREIRA 2002 ; BUI-XUAN, A. FERREIRA et JARRY 2003] :

> « Soit $G = (V, E)$ un graphe orienté avec V un ensemble de sommets et E un ensemble d'arêtes dont les extrémités appartiennent à V. Soit $S_G = (G_1, G_2, \ldots, G_T)$ un ensemble de sous-graphes de G. Le système $\mathbb{G}_E = (G, S_G)$ est appelé "graphe évolutif". »

$V(G) = \{A, B, C, D, E\}$
$E(G) = \{AB, AC, AD, BD, BE, CD, DE\}$



$V_1(G) = \{A, B, D\}$
$E_1(G) = \{AD, BD\}$

$V_2(G) = \{A, C, D\}$
$E_2(G) = \{AC, AD\}$

$V_3(G) = \{B, C, D, E\}$
$E_3(G) = \{BE, CD, DE\}$

$V_4(G) = \{A, B, C, D, E\}$
$E_4(G) = \{AB, AD, CD, DE\}$

FIGURE 3.4 : Illustration du modèle de dynamique séquentielle

Dans ce modèle, la dynamique se produit étape par étape, chaque étape correspondant à un graphe statique. La figure 3.4 propose un exemple de ce type de dynamique.

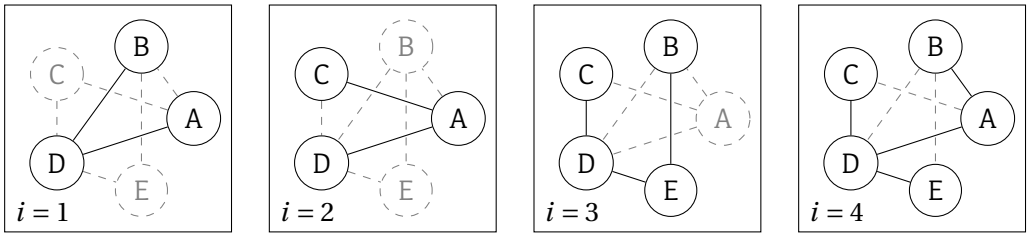
Dynamique cumulative

Une dynamique *cumulative* consiste à agréger la totalité du graphe dynamique en un graphe statique contenant les informations sur les *périodes de présence* de chaque élément dans le graphe. Ces périodes sont des segments du temps dans lequel le graphe est plongé qui indique quand un élément était présent dans le graphe. On notera alors le graphe $G = (V, E, P_V, P_E)$ où V (respectivement E) est l'ensemble des nœuds (des arêtes) apparaissant au moins une fois dans le graphe dynamique. P_V et P_E sont des fonctions permettant de définir la présence dans les graphes des nœuds (respectivement des arêtes). Cette dynamique est illustrée figure 3.5.

Contrairement à une dynamique séquentielle où l'échantillonnage impose d'avoir un temps discret, il est ici possible d'utiliser un temps continu. Nous utiliserons donc $\mathbb{T}$ afin de désigner le temps de manière générique, cet ensemble couvrant tout ou partie de l'ensemble $\mathbb{N}$ si l'on souhaite utiliser un temps discret, ou de l'ensemble $\mathbb{R}$ pour un temps continu. P_V et P_E sont alors des fonctions de $V \to \mathcal{P}(\mathbb{T})$ [3] (respectivement $E \to \mathcal{P}(\mathbb{T})$).

Dans [CORTES, PREGIBON et VOLINSKY 2003], les auteurs utilisent ce type de dynamique pour l'étude de grands graphes représentant des télécommunications. Une fenêtre de temps est définie, c'est à dire un segment de l'ensemble $\mathbb{T}$, puis un graphe statique correspondant à l'agrégation des événements survenus pendant cette fenêtre de temps est extrait à partir du graphe dynamique.

3. $\mathcal{P}(X)$ désigne l'ensemble des parties de X



$V(G) = \{A, B, C, D, E\}$
$E(G) = \{AB, AC, AD, BD, BE, CD, DE\}$

$P_V(A) = \{1, 2, 4\}$
$P_V(B) = \{1, 3, 4\}$
$P_V(C) = \{2, 4\}$
$P_V(D) = \{1, 4\}$
$P_V(E) = \{3, 4\}$

$P_E(AB) = \{4\}$
$P_E(AC) = \{2\}$
$P_E(AD) = \{1, 2, 3\}$
$P_E(BD) = \{1\}$
$P_E(BE) = \{3\}$
$P_E(CD) = \{3, 4\}$
$P_E(DE) = \{3, 4\}$

FIGURE 3.5 : Illustration du modèle de dynamique cumulative. Les éléments en pointillés ne sont pas présents dans le graphe pour le temps donné.

Dynamique par flux d'événements

On considère que la dynamique du graphe est définie par un flux d'événements généré par un processus P_G. Ce processus prend la forme d'une fonction qui à un intervalle temporel associe un ensemble ordonnée d'événements survenus dans cet intervalle. L'ensemble $\mathcal{U}$ représente l'ensemble de tous les événements possibles. Chaque événement $u \in \mathcal{U}$ est un triplet (ϑ, w, e) dans lequel :

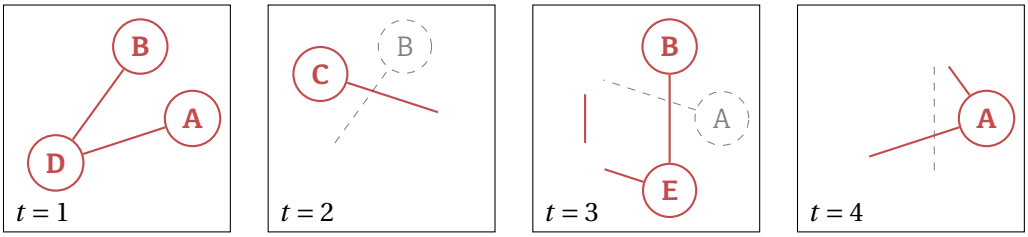
- ϑ désigne la date à laquelle l'événement survient ;
- w est un élément d'un ensemble $\mathcal{W}$ définissant l'ensemble des actions pouvant venir modifier le graphe ; cette action peut être l'ajout ou la suppression d'un élément (nœud ou arête) ou la modification d'un attribut. Dans ce dernier cas, w contient aussi la clef et la nouvelle valeur de l'attribut modifié ;
- e désigne l'élément impliqué dans l'événement.

Le processus P_G peut alors être défini :

$$\mathrm{P}_G \quad : \quad \begin{array}{ccc} \mathbb{T} \times \mathbb{T} & \rightarrow & \mathcal{P}(\mathcal{U}) \\ (t, t') & \rightarrow & \left\{ u = (\vartheta, w, e) \mid t < \vartheta \leq t' \right\} \end{array} \tag{3.2}$$

On introduit l'opérateur $\oplus$ entre un graphe $G = (V, E)$ et un ensemble d'événements $U \in \mathcal{U}$ qui permet d'appliquer à G les événements contenus dans U. À l'aide d'un processus itératif qui opère aux moments choisis t_i et en prenant un graphe initial G_0 au temps t_0, il nous est alors possible de construire le graphe G_i, $i > 0$:

$$G_i = G_{i-1} \oplus \mathrm{P}_G(t_{i-1}, t_i) \tag{3.3}$$



$u_1 = (1, AJOUT, A)$
$u_2 = (1, AJOUT, B)$
$u_3 = (1, AJOUT, D)$
$u_4 = (1, AJOUT, AD)$
$u_5 = (1, AJOUT, BD)$

$u_6 = (2, RETIRE, BD)$
$u_7 = (2, RETIRE, B)$
$u_8 = (2, AJOUT, C)$
$u_9 = (2, AJOUT, CA)$

$u_{10} = (3, RETIRE, AC)$
$u_{11} = (3, RETIRE, A)$
$u_{12} = (3, AJOUT, B)$
$u_{13} = (3, AJOUT, E)$
$u_{14} = (3, AJOUT, BE)$
$u_{15} = (3, AJOUT, DE)$
$u_{16} = (3, AJOUT, CD)$

$u_{17} = (4, RETIRE, BE)$
$u_{18} = (4, AJOUT, A)$
$u_{19} = (4, AJOUT, AB)$
$u_{20} = (4, AJOUT, AD)$

FIGURE 3.6 : Illustration du modèle de dynamique en flux. Les éléments rouges viennent d'être ajoutés dans le graphe, tandis que les éléments en pointillés ont été supprimés.

Contrairement aux dynamiques séquentielle et cumulative, l'indice i des G_i n'indique pas le temps mais le nombre d'itérations du processus permettant de générer le graphe. Un aperçu du fonctionnement de cette dynamique est donné figure 3.6.

On retrouve cette modélisation de la dynamique dans la littérature dans les problèmes de *ré-optimisation* qui consiste , pour un problème donné, à adapter une solution initiale aux changements survenus dans le graphe afin d'éviter de recalculer une solution à partir de zéro. C'est le cas des travaux de C. DEMETRESCU et G. F. ITALIANO [DEMETRESCU et ITALIANO 2006] sur le problème de recherche de l'ensemble des plus courts chemins [4].

Conclusion

Nous venons donc de voir différentes façons de modéliser la dynamique d'un graphe. Ces modèles se différencient en partie sur la manière dont le temps est représenté. L'approche séquentielle utilise un domaine de temps discret, chaque élément de la séquence représentant une étape de ce temps. Au contraire, l'approche cumulative permet de gérer le temps de façon continue pour peu que l'ensemble $\mathbb{T}$ représentant le temps soit lui-même continu. Cela est aussi possible en utilisant un modèle de dynamique par flux.

D'autre part, les modèles se différencient sur la façon de représenter les ensembles d'éléments. Les modèles séquentiel et cumulatif décrivent le contenu de ces ensembles alors que le modèle flux décrit la manière de les construire.

Dans nos travaux, nous avons choisi de voir la dynamique comme un flux. Ce modèle nous parait être le mieux adapté à notre contexte dans lequel il n'est pas possible de construire ni de connaître à l'avance l'ensemble de tous les nœuds du graphe et encore moins celui des

4. APSP, *All-Pairs Shortest Paths*

arêtes. Cette impossibilité est due d'une part au caractère non-déterministe de la trajectoire des systèmes que nous observons, et d'autre part à une de nos hypothèses de départ considérant que la quantité d'entités et d'interactions est trop importante pour pouvoir être réifiée en une seule structure.

3.1.3 Une approche stochastique des graphes : les chaînes de Markov

Nous avons vu jusqu'à présent la modélisation du graphe et de sa dynamique en s'appuyant sur la théorie des ensembles. Il est aussi possible de parvenir à cette modélisation à l'aide de probabilités. Il nous semble donc important de présenter cette approche.

Une chaîne de Markov est un processus opérant sur un ensemble d'états $S = s_1, s_2, ..., s_n$ qui permet de se déplacer d'un état à un autre par *étape* [GRINSTEAD et SNELL 1997]. On appelle *probabilité de transition* la probabilité p_{ij} de passer de l'état s_i à l'état s_j.

Les probabilités de transition sont contenues dans la *matrice de transition* $\mathbf{P}$ du processus. La probabilité de passer de l'état s_i à l'état s_j en k étapes correspond à l'élément ij de la matrice $\mathbf{P}^k$ et est noté $p_{ij}^{(k)}$.

Il est possible d'utiliser les chaînes de Markov afin de définir les graphes dynamiques sous la forme d'un processus stochastique. Pour cela, on considère l'ensemble d'états G_M contenant l'ensemble des graphes à M sommets ainsi qu'une suite de variables aléatoires $X_1, X_2, ..., X_n$ où chaque X_i appartient à G_M. On considère la probabilité initiale $p(x) = \mathbb{P}(X_1 = x)$ ainsi qu'une probabilité de transition $p(x, y) = \mathbb{P}(X_{m+1} = y | X_m = x)$.

La notion de temps est définie dans [RABERTO, RAPALLO et SCALAS 2011] comme une suite $J_1, \ldots, J_n, \ldots$ de délais entre les événements. La *date* T_n de l'événement n est alors :

$$T_n = \sum_{i=1}^{n} J_i$$

Nous allons reprendre un des exemples présentés dans [RABERTO, RAPALLO et SCALAS 2011] afin d'illustrer ce mode de représentation des graphes. Dans cet exemple, nous considérons un ensemble de M personnes représentant les nœuds du graphe, $\{1, \ldots, M\}$. Une information qui n'est connue à l'origine que par une personne doit être partagée jusqu'à être connue de tout le monde. À chaque étape **une** personne connaissant l'information choisit de la partager avec n'importe quelle autre personne ce qui ajoute une arête entre ces deux personnes. Le processus s'arrête lorsque chaque personne est au moins connectée à une autre. Les délais entre les étapes sont calculés en utilisant une distribution de MITTAG-LEFFLER dont la fonction de survie est :

$$\mathbb{P}(J > t) = E_\beta(-t^\beta) = \sum_{n=0}^{\infty} \frac{(-t^\beta)^n}{\Gamma(n\beta + 1)}$$

Le but de cette expérience est alors d'étudier le temps total nécessaire à la diffusion de l'information en fonction du paramètre β.

3.1.4 Vers une représentation formelle

La représentation formelle d'un graphe dynamique nécessite en premier lieu de représenter le temps dans lequel le graphe évolue. Ensuite, par rapport à ce temps, il faut être capable de représenter d'une part l'état de la structure du graphe à un temps donné, d'autre part l'état des données associées aux différents éléments du graphe.

Formalisme pour les réseaux tolérants aux délais

Dans [CASTEIGTS et al. 2012], A. CASTEIGTS *et al.* présentent un modèle nommé TVG (pour « *Time-Varying Graphs* ») afin d'unifier différents modèles présents dans la littérature dans certains domaines des systèmes dynamiques en particulier les réseaux tolérants aux délais, les réseaux de mobilité opportuniste ou encore les réseaux complexes du monde réel. La représentation d'un graphe G est $(V, E, \mathcal{T}, \rho, \zeta)$ avec :

— V, l'ensemble des entités ;
— E, l'ensemble des relations entre entités ;
— $\mathcal{T}$, une partie d'un domaine temporel $\mathbb{T}$;
— $\rho : E \times \mathcal{T} \to \{0, 1\}$, la fonction de présence des arêtes ;
— $\zeta : E \times \mathcal{T} \to \mathbb{T}$, la fonction de *latence* des arêtes.

La dynamique peut être étendue aux entités par l'ajout d'une fonction indicatrice de leur présence $\psi : V \times \mathcal{T} \to \{0, 1\}$ et une fonction de latence pour les entités $\varphi : V \times \mathcal{T} \to \mathbb{T}$.

Les fonctions de latence permettent, en ce qui concerne ζ, de définir le temps nécessaire pour traverser une arête à un temps t, et pour ce qui est de φ de définir les temps de traitements locaux.

Cette représentation est adaptée pour la modélisation d'infrastructure de communication, en particulier les réseaux mobiles. Elle contient de ce fait certains éléments comme les fonctions de latence qui sont propres aux domaines cibles comme les *réseaux tolérants aux délais*. Du fait que cette représentation soit dédiée à ces domaines particuliers, elle ne peut nous satisfaire là où nous cherchons à exprimer les choses de façon générale. C'est pourquoi nous allons désormais présenter une nouvelle représentation formelle permettant de représenter un graphe dynamique de manière générique : le temps dans lequel il est plongé, sa structure et les attributs liés aux différents éléments.

Proposition d'un formalisme générique

Temps Nous avons commencé à voir dans 3.1.2 que la définition d'un modèle de dynamique implique la définition de la temporalité dans laquelle le graphe est plongé. C'est au travers de cette ligne de temps que nous allons pouvoir définir l'évolution du graphe.

Le temps peut être une succession dénombrable ordonnée de valeurs t_i, comme c'est le cas pour une dynamique séquentielle, de telle sorte qu'à chaque t_i est associé un état G_i du graphe. On dira du temps qu'il est discret du fait qu'entre deux temps successifs, t_i et t_{i+1},

il n'est pas possible de définir un nouvel état du graphe. Ce type de temporalité est exprimé par l'ensemble $\mathbb{N}$ ou une partie de cet ensemble.

Au contraire, le temps peut être non dénombrable. Pour toutes valeurs du temps t et t' tel que $t < t'$ il est possible d'obtenir une nouvelle valeur t'' tel que $t < t'' < t'$. On considère alors le temps comme continu et il est possible de l'exprimer par l'ensemble $\mathbb{R}$.

Structure L'ensemble des nœuds du graphe dynamique est dépendant du temps. On le définira donc comme la fonction v qui a un temps t donné associe un ensemble de nœuds V_t. On considère que l'ensemble des nœuds possibles pour un graphe est infini mais dénombrable. On peut par conséquent utiliser l'ensemble $\mathbb{N}$ afin de le représenter, chaque nœud pouvant donc être associé à un entier. L'ensemble des nœuds du graphe est alors un élément de $\mathcal{P}(\mathbb{N})$. $\mathcal{P}(\mathbb{N})$ constitue l'ensemble des parties de $\mathbb{N}$, c'est à dire l'ensemble de ses sous-ensembles : $\mathcal{P}(\mathbb{N}) = \{U \mid U \subseteq \mathbb{N}\}$. Nous utiliserons donc ce dernier afin de représenter l'ensemble de tous les nœuds. La fonction v sera alors :

$$\begin{array}{rcccc} v & : & \mathbb{T} & \rightarrow & \mathcal{P}(\mathbb{N}) \\ & & t & \rightarrow & V_t \end{array} \tag{3.4}$$

De même, l'ensemble des arêtes du graphe dynamique est aussi dépendant du temps. Il sera défini à l'aide de la fonction e qui à un temps donné t associe un ensemble d'arêtes E_t :

$$\begin{array}{rcccc} e & : & \mathbb{T} & \rightarrow & \mathcal{P}(\mathbb{N} \times \mathbb{N}) \\ & & t & \rightarrow & E_t, \quad E_t \subseteq (V_t \times V_t) \end{array} \tag{3.5}$$

Il nous est donc possible de définir la partie structurelle d'un graphe dynamique G comme étant le triplet $(v, e, \mathbb{T})$.

Attributs Nous avons vu en 3.1.1 les fonctions p_V et p_E qui permettent de définir un ensemble de propriétés, les attributs, sur les éléments du graphe. Nous pouvons étendre la définition de ces fonctions pour y inclure la composante temporelle :

$$\begin{array}{rcccc} p_V & : & V \times K_V \times \mathbb{T} & \rightarrow & \mathbb{R} \\ p_E & : & E \times K_E \times \mathbb{T} & \rightarrow & \mathbb{R} \end{array} \tag{3.6}$$

Le graphe lui-même pouvant posséder des propriétés qui vont évoluer avec le temps, nous introduisons la fonction p_G qui, à un ensemble de clef K_G, permet d'associer des valeurs :

$$p_G : K_G \times \mathbb{T} \rightarrow \mathbb{R} \tag{3.7}$$

On peut par exemple considérer l'ensemble de clefs $K_V = \{poids, couleur, taille\}$. La fonction $p_V(n, couleur, t)$ permet alors de connaitre la couleur du nœud n au temps t.

En considérant le triplet $\mathcal{A} = (p_G, p_V, p_E)$, nous pouvons alors définir un graphe dynamique avec attributs comme étant le quadruplet $(v, e, \mathcal{A}, \mathbb{T})$.

Application aux modèles de dynamique

Considérons maintenant notre représentation formelle en fonction des différents modèles de dynamique que nous avons présentés en 3.1.2. Nous laisserons ici de côté la partie concernant les attributs.

Dynamique séquentielle $(G_0 = (V_0, E_0), \ldots, G_i = (V_i, E_i), \ldots, G_n)$ une séquence de graphes statiques. On définit alors le graphe dynamique correspondant par $G = (v, e, [0, n])$, avec :

— $v(t) = V_t$
— $e(t) = E_t$

Dynamique cumulative En considérant :

— le temps $\mathbb{R}^+$;
— l'ensemble V_A de tous les nœuds présents à au moins un instant $t \in \mathbb{R}^+$;
— E_A l'ensemble équivalent pour les arêtes ;
— la fonction indicatrice $\alpha : V_A \times \mathbb{R}^+ \to \{0, 1\}$ définissant si un nœud est présent dans le graphe à un temps donné ;
— la fonction indicatrice $\beta : V_A \times V_A \times \mathbb{R}^+ \to \{0, 1\}$ définissant de manière équivalente si une arête est présente dans le graphe à un temps donné.

On peut alors définir le graphe dynamique $G = (v, e, \mathbb{R}^+)$ avec :

— $v(t) = \{u \mid \alpha(u, t) = 1\}$;
— $e(t) = \big\{(u, v) \mid \beta(u, v, t) = 1\big\}$.

Dynamique par flux On considère un processus P_G correspondant à un graphe dynamique, ainsi qu'un graphe initial G_{t_0} correspondant à l'état du graphe au temps t_0. En reprenant l'opérateur $\oplus$ défini en 3.1.2, on peut alors construire le graphe $G_t = (V_t, E_t)$:

$$G_t = G_{t_0} \oplus \mathrm{P}_G(t_0, t)$$

On obtient alors le graphe dynamique $G = (v, e, \mathbb{T})$ avec :

$$\begin{aligned} v(t) &= V_t \\ e(t) &= E_t \end{aligned}$$

Pour résumer, nous considérons un graphe dynamique comme étant un processus dynamique fournissant un flux d'événements. Ces événements décrivent les modifications qui surviennent dans le graphe. Il est alors possible de construire le graphe à partir de sa version précédente en appliquant successivement les événements qui sont survenus entre temps. Le formalisme, que nous avons introduit ci-dessus, permet de décrire ce processus.

3.1.5 Centralité

La notion de *centre* est utilisée dans différents domaines scientifiques, comme la géométrie ou encore la physique, sous divers noms tels que barycentre, centre de masse, centroïde, centre de gravité, *etc*... Elle est l'expression du besoin de définir une zone d'équilibre dans un objet, qu'il soit pondéré ou non. La centralité n'est pas directement liée à la dynamique du graphe. Elle va cependant être une mesure importante pour une partie de nos travaux, lorsque nous chercherons à détecter les centroïdes des organisations.

Dans le domaine des graphes, on cherche principalement à mesurer à quel point chaque nœud se trouve éloigné du centre du graphe. On parle alors de mesure de centralité qui permet d'introduire une relation d'ordre entre les nœuds en fonction de cet éloignement vis à vis du *centre* du graphe. Il est alors possible de déterminer le ou les centres du graphe comme étant les nœuds qui minimisent ou maximisent (en fonction de la mesure choisie) cette mesure.

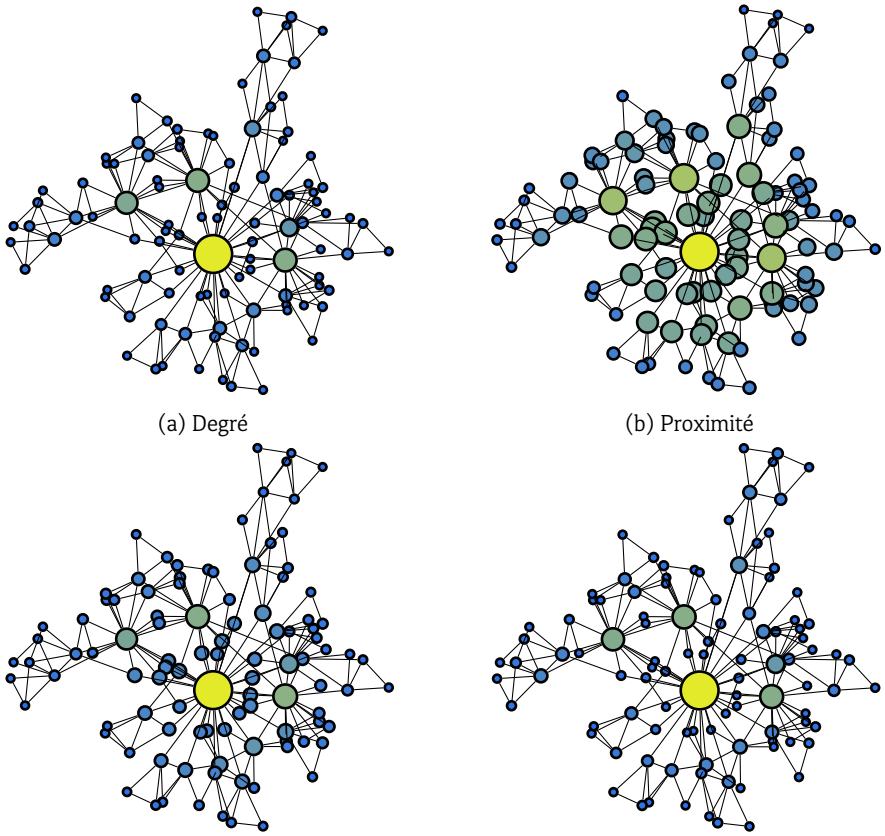
La centralité dans les graphes est l'expression du besoin de faire ressortir les informations les plus importantes qui, du fait de l'augmentation de la quantité d'informations contenue dans les graphes, se retrouvent noyées dans la masse. C'est par exemple le cas dans l'étude de réseaux sociaux [FREEMAN 1978] où la centralité permet de mieux cerner l'importance des nœuds dans le réseau et d'appréhender la diffusion d'informations à travers le réseau.

Dans la littérature, la centralité, les centres et les centroïdes font l'objet de nombreux travaux : de C. JORDAN donnant une preuve de leur existence dans chaque arbre en 1869 [JORDAN 1869] aux travaux de P. J. SLATER qui généralise leurs définitions [SLATER 1978], en passant par leur présence dans les livres de théorie des graphes [ORE 1962 ; Frank HARARY 1969 ; M. NEWMAN 2010].

Dans ce qui suit, nous allons aborder certaines mesures de centralité que sont : le degré, la proximité, l'intermédiarité, la centralité spectrale ou encore le *PageRank* utilisé par le moteur de recherche GOOGLE. De nombreux travaux scientifiques traitant de ces mesures, ainsi que d'autres mesures qui ne seront pas abordées ici, peuvent être consultées pour approfondir le sujet [M. NEWMAN 2010 ; CHIKHI 2010]. Certaines des mesures citées sont présentées dans la figure 3.7. Nous utiliserons par la suite les graphes G_1 et G_2 (cf. figure 3.8) pour illustrer les mesures de la centralité que nous présenterons.

Degré

La mesure de centralité la plus simple est le degré, consistant tout simplement à utiliser le degré des nœuds comme valeur de centralité, c'est à dire le nombre d'arêtes adjacentes au nœud considéré. On considère donc que plus un nœud est connecté à d'autres nœuds, plus ce nœud est important. L'inconvénient de cette mesure est de ne considérer que localement les nœuds ce qui peut aboutir à de fortes valeurs de centralité pour des nœuds qui seraient isolés (par exemple si ces nœuds sont connectés à de nombreuses feuilles du réseau).



(a) Degré (b) Proximité

(c) Vecteur propre (d) Page Rank

FIGURE 3.7 : Quelques exemples de mesure de la centralité. Plus la taille du nœud est importante (et plus sa couleur s'approche du jaune), plus la valeur de la centralité de ce nœud est importante.

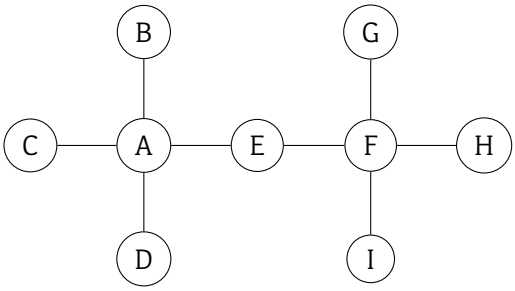
Si l'on considère la matrice d'adjacence A du graphe, le degré du nœud i se note alors :

$$c_d(i) = \sum_j A_{ij}$$

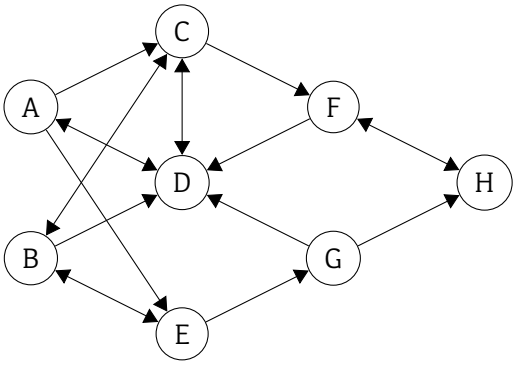
Lorsque le graphe est orienté, il devient nécessaire de considérer d'une part le degré entrant, et d'autre part le degré sortant. Pour chaque nœud i, on obtient deux mesures $c_{de}(i)$ et $c_{ds}(i)$:

$$c_{de}(i) = \sum_j A_{ji} \quad ; \quad c_{ds}(i) = \sum_j A_{ij}$$

La figure 3.9 montre l'application du degré aux graphes définis dans la figure 3.8. Actuellement, le degré reste à la base de nombreux travaux sur la centralité dans la littérature.



(a) G_1



(b) G_2

FIGURE 3.8 : Graphes jouets utilisés pour illustrer les différentes centralités (tirés de [CHIKHI 2010])

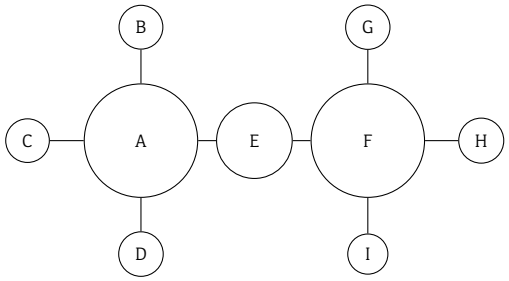
Proximité

La proximité, ou *closeness*, permet de tenir compte de l'ensemble du graphe pour le calcul de la centralité. La proximité d'un nœud i est égale à la moyenne des longueurs des plus courts chemins de i vers tous les autres nœuds du graphe. Autrement dit, la proximité $c_c(i)$ du nœud i d'un graphe $G = (V, E)$ est définie par l'équation :

$$c_c(i) = \frac{1}{|V|-1} \sum_{j \in V \setminus \{i\}} d(i,j) \tag{3.8}$$

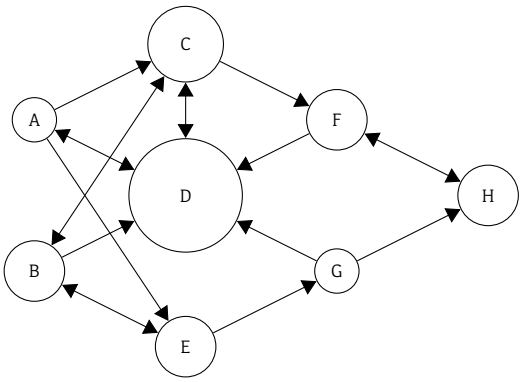
où $d(i,j)$ représente la longueur du plus court chemin entre les nœuds i et j.

L'inconvénient de cette méthode réside dans le fait qu'elle nécessite le calcul de l'ensemble des plus courts chemins du graphe. Ceci implique d'une part une complexité en $O(|V|^3)$ si l'on utilise l'algorithme de FLOYD-WARSHALL [FLOYD 1962]. On peut améliorer la complexité en utilisant l'algorithme de JOHNSON [JOHNSON 1977] qui nous permet d'obtenir en $O(|V| \cdot |E| + |V|^2 log|V|)$. D'autre part, cette méthode nécessite une approche globale qui est difficile dans le cas d'un graphe distribué.



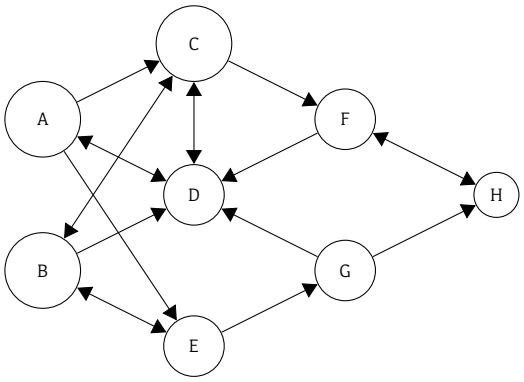
$$
\begin{aligned}
c_d(A) &= 4\\
c_d(B) &= 1\\
c_d(C) &= 1\\
c_d(D) &= 1\\
c_d(E) &= 2\\
c_d(F) &= 4\\
c_d(G) &= 1\\
c_d(H) &= 1\\
c_d(I) &= 1
\end{aligned}
$$

(a) Le degré appliqué au graph G_1



$$
\begin{aligned}
c_{de}(A) &= 1\\
c_{de}(B) &= 2\\
c_{de}(C) &= 3\\
c_{de}(D) &= 5\\
c_{de}(E) &= 2\\
c_{de}(F) &= 2\\
c_{de}(G) &= 1\\
c_{de}(H) &= 2
\end{aligned}
$$

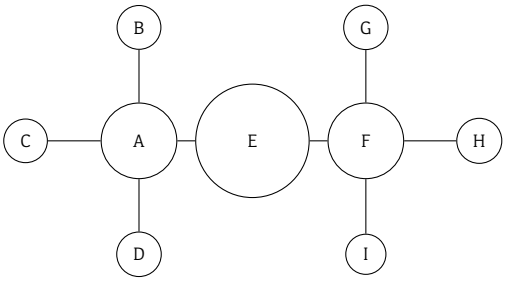
(b) Le degré entrant appliqué au graph G_2



$$
\begin{aligned}
c_{ds}(A) &= 3\\
c_{ds}(B) &= 3\\
c_{ds}(C) &= 3\\
c_{ds}(D) &= 2\\
c_{ds}(E) &= 2\\
c_{ds}(F) &= 2\\
c_{ds}(G) &= 2\\
c_{ds}(H) &= 1
\end{aligned}
$$

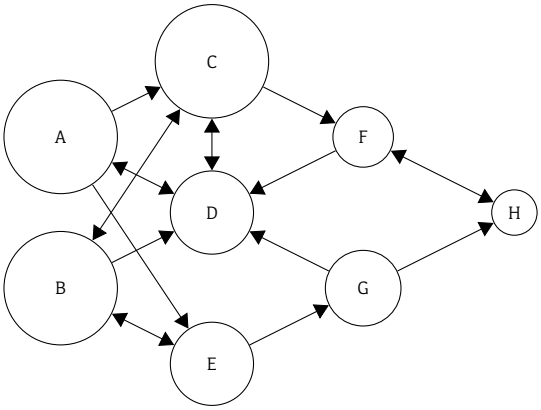
(c) Le degré sortant appliqué au graph G_2

FIGURE 3.9 : La centralité utilisant le degré appliquée aux graphes G_1 et G_2. La taille des nœuds est proportionnelle à la valeur de la centralité.



$$
\begin{aligned}
c_d(A) &= 0.067 \\
c_d(B) &= 0.045 \\
c_d(C) &= 0.045 \\
c_d(D) &= 0.045 \\
c_d(E) &= 0.071 \\
c_d(F) &= 0.067 \\
c_d(G) &= 0.045 \\
c_d(H) &= 0.045 \\
c_d(I) &= 0.045
\end{aligned}
$$

(a) G_1



$$
\begin{aligned}
c_{de}(A) &= 0.083 \\
c_{de}(B) &= 0.083 \\
c_{de}(C) &= 0.083 \\
c_{de}(D) &= 0.071 \\
c_{de}(E) &= 0.071 \\
c_{de}(F) &= 0.063 \\
c_{de}(G) &= 0.071 \\
c_{de}(H) &= 0.045
\end{aligned}
$$

(b) G_2

FIGURE 3.10 : Mesure de la proximité appliquée aux graphes G_1 et G_2 en utilisant l'équation 3.9. La taille des nœuds est proportionnelle à la valeur de la centralité.

De plus, la proximité est bien adaptée pour un graphe connexe dans lequel pour n'importe quelle paire de nœuds i, j il existe un chemin entre i et j. Dans le cas d'un graphe possédant plusieurs composantes connexes, ou dans le cas d'un graphe orienté dans lequel l'existence d'un chemin entre n'importe quelle paire de nœuds n'est pas assurée, on obtient des distances égales à l'infini. Une solution à ce problème est proposée dans [M. NEWMAN 2010] consistant à considérer l'inverse des distances plutôt que les distances elles-mêmes :

$$c_c(i) = \frac{1}{|V|-1} \sum_{j \in V \setminus \{i\}} \frac{1}{d(i,j)} \tag{3.9}$$

Intermédiarité

Lorsqu'une information est diffusée à travers un réseau, une route est calculée puis l'information emprunte cette route depuis le nœud émetteur jusqu'à sa destination. Chaque

nœud composant la route participe à la diffusion de cette information et l'absence de l'un d'entre eux pourrait mener à une diffusion plus longue voire à une absence de diffusion. À partir de ce point de vue, l'importance d'un nœud réside dans le rôle qu'il joue dans la diffusion d'information.

Si l'on considère que la diffusion d'une information d'un nœud A à un nœud B emprunte un plus court chemin entre ces deux nœuds, on peut alors, en calculant l'ensemble de tous les plus courts chemins du graphe, déterminer pour un nœud i la proportion de plus courts chemins $C_B(i)$ dont i fait partie. C_B est une mesure de centralité que l'on nomme intermédiarité, et qui est attribuée à FREEMAN [FREEMAN 1977]. On considère g_{ij} le nombre de plus courts chemins entre les nœuds i et j, et $g_{ij}(k)$ le nombre de plus courts chemins entre i et j dont le nœud k fait partie. On peut définir alors $b_{ij}(k)$ comme le rapport entre ces deux valeurs :

$$b_{ij}(k) = \frac{g_{ij}(k)}{g_{ij}} \tag{3.10}$$

On peut alors définir l'intermédiarité :

$$C_B(i) = \sum_{s,t \in V, s \neq t} b_{st}(i) \tag{3.11}$$

On peut normaliser cette centralité en utilisant le facteur $\frac{1}{n^2}$, ou $\frac{1}{n^2-n+1}$ [M. NEWMAN 2010] :

$$C_B(i) = \frac{1}{n^2} \sum_{s,t \in V, s \neq t} b_{st}(i) \tag{3.12}$$

Tout comme la proximité, les inconvénients de cette mesure sont liés à la nécessité de calculer l'ensemble des plus courts chemins du graphe.

Centralité spectrale

Il est intéressant de considérer que les valeurs de la centralité des nœuds peuvent être liées : plus un nœud est entouré de voisins ayant une centralité élevée, plus sa propre centralité devrait être importante. Ainsi, on peut définir la centralité $C_S(i)$ d'un nœud i par la centralité de ses voisins :

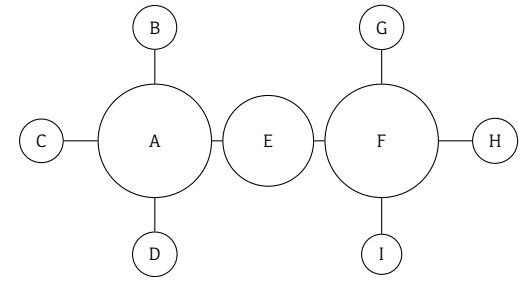
$$C_S(i) = \frac{1}{\lambda} \sum_{j \in V(G)} A_{ij} C_S(j) \tag{3.13}$$

où λ est une constante.

En considérant le vecteur $x = (C_S(0), C_S(1), \ldots, C_S(n))$, on peut ré-écrire l'équation précédente sous la forme :

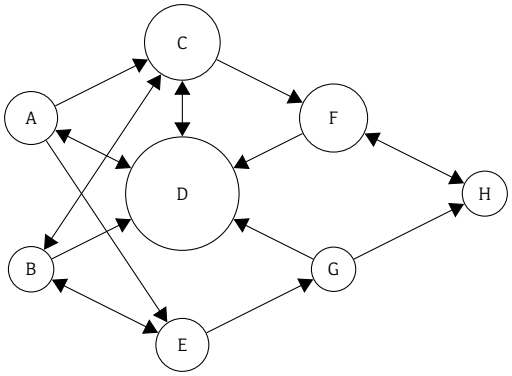
$$\lambda x = Ax \tag{3.14}$$

La seule solution positive de cette équation est le vecteur propre associé à la plus grand valeur propre λ de A.



$$
\begin{aligned}
c_d(A) &= 36\\
c_d(B) &= 0\\
c_d(C) &= 0\\
c_d(D) &= 0\\
c_d(E) &= 32\\
c_d(F) &= 36\\
c_d(G) &= 0\\
c_d(H) &= 0\\
c_d(I) &= 0
\end{aligned}
$$

(a) G_1



$$
\begin{aligned}
c_{de}(A) &= 7\\
c_{de}(B) &= 4.5\\
c_{de}(C) &= 10\\
c_{de}(D) &= 17\\
c_{de}(E) &= 7.5\\
c_{de}(F) &= 9\\
c_{de}(G) &= 3.5\\
c_{de}(H) &= 1.5
\end{aligned}
$$

(b) G_2

FIGURE 3.11 : Application de l'intermédiarité aux graphes G_1 et G_2

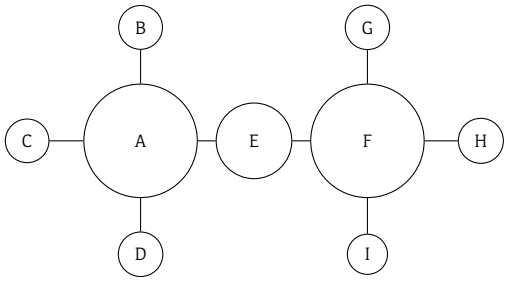
Page Rank

Le PageRank [BRIN et PAGE 1998] est une mesure de centralité utilisée par GOOGLE afin de permettre un classement des nombreuses pages web selon leur importance. Comparé à d'autres mesures cherchant à mettre en relation la valeur de la centralité d'un nœud avec celle de ces voisins comme la centralité spectrale évoquée précédemment ou encore la centralité de Katz [KATZ 1953], le PageRank tente de réduire l'impact de la centralité d'un nœud fortement connecté sur ses voisins. Autrement dit, la centralité d'un nœud doit être diffusée aux voisins dans une moindre mesure lorsque celui-ci est connecté à un milliard de nœuds que lorsqu'il n'est connecté qu'à une dizaine.

Le PageRank $C_{PR}(i)$ d'un nœud i peut s'exprimer sous la forme suivante :

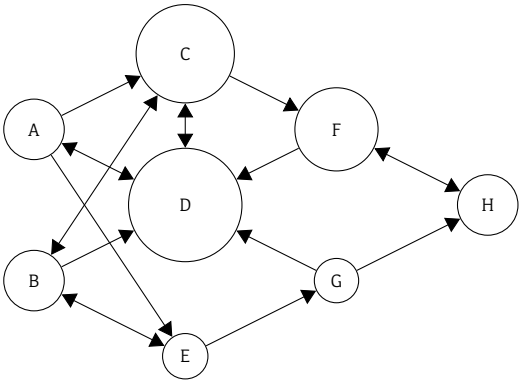
$$C_{PR}(i) = \sum_{j \in \mathcal{N}_i^{in}} \frac{C_{PR}(j)}{|\mathcal{N}_j^{out}|} \tag{3.15}$$

$\mathcal{N}_i^{in}$ et $\mathcal{N}_i^{out}$ représentent respectivement l'ensemble des nœuds pointant vers i, et l'ensemble des nœuds vers lesquels i pointe. $|\mathcal{N}_i^{out}|$ est donc égal au degré sortant du nœud



$$
\begin{aligned}
c_d(A) &= 0.2387\\
c_d(B) &= 0.0674\\
c_d(C) &= 0.0674\\
c_d(D) &= 0.0674\\
c_d(E) &= 0.1181\\
c_d(F) &= 0.2387\\
c_d(G) &= 0.0674\\
c_d(H) &= 0.0674\\
c_d(I) &= 0.0674
\end{aligned}
$$

(a) G_1



$$
\begin{aligned}
c_{de}(A) &= 0.1111\\
c_{de}(B) &= 0.1007\\
c_{de}(C) &= 0.1712\\
c_{de}(D) &= 0.2174\\
c_{de}(E) &= 0.0788\\
c_{de}(F) &= 0.1598\\
c_{de}(G) &= 0.0522\\
c_{de}(H) &= 0.1089
\end{aligned}
$$

(b) G_2

FIGURE 3.12 : Application du PageRank aux graphes G_1 et G_2

i.

Le PageRank est une version de la centralité spectrale qui intègre le dégré sortant des voisins. On peut interpréter le PageRank de manière probabiliste. On considère alors un "surfeur" aléatoire explorant successivement des sites web en cliquant sur les liens qu'il trouve. Il y a une probabilité $(1-d)$ qu'à un moment donné, ce surfeur cesse son activité. La constante d est appelée *facteur d'amortissement* [5]. Sa valeur recommandée est 0.85. Elle est intégrée dans le PageRank de la manière suivante :

$$C_{PR}(i) = \frac{1-d}{n} + d \cdot \sum_{j \in \mathcal{N}_i^{in}} \frac{C_{PR}(j)}{|\mathcal{N}_j^{out}|} \tag{3.16}$$

Bilan

Nous avons vu différentes approches permettant de mesurer la centralité des nœuds d'un graphe. L'ensemble des valeurs est résumé dans le tableau 3.1 de manière à pouvoir

5. « damping factor »

comparer les mesures entre elles.

Ces approches ne représentent qu'un fragment de l'ensemble des mesures de centralité existantes. On remarque cependant que le calcul de ces mesures peut nécessiter soit des données *locales* comme c'est le cas pour le degré, soit au contraire, des données globales (ensemble de plus courts chemins, matrice d'adjacence) pour le reste des mesures. L'utilisation de données locales permet un traitement décentralisé de la mesure et par conséquent de s'affranchir des problèmes posés par les *grands* graphes. En revanche, le risque lorsque l'on utilise de telles données est de trouver des optima locaux, en l'occurrence attribuer une forte centralité à des nœuds qui ne seraient centraux que *localement* et non pas à l'échelle du graphe. A contrario, l'utilisation de données globales permet des solutions valides pour l'ensemble du graphe mais implique une centralisation de la structure du graphe ainsi qu'une charge de calcul plus importante.

On constate dans le tableau 3.1 que le classement des nœuds du graphe G_1 est identique entre les différentes approches, à une exception près pour la proximité. La comparaison du degré avec les autres mesures est plus délicate en ce qui concerne le graphe G_2 du fait que ce graphe est orienté. Globalement, les classements sont différents, avec seulement une similarité dans la tête de classement de l'intermédiarité et du PageRank.

3.2 Structures

Il est régulièrement question dans ces travaux de *structures*, c'est pourquoi nous allons prendre le temps de développer ce concept. Nous considérons une structure comme un ensemble de nœuds et d'arêtes appartenant à un même graphe qui sera en règle générale associée à une propriété sémantique. La structure en elle-même est un objet général, qui peut contenir des arêtes sans leurs extrémités par exemple. Un exemple basique de structure est le plus court chemin entre deux nœuds A et B contenant l'ensemble des arêtes consécutives permettant d'aller de A à B ainsi que les extrémités de ces arêtes et dont la sémantique serait qu'il n'existe pas de structure plus petite permettant de joindre A et B.

La difficulté introduite par la dynamique est de maintenir la propriété propre à la structure au travers des modifications qui s'opèrent dans le graphe. En reprenant l'exemple de plus court chemin, il faut adapter le contenu de la structure afin que la propriété *"chemin le plus court"* reste toujours valable.

Une approche naïve de ce problème serait de recalculer une nouvelle solution au problème à chaque modification significative du graphe. Mais peut-on dans ce cas parler de *maintien* de la solution ? Non. Le maintien de la solution implique la réutilisation d'une solution précédemment calculée dans le but de réduire le temps de calcul nécessaire. De plus, on cherche à ce que cette solution varie le moins possible tant qu'elle reste valide : il peut y avoir plusieurs plus courts chemins entre deux nœuds, tant que la solution initialement retenue fait partie des solutions optimales il est préférable de la conserver plutôt que d'osciller entre différentes solutions. Ce dernier point est important lorsque l'on s'intéresse au taux de renouvellement de la solution.

	Degré	Proximité	Intermédiarité	PageRank
		G_1		
A	0.250000	0.139619	0.346154	0.238738
B	0.062500	0.095195	0.000000	0.067399
C	0.062500	0.095195	0.000000	0.067399
D	0.062500	0.095195	0.000000	0.067399
E	0.125000	0.149592	0.307692	0.118131
F	0.250000	0.139619	0.346154	0.238738
G	0.062500	0.095195	0.000000	0.067399
H	0.062500	0.095195	0.000000	0.067399
I	0.062500	0.095195	0.000000	0.067399
		G_2		
A	→ 0.166667 ← 0.055556	0.145626	0.116667	0.111132
B	→ 0.166667 ← 0.111111	0.145626	0.075000	0.100724
C	→ 0.166667 ← 0.166667	0.145626	0.166667	0.171158
D	→ 0.111111 ← 0.277778	0.124823	0.283333	0.217368
E	→ 0.111111 ← 0.111111	0.124823	0.125000	0.078776
F	→ 0.111111 ← 0.111111	0.109220	0.150000	0.159766
G	→ 0.111111 ← 0.055556	0.124823	0.058333	0.052230
H	→ 0.055556 ← 0.111111	0.079433	0.025000	0.108847

TABLE 3.1 : Comparaison des différentes centralités pour les graphes G_1 et G_2. Les valeurs ont été normalisées afin de permettre la comparaison. → symbolise le degré sortant, ← le degré entrant.

3.2.1 Communautés et organisations

Dans notre vie quotidienne, nous utilisons de manière intuitive le concept de communauté afin de désigner un groupe de personnes en fonction d'une caractéristique partagée par les membres du groupe. Il peut s'agir par exemple de la *communauté* scientifique, d'une communauté religieuse, ou encore la communauté open-source. Tous ces exemples montrent l'idée que nous avons d'une communauté : un ensemble d'individus qui interagissent entre eux grâce à un intérêt commun. Lorsque l'on considère une population, il existe donc un réseau d'interactions qui émerge des liaisons entre individus générées par divers centres d'intérêt.

Parmi les propriétés pouvant être attribuées aux structures, celles de *communauté* et d'*organisation* nous intéressent particulièrement. Ces deux propriétés sont étroitement liées et portent sur des groupes de nœuds fortement connectés comparés à l'ensemble du graphe. Dans un contexte statique, nous nommerons ces structures *communautés*. Lorsque la structure possède une base qui se maintient avec la dynamique, on parle alors d'*organisation*. Du fait du contexte dynamique dans lequel s'effectuent nos travaux, nous nous focaliserons uniquement sur la notion d'organisation par la suite.

Les structures d'organisation soulèvent deux problèmes majeurs. Tout d'abord, comment détecter ces structures ? Puis, comment maintenir les structures détectées au travers de la dynamique du graphe. Maintenir l'organisation signifie que cette dernière possède une signification lorsqu'on l'observe à une échelle macroscopique et qu'il faut adapter le contenu de la structure afin de conserver cette signification.

Nous allons considérer un exemple afin d'illustrer ce que nous entendons par *maintenir* l'organisation. Si l'on considère un graphe modélisant les interactions dans un réseau social : chaque nœud représente une personne et lorsque deux personnes discutent entre elles, une arête est créée entre les nœuds qui les représentent. Si ces mêmes personnes ne discutent plus un certain temps, l'arête est supprimée. Si l'on considère un centre d'intérêt I, on suppose que les personnes intéressées par I vont discuter entre elles et ainsi former une organisation. Au fil du temps, certaines personnes vont cesser d'être intéressées par I et au contraire d'autres vont le devenir, modifiant de ce fait la structure de l'organisation. L'organisation engendrée par I se maintient donc dans le temps même si sa structure change.

Définitions

On trouve dans [BOCCALETTI et al. 2006] un rappel sur les différentes définitions de *communauté*. Tout d'abord une définition *historique* qui considère que dans un graphe G une communauté est un sous-graphe G' de G dans lequel les nœuds sont fortement connectés, c'est à dire qu'il existe une forte cohésion entre les membres.

La définition la plus forte considère que les communautés sont des cliques, c'est à dire un ensemble de nœuds maximal tel que chaque membre est connecté à tous les autres. Cette définition peut être *affaiblie* par le concept de *k-clique* : chaque membre du sous-ensemble de nœuds est relié à tous les autres par au maximum k intermédiaires. Les auteurs présentent

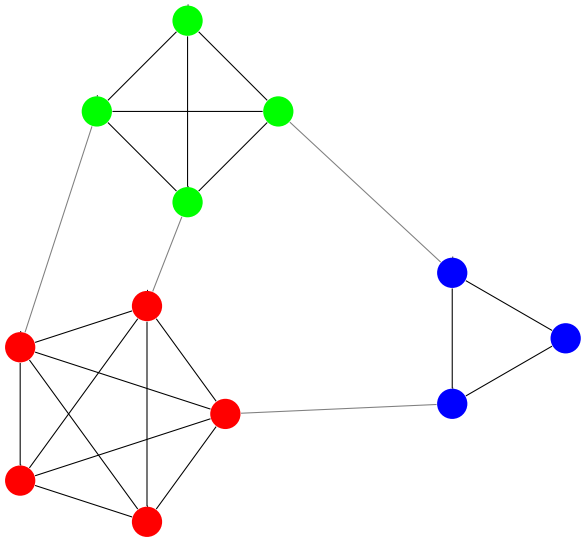


FIGURE 3.13 : 3 communautés formant des cliques

enfin la notion de *k-plex* afin d'affaiblir la définition d'une communauté : il s'agit d'un sous-graphe à n sommets dans lequel chaque sommet est connecté à au moins $n - k$ membres.

Enfin, une autre façon d'aborder la définition d'une communauté est présentée : il s'agit d'utiliser la fréquence des arêtes pour mesurer la cohésion entre les membres d'un groupe. On peut trouver plusieurs définitions de telles mesures dans [WASSERMAN et FAUST 1994].

3.2.2 Forme

La manière dont les nœuds d'une même structure interagissent entre eux et avec les nœuds extérieurs permet de donner une forme à la structure. Cette forme va nous permettre de *classer* les structures afin d'associer des propriétés aux différentes classes.

Les chemins forment une classe de formes dont les membres sont peu connectés entre eux (de une à deux connections) mais peuvent potentiellement être connectés à une quantité importante de nœuds extérieurs. Ce type de forme est fragile dans le sens où le retrait d'un seul élément peut entraîner une perte de connexité de la structure.

Les structures dont tous les membres sont connectés forment les *cliques* ou sous-graphes complets. Contrairement aux chemins, cette forme de structure est solide du fait qu'il est nécessaire de retirer au moins $n - 1$ éléments afin de déconnecter la structure.

Ces deux exemples, quelque peu extrêmes, nous permettent de mettre en avant que la *densité* de connections au sein d'une structure est un des éléments qui va permettre de définir sa forme. Plus la valeur de cette mesure sera élevée plus la structure sera difficile à déconnecter par la suppression d'arêtes ou de nœuds. Les chemins ont donc une densité faible, tandis que celle des cliques est maximale (il n'est pas possible de connecter davantage les nœuds d'une clique).

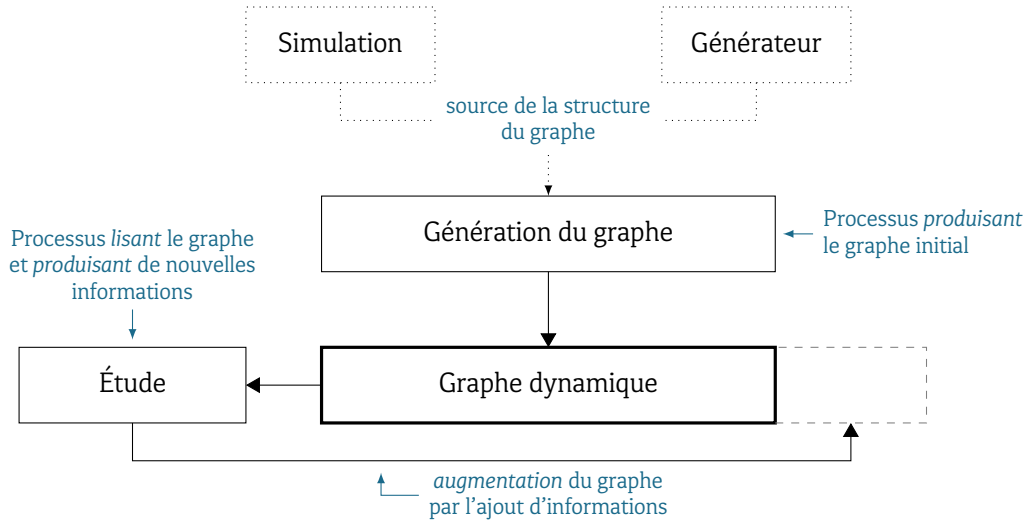


FIGURE 3.14 : Concept de processus. Un processus initial est utilisé pour générer le graphe. Un autre processus est utilisé pour analyser le graphe et l'augmenter grâce aux informations calculées.

3.3 Processus

Nous avons vu qu'un graphe dynamique est composé d'une structure de nœuds et d'arcs qui évolue au fil du temps. On peut alors se poser la question : quelle est la source de cette dynamique ? La source initiale de cette dynamique est la source même du graphe qui, dans le cadre de nos travaux, sera principalement une simulation. Cependant, la dynamique prend aussi ses sources dans d'autres mécanismes tels que des algorithmes qui permettent, en ajoutant des informations, *d'augmenter* le graphe.

D'une manière générale, nous désignons ces mécanismes sous le terme de *processus* qui pourront agir indépendamment du graphe produit ou au contraire utiliser le graphe existant dans le but de l'augmenter. Comme nous l'avons vu dans le paragraphe précédent, un processus peut désigner un générateur de la structure du graphe ou encore un algorithme maintenant une structure (arbre couvrant, plus court chemin) ou une mesure sur le graphe, mais il peut s'agir aussi d'une information telle que le flot pouvant parcourir un graphe. Nous présentons dans la figure 3.14 le concept le processus et de quelle manière ces processus viennent s'insérer dans la construction et l'étude du graphe.

Un processus ne possède pas de présence concrète dans le graphe. Il s'agit d'un objet abstrait dont la présence n'est perçue qu'au travers des informations qu'il produit à partir d'informations fournies par un graphe et/ou d'autres processus. Dans certains cas, le processus ne nécessite pas d'information en entrée comme dans le cas par exemple d'un générateur de graphe basé sur un algorithme [BARABÁSI et ALBERT 1999 ; DOROGOVTSEV et MENDES 2002]. Le processus peut utiliser les informations qu'il a lui-même produites dans

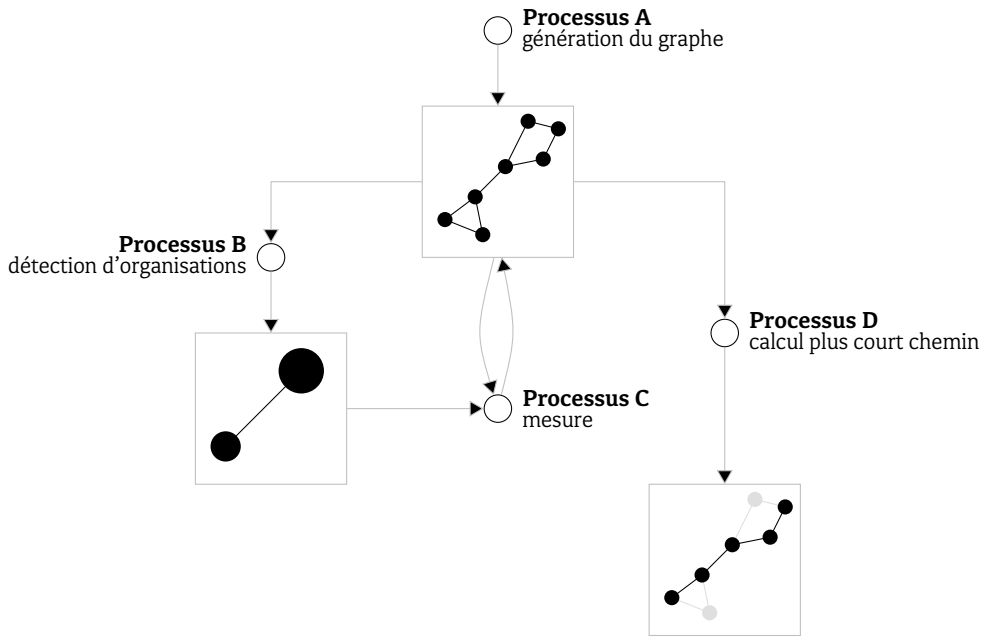


FIGURE 3.15 : Un graphe dynamique vu comme le résultat d'un ensemble de processus

le graphe afin d'affiner ces informations ou de les mettre à jour pour prendre en compte des changements qui seraient survenus dans le graphe : il y a alors une rétroaction du processus sur lui-même.

3.3.1 Exemple

La figure 3.15 nous présente un système dans lequel les processus A, B, C et D cohabitent afin de créer un graphe dynamique. Le processus A est un générateur qui est capable de fournir les informations nécessaires à la création de la structure du graphe. Les processus B et D permettent de maintenir des structures : une vue macro du graphe initial dans laquelle les nœuds représentent des organisations dans le cas du processus B, et un plus court chemin entre deux nœuds dans celui du processus D. Enfin, le processus C calcule une mesure en utilisant le graphe initial ainsi que la vue macro générée par le processus B puis retransmet les nouvelles informations mesurées afin d'augmenter les informations contenues initialement dans le graphe.

3.3.2 Propriétés

Une première propriété qui permet de caractériser les processus concerne la nature du lien qui permet de transmettre les informations. Cette transmission peut se faire de façon continue ou discrète comme cela est illustré dans la figure 3.16. Dans le cas d'une trans-

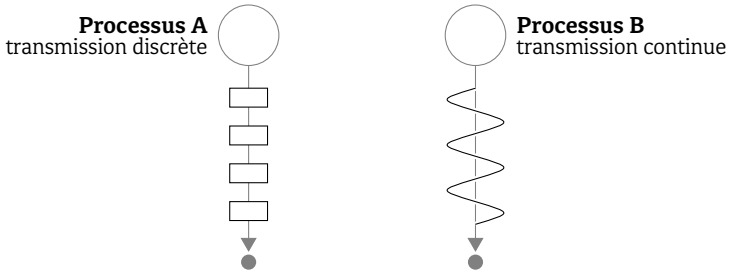


Figure 3.16 : Transmission discrète et continue : le processus A transmet les informations comme une suite de messages contrairement au processus B dont le résultat prend la forme d'un flux continu d'informations.

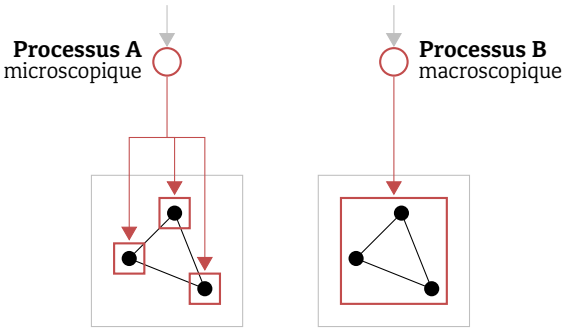


Figure 3.17 : Processus micro/macro.

mission discrète, les informations modifiant le graphe prennent la forme d'une séquence, c'est pourquoi on parlera de processus *itératif*. Au contraire, dans un processus *continu*, la transmission prend la forme d'un flux continu d'informations.

On peut différencier les processus en fonction de leur échelle d'action. On dira d'un processus qu'il opère à l'échelle *microscopique* lorsque les informations qu'il produit ont pour cible les nœuds ou les arêtes du graphe comme par exemple un processus qui mesurerait le degré des nœuds. Cependant, lorsque les informations produites concernent le graphe dans son intégralité, comme par exemple le degré maximum, on dira que le processus opère à une échelle *macroscopique*.

Une propriété des processus proche de celle de micro/macro est la notion de *local/global*. Un processus agissant localement ne nécessite qu'une partie locale du graphe pour produire de l'information. Au contraire, lorsqu'un processus agit globalement il a besoin de l'ensemble du graphe afin de pouvoir fonctionner.

Ces deux notions de micro/macro et local/global semblent proches au premier abord mais sont cependant bien distinctes : la notion de microscopique/macroscopique est liée à la sortie du processus tandis que celle de local/global concerne l'entrée du processus. Il est donc tout à fait possible de considérer un processus microscopique et global ou encore un processus

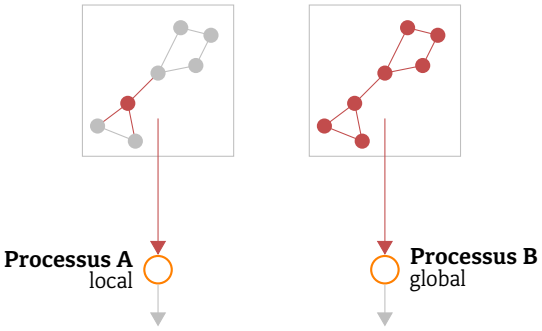


FIGURE 3.18 : Processus local/global.

macroscopique et local.

Références

BARABASI, A. et E. BONABEAU (2003). "Scale-Free Networks". In : *Scientific American*, p. 50–59.

BARABÁSI, Albert-László et Réka ALBERT (1999). "Emergence of Scaling in Random Networks". In : *Science* 286 (5439), p. 509–512. DOI : 10.1126/science.286.5439.509. URL : http://www.sciencemag.org/content/286/5439/509.abstract.

BOCCALETTI, S. et al. (2006). "Complex Networks : Structure and Dynamics". In : *Phys. Rep.* 424 (4-5), p. 175–308.

BORŮVKA, Otakar (1926). "O jistém problému minimálním (About a certain minimal problem)". In : *Práce mor. přírodověd. spol. v Brně* III (3), p. 37–58.

BRIN, Sergey et Lawrence PAGE (1998). "The anatomy of a large-scale hypertextual Web search engine". In : *Computer networks and ISDN systems* 30 (1), p. 107–117. ISSN : 0169-7552.

BUI-XUAN, B., A. FERREIRA et A. JARRY (avr. 2003). "Computing shortest, fastest, and foremost journeys in dynamic networks". In : *International Journal of Foundations of Computer Science* 14 (2), p. 267–285.

CASTEIGTS, A. et al. (2012). "Time-Varying Graphs and Dynamic Networks". In : *International Journal of Parallel, Emergent and Distributed Systems*. In press. DOI : 10.1080/17445760.2012.668546.

CHIKHI, Nacim Fateh (2010). "Calcul de centralité et identification de structures de communautés dans les graphes de documents." Université de Toulouse III. URL : http://thesesups.ups-tlse.fr/1364/.

CORTES, Corina, Daryl PREGIBON et Chris VOLINSKY (déc. 2003). "Computational Methods for Dynamic Graphs". In : *Journal of Computational & Graphical Statistics* 12 (4). ISSN 1061-8600, Online ISSN : 1537-2715, 950–970(21).

Demetrescu, Camil et Giuseppe F. Italiano (2006). "Fully dynamic all pairs shortest paths with real edge weights". In : *J. Comput. Syst. Sci.* 72 (5), p. 813–837. ISSN : 0022-0000. DOI : 10.1016/j.jcss.2005.05.005.

Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs". In : *Numerische Mathematik* 1, p. 269–271.

Dorogovtsev, S. N. et J. F. F. Mendes (2002). "Evolution of networks". In : *Adv. Phys*, p. 1079–1187.

Euler, Leonhard (1741). "Solutio problematis ad geometriam situs pertinentis". In : *Commentarii academiae scientiarum Petropolitanae* 8, p. 128–140.

Ferreira, Afonso (mai 2002). "On models and algorithms for dynamic communication networks : The case for evolving graphs". In : *4e rencontres francophones sur les Aspects Algorithmiques des Telecommunications (ALGOTEL'2002)*. Mèze, France.

Floyd, Robert W. (1962). "Algorithm 97 : Shortest Path". In : *Communications of the ACM* 5 (6), p. 345.

Ford, Lester Randolph (1956). *Network Flow Theory*.

Freeman, Linton C. (1977). "A set of measures of centrality based upon betweenness". In : *Sociometry* 40, p. 35–41.

— (1978). "Centrality in social networks conceptual clarification". In : *Social Networks* 1 (3), p. 215–239. ISSN : 0378-8733. DOI : 10.1016/0378-8733(78)90021-7. URL : http://www.sciencedirect.com/science/article/pii/0378873378900217.

Girvan, M. et M. E. J. Newman (2002). "Community structure in social and biological networks". In : *Proc. Natl. Acad. Sci. USA* 99, p. 8271–8276.

Grinstead, Charles M. et J. Laurie Snell (1997). *Introduction to Probability*. ISBN : 978-0-8218-9414-9. URL : http://www.dartmouth.edu/~chance/teaching_aids/books_articles/probability_book/book.html.

Harary, F. et G. Gupta (1997). "Dynamic Graph Models". In : *Mathematical and Computer Modelling* 25 (7), p. 79–87.

Harary, Frank (oct. 1969). *Graph Theory*. Published : Paperback. Westview Press. ISBN : 0201410338. URL : http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0201410338.

Johnson, Donald B. (1977). "Efficient Algorithms for Shortest Paths in Sparse Networks". In : *J. ACM* 24 (1), p. 1–13. ISSN : 0004-5411. DOI : 10.1145/321992.321993. URL : http://doi.acm.org/10.1145/321992.321993 (visité le 17/10/2013).

Jordan, Camille (1869). "Sur les assemblages des lignes". In : *J. Reine Angew. Math* 70, p. 185–190.

Katz, Leo (1953). "A New Status Index Derived from Sociometric Index". In : *Psychometrika*, p. 39–43.

Newman, M. E. J. (2003). "The Structure and Function of Complex Networks". In : *SIAM Review* 45, p. 167–256. DOI : 10.1137/S003614450342480.

Newman, Mark (2010). *Networks : An Introduction*. New York, NY, USA : Oxford University Press, Inc. ISBN : 0199206651, 9780199206650.

Ore, Oystein (1962). *Theory of graphs*. English. American Mathematical Society, Providence. v. P.

Pigné, Yoann (2008). "Modélisation et traitement décentralisé des graphes dynamiques : Application aux réseaux mobiles ad hoc". French. Université du Havre. URL : http://tel.archives-ouvertes.fr/tel-00371962/PDF/these.pdf.

Raberto, Marco, Fabio Rapallo et Enrico Scalas (2011). "Semi-Markov Graph Dynamics". In : *PLoS ONE* 6 (8).

Reynolds, Craig W. (août 1987). "Flocks, herds and schools : A distributed behavioral model". In : *SIGGRAPH Comput. Graph.* 21 (4), p. 25–34. ISSN : 0097-8930. DOI : http://doi.acm.org/10.1145/37402.37406. URL : http://doi.acm.org/10.1145/37402.37406.

Slater, Peter J. (1978). "Centers to centroids in graphs". In : *Journal of Graph Theory* 2 (3), p. 209–222.

Strogatz, S. H. et D. J. Watts (1998). "Collective dynamics of 'small-world' networks". In : *Nature* 393, p. 440–442.

Wasserman, Stanley et Katherine Faust (1994). *Social Networks Analysis*. Cambridge University Press.

Chapitre 4

Répartition de charge

Dans le cadre de la distribution d'une application, nous avons besoin de déterminer comment *répartir* les différents composants de l'application. Nous utilisons pour cela un algorithme de répartition de charge qui permet de calculer une répartition de manière à équilibrer le travail des différentes machines.

L'application que l'on souhaite distribuer est une simulation de système complexe dont les composants à répartir sont les entités du système. Les particularités de la répartition dont nous avons besoin sont d'une part la dynamique liée au système, et d'autre part la volonté de réduire la consommation de la bande passante du réseau. La quantité massive d'interactions

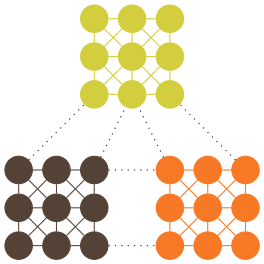


FIGURE 4.1 : Un exemple de partitionnement d'un graphe. Chaque couleur représente une partie. Les lignes hachurées représentent les arêtes entre les parties que l'on souhaite minimiser.

au sein du système, qui se traduisent par des communications distantes lorsque ce système est distribué, justifie ce besoin de tenir compte de cette contrainte.

4.1 Partitionnement du graphe

Nous avons vu précédemment que le réseau d'interactions formé par les entités peut être modélisé par un graphe dynamique. Répartir les entités sur un ensemble de machines consiste donc à calculer une classification des nœuds du graphe pour laquelle chaque classe représente une machine. La classe d'un nœud nous permet alors de savoir sur quelle machine l'entité, correspondant au nœud, doit s'exécuter. Notre problème consiste donc à déterminer cette classification de manière à ce que les tailles des classes soient similaires pour obtenir une répartition équilibrée. De plus, on cherche à réduire la quantité d'arêtes reliant des nœuds de classes différentes.

On considère un graphe $G = (V, E)$, et l'espace $\mathcal{S}_k$ représentant l'ensemble des solutions possibles au problème du partitionnement en k parties. Une solution $S \in \mathcal{S}_k$ est un ensemble $\{C_1, \ldots, C_k\}$, les C_i formant un recouvrement de V et étant deux-à-deux disjoints :

$$\bigcup_i C_i = V$$
$$\forall C_i, C_j \in S,\ C_i \cap C_j = \emptyset$$

Un exemple de partition de graphe est donné figure sur la 4.1. Trouver la meilleure partition est un problème NP-difficile [SANDERS et SCHULZ 2013], qui nécessite la mise en place d'heuristique afin d'en trouver une solution approchée. On commence par définir une fonction $Q : \mathcal{S}_k \to \mathbb{R}$ nous permettant de mesurer la qualité d'une solution et, de ce fait, de comparer des solutions deux-à-deux. Si, pour deux solutions S et S', $Q(S) < Q(S')$, alors on considère S' comme meilleure que S. On définit ensuite une fonction de voisinage $\mathcal{N}(S)$ qui, à partir d'une solution existante S, va créer un ensemble de solutions voisines de S, à l'aide de transformations locales. On obtient alors une solution localement optimale lorsque $\forall S' \in \mathcal{N}(S),\ Q(S') \leq Q(S)$.

Partitionner le graphe est un problème similaire à la détection de communautés dans un graphe. On peut en effet considérer cette détection comme la recherche de la meilleure partition du graphe dont on ignore le nombre de parties. Dans les deux problèmes on cherche à connaître la structure de plus haut niveau dont chaque nœud fait partie. C'est pourquoi dans ce qui suit, nous présenterons des méthodes de partitionnement mais aussi des méthodes de détection de communautés.

4.2 Approches statiques

4.2.1 Technique de la bissection récursive

La bissection récursive n'est pas en soi un algorithme de partitionnement mais une technique définissant une manière de procéder qui doit permettre de diminuer la complexité globale. Elle consiste à utiliser un algorithme de partitionnement pour diviser le graphe en deux parties. On applique ensuite récursivement l'algorithme sur chaque partie jusqu'à obtenir le nombre de parties souhaitées.

Le nombre de parties de la solution est alors sous la forme 2^k, la technique impose en effet que ce nombre soit une puissance de 2. On part du principe que la complexité de procéder k fois à un 2-partitionnement est significativement plus faible que d'utiliser directement un 2^k-partitionnement. De plus, à chaque étape, les deux parties créées peuvent être traitées indépendamment, rendant la technique naturellement parallélisable.

Il a été prouvé dans [Garey et Johnson 1979] que trouver la solution optimale à ce problème est *NP*-complet. C'est pourquoi il est nécessaire d'utiliser une heuristique permettant de trouver une solution approchée.

Heuristique de Kernighan-Lin

Une heuristique, dite *gloutonne*, consiste à améliorer une solution initiale établie arbitrairement. L'amélioration est effectuée en procédant à un échange de nœuds deux-à-deux entre les deux parties qui permettent de réduire la somme du poids des arêtes entre ces parties. On procède à l'échange tant qu'il est possible d'obtenir un gain significatif. La complexité de cette heuristique est $0(n^2)$.

L'algorithme de Kernighan-Lin [Kernighan et S. Lin 1970] propose d'améliorer cette méthode gloutonne en échangeant plus d'un sommet à la fois. Le résultat devient meilleur mais la complexité de l'algorithme augmente en $O(n^3)$. Il existe cependant une amélioration proposée par Fiduccia et Mattheyses dans [Fiduccia et Mattheyses 1982] qui permet d'obtenir une complexité en $O(|E(G)|)$, $E(G)$ désignant l'ensemble des arêtes du graphe G.

4.2.2 Technique multi-niveaux

En complément de la bissection récursive, il est possible d'adopter une approche *multi-niveaux*. Cela consiste à créer un graphe de taille plus petite que le graphe d'origine en fu-

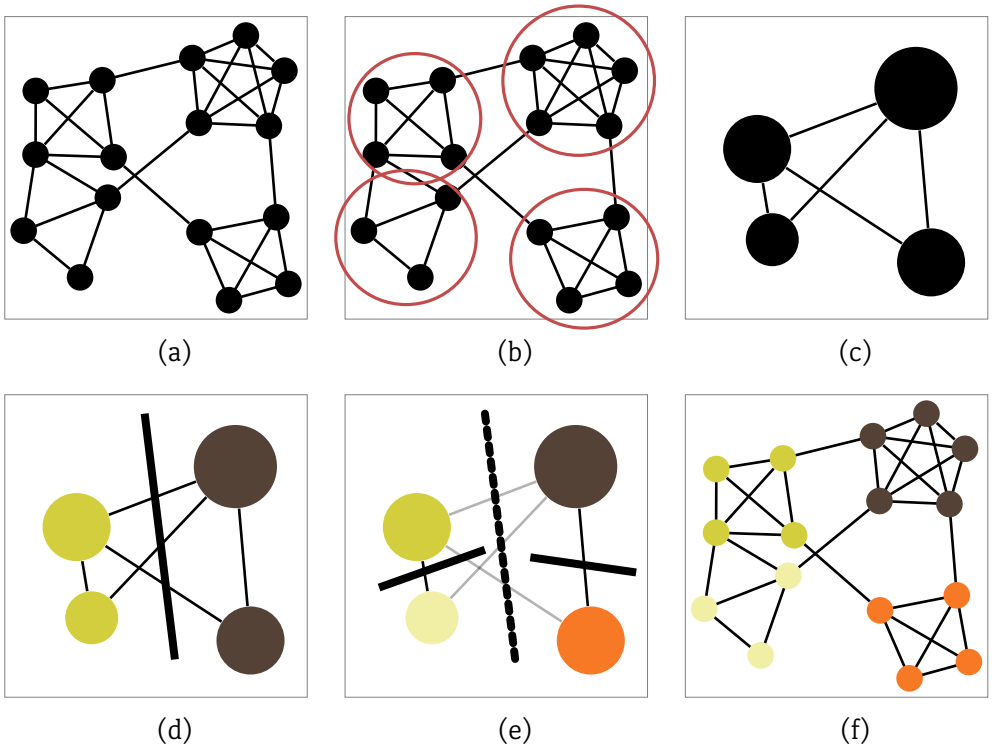


Figure 4.2 : Illustration de l'approche multi-niveaux appliquée au partitionnement. (a) présente le graphe initial, dans lequel on détermine un regroupement des nœuds en (b). Puis en (c) les nœuds des groupes sont fusionnés afin de créer un graphe d'un plus haut niveau. (d) et (e) montrent le calcul d'une partition par bissection. Le graphe est reconstruit en (f).

sionnant des sommets. Chaque sommet du graphe ainsi obtenu représente un groupe de sommets du graphe d'origine. Une fois qu'une partition est calculée sur ce graphe simplifié, il suffit de revenir au graphe d'origine. Cette procédure est présentée à l'aide d'un exemple trivial dans la figure 4.2.

Cette technique implique de devoir définir la façon dont les nœuds sont regroupés pour être fusionnés. Cela peut être fait de multiples façons, en regroupant par exemple les nœuds reliés entre eux par des arêtes ayant un poids élevé comme c'est le cas dans [Karypis et Kumar 1999], par exemple.

4.2.3 Classification hiérarchique

On peut étendre l'idée de l'approche multi-niveaux en créant une hiérarchie entre les partitionnements d'une partition composée des singletons contenant chaque nœud, à une partition contenant comme unique partie l'intégralité des nœuds du graphe. La construction

de cette hiérarchie peut se faire de manière *ascendante*. On commence alors avec une solution où chaque nœud représente une partie, puis on fusionne les parties deux-à-deux pour obtenir une nouvelle solution. Et ainsi de suite jusqu'à obtenir une seule partie contenant tous les nœuds. Au contraire, l'approche peut être *descendante*. On commence alors avec une solution contenant comme seule partie l'ensemble des nœuds, puis on divise la partie en deux, et ainsi de suite jusqu'à obtenir autant de parties que de nœuds.

On peut représenter cette hiérarchie, c'est à dire l'historique des fusions (hiérarchie ascendante), ou des divisions (hiérarchie descendante), à l'aide d'un *dendrogramme*. Un tel schéma est représenté sur la figure 4.3. À chaque étape, on calcule la qualité de la partition dans son état actuel. On peut alors connaître l'état pour lequel la qualité est maximale, ce qui nous donne le meilleur partitionnement.

Cette méthode nécessite la mise en place de deux mesures. Tout d'abord, une mesure de qualité que nous avons déjà abordée, qui nous permet de comparer des partitions afin d'évaluer laquelle est la meilleure. Ensuite, nous avons besoin d'une mesure de similarité $d(i, j)$ qui nous permette d'évaluer la similarité entre deux groupes de nœuds i et j. Cette mesure va ainsi nous permettre de choisir les nœuds, ou groupes de nœuds qui vont être fusionnés. Par exemple, dans le cas d'un graphe spatialisé, il peut s'agir de la distance euclidienne ou d'un dérivé ; dans ce cas, lorsque les groupes de nœuds contiennent plusieurs éléments, il est nécessaire d'appliquer un opérateur comme la moyenne ou le barycentre.

Mark E. J. Newman propose un algorithme de détection de communauté dans [M. E. J. Newman 2004] utilisant une méthode de *classification hiérarchique ascendante*, en présentant des résultats sur un graphe d'environ 50000 nœuds formés par un réseau de collaborations entre chercheurs. Il utilise une mesure de qualité Q, nommée *modularité*. En considérant e_{ij} la proportion d'arêtes connectant le groupe i au groupe j et $a_i = \sum_j e_{ij}$, la mesure de cette modularité est donnée par :

$$Q = \sum_i \left(e_{ii} - a_i^2\right) \tag{4.1}$$

L'algorithme procède par itérations. On considère une solution initiale où chaque nœud représente une communauté. Puis, à l'étape i, on choisit deux communautés à fusionner de telle sorte que $\Delta Q = Q_i - Q_{i-1}$ soit le plus grand possible, Q_i représentant la modularité à l'étape i. L'auteur montre ensuite que $\Delta Q = 2(e_{ij} - a_i a_j)$ ce qui peut être calculé en temps constant. La complexité d'une itération de l'algorithme est donc $O(n+m)$, n le nombre de nœuds, m le nombre d'arêtes. Ce qui amène à une complexité générale de $O((n+m)n)$.

Aaron Clauset, Mark E. J. Newman et Cristopher Moore proposent une amélioration de cet algorithme dans [Clauset, M. Newman et Moore 2004] en optimisant les structures de données nécessaires au calcul. Ils obtiennent ainsi une complexité $O(mdlogn)$, où d représente la profondeur du *dendrogramme* décrivant la hiérarchie des communautés. Ils présentent des résultats sur un graphe mettant en relation des livres à l'aide des informations de recommandations trouvées sur Amazon.com. Ils obtiennent ainsi un graphe de 400000 nœuds pour 2 millions d'arêtes. La table 4.1 présente les dix plus grandes communautés trouvées sur ce graphe.

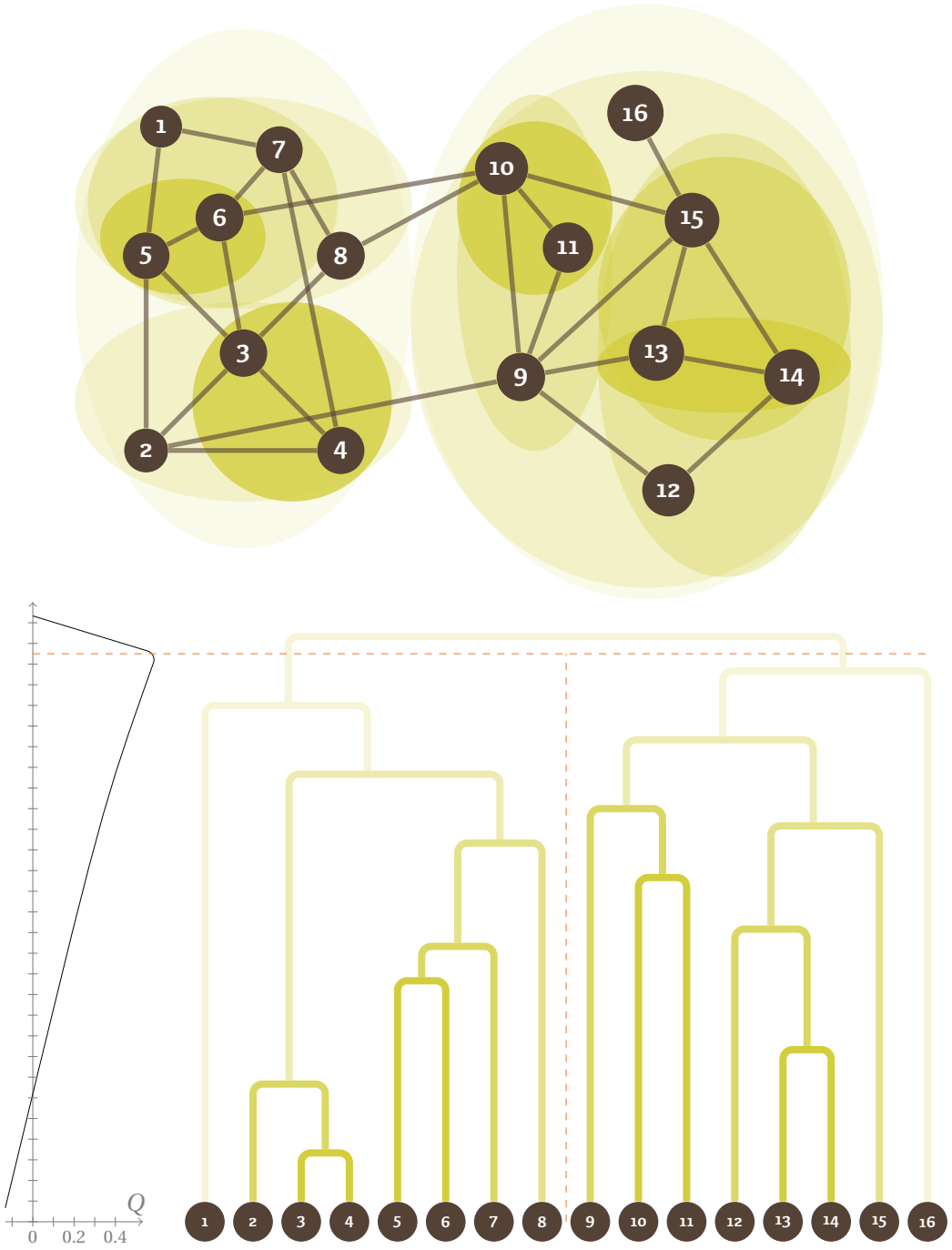


FIGURE 4.3 : Dendrogramme associé à une classification hiérarchique ascendante. La courbe de gauche représente l'évolution de la qualité Q au fur et à mesure des fusions. On observe que la qualité maximale est atteinte à l'avant dernière fusion, ce qui nous donne une bissection du graphe.

Rang	Taille	Description
1	114538	General interest : politics ; art/literature ; general fiction ; human nature ; technical books ; how things, people, computers, societies work, etc.
2	92276	The arts : videos, books, DVDs about the creative and performing arts
3	78661	Hobbies and interests I : self-help ; self-education ; popular science fiction, popular fantasy ; leisure ; etc.
4	54582	Hobbies and interests II : adventure books ; video games/comics ; some sports ; some humor ; some classic fiction ; some western religious material ; etc.
5	9872	classical music and related items
6	1904	children's videos, movies, music and books
7	1493	church/religious music ; African-descent cultural books ; homoerotic imagery
8	1101	pop horror ; mystery/adventure fiction
9	1083	jazz ; orchestral music ; easy listening
10	947	engineering ; practical fashion

TABLE 4.1 : Les 10 plus grandes communautés dans le réseau Amazon.com, représentant environ 87% des nœuds du réseau.

4.2.4 Théorie spectrale

Il existe également des méthodes de partitionnement utilisant la théorie spectrale, c'est à dire qu'elles utilisent les matrices et leurs valeurs/vecteurs propres. Ulrike VON LUXBURG dans [VON LUXBURG 2007] nous propose une introduction complète à ce type de partitionnement. Ces méthodes [POTHEN, SIMON et LIOU 1990 ; TERESCO, DEVINE et FLAHERTY 2006] nécessitent de construire une *matrice d'adjacence* qui représente les arêtes du graphe. Il s'agit d'une matrice carrée de dimension $n \times n$, où n correspond au nombre de nœuds dans le graphe. Chaque composant a_{ij} représente le poids de l'arête entre les nœuds i et j, ou, dans le cas d'un graphe non-pondéré, vaut 0 ou 1 pour indiquer la présence de l'arête. Si le graphe n'est pas orienté, sa matrice d'adjacence sera symétrique, c'est à dire que $\forall i, j \in V(G)$, $a_{ij} = a_{ji}$. La figure 4.4 nous montre un exemple de matrice d'adjacence. L'utilisation de la matrice d'adjacence permet d'augmenter la vitesse d'accès aux arêtes mais est coûteuse du point de vue de la mémoire nécessaire.

Matrice Laplacienne

On considère la matrice diagonale D dont chaque élément d_{ii} représente le degré du nœud. Dans le cas d'un graphe pondéré, d_{ii} représente la somme du poids des arêtes connec-

tées à i. On peut donc définir D de façon générale à partir de la matrice d'adjacence A :

$$d_{ij} = \begin{cases} \sum_{k=0}^{n} a_{ik}, & \text{si } i = j \\ 0, & \text{sinon} \end{cases}$$

La matrice *Laplacienne L non-normalisée* du graphe est alors :

$$L = D - A$$

Cette matrice peut être normalisée de deux façons [CHUNG 1997]. D'une part la Laplacienne normalisée symétrique L^{norm}, et d'autre part, une matrice L^{ma} correspondant à la matrice de transition d'une marche aléatoire sur le graphe :

$$L^{norm} = D^{-1/2} \cdot L \cdot D^{-1/2}$$
$$L^{ma} = D^{-1} \cdot L$$

Valeurs et vecteurs propres

Les méthodes spectrales utilisent les valeurs et vecteurs propres de la matrice Laplacienne, normalisée ou non, afin de calculer un partitionnement du graphe. Une *valeur propre* d'une matrice A est un réel λ pour lequel il est possible de trouver un vecteur $u \in \mathbb{R}^n$ tel que $A \cdot u = \lambda \cdot u$. Le vecteur u est alors appelé *vecteur propre* de A. Les valeurs propres de A sont les racines λ_i du *polynôme caractéristique* de A décrit par :

$$\det(A - \lambda \cdot I_n),$$

où det représente le déterminant de la matrice, et I_n la matrice identité de taille $n \times n$.

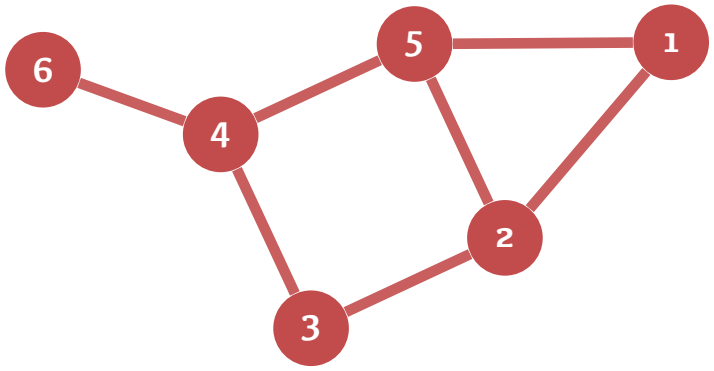
Calcul du partitionnement

La forme générale de la méthode de calcul permettant d'obtenir la partition nous est donnée dans l'algorithme 1. On calcule les vecteurs propres u_i de la Laplacienne, normalisée où non. Nous pouvons alors créer la matrice U donc les colonnes sont les vecteurs u_i. Les lignes de U vont former les vecteurs $\{y_1, \ldots, y_n\}$, chaque vecteur y_i étant associé au nœud i.

Nous allons ensuite partitionner l'espace formé par les points y_i en k classes C_i à l'aide d'un algorithme des k-moyennes. Cet algorithme permet de diviser un espace de n points en k parties uniformes. Sa version la plus populaire est l'algorithme de LLOYD [LLOYD 1982], aussi connu sous le nom d'itérations de VORONOI.

Dans le cas où la Laplacienne utilisée est L^{sym}, il est nécessaire de normaliser les lignes de U. Cela est dû au fait que l'utilisation de L^{sym} réduit la différence entre les valeurs propres, ce qui réduit la précision de l'algorithme des k-moyennes, et donc la qualité de la partition. Les nouvelles valeurs de U sont alors :

$$u'_{ij} = \frac{u_{ij}}{\sqrt{\sum_k u_{ik}^2}}$$



$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

$$L = \begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}$$

FIGURE 4.4 : Matrice d'adjacence et Laplacienne non-normalisée d'un graphe.

4.2.5 Approches bio-inspirées

Certaines techniques d'optimisation, inspirées de phénomènes que l'on trouve dans la nature, peuvent être utilisées afin de calculer une partition du graphe. C'est le cas du recuit simulé ou encore des algorithmes génétiques.

Recuit simulé

Le recuit simulé [KIRKPATRICK, VECCHI et GELATT 1983 ; ČERNÝ 1985] s'inspire de la technique du recuit utilisé en métallurgie afin de modifier les caractéristiques d'un métal. Elle consiste à augmenter progressivement la température du système puis à procéder à un refroidissement contrôlé. Appliquée à l'optimisation, cette technique consiste à minimiser l'énergie E que l'on attribue à la solution. On part d'une solution initiale dont on calcule la

Algorithme 1 : Calcul d'une partition à l'aide de la théorie spectrale

Entrées :
— G, un graphe dont une cherche une partition en k parties
Sorties :
— S, la solution

début
  $L \leftarrow$ calcul de la Laplacienne ;
  $u_1, \ldots, u_k \leftarrow$ calcul des k premiers vecteurs propres de L ;
  $U \leftarrow \begin{pmatrix} u_{1_1} & \ldots & u_{k_1} \\ \ldots & \ldots & \ldots \\ u_{1_n} & \ldots & u_{k_n} \end{pmatrix}$;
  pour $i \in \{1, \ldots, n\}$ **faire**
    $y_i \leftarrow i^{\text{ème}}$ ligne de U ;
  fin
  $\{C_1, \ldots, C_k\} \leftarrow$ k-means$(\{y_1, \ldots, y_n\})$;
  retourner $S = \{A_1, \ldots, A_k\}, A_i = \{j \mid y_j \in C_i\}$
fin

valeur initiale de E, E_0. On attribue au système une température T, initialement fixée à une valeur élevée T_0. À chaque étape, on procède à une modification de la solution en calculant, selon une méthode à définir, la variation d'énergie Δ_E que cette solution apporte. Si Δ_E est négatif, on procède à la modification de la solution ; sinon, on procède à cette modification avec une probabilité $e^{-\frac{\Delta_E}{T}}$. On procède à une diminution de la température T soit de manière continue, soit par palier au bout d'un certain nombre d'itérations. L'algorithme (cf.algorithme 2) se termine lorsque l'on atteint une température suffisamment basse.

Algorithme génétique

Un algorithme génétique [HOLLAND 1975] consiste à faire évoluer une population de solutions initiales de telle sorte que chaque nouvelle génération soit de meilleure qualité. Pour cela, on code les solutions sous la forme d'une chaîne, par analogie aux chaînes de nucléotides contenues dans l'ADN. On procède à la génération d'une nouvelle population par des mécanismes de *croisement* et de *mutation*. La figure 4.5 présente ces deux mécanismes ainsi que le codage de la solution.

Le croisement consiste à sélectionner deux individus A et B (voire plus) parmi la population actuelle, en fonction de leur qualité, et de les combiner afin de créer un nouvel individu.

Sélection Il s'agit du premier opérateur à définir, qui va directement influencer l'évolution de la population. Il englobe deux notions importantes : qui peut être sélectionné, et comment se passe la sélection. Dans [BÄCK 1994], T. BÄCK nous résume deux approches

Algorithme 2 : Recuit simulé appliqué au partitionnement

Entrées :
- G, un graphe G ;
- S_0, une solution initiale

Sorties :
- S, un partition de G

Données :
- T_0, la température initiale ;
- T_{arret}, la température à atteindre ;
- $\rho \in]0,1[$, le facteur de diminution de la température ;

```
1  début
2      T ← T_0 ;
3      S ← S_0 ;
4      tant que T > T_arret faire
5          pour S' ∈ N(S) faire
6              Δ_E ← Q(S) − Q(S') ;
7              si Δ_E < 0 alors
8                  S ← S' ;
9              fin
10             sinon si aléatoire() < e^(−Δ_E/T) alors
11                 S ← S' ;
12             fin
13         fin
14         T ← T × ρ ;
15     fin
16     retourner S
17 fin
```

qui définissent quelles populations sont en mesure de se reproduire. La première, nommée $(\mu + \lambda)$-sélection, implique que les parents **et** leurs descendants peuvent se reproduire. Au contraire, la sélection (μ, λ) indique que seuls les descendants participent à la reproduction, et dans ce cas, chaque population ne *vit* que pour une étape de l'algorithme. La sélection $(\mu+\lambda)$ permet de maintenir des individus de bonne qualité sur le long terme, mais de ce fait, nous augmentons le risque que la population de solutions se cantonne autour d'un optimum local.

Lorsque l'ensemble des individus sélectionnables est établi, il faut procéder à la sélection. Il existe différentes méthodes utilisant la qualité des individus et l'aléatoire. La *pression de sélection* définit le degré d'importance de la qualité dans la sélection. Plus elle sera élevée, meilleurs seront les individus sélectionnés. La méthode standard consiste à utiliser la qualité d'un individu de manière à ce que sa chance d'être sélectionné soit proportionnelle à sa

qualité. D'autres méthodes existent, comme par exemple la sélection par *tournoi* [MILLER et GOLDBERG 1995], dans laquelle on sélectionne s compétiteurs, dont on retient le meilleur.

Croisement On choisit ensuite un *locus* qui correspond à une position dans la chaîne codant la solution. La chaîne de A est alors coupée du début jusqu'au locus, puis est combinée avec la chaîne de B, elle-même coupée du locus à la fin de la chaîne.

L'intérêt de cette opération est remise en question [CHELLAPILLA et FOGEL 1999], du fait qu'il est nécessaire qu'elle soit adaptée au problème pour être efficace. Or si on utilise une méthode dédiée à un problème dans un contexte plus général, la qualité du résultat risque d'être pire que celui obtenu via une méthode aléatoire.

Mutation La mutation permet d'insérer de l'aléatoire en provoquant des modifications dans les chaînes codant les solutions. Ce mécanisme va ainsi permettre d'explorer plus en profondeur l'espace des solutions, en évitant de se cloisonner dans des optima locaux.

Ces mécanismes vont permettre la création d'une nouvelle population de solutions, de meilleure qualité que la précédente. On peut alors reproduire l'étape de création d'une nouvelle population jusqu'à atteindre une qualité souhaitée ou dépasser un nombre maximum itérations.

Colonie de fourmis

Nous verrons par la suite que nous utilisons un algorithme à base de colonie de fourmis pour le partitionnement du graphe (cf.10.2). Des approches similaires existent dans la littérature comme c'est le cas de celle proposée par Pascale KUNTZ et Dominique SNYERS dans [KUNTZ et SNYERS 1994] qui est appliquée aux graphes statiques. L'algorithme utilise q espèces d'*animats* pour déterminer une partition de taille p du graphe, avec $q \geq p$. Les animats vont se reproduire et mourir à un certain âge. La solution de partitionnement est donnée par la quantité d'animats sur les nœuds : lorsqu'une espèce est majoritairement présente sur un nœud, on dit alors que ce nœud est colonisé par l'espèce. L'homéostasie de ce système repose sur deux mécanismes de rétroaction : une rétroaction négative induite par la mort des animats lorsque leur âge dépasse une limite maximale, et une rétroaction positive grâce à la reproduction des animats lorsqu'ils se trouvent sur un nœud colonisé par leur espèce. Les animats se déplacent sur le graphe de manière à favoriser les nœuds colonisés par leur espèce et fuir le reste.

Dans [KUNTZ, LAYZELL et SNYERS 1997], les auteurs précédents en collaboration avec Paul LAYZELL proposent un algorithme utilisant des colonies de fourmis. Il s'agit cependant d'un modèle de fourmi différent de celui que nous allons manipuler du fait de l'absence de *phéromones*. Le graphe est projeté dans un espace que les fourmis vont pouvoir parcourir, avec la capacité de déplacer des nœuds. Cette méthode n'est pas sans rappeler celle du *tri du couvain* [FRANKS et SENDOVA-FRANKS 1992]. Les fourmis sont donc capables de transporter un nœud qui se trouverait sur leur position puis de le déposer à un autre point de l'espace après un certain nombre d'itérations. La probabilité de transporter un nœud augmente

Algorithme 3 : Algorithme génétique appliqué au partitionnement

Entrées :
- G, un graphe

Données :
- p, nombre d'individus par population
- Q_{max}, qualité maximale souhaitée
- itérations$_{max}$, nombre maximal d'itérations de l'algorithme

Sorties :
- S, une partition de G

```
1  début
2      P_0 ← ensemble initial de p solutions ;
3      itérations ← 0 ;
4      répéter
5          Q_max ← 0 ;
6          itérations ← itérations + 1 ;
7          pour tous les S_i ∈ P_0 faire
8              Q_i ← Q(S_i) ;
9              Q_max ← max(Q_max, Q_i) ;
10         fin
11         P_1 ← ∅ ;
           /* On normalise la suite des Q_i pour obtenir une suite dont
              la somme est égale à 1. Celle-ci est utilisée par la suite
              comme distribution de probabilités. */
12         d = normaliser({Q_i}, Q_max) ;
13         pour i ∈ {1, ..., p} faire
               /* la fonction croisement({S_i}, {p_i}) choisit deux individus à
                  l'aide de la distribution de probabilités {p_i}, puis
                  procède à un croisement de ces individus et retourne
                  « l'enfant » obtenu. */
14             C ← croisement(P_0, d) ;
               /* la fonction mutation(S) va modifier l'individu S par des
                  perturbations locales. */
15             C ← mutation(C) ;
16             P_1 ← P_1 + {C} ;
17             Q_max ← max(Q_max, Q(C)) ;
18         fin
19         P_0 ← sélection(P_0, P_1) ;
20     jusqu'à Q_max ≥ Q_fin ou itérations > itérations_max ;
21     retourner S ∈ P_0 tel que ∀S' ∈ P_0, Q(S) ≥ Q(S') ;
22 fin
```

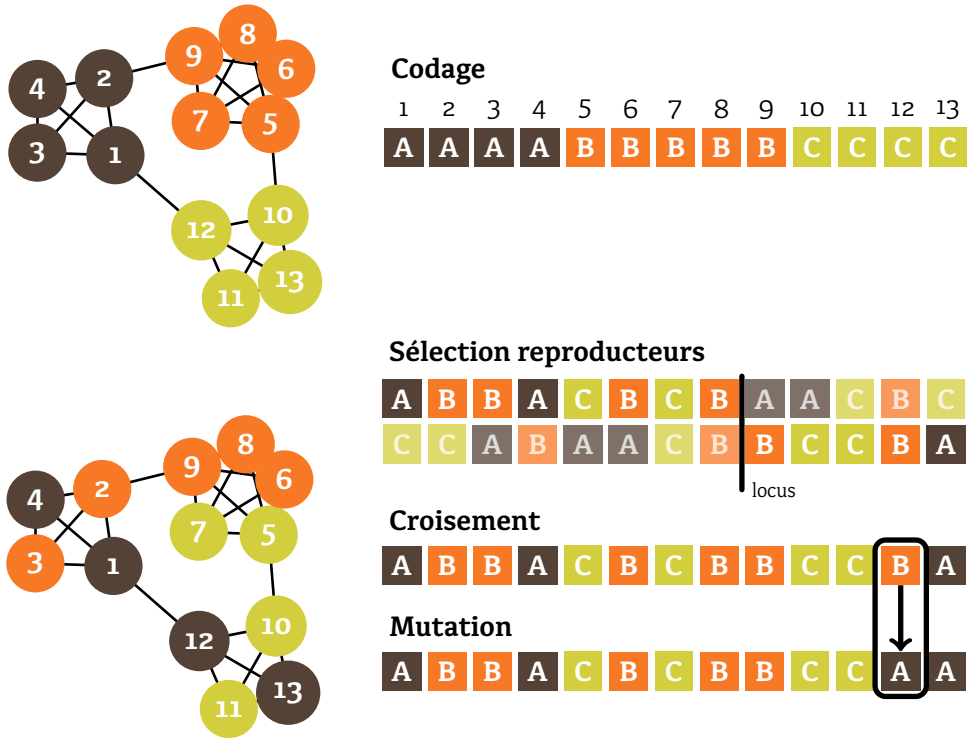


FIGURE 4.5 : Codage, croisement et mutation dans un algorithme génétique. On considère pour le codage que les nœuds peuvent être de numérotés de 1 à n, le code est alors une chaîne de caractères où le $i^{\text{ème}}$ caractère représente la classe du nœud i. Le croisement permet de combiner les codes de deux individus, et la mutation d'ajouter de l'aléatoire.

lorsque celui-ci est isolé. De même, la probabilité de déposer le nœud transporté augmente si la fourmi est entourée de nœuds similaires. La solution émerge du fait qu'en déplaçant les nœuds vers d'autres nœuds *similaires*, les fourmis vont faire que les nœuds d'une même partie du graphe vont être regroupés.

4.3 Approches dynamiques

Nous avons présenté jusqu'à présent des approches qui n'ont pas la capacité de prendre en compte la dynamique d'un graphe. Pour pouvoir en utiliser certains sur un graphe dynamique, cela nécessiterait de devoir calculer une nouvelle solution à partir de rien à chaque changement significatif qui surviendrait dans le graphe. D'autres peuvent être adaptés mais n'ont pas une approche prévue pour la dynamique et n'apportent donc pas les performances souhaitées. C'est pourquoi nous allons à présent nous intéresser aux approches permettant de gérer cette dynamique en adaptant une solution préalablement calculée aux modifications du graphe.

4.3.1 Approches par diffusion

Les algorithmes de *diffusion* constituent un premier type d'approche qui permet de prendre en compte la dynamique. Cependant, cette méthode ne manipule pas de graphe, mais seulement un ensemble d'entités.

On considère ici que le réseau formé par les ressources sur lesquelles nous souhaitons distribuer l'application n'est pas complet. Chaque ressource n'est donc pas en mesure de communiquer, et donc d'échanger une entité, avec n'importe quelle autre mais seulement avec un nombre restreint de voisins. Chaque ressource se voit attribuer une *température* qui correspond à la quantité d'entités qu'elle héberge actuellement, et donc à sa charge de calcul. Cette approche s'inspire alors de la manière dont la chaleur se diffuse des points chauds aux points froids pour échanger des entités entre les ressources de manière à équilibrer la température de ces ressources.

La diffusion peut se faire de deux façons. Tout d'abord, de manière *globale* comme c'est le cas dans [H.-C. Lin et Raghavendra 1992]. L'algorithme opère de manière centrale avec une vue globale sur les ressources qui lui permet de déterminer une solution de répartition des entités. Au contraire, l'algorithme de diffusion peut être lui même réparti sur l'ensemble des ressources. Chaque *instance* de l'algorithme ne possède alors qu'une vue locale des ressources : celle sur laquelle il est hébergé, et celles auxquelles cette dernière est connectée.

L'inconvénient des algorithmes de diffusion est lié au fait qu'il n'y a pas d'utilisation d'un graphe et donc de prise en compte des interactions qui existent entre les entités, or il s'agit d'un des objectifs que nous nous sommes fixés.

4.3.2 Approches par particules

L'approche particulaire proposée dans [Heiss et Schmitz 1995] permet d'ajouter aux algorithmes de diffusion cette prise en compte des interactions entre les entités. Cet algorithme s'inspire de la stabilisation que l'on obtient en plaçant des fluides de viscosités différentes dans un même bassin. Il considère un ensemble de tâches T, l'équivalent de nos entités, comme étant les particules de fluides ; les communications entre ces tâches sont utilisées pour définir la cohésion entre les particules. La charge potentielle des machines, quant à elle, est utilisée pour définir une force de gravitation. Le coût de la migration d'une tâche est alors défini comme une force de friction qui va permettre d'éviter les migrations intempestives et participer à la stabilisation du système.

Les auteurs définissent donc trois forces :

— l'équilibrage, qui va permettre de prendre en compte la charge des ressources ;
— la cohésion, elle correspond à la force induite par les communications entre les tâches ;
— la friction, son rôle est de limiter la quantité de migrations.

Ces forces sont respectivement pondérées par les constantes globales c_{lb}, c_{comm} et c_{frict}.

Pour une tâche t, dont le nombre d'instructions est s_t, et l'emplacement $loc(t) = j$, la

force d'équilibrage qui est exercée par une ressource k voisine de j est définie par :

$$f_{lb}^{j\to k}(t) = c_{lb}\frac{V_{load}^{j}+1}{V_{load}^{k}+1} \tag{4.2}$$

V_{load}^{k} représente la mesure de la charge de la ressource k et peut être définie de plusieurs façons, la plus précise étant :

$$V_{load}^{k} = \frac{1}{\mu_k}\sum_{t|loc(t)=k} s_t \tag{4.3}$$

μ_k permet de définir la vitesse de calcul de la ressource k, ce qui permet de prendre en considération les capacités de chaque ressource. La force de cohésion, quant à elle, est définie par :

$$f_{comm}^{j\to k}(t) = c_{comm}\left(V_{comm}^{j}(t) - V_{comm}^{k}(t)\right) \tag{4.4}$$

$V_{comm}^{k}(t)$ correspond au *potentiel de communication* de la tâche t sur la ressource k :

$$V_{comm}^{k}(t) = \sum_{t'\in T} a\left(k, loc(t')\right)\cdot c(t,t') \tag{4.5}$$

$a(j,k)$ définit le coût d'une communication entre les ressources k et j. L'intensité des communications entre deux tâches t et t' est donnée par $c(t,t')$. En cas d'absence de communications entre les tâches, $c(t,t')$ est nulle.

Enfin, la force de friction appliquée à la tâche t est la suivante :

$$f_{frict}^{j\to k}(t) = -c_{frict}d_t a(j,k) \tag{4.6}$$

d_t permet de définir le coût de la migration de la tâche t en mesurant la quantité de données qu'il serait nécessaire de transférer dans l'hypothèse de cette migration.

La force finale exercée sur la tâche t, située sur la ressource j, par la ressource k est donc :

$$f_{res}^{j\to k}(t) = f_{lb}^{j\to k}(t) + f_{comm}^{j\to k}(t) + f_{frict}^{j\to k}(t) \tag{4.7}$$

Nous avons vu des méthodes de partitionnement du graphes dans lesquelles il est nécessaire d'identifier des groupes de nœuds de manière à obtenir une faible quantité d'arêtes entre les groupes. La détection de groupes à l'intérieur desquels les membres entretiennent des interactions qui se démarquent des interactions avec l'extérieur du groupe relève du domaine de la détection de communautés.

4.3.3 Diffusion de labels

Dans [RAGHAVAN, ALBERT et KUMARA 2007], les auteurs proposent de détecter les communautés, dans des réseaux à grande échelle, à l'aide d'un mécanisme de *diffusion de labels*. Dans ce type d'algorithme, on considère que chaque nœud du réseau possède un *label* qui

définit la communauté dont il fait partie. Un nœud choisit alors de rejoindre la communauté la plus présente parmi ses voisins. On procède alors par itération, le nouveau label $\mathcal{L}'_x(t)$ d'un nœud x est calculé en fonction du label courant de ses voisins x_i :

$$\mathcal{L}'_x = f\left(\mathcal{L}_{x_1}, \ldots, \mathcal{L}_{x_k}\right) \tag{4.8}$$

La fonction f retourne le label dont la fréquence est la plus importante parmi les voisins. Pour pallier l'oscillation de labels qui peut apparaître en cas de présence de sous-graphe biparti, U. N. RAGHAVAN *et al.* proposent de désynchroniser le calcul du label d'un nœud en utilisant les labels de ses voisins qui ont déjà été calculés pour l'étape t :

$$\mathcal{L}'_x = f\left(\mathcal{L}'_{x_1}, \ldots, \mathcal{L}'_{x_m}, \mathcal{L}_{x_{m+1}} \ldots, \mathcal{L}_{x_k}\right) \tag{4.9}$$

Cet algorithme possède un inconvénient : il peut mener à la production de *super*-communautés. Ces structures sont liées à l'aspect *épidémique* de l'algorithme faisant que les communautés trop faiblement connectées sont absorbées pour former une communauté de plus haut niveau.

Ian X.Y. LEUNG *et al.* étudient et étendent, dans [LEUNG et al. 2009], l'algorithme par diffusion de labels cité ci-dessus, et en améliorent d'une part la complexité, et d'autre part la qualité de la détection. Chaque label $\mathcal{L}$ se voit attribuer un *score* $s_x(\mathcal{L})$ en chaque nœud x du réseau. Ce score va décroître à chaque itération. Le calcul du nouveau label de x devient alors :

$$\mathcal{L}'_x = \underset{\mathcal{L}}{\operatorname{argmax}} \sum_{x' \in \mathcal{N}_x} s_{x'}(\mathcal{L}_{x'}) \cdot d(x')^m \cdot w_{x,x'} \tag{4.10}$$

$\mathcal{N}_x$ correspond à l'ensemble des nœuds x' connectés à x par une arête dont le poids est donné par $w_{x,x'}$. $d(x)$ est une mesure permettant de comparer les nœuds entre eux ; il peut s'agir par défaut du degré. La constante m permet de relativiser l'importance de cette mesure d dans le calcul. La décroissance des scores est effectuée à l'aide de la constante δ de la façon suivante :

$$s'_x(\mathcal{L}'_x) = \left(\max_{x' \in \mathcal{N}_x(\mathcal{L}_x)} s_x(\mathcal{L}_{x'})\right) - \delta \tag{4.11}$$

où $\mathcal{N}_x(\mathcal{L}_x)$ désigne l'ensemble de voisins x' de x tel que $\mathcal{L}_{x'} = \mathcal{L}_x$.

Pour pallier la création de super-communautés, les auteurs proposent une seconde modification qui consiste à utiliser la distance d_G à l'origine du label $\mathcal{L}$, notée $\mathcal{O}(\mathcal{L})$, dans la mise à jour du score. Celle-ci devient alors :

$$s'_x(\mathcal{L}_x) = 1 - \delta\left(d_G(\mathcal{O}(\mathcal{L}_x), x)\right), \tag{4.12}$$

avec

$$d_G\left(\mathcal{O}(\mathcal{L}_x), x\right) = 1 + \min_{x' \in \mathcal{N}_x(\mathcal{L}_x)} d_G(\mathcal{O}(\mathcal{L}_x), x'). \tag{4.13}$$

Les auteurs montrent que l'algorithme ainsi modifié obtient de meilleurs résultats, temporel et qualitatif, sur le graphe d'un réseau social composé d'un million de nœuds pour 58 millions d'arêtes.

Références

Bäck, Thomas (1994). *Evolutionary algorithms in theory and practice*. T. Bäck.

Černý, V. (1985). "Thermodynamical approach to the traveling salesman problem : An efficient simulation algorithm". In : *Journal of Optimization Theory and Applications* 45, p. 41–51.

Chellapilla, Kumar et David B Fogel (1999). "Fitness distributions in evolutionary computation : motivation and examples in the continuous domain". In : *BioSystems* 54 (1), p. 15–29. ISSN : 0303-2647.

Chung, Fan RK (1997). *Spectral graph theory*. T. 92. AMS Bookstore. ISBN : 0821803158.

Clauset, Aaron, Mark Newman et Cristopher Moore (2004). "Finding community structure in very large networks". In : *Physical review E*. APS 70 (6), p. 066111.

Fiduccia, C. M. et R. M. Mattheyses (1982). "A linear time heuristic for improving network partitions". In : *ACM IEEE Design Automation Confe- rence, pages*, p. 175–181.

Franks, Nigel R. et A. B. Sendova-Franks (1992). "Brood sorting by ants : distributing the workload over the work-surface". In : *Behavioral Ecology and Sociobiology*. Springer 30 (2), p. 109–123.

Garey, Michael R et David S Johnson (1979). *Computers and intractability*. T. 174. Freeman New York.

Heiss, Hans-Ulrich et Michael Schmitz (1995). "Decentralized dynamic load balancing : The particles approach". In : *Information Sciences*. Elsevier 84 (1), p. 115–128.

Holland, John H. (1975). *Adaptation in Natural and Artificial Systems*. University Michigan Press.

Karypis, George et Vipin Kumar (1999). "A fast and high quality multilevel scheme for partitioning irregular graphs". In : *SIAM, Journal on Scientific Computing* 20 (1), p. 359–392.

Kernighan, B. W. et S. Lin (1970). "An Efficient Heuristic Procedure for Partitioning Graphs". In : *The Bell system technical journal* 49 (1), p. 291–307.

Kirkpatrick, Scott, Mario P Vecchi et D. Gelatt (1983). "Optimization by simulated annealing". In : *science* 220 (4598).

Kuntz, Pascale, Paul Layzell et Dominique Snyers (1997). "A colony of ant-like agents for partitioning in VLSI technology". In : *Proceedings of the Fourth European Conference on Artificial Life*. MIT, p. 417–424.

Kuntz, Pascale et Dominique Snyers (1994). "Emergent colonization and graph partitioning". In : *Proceedings of the third international conference on Simulation of adaptive behavior : from animals to animats 3*, p. 494–500.

Leung, Ian XY et al. (2009). "Towards real-time community detection in large networks". In : *Physical Review E*. APS 79 (6).

Lin, H.-C. et Cauligi S. Raghavendra (1992). "A dynamic load-balancing policy with a central job dispatcher (LBC)". In : *Software Engineering, IEEE Transactions on* 18 (2), p. 148–158.

Lloyd, Stuart (1982). “Least squares quantization in PCM”. In : *Information Theory, IEEE Transactions on* 28 (2), p. 129–137. ISSN : 0018-9448.

Miller, Brad L. et David E. Goldberg (1995). “Genetic Algorithms, Tournament Selection, and the Effects of Noise”. In : *Complex Systems* 9, p. 193–212.

Newman, Mark E. J. (2004). “Fast algorithm for detecting community structure in networks”. In : *Physical review E*. APS 69 (6), p. 066133.

Pothen, Alex, Horst D. Simon et Kang-Pu Liou (1990). “Partitioning sparse matrices with eigenvectors of graphs”. In : *SIAM, Journal on Matrix Analysis and Applications* 11 (3), p. 430–452.

Raghavan, Usha Nandini, Réka Albert et Soundar Kumara (2007). “Near linear time algorithm to detect community structures in large-scale networks”. In : *Physical Review E*. APS 76 (3), p. 036106.

Sanders, Peter et Christian Schulz (2013). “Think Locally, Act Globally : Highly Balanced Graph Partitioning”. In : *Experimental Algorithms*. Springer, p. 164–175. ISBN : 3642385265.

Teresco, J. D., K. D. Devine et J. E. Flaherty (2006). “Partitioning and Dynamic Load Balancing for the Numerical Solution of Partial Differential Equations”. In : *Numerical solution of partial differential equations on parallel computers*. Bruaset, Are Magnus. Springer.

Von Luxburg, Ulrike (2007). “A tutorial on spectral clustering”. In : *Statistics and computing* 17 (4), p. 395–416. ISSN : 0960-3174.

CHAPITRE 5

CALCUL DISTRIBUÉ

Jusqu'à présent, nous nous sommes intéressés à la modélisation des réseaux d'interactions dans le chapitre 3, puis à la répartition de charges dans le chapitre 4. Ces deux éléments nous fournissent la base théorique et algorithmique de la distribution. Dans ce chapitre, nous allons nous pencher sur les modèles existants de systèmes distribués, en tant qu'infrastructure, ainsi que sur l'algorithmique possible dans un tel environnement.

5.1 Système distribué

Avant de pouvoir procéder à un calcul distribué, il est nécessaire de mettre en place un environnement dans lequel nous allons pouvoir exécuter nos algorithmes. Vu de l'extérieur, cet environnement peut être considéré comme une machine de haut niveau. Nous allons donc examiner, dans un premier temps, de quoi cette machine est constituée, et comment définir l'architecture de son fonctionnement.

5.1.1 Le système distribué, cette *super*-machine

Un système distribué est composé de *machines* qui vont être reliées grâce à un réseau et combinées à des ressources : capacité de calcul, mémoire, stockage, voire des capteurs qui vont servir d'instruments de mesure permettant d'obtenir de l'information. Nous utilisons le terme de "machine" au sens large, c'est à dire qu'il désigne n'importe quel appareil, équipé d'au moins un *processeur* et de mémoire, qui lui permettent d'exécuter des flux d'instructions. Dans le cadre de nos travaux, ces flux d'instructions qui s'exécutent sur un ensemble de données forment, à un plus haut niveau, le comportement des entités. La manière dont les processeurs, d'une même machine, se synchronisent pour exécuter les instructions, permet une première classification des machines, proposée par Michael J. FLYNN en 1966 dans [FLYNN 1966] (cf.figure 5.1) :

SISD « Single Instruction, Single Data stream », les instructions sont exécutées de manière séquentielle sur une donnée à la fois ;

SIMD « Single Instruction, Multiple Data streams », une même instruction est exécutée sur plusieurs données différentes ;

MISD « Multiple Instructions, Single Data stream », plusieurs instructions sont exécutées sur une même donnée ;

MIMD « Multiple Instructions, Multiple Data streams », plusieurs instructions sont exécutées sur des données différentes.

Cette classification permet de définir comment les processeurs sont dépendants les uns des autres. Le type SISD correspond au fonctionnement d'une machine mono-processeur, telle que décrite dans l'architecture de VON NEUMANN [VON NEUMANN 1993]. Les types SIMD et MISD représentent des architectures multi-processeurs, où chaque processeur est lié aux autres soit pour l'instruction qui est exécutée, soit pour la donnée sur laquelle l'instruction est exécutée. Les processeurs dédiés aux graphismes sont un exemple d'architecture SIMD. Un processeur graphique est composé de nombreux *processeurs de flux* (plusieurs milliers sur des cartes haut de gamme récentes) fonctionnant en mode SIMD. On peut ainsi exécuter une même fonction (les instructions) sur les points (les données) en parallèle.

L'architecture MIMD représente un système composé de plusieurs processeurs capables de fonctionner indépendamment les uns des autres. Les ordinateurs personnels actuels sont principalement de type MIMD : bien qu'ils n'aient qu'un seul processeur physique, celui-ci est divisé en plusieurs processeurs logiques (les « cœurs ») qui sont capables de fonctionner

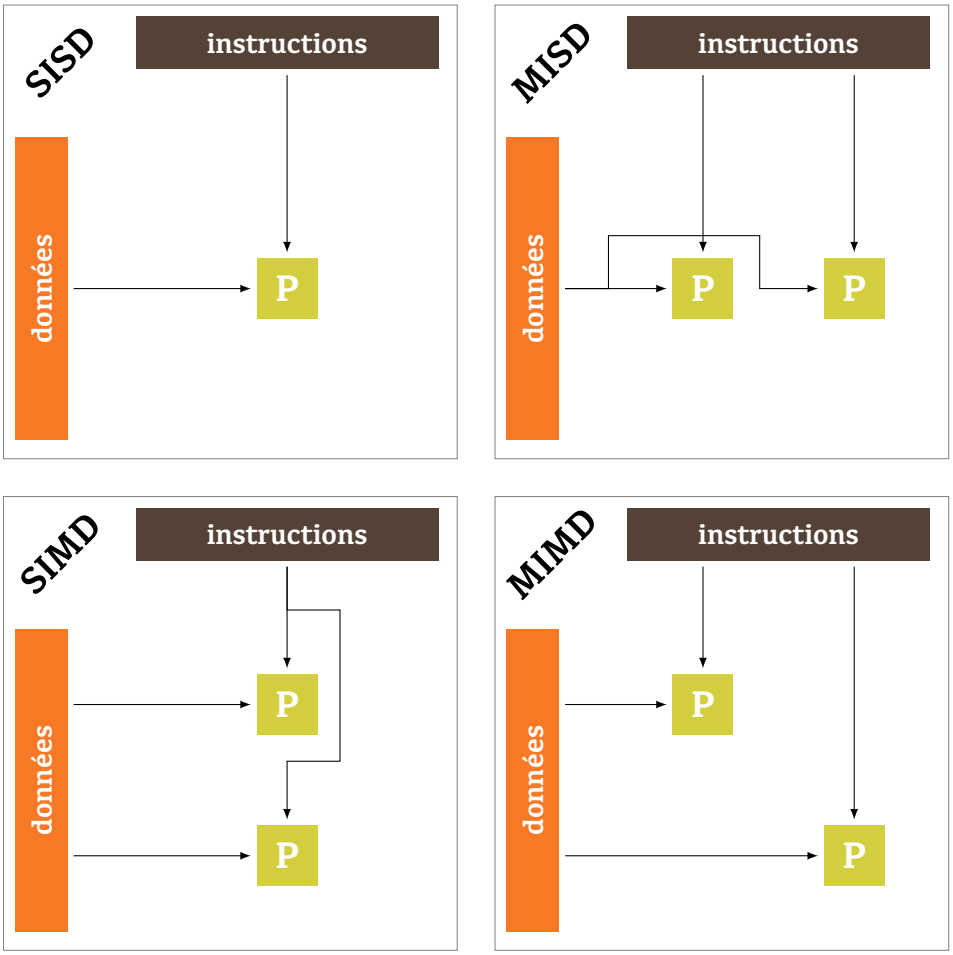


FIGURE 5.1 : Classification de l'architecture des machines proposée par Michael J. FLYNN. Les processeurs sont représentés par P.

indépendamment des autres. Dans [MORRISON 2003], Richard S. MORRISON résume cette architecture en quatre points :

— chaque processeur exécute sa propre séquence d'instructions ;
— chaque processeur travaille sur une partie différente du problème ;
— chaque processeur communique des données aux autres ;
— les processeurs peuvent avoir à attendre d'autres processeurs ou à attendre pour obtenir un accès à des données.

L'architecture d'un système distribué est de type MIMD. Nous avons en effet un problème divisé en différentes parties : les entités. Chaque entité possède sa propre séquence d'instructions et va être attribuée à un processeur. Les interactions entre les entités impliquent un échange de données entre les processeurs.

On peut voir l'architecture MIMD comme un ensemble d'architectures SISD dont il faudrait définir le couplage. E. E. Johnson propose une classification qui permet d'étendre type MIMD [Johnson 1988], et de prendre en compte ce couplage. Il utilise pour cela la manière dont la mémoire est répartie sur le système et des méthodes de communication entre les processeurs. La mémoire peut être commune à l'ensemble des machines, on dira qu'elle est *globale* ; ou elle peut être *distribuée* sur ce même ensemble. Les communications peuvent se faire directement à l'aide de variables, ou indirectement par transmission de messages. La classification qui en découle est donc :

GMSV « Global Memory, Shared Variables » ;
GMMP « Global Memory, Message Passing » ;
DMSV « Distributed Memory, Shared Variables » ;
DMMP « Distributed Memory, Message Passing ».

Le type GMSV regroupe principalement les systèmes où le couplage entre les processeurs est fort, comme dans le cas de machines multi-processeurs. Les systèmes distribués que nous manipulons ont un couplage faible entre les processeurs, la mémoire est répartie sur l'ensemble des machines, et les communications nécessitent l'envoi des messages ; c'est pourquoi les systèmes distribués, tels que nous les appréhendons, seront principalement de type DMMP.

5.1.2 Réseaux

Le réseau est un des éléments clefs du système. Grâce aux communications qu'il permet entre les machines, les entités vont avoir la possibilité d'interagir, tout en étant situées sur des machines différentes, et de migrer d'une machine à une autre.

Un réseau est caractérisé par la *latence* d'une communication qui transite par ce réseau, ainsi que par sa *bande passante*, c'est à dire la vitesse à laquelle il est possible de transférer de l'information. La latence d'une communication entre deux machines reliées à l'aide d'une fibre optique va être différente de celle nécessaire à l'établissement d'une connexion à un satellite via les ondes. La latence peut être due à une limitation technique, et dans ce cas être constante et prévisible. Elle peut aussi être liée à des conditions variables, comme par exemple dans le cas d'un réseau mobile où la connexion entre deux machines peut varier en fonction des obstacles, de la météo, *etc*... Les *réseaux tolérants aux délais* [Fall 2003 ; Casteigts et al. 2012] forment un domaine de recherche visant à gérer les problèmes liés à ce type de délais.

Chaque réseau possède une *topologie* qui établit les connections entre les machines. Cette topologie va donc nous permettre de définir comment circulent les données au sein du réseau. Elle est à la fois physique et logique, la topologie physique apportant un certain nombre de contraintes que la topologie logique doit respecter. Un réseau de type *Token Ring* par exemple, possède une topologie physique en forme d'étoile : chaque machine est connectée à un élément central appelé « Unité d'Accès au Média » (MAU, *Media Access Unit*). En revanche, la topologie logique du Token Ring forme un anneau : chaque machine peut communiquer avec deux autres, elle reçoit des informations de l'une, et peut en envoyer

à l'autre ; la communication est donc circulaire. On peut alors représenter cette topologie à l'aide d'un graphe, dont les nœuds représentent les machines, et où une arête entre deux machines symbolise la possibilité d'une communication directe.

Plusieurs réseaux peuvent être interconnectés afin de créer un réseau de plus grande envergure. L'exemple le plus connu est naturellement celui d'Internet, qui n'est autre qu'une interconnexion, à l'échelle mondiale, d'une multitude de sous-réseaux locaux.

5.1.3 Stockage

Les ressources désignent ce qui est nécessaire à l'exécution des entités, et ce qu'elles vont pouvoir exploiter. D'après ce que nous avons vu précédemment, les ressources principales sont, d'une part, les ressources de calcul, c'est à dire les processeurs, et d'autre part, la mémoire. Les processeurs vont permettre d'exécuter les entités et donc de leur donner la possibilité d'agir dans l'application distribuée, tandis que la mémoire est nécessaire pour stocker la structure représentant l'état de l'entité.

D'autres types de ressources peuvent être disponibles. L'espace de stockage en constitue une grande partie. On va distinguer :

— un stockage direct associé à une seule machine, c'est à dire que l'accès au stockage par cette machine ne nécessite pas l'utilisation du réseau ;
— un stockage centralisé accessible par l'ensemble des machines (ou limité à un groupe de machines) ;
— un stockage décentralisé.

On peut de plus distinguer un stockage sous forme de fichiers, d'un stockage dans une base de données. Un stockage centralisé implique la mise en place d'une machine dédiée (il peut s'agir éventuellement d'une simple façade virtuelle) à laquelle les machines vont pouvoir se connecter pour accéder aux ressources de stockage. Il peut s'agir par exemple de « serveurs de stockage en réseau » (N.A.S.) dont l'accès se fait à l'aide de systèmes de fichiers dédiés ; ou encore de « réseaux de stockage » (S.A.N.) qui opèrent à une couche inférieure aux fichiers, au niveau des blocs de données, permettant ainsi à une machine du système d'accéder à une zone de stockage distante de manière locale.

Un stockage distribué implique que ce stockage soit formé par l'agrégation de stockages directs associés à chaque machine du système. C'est le cas, par exemple, dans un réseau *peer-to-peer* où les fichiers sont séparés en morceaux qui sont hébergés, avec de la redondance, sur les machines. Une machine, souhaitant obtenir un fichier, doit alors obtenir une liste des machines possédant des morceaux de ce fichier, puis télécharger l'ensemble de ces morceaux afin de pouvoir reconstituer le fichier.

Le stockage distribué est un système dans le système. Son objectif est de fournir un accès concurrent à des données tout en permettant aux machines de s'abstraire de la gestion du matériel. Il doit donc être en mesure de gérer les pannes, et d'être extensible afin de s'adapter aux besoins des machines.

5.1.4 Intergiciel

L'intergiciel est la couche logicielle qui va venir s'insérer entre le système d'exploitation des machines et l'application distribuée. Il s'agit de la « glue » qui permet de maintenir ensemble les différentes parties de l'application. Il permet ainsi de gérer l'ensemble des ressources disponibles et fournir à cette application une abstraction de l'environnement. Si l'on voit le système distribué comme une machine de haut niveau, l'intergiciel en est son système d'exploitation, qui va permettre aux entités de communiquer entre elles et d'échanger de l'information. Il peut aussi être utilisé sur une unique machine pour mettre une communication entre différents processus.

Il est d'autant plus nécessaire lorsque les ressources sont hétérogènes. D'une part, concernant les machines, dont les ressources matérielles ainsi que le système d'exploitation peuvent différer. Et d'autre part, concernant la partie réseau qui peut inclure différents protocoles de communication que l'intergiciel doit être capable de gérer.

La technologie RMI (pour « Remote Method Invocation ») de Java [WOLLRATH, RIGGS et WALDO 1996] est un exemple d'intergiciel qui fournit les moyens d'invoquer un objet situé sur une machine distante. Cette technologie est décentralisée, chaque machine virtuelle Java hébergeant un serveur de registres, qui répertorie les entités hébergées et permet la communication entre les machines. Une entité (ici un objet) qui souhaite en contacter une autre doit connaître la localisation de celle-ci et son identifiant unique. À partir de ces informations, le registre local va fournir un objet représentant localement l'entité distante, mais dont les méthodes seront routées vers la dite entité.

Approche synchrone et asynchrone

Les intergiciels tels que RMI (CORBA [OMG 1991], DCOM [REDMOND 1997], *etc...*) rentrent dans la catégorie des « Remote Procedure Call » (littéralement *Appel de Procédure Distante*). Ils permettent d'abstraire l'invocation de morceaux de code distants. L'inconvénient vient du fait que cette technique est généralement *synchrone*, l'invocateur devant attendre la fin de l'exécution de la procédure invoquée. Or, si cela ne pose guère de problème lorsque la procédure est locale, il en est tout autre lorsque celle-ci est distante : les latences induites par les communications nécessaires peuvent avoir un impact fortement négatif sur les performances globales de l'application distribuée.

Une solution a ce problème est d'utiliser un type de communication *asynchrone* comme c'est le cas dans le modèle d'intergiciel *orienté message* (MOM, « Message-oriented Middleware ») [CURRY 2004]. Dans ce modèle, l'invocation, par une entité source, d'une procédure d'une entité cible prend la forme d'un message. L'entité source émet donc un message contenant les informations nécessaires à l'invocation, puis le message est transmis jusqu'à la machine hébergeant l'entité cible. Dès que cette dernière devient disponible, elle peut alors exécuter la procédure puis répondre à l'entité source. Le caractère asynchrone de cette technique vient du fait qu'une fois le message émis, l'entité source peut continuer à effectuer d'autres tâches et revenir sur le traitement en cours une fois que la réponse de la cible

est disponible. On obtient alors un état *non-bloquant* de l'entité source lorsqu'elle procède à une invocation distante. Ce modèle permet donc de gérer des réseaux possédant une latence élevée.

Approche décentralisée

Certains intergiciels ont une architecture *décentralisée*. C'est le cas d'intergiciels utilisant des protocoles réseaux *pair-à-pair* [JUNGINGER et LEE 2004], ou d'intergiciels dédiés aux grilles [LASZEWSKI et AMIN 2004] (cf. 5.2.2 pour une présentation des grilles). Une approche décentralisée permet d'augmenter la *tolérance aux fautes* du système qui n'est plus dépendant de quelques points centraux. De plus, elle permet de mieux répartir la charge du réseau du fait qu'une connexion peut être établie directement avec la machine visée, sans nécessiter le passage par un serveur central, véritable goulot d'étranglement des communications.

Il existe des intergiciels *hybrides* [SCHOLLMEIER 2001] qui combinent les avantages des deux approches, centralisée et décentralisée. Ceci en utilisant un réseau pair-à-pair pour les communications, sur lequel s'ajoutent certains services ayant une architecture client/serveur.

5.2 Différents types de systèmes distribués

Plusieurs éléments vont permettre de caractériser un système distribué. Tout d'abord, la façon dont sont réparties les ressources. Celles-ci peuvent être *homogènes*, c'est à dire que chaque machine va apporter les mêmes ressources ; ou au contraire *hétérogènes*, dans ce cas les ressources peuvent être variables en fonction des machines.

5.2.1 Grappe de calcul

Une *grappe de calcul*, encore appelée *ferme de calcul*, ou tout simplement par son nom anglais *cluster*, désigne un système distribué dans lequel l'ensemble de machines qui le composent, apparaît comme une unique machine. Le réseau, reliant les machines entre elles, est particulièrement rapide comparé à d'autres modèles de systèmes distribués. La topologie logique d'une grappe est centralisée. On peut voir sur la figure 5.2 une création « maison » d'une grappe à l'aide d'ordinateurs à bas coût, les RASPBERRY PI [1].

Ce type d'architecture est particulièrement adaptée pour des services nécessitant une haute disponibilité ou pour du calcul intensif. Une grappe est un supercalculateur qui est dédié au calcul. Chaque machine possède son propre système d'exploitation. Un répartiteur de charge est placé en amont de ces machines, et reçoit les demandes de calcul.

Par exemple, une grappe est idéale pour servir les requêtes d'un service web. Les requêtes sont envoyées au répartiteur de charge qui les distribue aux machines composant la grappe de manière à répartir au mieux la charge. On peut ainsi maintenir une qualité de service optimale même en cas de trafic important.

1. http://www.raspberrypi.org/

Figure 5.2 : Une grappe réalisée avec des Raspberry Pi.

L'ensemble des machines composant la grappe est évolutif : on peut ajouter ou retirer des machines pour adapter dynamiquement la dimension de cette grappe pour que sa taille corresponde aux besoins réels actuels.

5.2.2 Grille de calcul

Là où une grappe de calcul repose sur du matériel dédié, la grille va, quant à elle, regrouper diverses machines disponibles pour former le système distribué. Le réseau, reliant les machines entre elles, est donc potentiellement de plus grande envergure, et possède donc des performances plus faibles que le réseau dédié d'une grappe.

Le terme de *grille* est présenté par Ian Foster et Carl Kesselman, dans un livre dédié aux grilles de calcul [Foster et Kesselman 2003], comme une analogie avec le réseau électrique (« power grid » en anglais) qui permet de fournir un accès universel à une ressource :

> “ The word Grid is used by analogy with the electric power grid, which provides pervasive access to electricity and, like the computer and a small number of other advances, has had a dramatic impact on human capabilities and society. ”

Une grille de calcul offre une approche décentralisée [Foster 2002] permettant de faire collaborer des machines, qui n'étaient pas initialement prévues à cet effet, afin de participer à une tâche de plus haut niveau. Cette collaboration peut se faire à une échelle mondiale grâce à Internet. Le projet SETI@home [2], initié par le Space Science Laboratory de l'université de Californie à Berkeley, permet à des volontaires de mettre à disposition leur machine afin de former une grille de calcul permettant d'analyser des échantillons de radiotéléscopes afin de détecter d'éventuelles traces d'intelligence extra-terrestre. Au delà de cet objectif, il s'agit aussi d'une preuve de concept qu'il est possible, grâce à la collaboration de volontaires, de créer à faible coût une capacité de calcul de grande envergure.

La grille ouvre la porte à l'informatique *ubiquitaire* (« ubiquitous computing » ou encore « pervasive computing »). Un utilisateur peut participer et interagir avec une grille depuis n'importe où, pour peu qu'une connexion soit disponible, et avec n'importe quel appareil contenant l'intergiciel adéquat. Cette interaction pouvant potentiellement se faire de façon à ce que l'utilisateur n'en ait pas conscience. Le concept d'informatique ubiquitaire a été introduit par Mark Weiser [Weiser 1991 ; Weiser 1993] qui prédisait en 1991 que les équipements matériels et logiciels connectés seraient un jour tellement bien intégrés dans notre environnement qu'on n'aurait plus conscience de leur présence :

> « Specialized elements of hardware and software, connected by wires, radio waves and infrared, will be so ubiquitous that no one will notice their presence. »

5.2.3 Cloud computing

Un type de système distribué, que beaucoup côtoient sûrement sans s'en rendre compte, est le « cloud computing » , ou *informatique dans le nuage*. Si le terme est sujet à un fort engouement, sa compréhension n'en est pas pour autant des plus claires, du fait de l'abondance des définitions que l'on trouve à son propos.

La NIST [3] donne, dans [Mell et Grance 2011], la définition suivante du cloud computing :

> « Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. »

Le « nuage » est un ensemble flou de ressources, que l'on peut facilement dimensionner, et auxquelles on accède sous forme de services depuis n'importe quel point d'entrée du réseau. Ian Foster *et al.* nous donnent une définition similaire qui plonge ce nuage dans l'Internet [Foster, Zhao et al. 2008] :

> « A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet. »

2. « Search for Extra-Terrestrial Intelligence », http://setiathome.ssl.berkeley.edu/
3. National Institute of Standards and Technology

On retrouve à nouveau la notion d' *extensibilité* associée au cloud computing. Contrairement aux grilles, cette extensibilité semble principalement nécessaire pour des raisons économiques. Ce contexte économique est aussi reflété par le fait que nous ne sommes plus dans une optique de collaboration afin d'atteindre un objectif, mais plutôt dans une optique de *services* rendus à des utilisateurs. La notion de service est omniprésente dans le cloud computing. On parlera ainsi de « Software as a Service » (*Saas*), « Platform as a Service » (*Paas*) et même de « Infrastructure as a Service » (*IaaS*). Ces concepts sont représentés dans la figure 5.3. Le terminal d'un utilisateur ne possède plus, à proprement parler, d'applications installées mais d'une interface qui lui permet d'accéder à un service. Si l'on considère l'ordinateur, ou le smartphone, d'une grande partie des utilisateurs aujourd'hui, un simple navigateur web suffit à accéder à des fonctionnalités qui auraient nécessitées des applications dédiées il y a une dizaine d'années (emails, calendrier, stockage, cartes géographiques, *etc*...).

On trouvera dans [Sadashiv et Kumar 2011], une comparaison détaillée des différences entre les modèles de grappe, grille et cloud computing.

5.2.4 Écosystème computationnel

Nous avons vu jusqu'à présent, trois modèles de système distribué permettant de répartir les parties d'une application distribuée sur un ensemble de machines. Ces modèles supportent une dynamique qui est principalement liée au souci d'adapter la taille du système à la dimension du problème.

Les applications que nous considérons sont d'un type particulier. Il s'agit en effet de simulations de système complexe. Cela implique deux points importants concernant le système distribué. Tout d'abord, les interactions existant entre les entités vont tenir le rôle principal dans l'application, comme nous l'avons vu dans le chapitre 3. Ensuite, de ces interactions va émerger de l'organisation qui va venir structurer le réseau d'interactions et donc apporter de nouvelles propriétés à l'application distribuée. À ces deux points s'ajoute une forte dynamique à la fois des entités mais aussi des interactions. Les entités ont la particularité d'être autonomes.

Du fait de l'étroite ressemblance entre les simulations que l'on souhaite distribuer et les écosystèmes biologiques, J. O. Kephart, T. Hogg et B. A. Huberman ont introduit le modèle d'*écosystème computationnel* [Huberman 1988 ; Kephart, Hogg et Huberman 1989]. Ils s'appuient en particulier sur les mécanismes de compétition et de collaboration entre les ressources, ainsi que sur la capacité d'adaptation du système aux perturbations. Pour cela, le système nécessite un contrôle décentralisé, ainsi qu'un mode de communication asynchrone entre les entités.

Dans [Jacyno et Bullock 2008], Mariusz Jacyno et Seth Bullock proposent une approche thermodynamique des écosystèmes computationnels. Ils cherchent en particulier à étudier et à comprendre les principes thermodynamiques qui régissent l'auto-organisation dans les systèmes naturels afin d'appliquer ses principes à un écosystème computationnel. Celui-ci s'organise alors autour d'un flux entrant d'énergie (un flux d'information) et produit

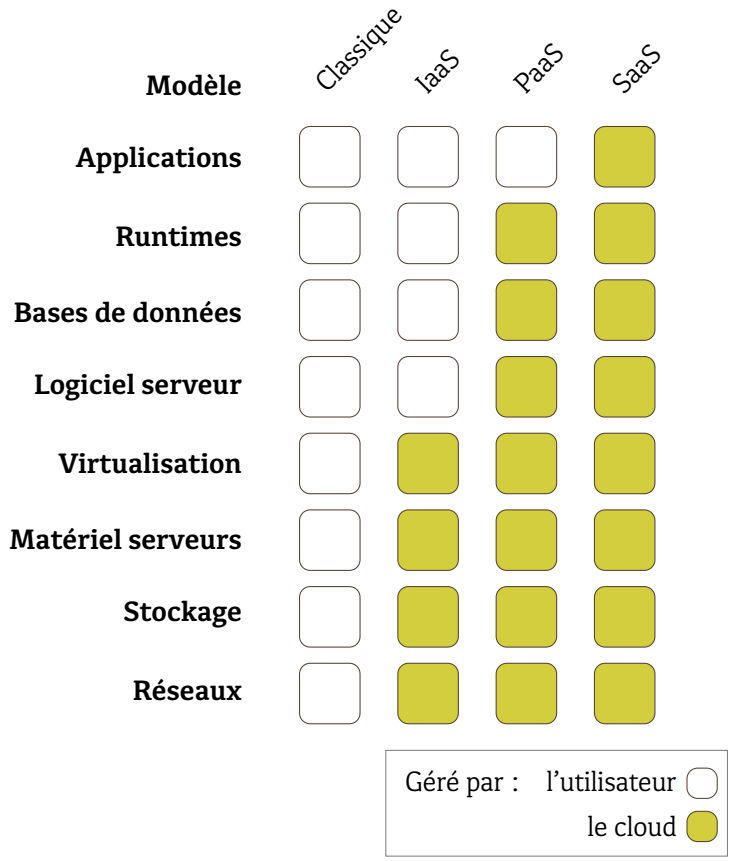


FIGURE 5.3 : Répartition, entre utilisateur et fournisseur du cloud, de la gestion des différentes couches d'un système distribué en fonction du modèle (IaaS, PaaS, SaaS).

en retour de la chaleur (le *débit* du système, c'est à dire le nombre de tâches achevées par unité de temps).

Le modèle d'écosystème computationnel se rapproche d'un modèle de grille dont les tâches seraient des entités autonomes et qui se focaliserait sur les interactions entre ces entités. Ce rapprochement est principalement dû à la volonté d'utiliser une infrastructure existante composée de l'agrégation d'appareils de tous les jours, plutôt que d'une infrastructure dédiée. Bien que ces deux modèles aient une approche décentralisée des communications, l'écosystème computationnel va plus loin dans cette démarche en décentralisant le contrôle de l'application elle-même. Contrairement aux grilles et au cloud computing, un écosystème computationnel ne s'oriente pas vers le calcul ou vers le service, mais considère les choses d'une manière plus générale, ce qui permet éventuellement une cohabitation des deux aspects.

Les propriétés que l'on souhaite voir attachées à un écosystème computationnel sont

l'adaptabilité, la robustesse et la résilience. L'aspect adaptatif signifie que le système est capable de faire preuve d'apprentissage qu'il va pouvoir mettre à profit afin de réagir efficacement aux perturbations. En partie grâce à cette adaptabilité, l'écosystème computationnel tend à être robuste, c'est à dire capable de résister aux pannes ou aux défaillances pouvant survenir dans le système. Enfin, dans le cas où des perturbations importantes ont fait que le système a quitté sa trajectoire, celui doit être en mesure de retrouver une trajectoire similaire stable, c'est à dire faire preuve de résilience. On tente d'obtenir ces propriétés à l'aide de mécanismes de redondance et d'apprentissage collectif. Il s'agit bien sûr de la vision globale d'un écosystème computationnel idéal dont nous sommes encore loin.

5.3 Algorithmique

La partie 5.1 précédente nous a permis de décrire une machine de haut niveau, composée de l'agrégation de nombreuses machines, décuplant les capacités d'une simple machine, en particulier la puissance de calcul et la mémoire. Ce système distribué ainsi formé va cependant modifier nos méthodes algorithmiques, habituées à être appliquées sur des machines monoprocesseurs.

Un algorithme centralisé considère une séquence d'instructions qui vont être appliquées sur un ensemble de données entrantes, afin de produire une ou plusieurs données sortantes. Un tel algorithme ne pourra donc pas profiter pleinement de la puissance de calcul fournie par un système distribué. Il est nécessaire pour cela de décomposer l'algorithme en un ensemble de parties, les entités, qui vont pouvoir être réparties dans le système. Ces entités vont s'exécuter indépendamment, communiquer entre elles, et finalement aboutir à un résultat. On parle alors d'*algorithmes distribués*.

Gerard Tel nous fournit, dans son livre dédié à ce sujet [Tel 2000], une description des trois points principaux qui distinguent les algorithmes distribués des algorithmes centralisés :

— absence de connaissance sur l'état global du système ;
— absence d'échelle de temps globale ;
— non-déterminisme.

Chaque entité d'un algorithme distribué ne peut utiliser qu'une quantité restreinte de connaissances, dites *locales*, qui sont celles contenues sur la machine hébergeant l'entité. Ces informations peuvent être étendues à celles que possèdent les voisins directs mais l'entité n'a pas la possibilité de connaître l'état global du système dans lequel elle évolue. Du fait de l'hétérogénéité de la vitesse des processeurs et de l'absence de contrôle centralisé, chaque entité possède sa propre échelle de temps désynchronisée des autres.

5.3.1 Classification

Les algorithmes distribués peuvent être classés d'après plusieurs critères, selon Nancy Lynch dans [Lynch 1996] :

— la méthode de communication entre processus ;
— le modèle temporel ;

— le modèle de gestion des fautes ;
— les problèmes abordés.

La classification proposée est basée sur le modèle temporel qui semble être le critère permettant de faire la meilleure distinction entre les algorithmes. On obtient ainsi trois classes d'algorithmes :

— modèle synchrone ;
— modèle asynchrone ;
— modèle hybride partiellement synchrone.

Modèle synchrone

Le modèle synchrone est peu probable dans le cas d'un système distribué car, comme nous l'avons vu, il n'y a pas de synchronisation au niveau matériel entre les processeurs du système. Cette synchronisation peut cependant être simulée au niveau logiciel, ce qui justifie la présence de ce modèle d'algorithmes.

On utilise pour cela une méthode de diffusion de messages dans un graphe. Le graphe est celui dont les nœuds représentent les processeurs et dont les arêtes sont données par la topologie logique du réseau. Chaque processeur possède un état qu'il va diffuser à ses voisins. Chaque processeur met à jour son état à l'aide d'une fonction de transition qui considère l'état courant et l'état des voisins. On peut alors définir une *étape* qui consiste pour chaque processeur à :

1. générer le message qui sera envoyé aux voisins ;
2. envoyer ce message ;
3. lire les messages des voisins ;
4. appliquer la fonction de transition.

La fonction de transition permet d'exécuter le code des entités attachées au processeur. On peut, grâce à ce mécanisme, diffuser un message d'arrêt par exemple, qui permettrait de signaler la fin de l'algorithme.

Cette technique permet de définir deux types de complexité. Tout d'abord, une complexité temporelle correspondant au nombre d'*étapes* nécessaires afin la finalisation de l'algorithme. Puis une complexité en terme de communication qui correspond au nombre total de messages non-nuls envoyés.

Modèle asynchrone

Le modèle général permettant de décrire le fonctionnement asynchrone est donné par « l'automate d'entrée/sortie » introduit dans [Lynch et Tuttle 1987]. Il s'agit d'une machine à état dont les transitions sont associées à des actions de type *entrée*, *sortie*, ou *interne*.

Un automate e/s A est défini par sa signature $S = sig(A)$. S est un triplet contenant l'ensemble des actions d'entrée $in(S)$, l'ensemble des actions de sortie $out(S)$, et enfin celui des actions internes $int(S)$. L'ensemble des actions externes est défini comme l'union $ext(S) =$

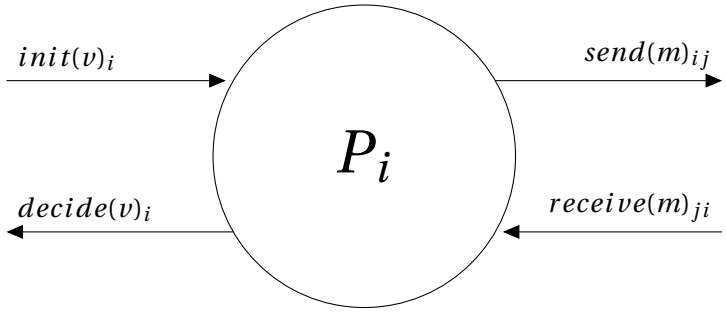


Figure 5.4 : Un automate e/s de processus

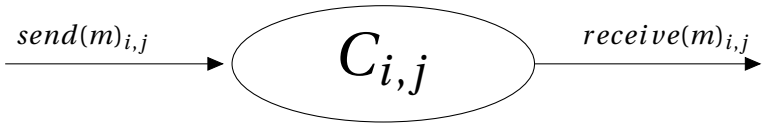


Figure 5.5 : Un automate e/s de communication

$in(S) \cup out(S)$, et celui des actions localement contrôlées $local(S) = out(S) \cup int(S)$. L'ensemble $act(S)$ désigne l'ensemble des actions. L'automate A est alors constitué des éléments suivants :

- $sig(A)$, la signature de A ;
- $states(A)$, l'ensemble des états de A ;
- $start(A) \subseteq states(A)$, l'ensemble des états initiaux ;
- $trans(A)$, un ensemble de relations de transitions ;
- $tasks(A)$, une relation d'équivalence avec $local(sig(A))$; une tâche va permettre de procéder à plusieurs actions locales en une seule opération.

Afin de modéliser le système distribué, il est nécessaire de définir deux types d'automates e/s. Tout d'abord, l'automate associé à un processus P_i, symbolisé figure 5.4. Son état initial est donné par l'action $init(v)_i$ où v représente une donnée d'entrée. Cette initialisation va déclencher le processus qui aboutira sur une action finale $decide(v)$. Pour y parvenir, le processus a la possibilité de communiquer avec d'autres automates grâce aux actions $send(m)_{ij}$ et $receive(m)_{ji}$.

Ensuite, un automate e/s de communication $C_{i,j}$ qui va permettre de définir la communication entre des processus P_i et P_j. Ce type d'automate est représenté dans la figure 5.5, il dispose d'une action d'entrée $send(m)_{ij}$ et d'une de sortie $receive(m)_{ij}$ où m représente le message à transmettre. Un automate e/s de communication permet par exemple de représenter un canal de communication en mode FIFO [4] et de définir ainsi l'ordre dans lequel les messages envoyés seront lus.

On peut alors définir un algorithme asynchrone grâce à une composition de ces automates. On distinguera un modèle asynchrone à mémoire partagée, d'un modèle à réseau

4. « First In, First Out »

asynchrone. Dans le premier, les processus peuvent communiquer instantanément à l'aide de variables partagées. Dans le second, les processus communiquent au travers d'un réseau.

Modèle partiellement synchrone

On considère dans ce modèle que, bien qu'il n'y ait pas de synchronisation matérielle entre les processeurs, chaque processeur peut accéder à une horloge commune et synchroniser son exécution par rapport à cette horloge. On peut alors reprendre le modèle précédent d'automate en modifiant les propriétés de l'ensemble des tâches : celles-ci doivent alors être bornées en temps.

5.3.2 Marche aléatoire dans un graphe

Nous avons abordé, dans ce qui précède, une manière générale et formelle permettant de modéliser un algorithme distribué. Dans le cadre de nos travaux, nous manipulons un graphe qui est réparti dans un système distribué, un écosystème computationnel, et nous souhaitons pouvoir appliquer des algorithmes sur ce graphe.

Le système que nous utilisons est décentralisé, tant au niveau des communications entre les machines, qu'au niveau du contrôle. Nous sommes donc dans le cas d'un modèle asynchrone d'algorithme, dans lequel chaque processeur va avoir la possibilité d'exécuter des actions internes et d'échanger des informations avec d'autres processeurs.

Une marche aléatoire consiste en un ensemble de particules qui vont se déplacer aléatoirement dans le graphe. On initialise cet ensemble en répartissant les particules sur l'ensemble du graphe. Puis chaque particule va choisir à chaque étape un nœud parmi les nœuds voisins de sa position. L'action interne d'un processeur consiste dans ce cas à calculer les nouvelles positions de particules. Si certaines particules se déplacent sur une partie du graphe située sur un autre processeur, il y a alors une action d'envoi de ces particules vers les processeurs correspondants. Puis une action de réception des particules ayant choisi de se déplacer vers la partie du graphe hébergée par le processeur. La marche aléatoire d'une particule a est alors la suite

$$X_0^a, X_1^a, \ldots, X_k^a$$

où X_i^a est un nœud du graphe et telle que X_{i+1}^a a été choisi dans le voisinage de X_i^a avec une probabilité $p_a(X_i^a, X_{i+1}^a)$.

On peut alors obtenir des informations sur le graphe en procédant à une analyse statistique des déplacements des particules. Par exemple, l'étude de la fréquence de passage des particules sur les arêtes peut donner des informations concernant les points de congestion du graphe.

Les marches aléatoires sont un type d'algorithme intéressant dans le cas des algorithmes distribués du fait qu'elles ne sont pas inquiétées par les contraintes que ceux-ci soulèvent. En particulier, une marche aléatoire ne nécessite pas de connaissance sur l'état global du système, chaque particule n'utilise que des informations locales pour choisir sa future destination. Elles sont, de plus, facilement adaptables à la dynamique du système.

Il est possible d'adapter une marche aléatoire à un problème spécifique en insérant un *biais* dans la probabilité $p_a(X_i^a, X_{i+1}^a)$. Nous allons présenter, dans ce qui suit, un type d'algorithme utilisant une telle marche biaisée, les algorithmes fourmis.

5.3.3 Algorithmes fourmis

Les algorithmes fourmis ont été initialement introduits par Marco Dorigo dans [Dorigo 1992], puis formalisés sous le nom de « Ant Colony Optimization » (ACO) dans [Dorigo, Di Caro et Luca M Gambardella 1999]. Depuis lors, ces algorithmes sont la source de nombreuses contributions scientifiques. On trouvera un état de l'art détaillé par Antoine Dutot et Yoann Pigné dans [Dutot et Pigné 2009]. L'idée est de reproduire le comportement de fourmis naturelles qui se déplacent dans leur environnement en étant attirées et en déposant des molécules chimiques, appelées *phéromones*. On introduit donc une colonie de fourmis artificielles dans un graphe, lesquelles , grâce à leurs déplacements, vont permettre de résoudre certains problèmes combinatoires. Le problème initial était celui du voyageur de commerce, dans lequel on recherche un cycle hamiltonien de poids minimum dans le graphe.

Les fourmis vont donc parcourir le graphe, à l'image d'une marche aléatoire. Elles vont, de plus, déposer des phéromones sur les arêtes qu'elles traversent. Ces phéromones s'évaporent à chaque itération, c'est à dire que l'on applique un facteur $\rho \in]0,1[$ à la concentration en phéromones de chaque arête. Le choix de l'arête à traverser est biaisé par la concentration en phéromones des arêtes, ainsi que, éventuellement, par le poids, ou la distance pondérant cette arête.

Une solution est obtenue en extrayant la piste de phéromones dont la concentration est la plus forte. Une piste de phéromones correspond à un chemin dans le graphe. Chaque fourmi renforce donc une piste, et donc une solution, en déposant une quantité de phéromones sur son passage. Il existe plusieurs améliorations de ce renforcement, comme l'algorithme *Ant-Q* [Luca Maria Gambardella et Dorigo 1995] ou encore *Max-Min Ant System* [Stützle et Hoos 2000].

Les algorithmes fourmis, dans leur expression la plus proche du modèle naturel, semblent donc être un outil adapté aux systèmes distribués. C'est le cas par exemple dans [Schoonderwoerd et al. 1997], les auteurs présentent *Ant-Based Control*, un algorithme dont les fourmis sont autonomes et le fonctionnement complètement décentralisé. Cet algorithme est utilisé pour répartir la charge d'un réseau de communication, les fourmis collaborant pour mettre à jour les tables servant à router les appels téléphoniques.

Modèle

Un algorithme fourmis repose sur un processus itératif dont chaque cycle permet de déplacer les fourmis dans un graphe $G = (V, E)$ et de procéder à l'évaporation des phéromones. L'ensemble des fourmis au temps t est noté $\mathcal{F}(t)$. La quantité de phéromone à l'instant t sur l'arête e est notée $\tau_e(t)$. À chaque cycle cette quantité de phéromone est mise à jour en

tenant compte de l'évaporation et du passage des fourmis :

$$\tau_e(t+1) = \rho \times \tau_e(t) + \Delta\tau_e(t) \tag{5.1}$$

La constante $\rho \in]0,1[$ définit le facteur d'évaporation des phéromones $(1-\rho)$. $\Delta\tau_e(t)$ correspond à la quantité de phéromone déposée par les fourmis ayant traversé l'arête e au temps t. Pour une fourmi k, cette quantité est notée $\Delta\tau_e^k(t)$. Cette quantité est nulle si la fourmi k n'a pas traversé l'arête ij. On obtient alors :

$$\Delta\tau_e(t) = \sum_{k \in \mathcal{F}(t)} \Delta\tau_e^k(t) \tag{5.2}$$

Nous devons ensuite définir la loi de probabilité qui va être utilisée par une fourmi pour choisir une arête à traverser. En considérant une fourmi k, $pos_k \in V$ désigne la position actuelle de k. On peut alors définir ω_k comme l'ensemble des arêtes que peut traverser la fourmi k :

$$\omega_k = \{(i,j) \in E \mid i = pos_k\}$$

On définit alors la probabilité $p_e^k(t)$, la probabilité pour une fourmi k de traverser l'arête e au temps t :

$$p_e^k(t) = \begin{cases} \dfrac{[\tau_e(t)]^\alpha \cdot [\eta_e]^\beta}{\sum\limits_{e' \in \omega_k} [\tau_{e'}(t)]^\alpha \cdot [\eta_{e'}]^\beta} & \text{si } e \in \omega_k \\ 0 & \text{sinon} \end{cases} \tag{5.3}$$

Le paramètre η_e permet d'utiliser une propriété des arêtes pour ajuster leur importance dans la loi de probabilité. Il peut s'agir par exemple de la longueur, comme défini dans *Ant System*, on a alors $\eta_e = 1/d_e$ où d_e est la longueur de e, ce qui augmente la probabilité de choisir les arêtes dont la longueur est faible.

Le déroulement du cycle d'un algorithme fourmis au temps t est donné par l'algorithme 4.

Le nombre initial de fourmis peut être fixé en fonction du nombre de nœuds. On définit alors une quantité κ de fourmi par nœud, et $N(v)$ le nombre de fourmis présentes sur le nœud v. Lorsqu'un nœud est ajouté dans le graphe, on ajoute κ fourmi sur ce nœud. Lorsqu'un nœud v est supprimé, nous avons plusieurs cas de figure :

- $N(v) < \kappa$, on supprime κ fourmis dans le graphe en commençant par celles du nœud supprimé ;
- $N(v) = \kappa$, on supprime juste les fourmis du nœud ;
- $N(v) > \kappa$, on supprime κ fourmis du nœud puis on répartit les $N(v) - \kappa$ restante aléatoirement dans le graphe.

En respectant ce mécanisme, nous obtenons une démographie stable de la population des fourmis, qui reste constamment proportionnelle d'un facteur κ au nombre de nœuds.

Algorithme 4 : Cycle d'un algorithme fourmis

Données :
- $\mathcal{F}$, l'ensemble des fourmis ;
- $\mathcal{E}$, l'ensemble des arêtes du graphe ;
- t, la date courante.

début
  pour chaque $k \in \mathcal{F}$ **faire**
    pour chaque $e \in \omega_k$ **faire**
      calculer $p_e^k(t)$;
    fin
    /* La méthode choisir() permet de sélectionner aléatoirement un élément de l'ensemble donné en premier paramètre en fonction de la distribution de probabilités donnée en second paramètre */
    $e \leftarrow$ `choisir`$(\omega_k, \left\{p_{e_0}^k(t), \ldots, p_{e_i}^k(t), \ldots\right\})$;
    traverser e ;
    $\Delta\tau_e \leftarrow \Delta\tau_e + \Delta\tau_e^k$;
  fin
  pour chaque $e \in \mathcal{E}$ **faire**
    $\tau_e \leftarrow \tau_e \times \rho + \Delta\tau_e$;
  fin
fin

Références

Casteigts, A. et al. (2012). "Time-Varying Graphs and Dynamic Networks". In : *International Journal of Parallel, Emergent and Distributed Systems*. In press. DOI : 10.1080/17445760.2012.668546.

Curry, Edward (2004). "Message-oriented middleware". In : *Middleware for communications*, p. 1–28.

Dorigo, Marco (1992). "Optimization, learning and natural algorithms". In : *Ph. D. Thesis, Politecnico di Milano, Italy*.

Dorigo, Marco, Gianni Di Caro et Luca M Gambardella (1999). "Ant algorithms for discrete optimization". In : *Artificial life* 5 (2), p. 137–172.

Dutot, Antoine et Yoann Pigné (2009). "Tour d'horizon des problèmes combinatoires traités par les fourmis artificielles". In : *Fourmis artificielles 1, Des bases de l'optimisation aux applications industrielles*, p. 71–100.

Fall, Kevin (2003). "A delay-tolerant network architecture for challenged internets". In : Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications. ACM, p. 27–34. ISBN : 1581137354.

Flynn, Michael J (1966). "Very high-speed computing systems". In : *Proceedings of the IEEE* 54 (12), p. 1901–1909. ISSN : 0018-9219.

Foster, Ian (2002). "What is the grid ?-a three point checklist". In : *GRIDtoday* 1 (6).

Foster, Ian et Carl Kesselman (2003). *The Grid 2 : Blueprint for a new computing infrastructure*. Access Online via Elsevier. ISBN : 0080521533.

Foster, Ian, Yong Zhao et al. (2008). "Cloud computing and grid computing 360-degree compared". In : Grid Computing Environments Workshop, 2008. GCE'08. Ieee, p. 1–10. ISBN : 1424428602.

Gambardella, Luca Maria et Marco Dorigo (1995). "Ant-Q : A reinforcement learning approach to the traveling salesman problem". In : ICML, p. 252–260.

Huberman, Bernado A (1988). *The ecology of computation*. Elsevier Science Inc.

Jacyno, Mariusz et Seth Bullock (2008). "Energy, entropy and work in computational ecosystems : A thermodynamic account". In :

Johnson, Eric E (1988). "Completing an MIMD multiprocessor taxonomy". In : *ACM SIGARCH Computer Architecture News* 16 (3), p. 44–47. ISSN : 0163-5964.

Junginger, Markus Oliver et Yugyung Lee (2004). "Peer-to-peer middleware". In : *Middleware for Communications*, p. 81–107. ISSN : 0470862084.

Kephart, Jeffrey O, Tad Hogg et Bernado A Huberman (1989). "Dynamics of computational ecosystems". In : *Physical Review A*.

Laszewski, Gregor von et Kaizar Amin (2004). "Grid middleware". In : *Middleware for Communications*, p. 109.

Lynch, Nancy A (1996). *Distributed algorithms*. Morgan Kaufmann. ISBN : 0080504701.

Lynch, Nancy A et Mark R Tuttle (1987). "Hierarchical correctness proofs for distributed algorithms". In : Proceedings of the sixth annual ACM Symposium on Principles of distributed computing. ACM, p. 137–151. ISBN : 089791239X.

Mell, Peter et Timothy Grance (2011). "The NIST definition of cloud computing (draft)". In : *NIST special publication* 800 (145), p. 7.

Morrison, Richard S. (2003). *Cluster Computing : Architectures, Operating Systems, Parallel Processing & Programming Languages*.

OMG, Committee (1991). *The Common Object Request Broker : Architecture and Specification*. OMG Document Revision 1.

Redmond, Frank E (1997). *Dcom : Microsoft Distributed Component Object Model with Cdrom*. IDG Books Worldwide, Inc. ISBN : 0764580442.

Sadashiv, Naidila et SM Dilip Kumar (2011). "Cluster, grid and cloud computing : A detailed comparison". In : Computer Science & Education (ICCSE), 2011 6th International Conference on. IEEE, p. 477–482. ISBN : 1424497175.

Schollmeier, Rüdiger (2001). "A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications". In : Peer-to-Peer Computing, 2001. Proceedings. First International Conference on. IEEE, p. 101–102. ISBN : 0769515037.

Schoonderwoerd, Ruud et al. (1997). "Ant-based load balancing in telecommunications networks". In : *Adaptive behavior* 5 (2), p. 169–207. ISSN : 1059-7123.

Stützle, Thomas et Holger H Hoos (2000). “MAX–MIN ant system”. In : *Future generation computer systems* 16 (8), p. 889–914. issn : 0167-739X.

Tel, Gerard (2000). *Introduction to distributed algorithms*. Cambridge university press. isbn : 0521794838.

Von Neumann, John (1993). “First Draft of a Report on the EDVAC”. In : *Annals of the History of Computing, IEEE* 15 (4), p. 27–75. issn : 1058-6180.

Weiser, Mark (1991). “The computer for the 21st century”. In : *Scientific american* 265 (3), p. 94–104. issn : 0036-8733.

— (1993). “Some computer science issues in ubiquitous computing”. In : *Communications of the ACM* 36 (7), p. 75–84. issn : 0001-0782.

Wollrath, Ann, Roger Riggs et Jim Waldo (1996). “A Distributed Object Model for the Java TM System”. In : *Computing Systems* 9, p. 265–290. issn : 0895-6340.

Troisième partie

Modèle

CHAPITRE 6

MODÉLISATION DES INTERACTIONS

Tout ce que nous avons vu jusqu'à présent tourne autour de la notion d'interaction. Au point que ce sont ces interactions qui semblent jouer un rôle majeur dans le réseau. Les entités en sont réduites à jouer le rôle secondaire d'attributs, nécessaire certes, permettant de définir les acteurs des interactions. Ces interactions forment la syntaxe qui décrit le réseau, lui donnant ainsi une forme et sont à l'origine de l'émergence des organisations qui vont participer à structurer le graphe.

Derrière cette syntaxe se cache la sémantique de la simulation qui ne nous est pas accessible. Nous devons donc nous contenter d'étudier cette syntaxe ainsi que la sémiologie de la simulation qui s'exprime au travers de phénomènes qui apparaissent dans le graphe comme par exemple sa structuration. C'est grâce à l'analyse de ces éléments que nous pourrons réussir à améliorer notre gouvernance du système. Pour cela nous avons besoin de comprendre les mécanismes qui entourent les différentes interactions, ainsi qu'une meilleure appréhension des interactions elles-mêmes et de leurs propriétés, cela afin d'en fournir une meilleure

modélisation.

6.1 Qu'est ce que l'interaction ?

L'encyclopédie communautaire, WIKIPÉDIA [1], commence par définir une interaction de la sorte :

> « Une interaction, dans le langage courant, est l'action ou l'influence réciproque qui peut s'établir entre deux objets ou plus. Une interaction est toujours suivie d'un ou plusieurs effets. En physique, en chimie, ou en biologie, une interaction a pour effet de produire une modification de l'état des objets en interaction. »

Une interaction définit donc un lien de causalité qui se crée entre au moins deux entités. Une entité, de par son comportement, va alors déclencher une action qui va produire un effet sur d'autres entités, pouvant introduire une modification de leur état.

L'interaction peut aussi exister à différents niveau de représentation du système ainsi que entre ces niveaux. Comme nous le verrons dans ce qui suit en 6.1.2, il existe en effet une interaction ascendante et descendante entre les niveaux microscopique et macroscopique.

6.1.1 Interactions & Émergence

Les interactions sont responsables de l'apparition de propriétés, propres à des groupes ou au système lui-même, que l'on ne peut expliquer par la seule somme de celles des membres du groupe. On qualifie alors ces propriétés d'émergentes et on parlera plus généralement de *l'émergence* qui existe au sein du système. Ce phénomène d'émergence est classé de différentes manière dans la littérature, Damien OLIVIER nous en donne une présentation dans [OLIVIER 2006].

Le dictionnaire de Philosophie de Cambridge, [AUDI 1995], oppose l'émergence *descriptive* à l'émergence *explicative*. La première implique la présence de propriétés globales qui ne peuvent pas être définies à partir de la somme des propriétés des parties. La seconde, l'émergence dite explicative, énonce l'émergence à l'aide des lois qui régissent le système : les lois des situations les plus complexes dans le système ne sont déductibles d'aucune façon par composition des lois des situations les plus simples. Nous considérerons pour notre part, l'émergence *forte* et l'émergence *faible*.

L'émergence forte [CHALMERS 1997 ; CHALMERS 2006] se base sur l'apparition d'organisations, en tant que structure composée d'entités, sur une échelle macroscopique. Ces organisations possèdent leurs propres propriétés qui ne sont pas à *déductibles* des propriétés des entités de l'échelle microscopique qui composent l'organisation.

La définition de l'émergence faible nous est donnée dans [BEDAU 1999] : on considère que l'émergence est faible lorsqu'il est possible d'obtenir l'organisation du niveau macroscopique à partir d'une simulation utilisant le niveau microscopique ainsi que des conditions

1. consultée le 14 novembre 2013, https://fr.wikipedia.org/wiki/Interaction

externes au système considéré. David J. CHALMERS, quant à lui, exprime dans [CHALMERS 2006] le fait que l'émergence faible implique que le niveau macroscopique peut être obtenu à partir du niveau microscopique mais que certaines propriétés apparaissent qui sont *inattendues* compte tenu les principes qui régissent le niveau microscopique.

6.1.2 Causalité ascendante et descendante

L'interaction, désignée de manière générale, permet donc l'émergence de nouvelles propriétés du système. Les interactions sont tout d'abord l'action d'une entité sur une autre, voire sur plusieurs autres. Il s'agit d'un lien de causalité qui se crée entre une cause et un effet. Puis, ces interactions vont permettre de structurer le graphe et de ce fait, aboutir à la création d'organisation. Ces liens ne peuvent cependant pas expliquer à eux seuls l'émergence survenant dans le système.

L'émergence nécessite d'une part la présence de boucles de rétroaction, qui impliquent que l'effet engendré par une cause va en retour agir sur cette même cause, entrainant ainsi la création d'une boucle de causalité. De plus, les interactions existent aussi entre une organisation et les entités. D'une part de manière ascendante, les entités vont causer des effets sur leur organisation qui vont modifier les propriétés de celle-ci. D'autre part, ces interactions existent aussi de manière descendante, c'est à dire que le comportement du groupe va causer des effets sur ces individus.

On peut, pour illustrer cette notion, considérer l'exemple d'un groupe d'individus qui s'organise autour d'une certaine idéologie. Les évolutions des idées de chacun vont permettre de faire évoluer l'idéologie globale du groupe, il s'agit alors d'une causalité ascendante. Mais cette idéologie globale va aussi influencer les idées de chaque membre du groupe, et dans ce cas on parle d'une causalité descendante.

6.2 Propriétés des interactions

Nous avons tenté de définir, dans ce qui précède, ce qu'est l'interaction, d'une façon générale. Une interaction a besoin de s'appuyer sur un vecteur afin de lui permettre d'atteindre sa cible. Il peut s'agir de la voix, d'un message, mais aussi de molécules, ou d'éléments abstraits comme la proximité entre les entités. Grâce à ce vecteur et à la façon dont il permet de propager l'interaction, nous pouvons proposer certaines propriétés des interactions.

6.2.1 Interaction directe ou indirecte

Lorsqu'une entité a connaissance d'une autre et agit de manière à provoquer des modifications immédiates sur cette autre entité, on pourra dire de cette interaction qu'elle est *directe*. Lorsqu'une personne parle à une autre, par exemple, il s'agit d'une interaction directe, il n'y a pas d'intermédiaire entre ces personnes.

Au contraire, l'interaction peut nécessiter l'intervention d'un intermédiaire (voire de plusieurs). On qualifie, dans ce cas, cette interaction d'*indirecte*. De telles interactions néces-

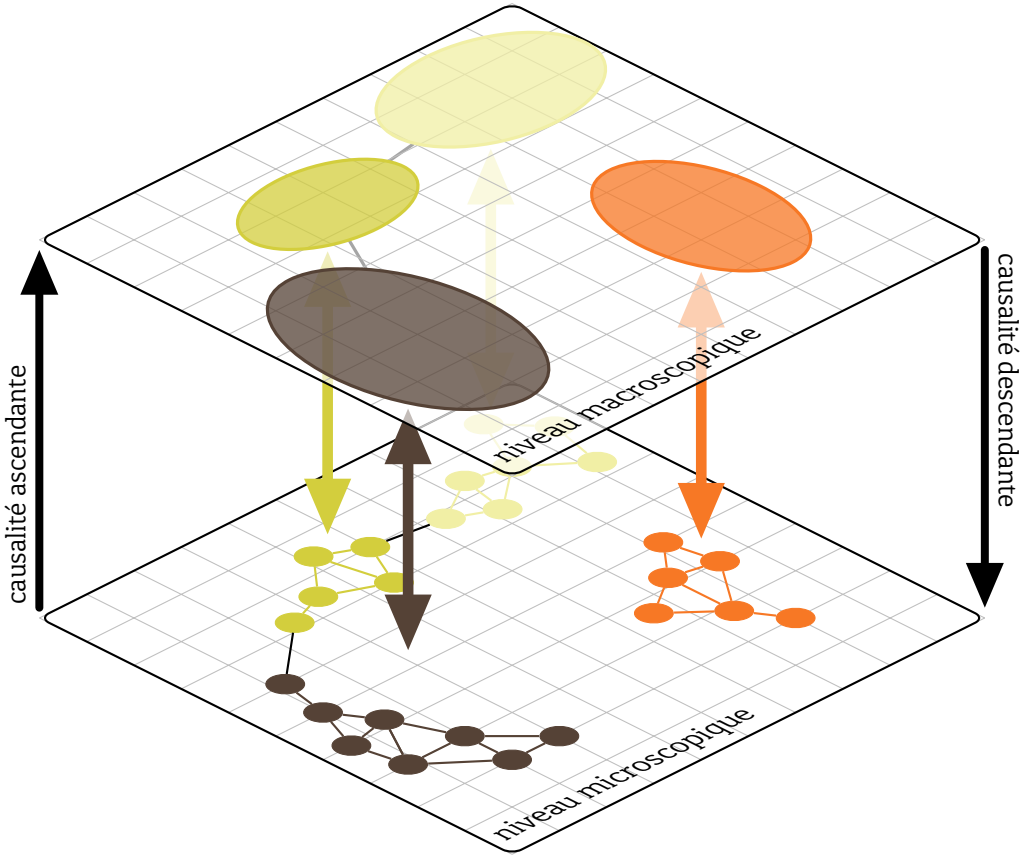


Figure 6.1 : Causalité ascendante et descendante.

sitent la création d'une information qui va être transmise indirectement à une entité, ou un groupe d'entités. L'environnement peut servir de vecteur pour la transmission de cette information comme c'est le cas avec le mécanisme de stigmergie dont nous parlions en 1.1. Dans ce dernier cas, l'interaction indirecte se fait entre une entité et un groupe d'entité. Pour illustrer cette idée, nous pouvons considérer, par exemple, une personne qui collerait une affiche dans un lieu de passage : l'information contenu sur l'affiche est alors diffusée de manière indirecte aux personnes qui prendront le temps de lire cette affiche. Mais une interaction indirecte peut aussi être ciblée entre deux entités. Pour reprendre l'exemple de la communication entre deux personnes, l'information que l'on transmettait directement à l'oral peut être transmise de manière indirecte au travers d'une lettre ou d'un courriel.

6.2.2 Interaction synchrone ou asynchrone

Il est possible de caractériser les interactions en fonction de l'état qu'elles provoquent sur les entités qui en sont à l'origine. En considérant deux entités *source* et *cible*. *source* est

à l'origine d'une interaction qui va déclencher chez *cible* une réaction qui aboutira sur une réponse à *source*. Cette réponse permet soit de retourner une information, soit de simplement valider le fait que la réaction a eu lieu. *source* a besoin de cette réponse pour effectuer un certain traitement.

Dans le cas où *source* rentre dans un état bloquant en l'attente de la réponse de *cible*, on parlera alors d'interaction *synchrone*. État bloquant implique que l'entité n'effectue plus aucune action tant que son état ne s'est pas débloqué.

L'entité *source* peut aussi décider que certaines actions peuvent être traitées sans attendre la réponse de *cible*. Elle traite alors tout ou partie de ces actions et lorsque la réponse de *cible* est disponible, elle effectue le traitement correspondant. On parle dans ce cas d'interaction *asynchrone*. Ces deux types d'interactions sont illustrés dans la figure 6.2.

Il est important de noter que même si les notions de synchrone/asynchrone peuvent sembler liées aux notions de directe/indirecte que nous avons vu précédemment, il n'en est rien. Les notions directe/indirecte permettent de définir le moyen qui permet l'interaction tandis que les notions de synchrone/asynchrone définissent une propriété concernant les entités lorsqu'elles interagissent. Ainsi il est possible qu'une interaction indirecte soit synchrone et qu'une interaction directe soit asynchrone.

6.2.3 Interaction ciblée ou diffuse

L'interaction initiée par une entité peut être dirigée vers une autre entité en particulier. C'est le cas lorsqu'une personne parle à une autre, ou qu'une machine démarre une communication avec une machine spécifique. On peut dire alors que cette interaction est *ciblée*.

Cependant, l'interaction peut prendre une autre forme. Une entité peut en effet initier une interaction sans avoir la connaissance de qui sera impliquée dans cette interaction. Lorsqu'une fourmi dépose de la phéromone dans l'environnement par exemple, cette fourmi ne cible personne en particulier mais diffuse un message qui va potentiellement lui permettre d'interagir avec un grand nombre d'individu. Si une personne au milieu d'une foule crie un message pour prévenir d'un danger, elle ne cible personne en particulier mais atteint tout le monde autour d'elle. On qualifie de telles interactions de *diffuse*.

6.3 Représentation et manipulation des interactions

Dans l'ensemble de ces travaux, les interactions sont modélisées à l'aide d'un graphe dynamique composé d'un ensemble de nœuds représentant les entités intervenant dans les interactions, ainsi que d'un ensemble d'arêtes représentant les interactions entre les entités.

La difficulté de la représentation réside dans la manière de créer et supprimer les arêtes modélisant les interactions. Celle-ci pourrait paraître triviale, en particulier dans le cas d'interactions directes : l'arête est créée en début de communication, puis supprimée lorsque cette communication prend fin. Cependant, dans le cas de communications de très courte durée ou dans le cas d'interactions indirectes dans lesquelles deux courtes communications

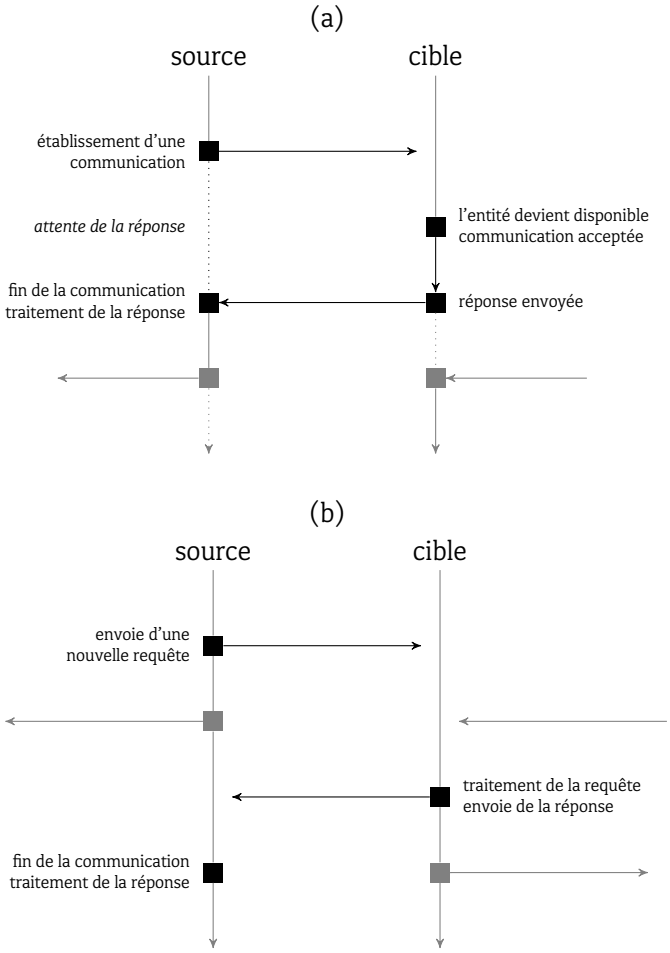


Figure 6.2 : Interaction (a) synchrone et (b) asynchrone entre une entité source et une entité cible.

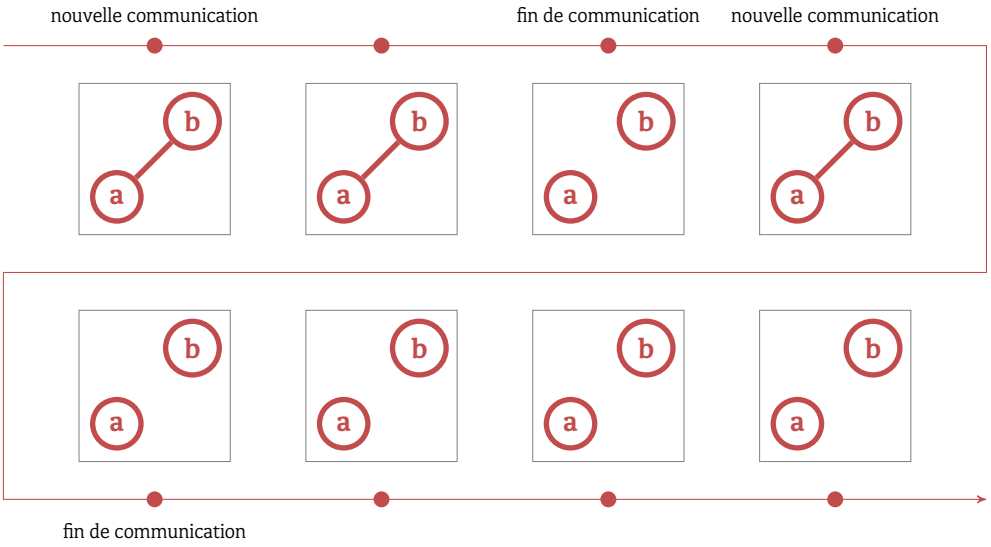


FIGURE 6.3 : Création d'interactions naïve

(envoi du message, réception de la réponse) interviennent, ce mode de représentation aboutit à l'oscillation de la présence de l'arête correspondante dans le graphe. Or nous avons vu que l'interaction joue un rôle majeur dans notre modèle, ce qui nous amène à deux constats :

1. pour un observateur humain, la visualisation du modèle peut être faussée par un temps d'apparition trop court, donnant l'impression que deux entités interagissent peu ou pas ;
2. d'un point de vue algorithmique, l'oscillation de la présence des arêtes peut entraîner l'oscillation de la solution et fausser la stabilisation de l'algorithme (dans le cas d'un algorithme dynamique itératif).

Pour pallier ces problèmes, nous introduisons une persistance de l'interaction dans le graphe. À chaque arête e est associée une valeur que nous noterons $\rho(e)$. Lorsqu'une nouvelle interaction entre deux entités commence, deux cas de figure peuvent se présenter :

— l'arête e correspondante n'existe pas ; elle est alors créée avec $\rho(e) = \rho_{init}$;
— l'arête e existe ; la valeur de $\rho(e)$ est alors incrémentée d'une valeur δ_ρ.

On considère un processus itératif qui permet, à espacement temporel régulier, d'appliquer un facteur ϖ aux valeurs $\rho(e)$ de chaque arête e. La valeur de ϖ est incluse dans l'intervalle $]0;1[$. Lorsque la valeur ρ d'une arête e passe en dessous d'un seuil ρ_{del}, l'arête e est alors supprimée.

Ce mécanisme permet de supprimer les oscillations des arêtes dans le graphe tout en conservant un modèle cohérent. Les figures 6.3 et 6.4 illustrent les différences qu'apportent la persistance dans la présence d'une arête entre deux nœuds.

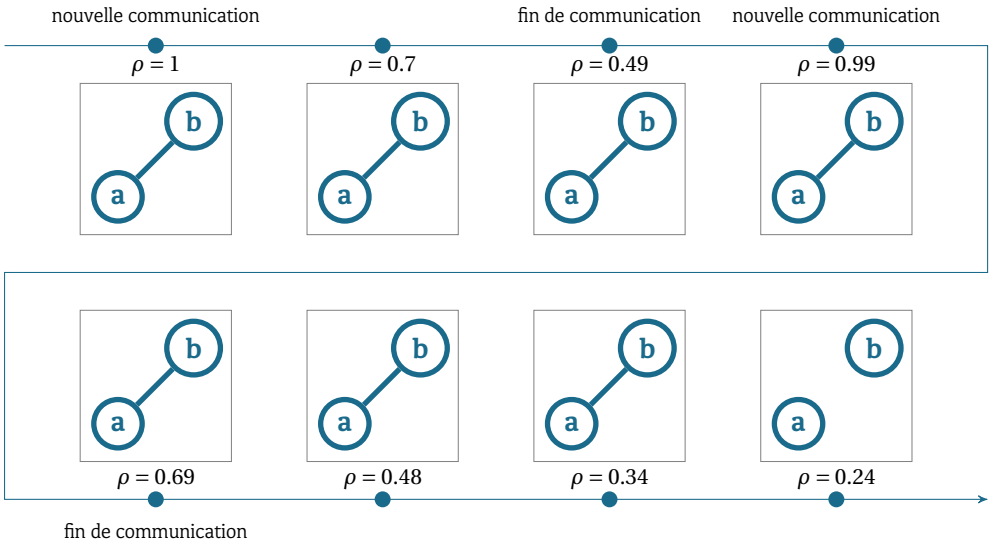


FIGURE 6.4 : Création d'interactions avancée avec $\rho_{init} = 1$, $\delta_\rho = 0.5$, $\varpi = 0.7$ et $\rho_{del} = 0.3$.

Références

AUDI, Robert (1995). *The Cambridge Dictionary of Philosophy*. en. Cambridge University Press. 2390 p. ISBN : 9781107268616.

BEDAU, M. A. (1999). "Weak Emergence". In : TOMBERLIN, James E. *Philosophical Perspectives, Mind, Causation And World*. T. 11, p. 375–399. ISBN : 9780631207931.

CHALMERS, David J. (1997). *The Conscious Mind : In Search of a Fundamental Theory*. ISBN : 0195117891.

— (2006). "Strong and weak emergence". In : *The reemergence of emergence*, p. 244–256.

OLIVIER, Damien (2006). *Modélisation informatique de systèmes à base d'interactions et détection d'organisations. Modèles du vivant*. Habilitation à Diriger des Recherches de l'Université du Havre.

CHAPITRE 7

MODÉLISATION DES ORGANISATIONS

Nous avons vu dans le chapitre 6 que l'interaction était une pièce maîtresse dans l'étude de la dynamique des graphes, et qu'elle permet l'émergence de structures dans le graphe que nous nommons *organisations*. Ce chapitre va s'intéresser à ces organisations, leur modélisation, et leur manipulation.

7.1 Qu'est ce qu'une organisation ?

Les interactions au sein d'un réseau d'entités permettent de définir des structures. Il peut s'agir d'une densité des interactions plus importante à l'intérieur de la structure qu'avec les entités de l'extérieur, ou de propriétés spécifiques permettant de donner un sens à la structure. Du fait de la dynamique du réseau d'interactions, les éléments de la structure vont évoluer au fil du temps, mais la sémantique va rester la même. On nomme une telle struc-

ture *organisation*. Lorsque l'apparition d'une organisation n'est pas due à un comportement directement programmé du système mais qu'au contraire cette organisation émerge du comportement de bas niveau des entités, on parle alors d'*auto-organisation*.

Par exemple, un algorithme qui calcule un plus court chemin entre deux nœuds d'un graphe et qui adapte la structure calculée aux modifications survenant dans le graphe. Le plus court chemin est alors une organisation dont les éléments forment un chemin, de telle sorte que ce chemin soit le plus court possible. Cette organisation est le fruit du calcul d'un algorithme et n'est donc pas une auto-organisation. En revanche, si l'on considère le graphe de la figure 3.1c qui présente un ensemble de livres sur l'informatique disponible sur Amazon, qui interagissent lorsqu'ils sont achetés ensemble, on voit ici se dégager des auto-organisations, qui correspondent à différents domaines informatiques (programmation Java, Linux, base de données, *etc*...).

7.2 Du micro au macro : le changement d'échelle

La détection des organisations dans un graphe nous permet de définir pour chaque nœud, l'organisation dont il fait partie. Ainsi pour un graphe $G = (V, E)$, on obtient un ensemble $V^M = \{C_i\}$ tel que :

$$\begin{gathered} \forall C_i \in V^M, \quad C_i \subset V \\ \bigcup_i C_i = V \\ \forall C_i, C_j, C_i \neq C_j, \quad C_i \cap C_j = \emptyset \end{gathered}$$

On peut alors définir l'ensemble E^M des arêtes existant entre les organisations :

$$E^M = \left\{(C_i, C_j) \mid \exists u \in C_i, v \in C_j, \exists\, e = (u, v) \in E\right\}$$

Une arête existe entre deux organisations C_i et C_j s'il existe au moins une arête dans le graphe G entre un nœud contenu dans C_i et un contenu dans C_j. Le graphe $G^M = (V^M, E^M)$ ainsi obtenu est appelé une *vue macroscopique* du graphe G. S'il est possible de construire une ou plusieurs vue macroscopique d'un graphe, on peut alors parler d'un graphe *multi-échelle*.

Nous ne tenons pas compte, dans ces travaux, du chevauchement possible des organisations. C'est à dire qu'il est possible que la propriété $\forall C_i, C_j, C_i \neq C_j,\ C_i \cap C_j = \emptyset$ ne soit pas respectée si un nœud s'avère appartenir à plusieurs organisations. Si l'on considère par exemple, un réseau d'interactions formés par des personnes, celles-ci peuvent être connectées ensemble de différentes façons : membres d'une même famille, du même école, de tel ou tel réseau social, *etc*... G. Palla *et al.*, dans [Palla et al. 2005], ont été les premiers à prendre en compte ce problème dans la détection statique d'organisations. Pour approfondir le domaine du chevauchement d'organisations, on peut se référer à un état de l'art sur le sujet dans [Xie, Kelley et Szymanski 2011].

On parle de changement d'échelle lorsque l'on passe d'un graphe G, que l'on nomme la vue *microscopique*, à une vue macroscopique G^M de G. Nous pouvons donc appliquer à cette vue macroscopique les méthodes d'analyses qu'il est possible d'appliquer à un graphe.

7.2.1 Granularité de la vue sur le modèle

Nous avons donc un graphe initial qui forme la vue microscopique, c'est à dire que ces composants sont atomiques. À partir de cette vue microscopique, nous pouvons construire une vue macroscopique, mais cette nouvelle vue peut aussi servir de base au calcul d'une autre vue macroscopique de plus haut niveau. Le changement d'échelle peut donc être récursif.

On parle alors de la *granularité* de la vue que l'on a sur le modèle. Il s'agit d'une information qui nous permet de définir à quel niveau de complexité nous observons un modèle. Lorsque cette granularité est *microscopique*, ces composantes sont atomiques, c'est à dire qu'elles sont les plus petites possibles, et correspondent donc aux entités initiales du réseau d'interactions. Plus la granularité augmente, plus les nœuds du graphe représentent une agrégation complexe de ces entités (organisations, organisations d'organisations, *etc...*).

Dans la figure 7.1, on peut observer un graphe initial, de couleur foncée ■, qui représente la vue microscopique du modèle. On peut procéder à un premier regroupement des nœuds pour former une vue macroscopique, représentée en couleur vert foncé ■. Il est ensuite possible de ce servir de cette vue macroscopique comme base afin de créer une nouvelle vue de plus haut niveau, représentée en clair ■. On obtient ainsi trois vues de granularité différentes sur le modèle.

7.2.2 Intérêt du changement d'échelle

L'intérêt du changement d'échelle est tout d'abord analytique. L'organisation n'apparait qu'à partir d'une certaine granularité, c'est pourquoi rester à l'échelle microscopique n'est pas pertinent lorsque l'on souhaite observer le fonctionnement émergent du système. Cette échelle est en revanche nécessaire pour comprendre les mécanismes intrinsèques permettant d'expliquer le comportement global. Si l'on prend l'exemple du corps humain, observer les atomes qui le composent ne nous permet pas de comprendre les mécanismes complexes qui le régissent comme la motricité, les différents sens, *etc...* Nous avons déjà besoin d'augmenter la complexité de notre échelle de représentation (molécules, organes, *etc...*), c'est à dire de passer à des échelles macroscopiques, afin de comprendre le fonctionnement du corps humain. Cette analyse est simplifiée du fait qu'une vue macroscopique soit un graphe, ce qui nous permet d'exploiter les outils d'analyse ou les algorithmes prévus pour être exploités sur des graphes.

Le changement d'échelle a aussi un intérêt en terme de visualisation d'informations. Pour un observateur humain, la visualisation du modèle à une échelle macroscopique permet de clarifier une vue qui peut s'avérer complexe du fait du nombre important d'entités composant les simulations. Cette clarification passe par la réduction ou le regroupement d'informations visible par l'observateur, que ce soit en terme d'éléments du graphes (nœuds, arêtes) que de données sur ces éléments. Dans [Auber et al. 2003], David Auber *et al.* s'intéresse tout particulièrement à la visualisation de graphes petit-mondes dans lesquels plusieurs échelles sont présentes. Ils proposent une métrique permettant d'identifier les arêtes inter-organisations,

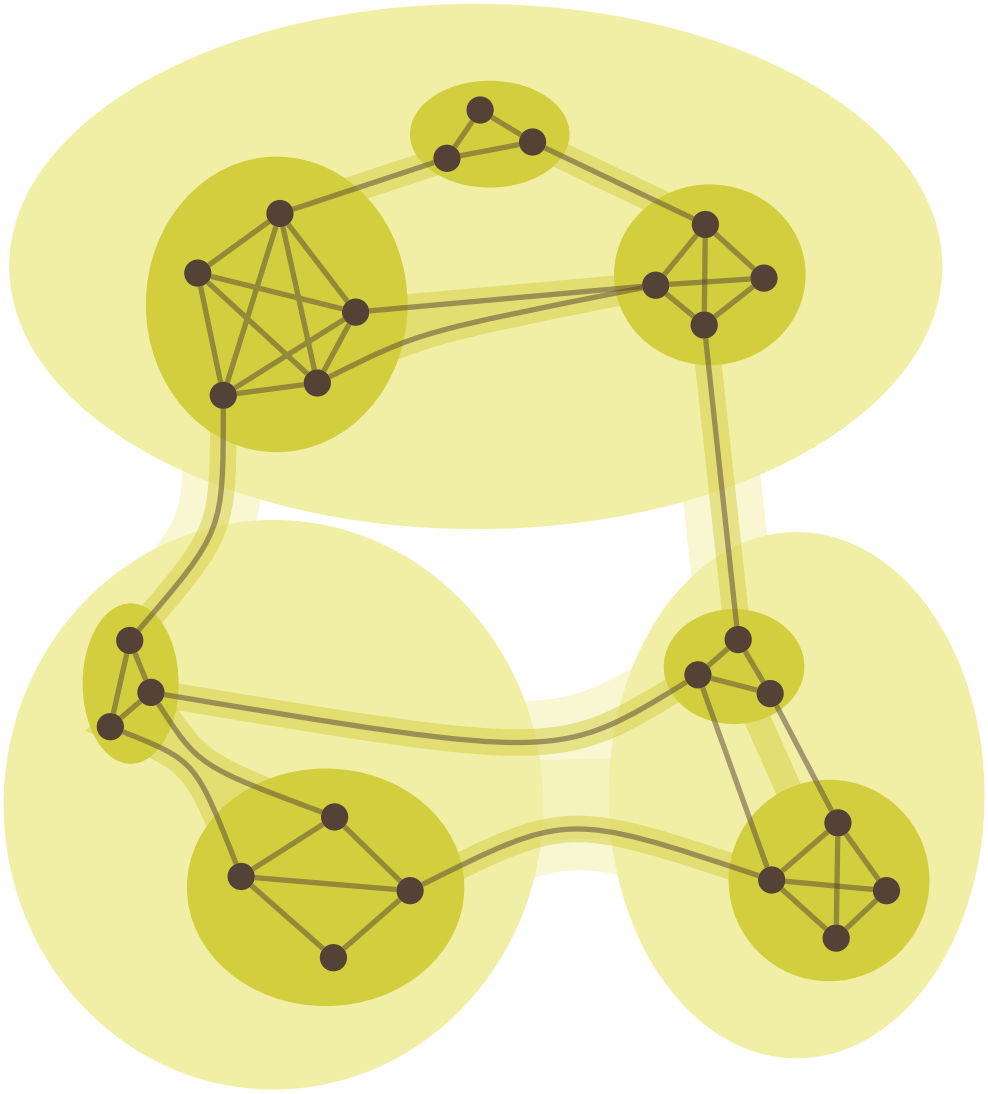


Figure 7.1 : Un graphe initial représentant l'échelle microscopique ■, et deux vues macroscopiques, ■ et ■, de ce graphe.

et qui peut être utilisée par l'interface graphique afin d'améliorer le rendu et la navigation.

7.2.3 Processus

Le changement d'échelle est effectué au travers d'un processus qui va nous permettre de créer la vue macroscopique. Il peut s'agir d'un algorithme de détection d'organisations ou tout autre algorithme capable de fournir une surjection de l'ensemble initial de nœuds vers un ensemble de structures. Ce processus agit donc comme le générateur du graphe correspond à l'échelle macroscopique.

Outre les processus « classiques » applicables aux graphes qui vont utiliser uniquement le graphe de la vue macroscopique, nous pouvons considérer des processus qui vont être en mesure d'utiliser à la fois la vue microscopique et macroscopique pour effectuer des mesures.

7.3 Réification des organisations

La création d'une vue macroscopique passe par plusieurs étapes. La première est création d'une classification des nœuds qui va permettre de déterminer de quelle organisation chaque nœud fait partie. Cette classification est donnée par une algorithme de détection d'organisations sous la forme d'une valeur associée à chaque nœud, correspondant à l'index d'une organisation. La seconde étape est de reconstruire les structures correspondant aux organisations à l'aide de la classification. Enfin, la dernière étape consiste à construire l'ensemble des arêtes existant entre les organisations. Chaque arête entre deux organisations A et B peut être pondéré par la quantité totale d'arêtes existant à l'échelle microscopique entre les nœuds de A et ceux de B.

7.3.1 Propriété de connexité

La difficulté de la création de la vue macroscopique réside dans la seconde étape de sa construction. On considère que le sous-graphe induit par les nœuds d'une même organisation doit être connexe, c'est à dire qu'il existe un chemin dans ce sous-graphe entre n'importe quelle paire de nœuds de l'organisation. Le problème que nous abordons ici est donc de fabriquer une vue macroscopique à partir d'une classification en s'assurant de cette propriété de connexité.

Il y a deux possibilités pour l'algorithme de détection :

1. soit le nombre d'organisations est un paramètre de l'algorithme ;
2. soit au contraire l'algorithme doit déterminer de lui-même le nombre d'organisations.

Si l'algorithme est libre de fixer le nombre d'organisations, on peut supposer que la propriété de connexité est respectée. Nous nous intéressons donc au cas où le nombre d'organisations est imposé. Dans ce cas, il est possible que l'algorithme fusionne des organisations afin d'aboutir au nombre désiré, comme illustré dans l'exemple de la figure 7.2. C'est le cas dans

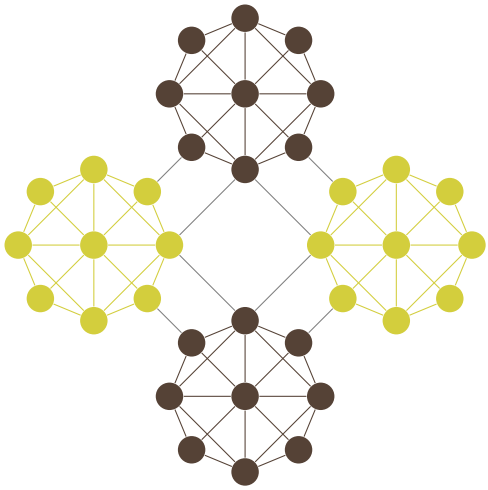


Figure 7.2 : Exemple de détection d'organisations non-connexes. Le nombre d'organisations à détecter est fixé à deux, cependant on observe quatre organisations, l'algorithme doit fusionner certaines de ces organisations.

l'algorithme AntCo^2 présenté dans la partie 10.2, où des colonies de fourmis numériques détectent chacune un ensemble d'organisations.

Il est alors nécessaire, pour fournir une vue respectant la propriété de connexité des organisations, d'effectuer un traitement permettant de récupérer les composantes connexes de chaque sous-graphe d'une organisation. Il s'agit de la seconde étape de la création de la vue macroscopique, qui nous permet d'obtenir les structures finales des organisations.

7.3.2 Fusion d'organisations connexes

Cependant, l'algorithme qui fournit la classification peut avoir fusionné des organisations entre lesquelles il existe des arêtes. Dans un tel cas, la détection des composantes connexes n'est d'aucune utilité.

Afin de détecter ces cas de figure, nous proposons l'approche suivante. Nous considérons qu'une organisation est une structure robuste. Cette robustesse est caractérisée par la présence d'un unique centroïde de l'organisation. Nous définissons cette notion en détail dans la partie 8.2.

En ayant à disposition un algorithme qui nous permette de détecter les centres d'équilibre dans un graphe, que nous nommons les centroïdes, nous pouvons alors appliquer cet algorithme à chaque organisation afin de déterminer si l'organisation possède plusieurs centroïdes. Dans ce cas, on considère que l'organisation est une fusion de plusieurs sous-organisations (une par centroïde) et qu'il faut donc procéder à une mitose, c'est à dire à séparer les sous-organisations. Nous proposons dans le chapitre 8 suivant, un algorithme permettant de détecter dynamiquement les centroïdes d'un graphe à l'aide d'un algorithme fourmis.

CHAPITRE 8

DÉTECTION DES CENTROÏDES

8.1 Présentation

Dans le chapitre 7 précédent, nous exposions une approche dont le but est de permettre d'améliorer la robustesse du système en détectant les organisations qui seraient composées de l'agrégation de plusieurs sous-organisations, ce qui fragiliserait l'ensemble. Cette technique repose sur la détection de points d'équilibre, les centroïdes, à l'intérieur de l'organisation, qui nécessite par conséquent une mesure qui nous permette d'évaluer la centralité des nœuds.

Nous avons présenté dans la partie 3.1.5 une description de la notion de centralité dans un graphe. Il s'agit de mesurer l'importance des nœuds dans le graphe, en considérant par

exemple la diffusion d'information : plus un nœud est un point de passage obligé pour l'acheminement d'une information entre deux nœuds, plus sa centralité sera importante. Nous avons présenté plusieurs mesures afin d'illustrer cette notion, qui était basées sur le degré, les plus court chemins, ou sur le voisinage.

La plupart de ces méthodes nécessitent des considérer le graphe d'une manière globale, que ce soit au travers du calcul de l'ensemble des plus courts chemins, ou par la construction de la matrice d'adjacence du graphe. Seul une mesure se basant sur le degré peut permettre une approche locale. Cependant, la structure d'une organisation peut être complexe et ne pas reposer que sur le degré de ses membres. Nous avons donc besoin d'étendre la portée des connaissances que nous avons des nœuds, sans entrer dans une approche globale. La considération d'une connaissance globale du graphe impliquerait une remise en question permanente de la validité de cette connaissance du fait de la dynamique de ce graphe.

Ceci explique notre motivation pour proposer une nouvelle approche permettant de ne pas nécessiter de connaissances globales du graphe, de tenir compte du voisinage, et d'être capable de s'adapter à la dynamique du graphe. Nous proposons donc, dans ce qui suit, une approche fourmis du calcul de la centralité, plus particulièrement de la recherche des centroïdes d'un graphe.

8.2 Centralité

Nous complétons ici l'état de l'art, que nous avons vu sur la centralité dans la partie 3.1.5, en développant les concepts de centre et de centroïde. Nous commençons tout d'abord par définir les notions nécessaires avant de décrire en détails les notions qui nous intéressent.

8.2.1 Définitions

Distance entre deux nœuds

La distance séparant deux nœuds a et b correspond au nombre d'arcs composant le plus court chemin entre a et b. Elle est notée $d(a, b)$.

Excentricité

L'excentricité d'un nœud v est la distance maximale existant entre v et n'importe quel autre nœud du graphe [Frank HARARY 1969]. Cette mesure, décrite dans l'équation 8.1, est notée $e(v)$.

$$e(v) = \max_{x \in V(G)} (d(v, x)) \tag{8.1}$$

Distance d'un nœud

La *distance* d'un nœud v, notée $d(v)$, est la somme des distances entre v et les autres nœuds du graphe, telle que définie dans l'équation 8.2.

$$d(v) = \sum_{x \in V(G)} d(v, x) \tag{8.2}$$

8.2.2 Centres et centroïdes

Le terme *centroïde* est utilisé dans plusieurs domaines scientifiques tels que la géométrie ou la physique. Il est aussi utilisé dans les sciences humaines pour agréger, par exemple, un ensemble de points dans l'espace.

Le *centroïde* d'un objet est son barycentre ou son centre géométrique. Dans le cas d'un objet pondéré dont la densité est uniforme , le centroïde sera alors le *centre de masse*. Le centroïde évoque l'idée d'un point d'équilibre de l'objet.

Les centres et centroïdes font l'objet de nombreux travaux dans la littérature. De C. Jordan donnant une preuve de leur existence dans chaque arbre [Jordan 1869] à P.J. Slater généralisant leurs définitions [Slater 1978], en passant par des parties traitant de ce concept dans les livres sur la théorie des graphes [Ore 1962 ; Frank Harary 1969].

Dans ce qui suit, nous allons décrire le concept de centroïde appliqué aux graphes en commençant par définir les notations qui sont utilisées. G désigne un graphe connexe dont $V(G)$ est l'ensemble de ses sommets et $E(G)$ l'ensemble de ses arcs. T désigne un arbre, c'est à dire un graphe dépourvu de cycle. $op(e, u)$, avec $e = (u, v)$ un arc, désigne le nœud v qui est l'opposé de u par rapport à l'arc e.

Définition formelle

Les concepts de *centres* et *centroïdes* ont d'abord été définis pour les arbres [Jordan 1869 ; Ore 1962 ; Frank Harary 1969] puis étendus aux graphes.

D'après [Frank Harary 1969], un nœud c_0 est considéré comme un *point central* ou *centre* de G si son excentricité $e(c_0)$ est minimale pour G, cf. équation 8.3.

$$e(c_0) = \min_{v \in V(G)} (d(v)) \tag{8.3}$$

Le *centre de gravité* d'un graphe G est un ensemble de nœuds v qui minimise la fonction $m()$ décrite dans l'équation 8.4. Le *centre de masse* d'un arbre T est un nœud dont le poids est minimal.

$$m(v) = \frac{1}{|V(G)|} \sum_{x \in V(G)} d(v, x)^2 \tag{8.4}$$

[Frank Harary 1969] définit le *centroïde* d'un graphe comme un ensemble de nœuds appelés *points du centroide*. Dans [Truszczynski 1985], le centroïde est considéré comme

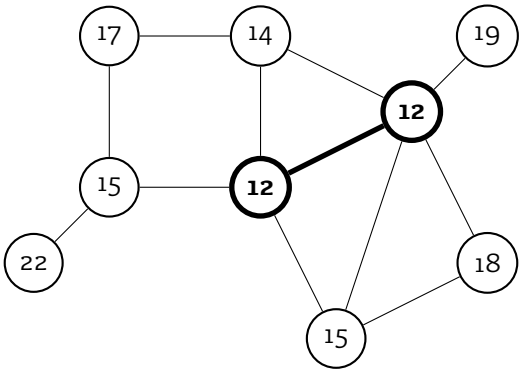


FIGURE 8.1 : Distances des nœuds. La structure en gras représente le sous-graphe induit par les nœuds dont la distance est minimale, ce qui correspond au centroïde de ce graphe.

étant le sous-graphe induit par les points du centroïde, c'est à dire le sous-graphe H de G tel que $V(H)$ soit l'ensemble de points du centroïde et

$$E(H) = \{(u, v) \mid (u, v) \in E(G);\ u, v \in V(H)\}$$

Il existe dans la littérature plusieurs façons de caractériser les points du centroïde. Dans [Frank HARARY 1969], les points d'un arbre T sont définis comme étant les nœuds dont le poids est minimal, i.e. les centres de masse de T. Dans le cas d'un graphe connexe et non-dirigé, [SLATER 1978] définit les points du centroïde comme étant les nœuds dont la distance est minimale, cf. figure 8.1.

Enfin, d'une manière plus générale, P.J. SLATER définit le *k-centrum* d'un graphe G [SLATER 1978] qui permet de lier les définitions de centre et centroïde. On note $r_k(v)$ la somme des k plus grandes distances entre un nœud v est les autres nœuds du graphe tel que définit dans l'équation 8.5. L'ensemble des nœuds minimisant r_k est appelé le $k-centrum$ de G et est noté $C(G;k)$. Si $k = 1$, alors $r_1(v)$ correspond à l'eccentricité de v, par conséquent $C(G;1)$ est le centre de G. Si $k = n$ avec $n = |V(G)|$, alors $r_n(v)$ est la distance de v et $C(G;n)$ le centroïde de G.

$$r_k(v) = \max_{S \subseteq V(G); |S|=k} \left\{ \sum_{s \in S} d(v, s) \right\} \tag{8.5}$$

8.3 Des fourmis et des centroïdes

L'algorithme 5 vu précédemment pour le calcul du centroïde permet de fournir le centroïde exact du graphe mais possède cependant quelques particularités qui font que cet algorithme est difficilement adaptable aux graphes dynamiques, alors qu'il s'agit du contexte dans lequel nous nous plaçons :

Algorithme 5 : Calcul classique du centroïde

```
Data :
    - G, le graphe, E(G) l'ensemble des arcs
    - A, ensemble de fourmis
1  begin
2      centroide = [] ;
3      min = +∞ ;
4      forall the node n ∈ V(G) do
5          d = Σ_{u∈V(G)/{n}} d(n,u) ;
6          if d < min then
7              centroide = [n] ;
8              min = d ;
9          end
10         else if d = min then
11             centroide.ajouter(n) ;
12         end
13         return centroide ;
14     end
15 end
```

1. la complexité en $O(|V|^3)$ [Floyd 1962], qui peut toutefois être réduite à $O(|V| \times |E| + |V|^2 \log|V|)$ [D. B. Johnson 1977], due à la nécessité de calculer l'ensemble des plus courts chemins du graphe ;
2. il ne permet pas d'adapter une solution précédemment calculée aux changements survenus dans le graphe ;
3. il est nécessaire d'avoir une connaissance globale du graphe.

.

Nous nous intéressons donc à définir une heuristique qui nous permettra de trouver une solution approchée du centroïde d'un graphe en utilisant uniquement des informations locales. Il s'agit d'un algorithme fourmis [Dorigo, Maniezzo et Colorni 1996] dans lequel une colonie de fourmis explore le graphe, en utilisant la stigmergie comme moyen de communication au travers de phéromones qui sont déposées sur chaque arc traversé.

L'idée générale de cet algorithme est que chaque nœud possède une masse que les fourmis peuvent déplacer par fragment vers d'autres nœuds. Le but final est d'augmenter la masse des nœuds du centroïde en réduisant de plus en plus la masse de ceux qui s'en éloignent.

Une fourmi a est composée des attributs suivants :

— pos_a, correspond au nœud sur lequel la fourmi se situe actuellement ;
— $phn_{max}(a)$ et $mass_{max}(a)$ qui seront décrit dans la suite ;
— $tmass_a$, la masse transportée par a ;

— une liste tabou de nœuds correspondant aux positions précédentes de a, qui est utilisée afin d'éviter un retour en arrière.

Pour chaque arête e, la mesure $phe(e)$ correspond au taux de phéromone actuel de e. On peut ensuite définir, pour chaque nœud n, un taux de phéromone $phn(n)$ qui est la somme des $phe(e_i)$ où les e_i correspondent aux arêtes adjacentes à n. Nous utilisons dans la suite la taux de phéromone des nœuds plutôt que celui des arêtes afin de prendre en compte le passage des fourmis sur les nœuds et non sur les arêtes.

Les phéromones s'évaporent à chaque étape de l'algorithme suivant un facteur $\rho \in [0;1]$. Si $phe_i(e)$ correspond au taux de phéromones de e au temps i alors nous avons :

$$phe_{i+1}(e) = phe_i(e) \times \rho, \forall e \in E(G)$$

Les attributs $phn_{max}(a)$ et $mass_{max}(a)$ représente respectivement le taux de phéromones maximal et la masse maximale que la fourmi a a rencontré durant son exploration du graphe. Afin de maintenir ces attributs à une valeur représentative, on utilise le même principe d'évaporation que pour les phéromones : les valeurs décroissent au fur et à mesure ce qui permet de les adapter aux changements qui surviennent dans le graphe. Les valeurs de ces attributs sont mis à jour au début du cycle d'une fourmi en considérant les arêtes adjacentes à la position actuelle de la fourmi, ainsi que leur extrémité opposée.

À chaque étape de l'algorithme, chaque fourmi doit choisir une arête adjacente à sa position actuelle qu'elle devra alors traverser. La distribution de probabilité P permettant de définir la probabilité $p(e_i)$ de chaque arête $e_i = (pos_a, v)$ d'être choisie est basée sur le taux de phéromone $phn(v)$ ainsi que sur une fonction $\eta_a(e_i)$, définie par l'équation 8.6, qui permet de prendre en compte la masse du nœud se trouvant à l'autre extrémité de l'arête. L'expression de $p(e_i)$ est donnée par l'équation 8.7. L'importance du taux de phéromone par rapport à la masse peut être contrôlée à l'aide du paramètre α. La fonction $\eta_a(e)$ retourne une valeur comprise entre $\lambda \in [0,1[$ et 1. Le paramètre λ permet de définir un seuil minimal et ainsi permettre de continuer l'exploration de nœuds dont la masse est faible voire nulle. On utilise l'opérateur op pour désigner l'extrémité opposé d'une arête. Ainsi, si $e = (u, v)$ alors $e \mathrm{op}\, u = v$ et $e \mathrm{op}\, v = u$.

$$\eta_a(e) = \lambda + (1-\lambda) \times \frac{mass(op(e, pos_a))}{mass_{max}(a)} \tag{8.6}$$

$$p(e) = phn(e \mathrm{op}\, pos_a))^{\alpha} \times \eta_a(e) \tag{8.7}$$

La seconde partie du comportement d'une fourmi est sa capacité à prélever une partie de la masse d'un nœud afin de la répartir sur d'autres nœuds. Chaque fourmi a possède un attribut $tmass_a$ qui correspond à la masse qui est actuellement transportée par a. Si cette masse est nulle, la fourmi tente de prélever une quantité de masse q de sa position courante comprise entre les paramètres $tmass_{min}$ et $tmass_{max}$. Dans le cas contraire, la fourmi

dépose une partie de la masse qu'elle transporte sur sa position actuelle n, cette quantité étant définie par la fonction $drop(a, n)$ dans l'équation 8.8.

$$drop(a, n) = tmass_a \times \frac{phn(n)}{phn_{max}(a)} \tag{8.8}$$

L'étape d'une fourmi est décrite dans l'algorithme 7 tandis que l'algorithme 6 présente l'exécution globale de l'algorithme fourmi.

Algorithme 6 : Algorithme général

Data :
- G, le graphe, $E(G)$ l'ensemble des arêtes
- A, ensemble de fourmis

```
1  begin
2      repeat
3          forall the arête e ∈ E(G) do
4              phe(e) ← phe(e)×ρ ;
5          end
6          forall the fourmi a ∈ A do
7              exécuter l'algorithme 7 sur a ;
8          end
9      until condition d'arrêt soit atteinte ;
10 end
```

8.4 Application aux organisations

Les organisations dans un graphe dynamique G telles que nous les avons présentées dans 3.2.1 forment des sous-graphes de G et nous avons vu dans le chapitre 4 que le but de la détection de ces organisations est de fournir une partition de l'ensemble des nœuds du graphe qui soit représentative de la cohésion qui existe entre ces nœuds. Dans le cas d'un graphe dynamique, on peut souhaiter que ces structures soient robustes avec un taux de renouvellement faible, c'est à dire que les éléments composant la structure varient peu au cours du temps.

Les structures modélisant les organisations, fournies par l'algorithme de détection d'organisations, peuvent comporter des nœuds qui affaiblissent la robustesse de leurs organisation. C'est le cas de l'organisation présentée dans la figure 8.2 : la suppression d'un des éléments de la partie faible en orange déconnecte l'organisation.

La détection des *parties faibles* des organisations pourrait permettre d'améliorer la détection de ces organisations en fournissant un résultat plus stable. La figure 8.3 montre comment augmenter la robustesse des organisations.

Algorithme 7 : Étape d'une fourmi

Données :
- a, la fourmi
- transport, masse transportée par a
- pos, position de a
- m_p, masse de pos

```
1  début
2      pour tous les arête e_i adjacente à pos faire
3          calculer p(e_i) ;
4      fin
5      construire la distribution de probabilité P à partir des p(e_i) ;
6      choisir aléatoirement une arête e en fonction de P ;
       /* traverser l'arête choisie */
7      pos ← op(e, pos) ;
       /* mettre à jour les valeurs maximales */
8      pour tous les arête (pos, v) faire
9          phn_max(a) ← max(phn_max(a), phn(v)) ;
10         tmass_max(a) ← max(mass_max(a), masse(v)) ;
11     fin
12     si transport = 0 alors
13         alea ← random(tmass_min, tmass_max) ;
14         q ← min(alea, masse(pos)) ;
15         transport ← transport + q ;
16         prendreMasse(q) ;
17     sinon
18         q ← drop(a, pos) ;
19         transport ← transport- q ;
20         deposerMasse(q) ;
21     fin
22 fin
```

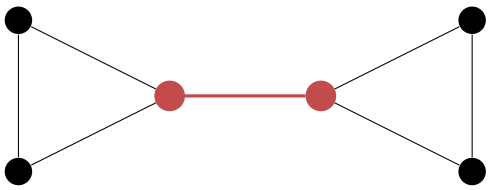


FIGURE 8.2 : une organisation comportant une partie faible (en orange).

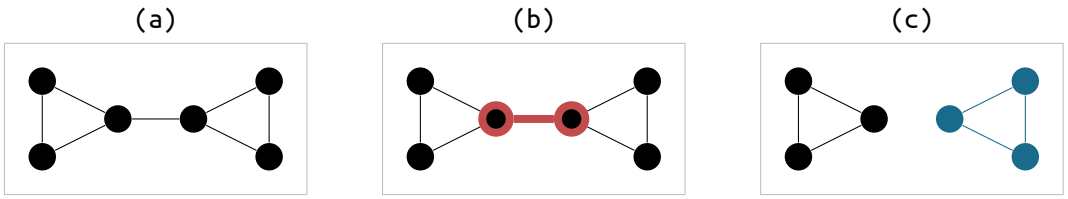


FIGURE 8.3 : (a) une organisation est fournie par un algorithme de détection d'organisations, (b) une partie *faible* est détectée, (c) l'organisation initiale est scindée en deux nouvelles organisations sans partie faible.

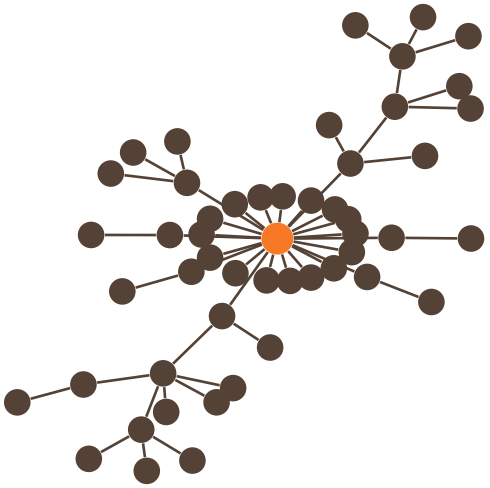


FIGURE 8.4 : Dans cet arbre de 50 nœuds, les fourmis ont détecté le centroïde optimal (représenté en orange sur la figure), c'est à dire le même centroïde que celui calculé par l'algorithme classique

8.5 Résultats

Nous avons présenté dans [DUTOT, OLIVIER et SAVIN 2011] des résultats obtenus à l'aide de cet algorithme. Nous avons comparé les résultats de notre algorithme fourmis avec les résultats obtenus en utilisant l'algorithme 5 *classique* du centroïde, le calcul des plus courts chemins se faisant grâce à l'algorithme de FLOYD-WARSHALL. Le but est ici de vérifier que ce nouvel algorithme est capable de trouver une solution approchée du centroïde d'un graphe. La figure 8.4 par exemple, montre un graphe dans lequel les fourmis ont réussi à trouver le centroïde optimal. Les paramètres de l'algorithme sont donnés dans la table 8.5.

8.5.1 Condition d'arrêt

Nous avons, dans un premier temps, comparé les résultats sur des graphes statiques. L'algorithme fourmis étant prévu pour être exécuté de manière continue, il est nécessaire de définir une condition d'arrêt afin de le stopper. Pour chaque nœud, nous définissons une mesure de *stabilisation*, décrite dans l'équation 8.9, qui nous permet de connaitre l'évolution de la masse d'un nœud. Cette mesure est normalisé entre 0 et 1, 0 signifiant que la masse du nœud évolue fortement, 1 que la masse est stable. Nous pouvons définir une mesure de stabilisation globale, décrite dans l'équation 8.10, qui est une moyenne de la mesure de stabilisation de l'ensemble des nœuds. La condition d'arrêt consiste à définir un seuil ϵ qui est la limite supérieure à atteindre pour la stabilisation globale afin d'arrêter l'algorithme. Pour ces expérimentations, ϵ est fixé à 0.99.

$$\begin{aligned} diff_t(n) &= \frac{|mass_{t-1}(n) - mass_t(n)|}{mass_t(n)} \\ stab_t(n \in V(G)) &= min(1, 1 - diff_t(n)) \end{aligned} \tag{8.9}$$

$$stab_t(G) = \frac{\sum_{n \in V(G)} stab_t(n)}{|V(G)|} \tag{8.10}$$

8.5.2 Comparaison des résultats

Nous avons utilisé différents modèles de graphes statiques pour ces expériences :

1. des arbres construits à l'aide du modèle d'attachement préférentiel de BARABASÍ-ALBERT [BARABÁSI et ALBERT 1999] ;
2. des graphes construits à l'aide du modèle de DOROGOVTSEV-MENDÈS [DOROGOVTSEV et MENDES 2002] ;
3. des grilles à deux dimensions, pleines et avec des vides aléatoires.

La comparaison des résultats se concentrent sur la différence entre les centroïdes calculés C_A et C_C ainsi que sur les temps d'exécution t_A et t_C, respectivement de l'algorithme fourmis et de l'algorithme classique. La différence Δt entre t_A et t_C , est exprimé comme le gain que t_A apporte par rapport à t_C :

$$\Delta t = \frac{t_A - t_C}{t_C}$$

La différence entre les centroïdes calculés repose sur deux points. D'une part la différence Δs du nombre de nœuds contenus dans les centroïdes C_A et C_C :

$$\Delta s = |C_A| - |C_C|$$

Et d'autre part, nous évaluons la différence Γ entre le contenu des centroïdes en mesurant pour chaque nœud v de $C_A \backslash C_C$, les nœuds de C_A qui ne sont pas dans C_C, la plus courte

TABLE 8.1 : Résultats pour les arbres (modèle d'attachement préférentiel). Les mesures sont des moyennes sur 1000 itérations. $\overline{xx}$ symbolise la moyenne et σ la variance.

nœuds	51	101	501	1001
arêtes	50	100	500	1000
$\overline{\Delta t}$	-37.1%	-84.2%	-98.6%	-99.0%
$\sigma(\Delta t)$	38.03%	5.07%	0.51%	0.00%
$\overline{\Delta s}$	+0.5	+0.4	+0.5	+0.5
$\sigma(\Delta s)$	1.13	0.77	0.77	0.87
$\overline{\Gamma}$	7.7%	6.7%	5.0%	4.0%
$\sigma(\Gamma)$	8.6%	7.3%	5.2%	4.1%

TABLE 8.2 : Résultats pour les graphes sans échelle (modèle de DOROGOVTSEV-MENDES). Les mesures sont des moyennes sur 1000 itérations. $\overline{xx}$ symbolise la moyenne de la mesure xx et $\sigma(xx)$ la variance.

nœuds	53	103	503	1003
arêtes	103	203	1003	2003
$\overline{t_\Delta}$	-34.2%	-79.1%	-97.9%	-99.0%
$\sigma(t_\Delta)$	19.84%	5.28%	0.30%	0.00%
$\overline{s_\Delta}$	+1.5	+0.7	+0.6	+0.6
$\sigma(s_\Delta)$	2.67	1.13	0.83	0.68
$\overline{\Gamma}$	10.4%	6.2%	4.3%	3.7%
$\sigma(\Gamma)$	9.4%	5.8%	3.8%	3.4%

distance $\gamma(v)$ entre v et le nœud de C_C le plus proche, normalisée par le diamètre du graphe. Autrement dit :

$$\gamma(v) = \min_{u \in C_C} \frac{d(u,v)}{\mathrm{diam}(G)}$$

Γ est alors la valeur moyenne de $\gamma(v)$ pour $v \in C_A$: si cette mesure est nulle, alors les deux centroïdes sont confondus ; au contraire, plus la valeur augmente plus les centroïdes sont éloignés.

8.5.3 Conclusion

Nous proposons une approche fourmis pour la recherche du centroïde d'un graphe. Il s'agit d'une marche aléatoire biaisée par le déplacement d'une masse virtuelle dans le graphe. Le but n'est pas ici de calculer le centroïde exact du graphe mais une valeur approchée qui puisse être calculée de manière décentralisée. L'objectif étant d'appliquer cet algorithme en complément d'une détection d'organisations elle-même utilisée pour la répartition de la

Table 8.3 : Résultats pour les grilles. Les mesures sont des moyennes sur 1000 itérations. $\overline{xx}$ symbolise la moyenne de la mesure xx et $\sigma(xx)$ la variance.

nœuds	36	121	256
arêtes	60	220	480
$\overline{t_\Delta}$	-15.9%	-89.3%	-96.7%
$\sigma(t_\Delta)$	31.99%	2.48%	0.78%
$\overline{s_\Delta}$	+2.9	+11.4	+13.3
$\sigma(s\Delta)$	3.51	6.83	9.69
$\overline{\Gamma}$	17.4%	17.8%	13.8%
$\sigma(\Gamma)$	4.7%	2.4%	1.8%

Table 8.4 : Résultats pour les grilles incomplètes. Les mesures sont des moyennes sur 1000 itérations. $\overline{xx}$ symbolise la moyenne de la mesure xx et $\sigma(xx)$ la variance.

nodes	22	93	214
edges	53	292	730
$\overline{t_\Delta}$	+197.8%	-77.8%	-94.4%
$\sigma(t_\Delta)$	122.77%	3.52%	0.88%
$\overline{s_\Delta}$	+4.1	+9.2	+13.7
$\sigma(s_\Delta)$	2.81	5.87	8.87
$\overline{\Gamma}$	25.3%	19.4%	15.1%
$\sigma(\Gamma)$	9.1%	5.2%	2.0%

Table 8.5 : Valeur des paramètres

ϵ	0.99
λ	0.50
α	1.00
dépôt de phéromone	0.03
évaporation de phéromone ρ	0.86

charge d'une simulation, la puissance de calcul requise doit être minimale afin de maximiser les ressources disponibles pour la simulation principale.

Les résultats sur les graphes statiques nous permettent d'observer que l'algorithme fourmis est significativement plus rapide, dans les conditions de l'expérimentation, que l'algorithme classique ; le gain obtenu est d'autant plus important que le graphe est grand. Notons qu'il s'agit là de graphe statique ; dans le cas de graphes dynamiques, l'algorithme fourmis est capable d'adapter sa solution, alors que l'algorithme classique nécessite d'être recalculé entièrement.

Concernant la qualité de la solution, la dégradation de la solution introduite par l'algorithme fourmis est faible en ce qui concerne les arbres et les graphes sans-échelle, les centroïdes calculés étant suffisamment proche pour que le résultat soit satisfaisant au vu de l'objectif recherché. Dans le cas des graphes dont la topologie est une grille (complète ou incomplète), l'algorithme fourmis fournit des résultats qui s'avèrent plus éloignés du résultat optimum. Cependant, cela est prévisible. En effet, l'heuristique repose sur l'hypothèse que le graphe est structuré, et que par conséquent, les fourmis vont pouvoir déplacer la masse vers les nœuds qui sont les plus à l'intérieur de la structure. Or, dans le cas de grille, la structure est absente, la partie locale du graphe dont une fourmi a connaissance est la même, peu importe la position de la fourmi dans le graphe (en dehors des bords de la grille). L'algorithme fourmis semble donc avoir un effet de bord qui est celui de nous permettre d'évaluer la force de la structuration à l'intérieur du graphe. Il s'agit bien sûr d'une hypothèse qui nécessite des investigations supplémentaires afin d'être vérifiée.

CHAPITRE 9

OUTIL DE MODÉLISATION

Comme nous l'avons vu dans ce qui précède, l'étude des écosystèmes computationnels nécessite un modèle, et celui de graphe semble être le plus approprié. Du fait que le temps dans lequel le système est plongé va jouer un rôle majeur dans l'émergence d'organisations, nous devons considérer des graphes dynamiques afin de tenir compte de l'évolution des différents éléments. Nous avons vu dans la partie 3.1.2 que la dynamique pouvait être modélisée de différentes façons : sous la forme d'une séquence, par un modèle agrégatif, ou encore sous la forme d'un flux d'événements. C'est sous cette dernière forme que nous avons choisi de manipuler la dynamique. À ce jour, peu d'outils sont disponibles pour nous permettre de manipuler ce type de dynamique, et aider ainsi les chercheurs et les développeurs dans la création d'algorithmes distribués dans le cadre de graphes dynamiques, permettant de me-

surer des propriétés de ces graphes ou encore d'en fournir une visualisation. Ceci explique notre motivation pour lancer le projet GraphStream, une bibliothèque logicielle en Java™ pour la manipulation de graphes dynamiques.

Le but de GraphStream n'est donc pas de remplacer les autres bibliothèques de graphes, déjà nombreuses en Java et d'autres langages, mais de fournir les éléments spécifiques à la manipulation de la dynamique, permettant ainsi une API facile d'utilisation pour la modélisation de l'évolution de réseaux d'interactions et des processus qui opèrent sur ou à l'intérieur des ces derniers. Dans ce but, GraphStream propose un ensemble d'outils permettant l'importation et l'exportation de la dynamique des graphes dans de nombreux formats ainsi que sous la forme d'une connection entre une source d'entrée et une de sortie. Toutefois, ses fonctionnalités principales sont la génération de graphes dynamiques, la conception d'algorithmes, et le développement de simulations, des prototypes aux problèmes à grande échelle. Sa gestion du temps permet à GraphStream de contrôler la dynamique du graphe, tandis que l'introduction de style dans la description des graphes permet d'élargir la manière de les visualiser, leur structure, leur dynamique ainsi que leurs propriétés. Enfin, sa flexibilité permet aux utilisateurs d'introduire de nouvelles métriques afin d'enrichir l'ensemble des mesures existantes.

Le fonctionnement interne de GraphStream repose sur un modèle événementiel qui forme le cœur de la bibliothèque et fournit une manière de décrire la dynamique du graphe. Nous allons décrire dans la prochaine section le modèle utilisé dans la bibliothèque. Nous exposons ensuite dans 9.2 une description de différents algorithmique contenus dans GraphStream, en particulier des générateurs de graphes dynamiques, des mesures, *etc*... Puis, nous décrivons dans 9.3 les composants permettant l'importation, l'exportation sous forme de fichier, ainsi que la visualisation de graphes.

9.1 Modèle

Afin de manipuler la dynamique, GraphStream ne modélise pas directement un graphe comme une structure de données mais comme un flux d'événements décrivant des modifications du graphe. Ceci correspond au formalisme d'une dynamique par flux d'événements que nous avons vu dans la partie 3.1.2. Il existe un nombre limité d'événement :

- ajout ou suppression de nœuds ;
- ajout ou suppression d'arêtes ;
- ajout, modification ou suppression de propriétés, les attributs, associées aux nœuds, arêtes ou au graphe lui-même ;
- évolution de l'indice temporel.

On peut classer ces événements en deux catégories principales. D'une part, les événements qui vont décrire une modification de la structure du graphe ; et d'autre part, les événements décrivant une modification de l'état interne d'un élément par la modification de la valeur d'une propriété qui lui est attachée. On peut considérer la temporalité comme une propriété du graphe, l'évolution de l'indice temporel rentre donc dans la seconde catégorie.

Ce flux d'événements décrit complètement le graphe et son évolution. Dans GraphStream, certains composants vont être capables de produire un tel flux, nous les appelons des *sources*. D'autres composants seront en mesure de consommer les événements, nous les appelons des *puits*. Le flux d'événements existe donc entre une source et plusieurs puits. Il est possible qu'un composant soit à la fois une source et un puits, nous les appelons des *canaux*.

Actuellement, le temps dans GraphStream est modélisé comme un simple nombre croissant qui peut être diffusé à n'importe quel moment dans le flux d'événements. Cela signifie qu'on considère que les événements qui surviennent entre deux diffusions de ce marqueur temporel apparaissent en même temps, bien qu'ils restent ordonnés dans le flux à l'aide d'un identifiant.

Cependant, GraphStream permet aussi de modéliser les nœuds, arêtes et graphes en tant qu'objets que l'on peut manipuler. Il s'agit d'objets tampons qui permettent de connaître l'état courant de l'élément. Suivant notre modèle événementiel, un graphe est un objet auquel est appliqué une séquence de mises à jour correspondant aux événements, de manière similaire à celle décrite dans [Demetrescu et al. 2010]. La bibliothèque fournit les objets graphes comme des canaux. Ils sont capables de recevoir des événements, et donc d'adapter leur structure, faite d'objets nœuds et d'arêtes, qui reflète l'état du graphe au temps présent. Ces objets graphes sont aussi capables de produire un flux d'événements qui décrit toutes les modifications du graphe.

En effet, les sources, puits et canaux peuvent eux-même être connectés en tant que graphe (une autre sorte de graphe, comme l'on peut le voir dans la figure 9.1), en suivant le concept d'observateur [Vlissides et al. 1995]. Ce concept est utilisé pour créer un flux d'événements entre une source (le sujet) et plusieurs puits (les observateurs). Par exemple, une application habituelle de ce concept consiste en une source de type fichier, qui lit les événements du graphe depuis un système de fichiers, connecté à un objet graphe qui maintient l'état courant du graphe en tant que structure de données que l'on peut explorer, et qui est lui-même connecté à une vue qui représente graphiquement l'état du graphe.

Le réseau formé par les sources, canaux, et puits peut contenir des cycles. Par exemple, il est parfois nécessaire de connecter l'affichage sur le graphe affiché afin d'obtenir les coordonnées des nœuds, éventuellement modifiées par l'utilisateur, qui peuvent ensuite être utilisées par des algorithmes. Un tel réseau est donné en exemple dans la figure 9.1.

Il existe plusieurs implantations de l'objet graphe, qui peuvent être utilisées en fonction des exigences : il faut choisir entre rapidité d'exécution, consommation de la mémoire, et temps d'accès aux éléments.

Dans GraphStream, les algorithmes peuvent opérer sur les objets graphes ou directement sur le flux d'événements. La bibliothèque fournit différents composants que nous décrivons dans ce qui suit.

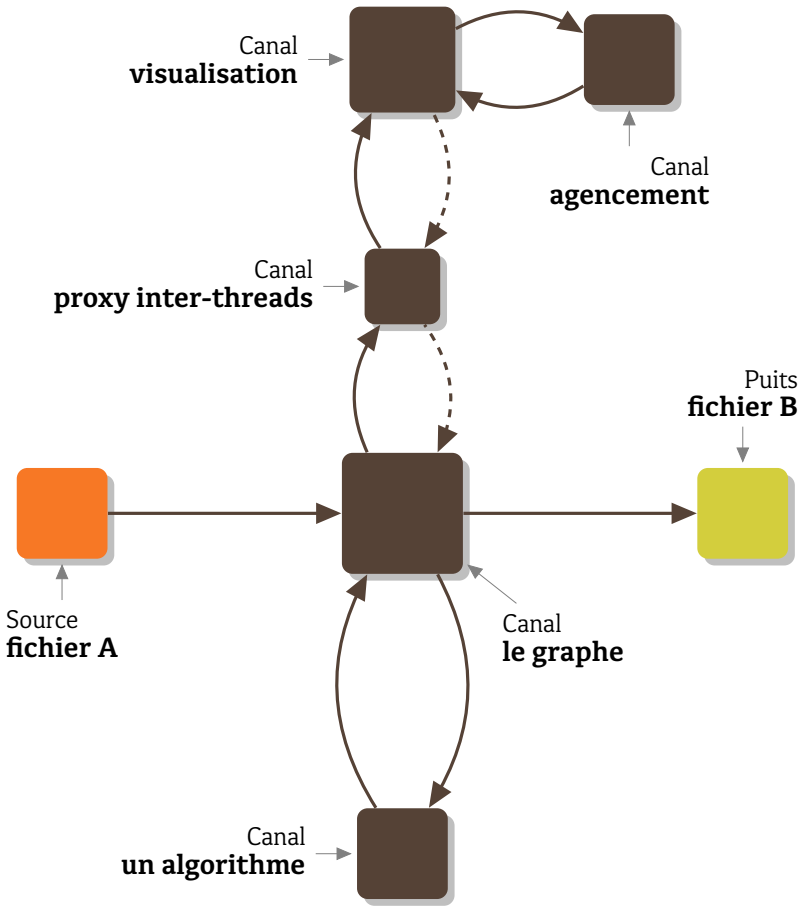


Figure 9.1 : Un exemple basique de réseau composé d'une source, de canaux, et d'un puits. Un fichier "A" est lu par une source afin de créer un graphe. On initialise une vue sur ce graphe dans un thread dédié aux graphismes. La vue communique avec le graphe au travers d'un proxy inter-threads. Elle utilise un algorithme d'agencement afin de positionner les nœuds. De plus, on exécute un algorithme sur le graphe qui répercute le fruit de son calcul directement dans le graphe. Enfin, le graphe, augmenté des données de l'algorithme, est stocké dans un fichier "B".

FIGURE 9.2 : Un générateur de graphe dynamique basé sur le modèle des boids.

9.2 Algorithmique

9.2.1 Générateurs

Les générateurs sont des sources particulières qui vont être capables de produire un flux d'événements à partir de rien, de manière algorithmique. La plupart des générateurs produisent un graphe en suivant un modèle donné, comme par exemple le modèle BARABASÍ-ALBERT [BARABÁSI et ALBERT 1999], celui de WATTS-STROGATZ [WATTS et STROGATZ 1998], ou encore celui de DOROGOVTSEV-MENDÈS [DOROGOVTSEV et MENDES 2002]. Mais il peut aussi s'agir de générateurs de graphes aléatoires, de grilles ou de tores, ou de graphes complètement connectés par exemple. Ils essayent, quand cela est possible, d'être dynamique, soit en permettant au graphe généré de grandir progressivement avec le temps, ou bien en reposant sur un modèle totalement dynamique qui permette l'ajout et/ou la suppression d'éléments. Nous avons par exemple adapté la simulation de boids (cf. annexe A) afin de l'utiliser comme un générateur (cf. la figure 9.2).

Nous travaillons aussi sur des générateurs plus spécifiques. Par exemple, certains générateurs permettent de transformer un type de données spécifiques en tant que graphe, comme c'est le cas des *shapefiles* représentant des réseaux routiers. C'est aussi le cas d'un gé-

nérateur permettant de générer un sous-graphe du Web à partir de une ou plusieurs adresses qu'il va être capable de suivre.

9.2.2 Métriques

Un autre type d'algorithmes que l'on peut exécuter sur un graphe dynamique est celui des mesures dynamiques. L'utilisateur a la possibilité de connecter une telle métrique sur un graphe, puis de suivre l'évolution des valeurs mesurées. Un exemple de ce type d'algorithmes est celui des *composantes connexes* qui permet de compter, à chaque pas de temps du graphe, le nombre de composantes connexes présentes.

On trouvera aussi dans GraphStream différentes mesures de centralité, en particulier la plupart des mesures exposées dans la partie 3.1.5 de ce manuscrit, comme la proximité, l'intermédiarité, ou encore le PageRank. Un algorithme de marche aléatoire est aussi disponible, permettant par exemple de mesurer la centralité par une approche stochastique (cf.la figure 9.3). La mesure de la modularité, ainsi que la mesure Surprise [Aldecoa et Marín 2011], permettant toutes deux d'évaluer la qualité d'une partition d'un graphe, sont présentes dans la bibliothèque.

De tels algorithmes essayent, quand cela est possible, d'éviter d'avoir à recalculer la solution entière de zéro mais au contraire d'adapter la solution précédente aux changements du graphes.

9.2.3 Recherche de structures

Une dernière catégorie importante d'algorithmes que nous proposons dans GraphStream concerne les algorithmes permettant de trouver des structures dans le graphe. Ces structures peuvent être des plus courts chemins comme c'est le cas des algorithmes Dijkstra [Dijkstra 1959], son extension A* [Hart, Nilsson et Raphael 1968], mais aussi celui de Bellman-Ford [Ford 1956 ; Bellman 1956] ou encore Floyd-Warshall [Floyd 1962 ; Warshall 1962]. Il peut aussi s'agir d'arbres couvrant qui peuvent être calculé par l'intermédiaire de l'algorithme de Prim [Prim 1957] ou de Kruskal [Kruskal 1956].

9.3 Exportation, importation et visualisation

9.3.1 Formats de fichier

Nous essayons de permettre l'utilisation de la plupart des formats de fichier permettant l'enregistrement d'un graphe. Ainsi GraphStream comprend le format GML [Passau 2011], Pajek [Batagelj et Mrvar 2004], GEXF le format du logiciel Gephi [Gephi 2009], le format TLP du logiciel Tulip [Tulip 2013], DOT celui de Graphviz [Graphviz 2013], mais encore NCol, LGL ou de simples listes d'arêtes par exemple. Dans la mesure du possible, ces formats sont disponibles à la fois en lecture, et en écriture.

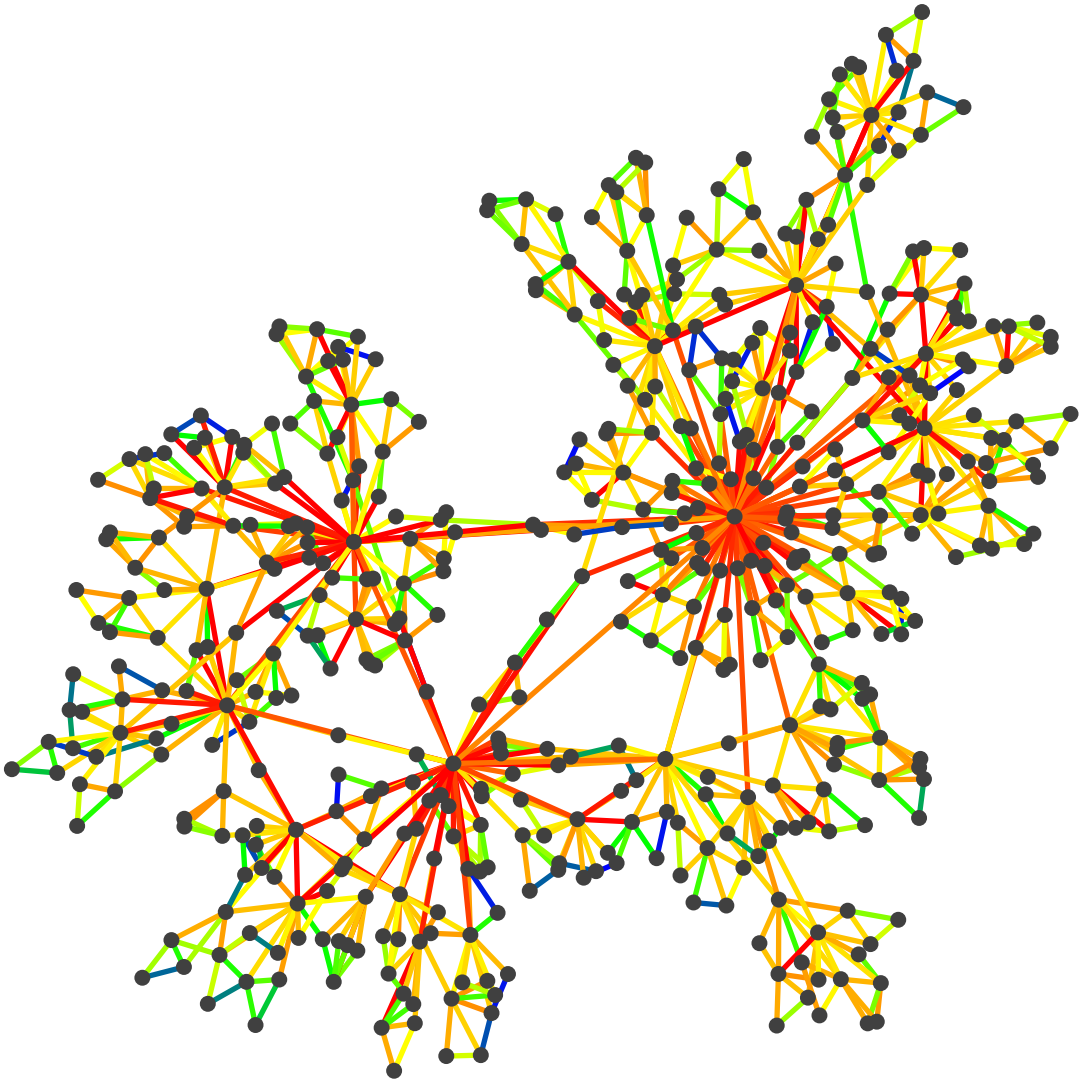


FIGURE 9.3 : Résultat d'une marche aléatoire sur un graphe généré à l'aide de l'algorithme de DOROGOVTSEV-MENDES. Les couleurs indiquent la fréquence de passage des entités sur l'arête, plus la couleur tend vers la rouge, plus le passage est important. On peut interpréter ceci en terme de centralité : plus un nœud est connecté à des arêtes dont la fréquence est forte, plus sa centralité est élevée.

```
001 DGS004
002 exemple 0 0
003
004 cg "propriété du graphe"=10
005
006 an "A"
007 an "B" label:"Je suis B"
008 an "C"
009
010 ae "AB" "A" "B"
011 ae "BC" "B" > "C"
012
013 st 2.0
014
015 dn "B"
```

Figure 9.4 : Exemple de fichier DGS. On construit un graphe à trois nœuds A, B et C puis une première arête AB entre les nœuds A et B, et un arc BC de B à C. À l'étape de temps 2, on supprime le nœud B.

DGS : Dynamic Graph Stream

La plupart des formats présentés ci-dessus ne permettent pas de tenir compte de la dynamique. Le format GEXF propose une gestion cumulative mais aucun ne permet de voir la dynamique comme un flux d'événements. C'est pourquoi, afin de permettre un stockage complet de la dynamique, GraphStream introduit son propre format nommé DGS [GraphStream 2013]. Il s'agit d'un format texte, lisible par l'œil humain, dans lequel chaque ligne décrit un événement. Ce format permet de décrire l'ensemble des événements de modification de la structure et des attributs que nous avons vu précédemment. Il permet de stocker différents types de valeurs des attributs (chaînes de caractères, entiers, réels, couleurs) mais aussi des types complexes (tableaux, tables de hachage). La figure 9.4 présente un exemple de fichier DGS.

Une sortie DGS peut être connectée sur un graphe au début d'une simulation et enregistrer ainsi l'ensemble des changements qui vont s'effectuer pour toute la durée de cette simulation. On peut ainsi, grâce à une entrée DGS qui lirait le fichier produit, rejouer la totalité de la simulation par la suite.

Le DGS est un format où chaque ligne correspond à un événement du graphe. Le ficher commence par une entête composée de six caractères qui permet de définir la version utili-

sée : "`DGS004`". Une ligne décrivant un événement commence par deux caractères qui vont définir le type :

`an, cn, dn` permettent respectivement d'ajouter, modifier, et supprimer un nœud. Ils sont directement suivis de l'identifiant du nœud concerné ;

`ae, ce, de` permettent quant à eux d'effectuer les mêmes opérations sur les arêtes. "`ae`" est de plus suivi par les identifiants des extrémités de l'arête entre lesquels peut s'ajouter le caractère "`>`" ou "`<`" afin de préciser la direction de l'arête ;

`cg` permet de modifier les attributs du graphe ;

`st` permet de modifier l'indice temporel ;

`cl` permet d'effacer le graphe.

La modification des attributs des éléments grâce aux types `cn, ce, cg` s'effectue sous la forme "`clef:valeur`" ou "`clef=valeur`". Les valeurs possibles sont des entiers, des réels, des chaînes de caractères, des couleurs, mais aussi des tableaux et des dictionnaires. Les tableaux sont entourés par les caractères "`[`" et "`]`", tandis que les dictionnaires le sont par les caractères "`{`" et "`}`". Les valeurs ou associations clef/valeur, de ces types complexes sont séparées par par le caractère "`,`".

L'indice temporel, défini par le type "`st`", est actuellement donné sous la forme d'un réel. Dans les futurs versions, le format de l'indice pourra être personnalisé pour permettre par exemple l'utilisation de dates.

9.3.2 Visualisation

La visualisation du graphe est une pièce importante d'une bibliothèque de graphe. Ivan Herman, Guy Mélançon et M. Scott Marshall en présentent une vue d'ensemble dans [Herman, Melançon et Marshall 2000]. La dynamique liée au réseau d'interactions apporte ses propres contraintes, comme cela est présenté par exemple par S. Bender-deMoll et D. A. McFarland dans [Bender-deMoll et McFarland 2006]. GraphStream fournit les outils de base pour permettre la visualisation d'un graphe. Ces outils sont composés d'un algorithme d'agencement des nœuds dans l'espace, d'une visionneuse, et d'un système de style qui permet de personnaliser l'affichage. Le style est inspiré des feuilles de style utilisées en HTML, le CSS [1]. On peut alors définir le style du graphe, des nœuds et des arêtes. Il est aussi possible de personnaliser le style d'un élément en particulier via son identifiant, ou d'un groupe d'éléments à l'aide de classes. Certaines propriétés comme la couleur ou la taille des éléments peuvent être dynamique ce qui permet la visualisation de l'évolution de mesures associées aux éléments. La figure 9.5 présente un exemple exploitant la personnalisation de l'affichage.

L'affichage permet les interactions entre l'utilisateur et le graphe, et peut ainsi communiquer les événements liés à l'interface graphique (clics de la souris, position des nœuds) à un algorithme ou une simulation tournant sur le graphe. Du fait que GraphStream est une

1. Cascading Style Sheet

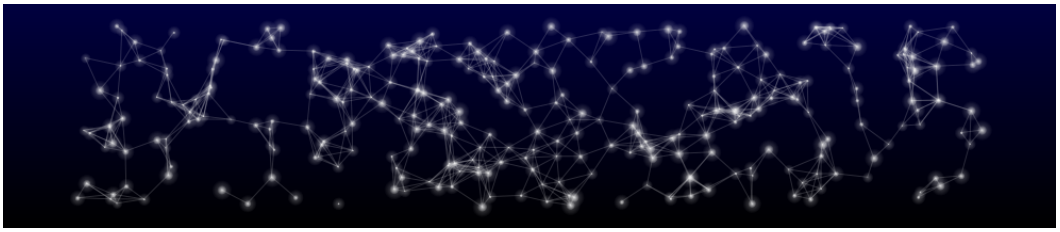


Figure 9.5 : Affichage avancé dans GraphStream

bibliothèque et non un programme final, l'affichage peut être inséré dans n'importe quelle application Java.

9.3.3 Communications réseaux, inter-threads et inter-plateformes

La bibliothèque fournit un protocole réseau, nommé NetStream, qui va permettre la diffusion des événements entre des sockets. Il s'agit d'un format binaire optimisé pour réduire la quantité de données et ainsi augmenter les performances. Ce protocole permet de connecter une source et un puits situés sur différentes machines, ou encore de faire communiquer GraphStream avec un programme écrit dans un langage différent. Il est possible, par exemple, d'héberger un graphe sur un serveur en utilisant GraphStream et de créer un affichage dans le navigateur d'un client grâce à une implantation Javascript de NetStream.

GraphStream permet aussi de connecter une source et un puits qui sont exécutés dans des threads différents. On utilise pour cela un système de boite aux lettres qui sert d'intermédiaire entre les threads. Nous utilisons, par exemple, un canal inter-threads afin de permettre de séparer l'affichage du graphe du traitement qui s'effectue sur ce graphe. Ainsi la vue peut être exécutée de manière transparente dans son propre thread afin de décharger l'utilisateur de la gestion des mécanismes liés à la visualisation.

Interactions avec NetLogo

Tandis que le protocole NetStream permet d'utiliser GraphStream avec des plateformes qui sont écrites dans un autre langage de programmation que le Java, il est tout à fait possible d'intégrer la bibliothèque directement dans des plateformes qui, comme NetLogo par exemple, sont écrites en Java.

Nous avons mis au point une extension NetLogo qui permet de modéliser le réseau d'interactions, formés par les tortues, en un graphe. Cela nous permet d'exploiter l'ensemble de l'algorithmique disponible dans GraphStream afin de l'appliquer dans NetLogo. Cela nous permet aussi de produire des graphes dynamiques à l'aide de simulations définies grâce au logiciel NetLogo qui est un outil développé dans ce but, et qui possède un large éventail d'exemples.

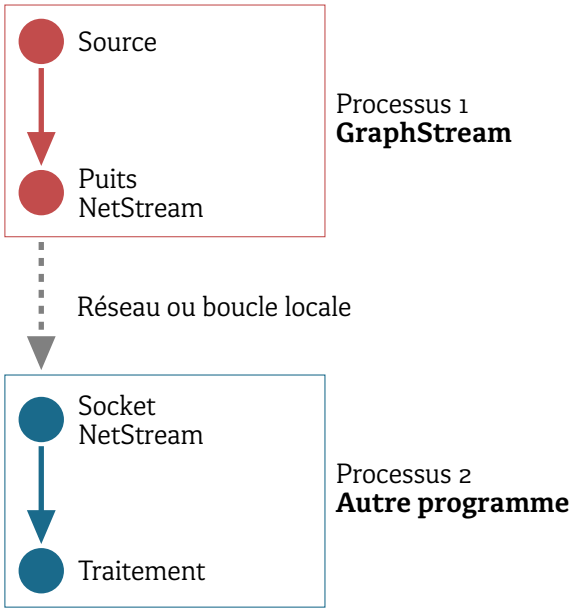


FIGURE 9.6 : Le protocole NetStream. Un programme GRAPHSTREAM est lancé dans lequel une source produit un graphe qui est envoyé à un puits NetStream. Un autre programme est lancé, éventuellement sur une machine différente. Ce programme inclut un socket qui se connecte au puits NetStream et qui va être capable d'interpréter le protocole et de traiter le graphe dynamique reçu.

L'extension GRAPHSTREAM pour NetLogo s'accompagne de la possibilité d'afficher le graphe créé par GRAPHSTREAM directement dans l'interface de NetLogo, comme présenté dans la figure 9.7.

9.4 Comment l'utiliser

GRAPHSTREAM est une bibliothèque gratuite et libre, *free as in freedom*. La bibliothèque est divisée en plusieurs *modules*, afin de s'adapter aux besoins des utilisateurs. Le *cœur* de GRAPHSTREAM constitue le module principal `gs-core` qui se veut minimal tout en permettant de manipuler un graphe dynamique, de lire la plupart des formats de fichier, et de fournir un affichage minimal. Les algorithmes sont contenus dans un module `gs-algo`, en dehors de certains algorithmes qui disposent de leur propre module, comme c'est le cas de AntCo^2. Un affichage offrant plus de fonctionnalités est également disponible dans un module `gs-ui`. D'autres modules spécifiques sont aussi disponibles, par exemple un projet permettant de modéliser les réseaux routiers comme c'est le cas dans la figure 9.8, qui présente une figure extraite de [NABAA et al. 2009] présentant une marche aléatoire dans un graphe modélisant le réseau routier du Havre.

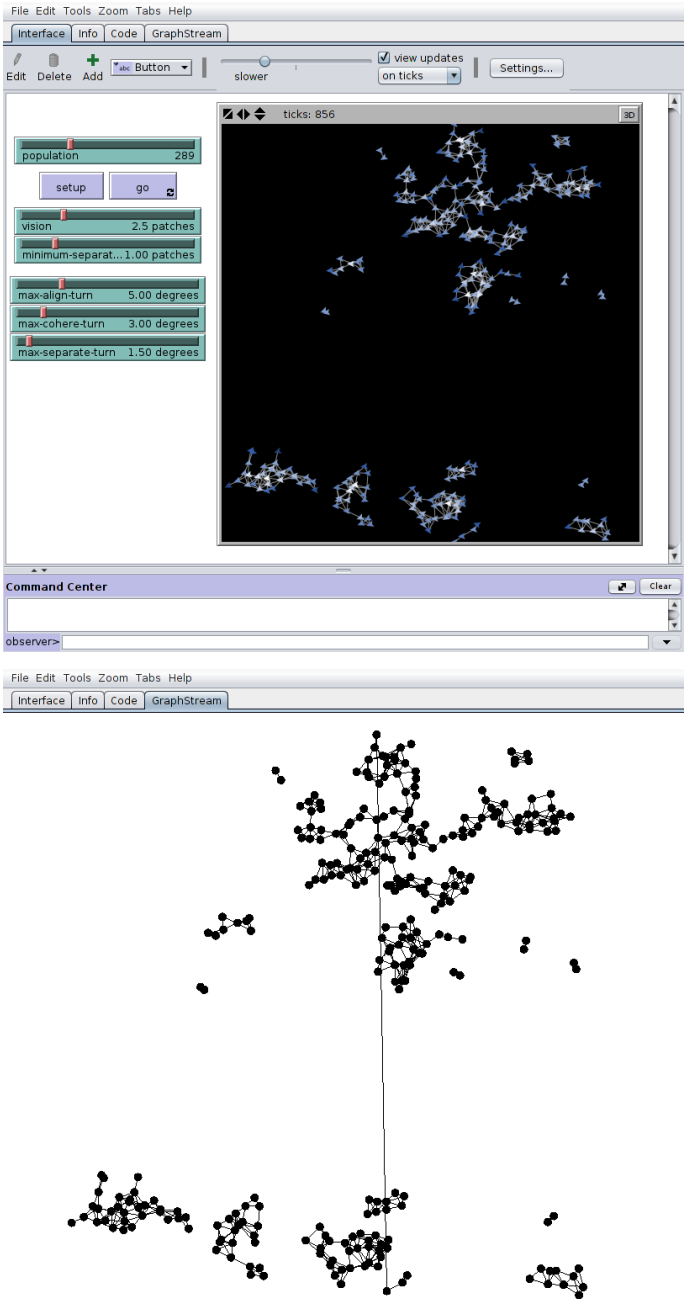


FIGURE 9.7 : Intégration de GRAPHSTREAM dans NetLogo

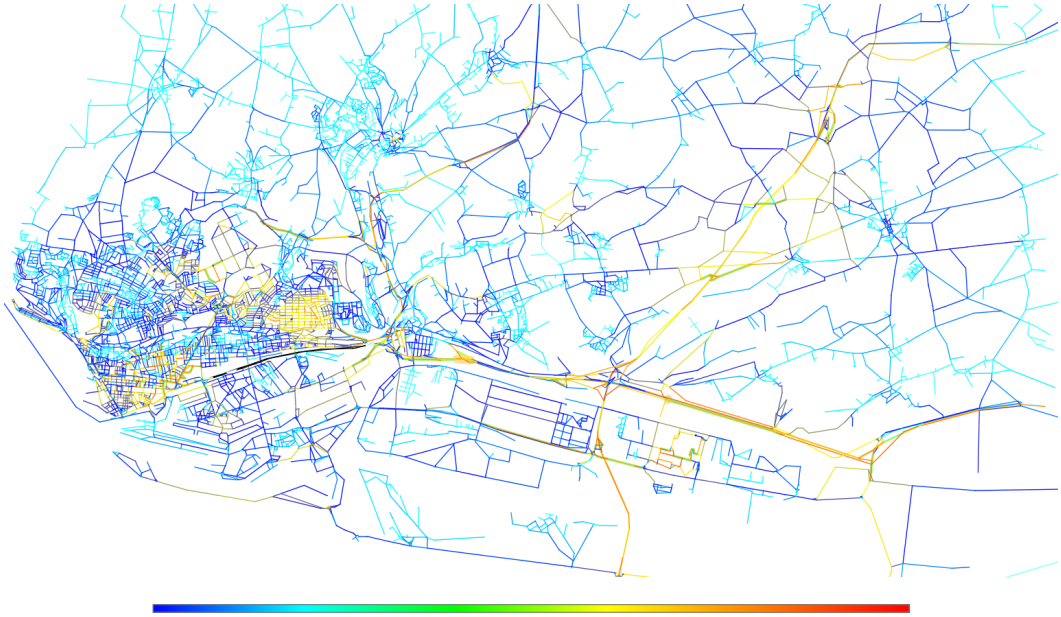


FIGURE 9.8 : Le réseau routier du Havre et de ses environs modélisé en tant que graphe par GRAPHSTREAM. On applique sur ce graphe une marche aléatoire « temporisée » c'est à dire que les entités qui l'utilisent vont à la vitesse maximale autorisée sur les arcs. Les entités possèdent, de plus, une mémoire des arcs traversés (la taille de la mémoire est fixée à 40 pour cet exemple) qui agit comme une liste d'arcs tabous qui ne peuvent être utilisés qu'en dernier recours.

Les différents modules peuvent être téléchargés depuis le site web de GRAPHSTREAM, http://graphstream-project.org, qui procure aussi une documentation complète, ainsi que la documentation de l'API et de nombreux tutoriels. Ils sont aussi disponibles dans le répertoire central de Maven, pour les projets utilisant cette méthode de production. Enfin, les sources et les versions en cours de développement sont hébergées et disponibles sur Github, https://github.com/graphstream.

Références

ALDECOA, Rodrigo et Ignacio MARÍN (2011). "Deciphering network community structure by surprise". In : *PloS one* 6 (9), e24195. ISSN : 1932-6203.

BARABÁSI, Albert-László et Réka ALBERT (1999). "Emergence of Scaling in Random Networks". In : *Science* 286 (5439), p. 509–512. DOI : 10.1126/science.286.5439.509. URL : http://www.sciencemag.org/content/286/5439/509.abstract.

BATAGELJ, Vladimir et Andrej MRVAR (2004). *Pajek—analysis and visualization of large networks*. Springer. ISBN : 3642622143.

Bellman, Richard (1956). *On a routing problem*. DTIC Document.

Bender-deMoll, Skye et Daniel A McFarland (2006). "The art and science of dynamic network visualization". In : *Journal of Social Structure* 7 (2), p. 1–38.

Demetrescu, Camil et al. (2010). "Dynamic graph algorithms". In : Algorithms and theory of computation handbook. Chapman & Hall/CRC, p. 9–9. ISBN : 1584888229.

Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs". In : *Numerische Mathematik* 1, p. 269–271.

Dorogovtsev, S. N. et J. F. F. Mendes (2002). "Evolution of networks". In : *Adv. Phys*, p. 1079–1187.

Floyd, Robert W. (1962). "Algorithm 97 : Shortest Path". In : *Communications of the ACM* 5 (6), p. 345.

Ford, Lester Randolph (1956). *Network Flow Theory*.

Gephi (2009). *GEXF File Format*. URL : http://gexf.net/format/ (visité le 04/11/2013).

Graphstream (2013). *GraphStream - The DGS File Format*. URL : http://graphstream-project.org/doc/Advanced-Concepts/The-DGS-File-Format_1.1/ (visité le 04/11/2013).

Graphviz (2013). *The DOT Language | Graphviz - Graph Visualization Software*. URL : http://www.graphviz.org/content/dot-language (visité le 04/11/2013).

Hart, Peter E, Nils J Nilsson et Bertram Raphael (1968). "A formal basis for the heuristic determination of minimum cost paths". In : *Systems Science and Cybernetics, IEEE Transactions on* 4 (2), p. 100–107. ISSN : 0536-1567.

Herman, Ivan, Guy Melançon et M. Scott Marshall (2000). "Graph visualization and navigation in information visualization : A survey". In : *Visualization and Computer Graphics, IEEE Transactions on* 6 (1), p. 24–43. ISSN : 1077-2626.

Kruskal, Joseph B (1956). "On the shortest spanning subtree of a graph and the traveling salesman problem". In : *Proceedings of the American Mathematical society* 7 (1), p. 48–50. ISSN : 0002-9939.

Nabaa, Michel et al. (2009). "Exploitation of a displacement survey to detect road network use vulnerability". In : ICCSA 2009 Conference Proceedings Special Sessions, p. 151–155.

Passau, Universität (2011). *Graph Modelling Language*. Projects. URL : http://www.fim.uni-passau.de/en/fim/faculty/chairs/theoretische-informatik/projects.html (visité le 11/03/2013).

Prim, Robert Clay (1957). "Shortest connection networks and some generalizations". In : *Bell system technical journal* 36 (6), p. 1389–1401.

Tulip (2013). *Tulip software graph format (TLP) | Tulip*. URL : http://tulip.labri.fr/TulipDrupal/?q=tlp-file-format (visité le 04/11/2013).

Vlissides, John et al. (1995). "Design patterns : Elements of reusable object-oriented software". In : *Reading : Addison-Wesley* 49, p. 120.

Warshall, Stephen (1962). "A theorem on boolean matrices". In : *Journal of the ACM (JACM)* 9 (1), p. 11–12. ISSN : 0004-5411.

WATTS, Duncan J et Steven H STROGATZ (1998). “Collective dynamics of ‘small-world’networks”. In : *nature* 393 (6684), p. 440–442. ISSN : 0028-0836.

Quatrième partie

Distribution

CHAPITRE 10

RÉPARTITION DE CHARGES

Dans ce qui précède, nous avons vu comment modéliser un système distribué dont le but est la simulation de systèmes complexes. Nous allons maintenant nous intéresser à la distribution de ce système qui repose sur deux points principaux. Nous présentons dans un premier temps notre contribution à la répartition de charges, telle que nous l'avons définie dans le chapitre 4, et qui permet de repartir les différentes entités sur les ressources de calcul disponible de façon à équilibrer la charge de ces ressources. Ensuite dans le chapitre suivant, nous nous focalisons sur notre apport à la couche logicielle, nommée intergiciel, permettant au système de s'abstraire de la partie distribution.

10.1 Vue d'ensemble

La répartition de charges est une technique intervenant dans différents domaines et permettant de manière générale de répartir des tâches sur un ensemble de ressources capables de les effectuer, et cela de manière à exploiter au mieux les ressources. La répartition de charges peut par exemple être utilisée devant un site web pour diriger les requêtes HTTP vers différents serveurs afin d'exploiter la puissance de plusieurs serveurs et ainsi assurer une meilleure qualité de service pour les utilisateurs.

Bien que dans notre cas, la répartition de charges consiste à répartir des entités plutôt que des requêtes http, le rôle d'un algorithme de répartition de charges reste d'équilibrer la charge de calcul d'un ensemble de machines disponibles. Du fait que ces entités interagissent, elles produisent des communications entre les machines qui peuvent surcharger le réseau. C'est pourquoi nous cherchons dans nos travaux, en complément d'une répartition équilibrée de la charge calcul, à optimiser la charge réseau. Pour cela, nous réduisons les communications distantes, que nous appelons aussi communications *effectives* et qui correspondent aux interactions entre des entités situées sur des machines différentes, en privilégiant au contraire les communications locales, lorsque les entités impliquées dans l'interaction se trouvent sur la même machine.

10.1.1 Pourquoi une répartition dynamique ?

Nous avons vu dans le chapitre 4 un état de l'art sur la répartition de charges statique et dynamique. Dans le cadre de nos travaux, une répartition statique implique de calculer une solution de répartition pour le graphe d'interaction à un temps t. Nous pouvons ensuite conserver cette solution tout au long de la distribution ou en calculer une nouvelle à partir de zéro lorsque des conditions qui restent à définir sont remplies (délai, événement déclencheur, etc...).

Si la première solution statique est adaptée à un système dont l'ensemble des entités est lui aussi statique et où l'on ne cherche pas à minimiser la charge réseau, il devient impossible de l'envisager dans notre contexte. Tout d'abord parce que les nouvelles entités qui vont apparaître au cours de la distribution, n'étant pas présentes au calcul de la solution de répartition, n'auront pas de ressource de calcul attribuée. Ces entités ainsi que celles qui vont disparaître vont participer à déséquilibrer la charge de calcul au fur et à mesure. Au niveau de la charge réseau, si dans le meilleur des cas le répartiteur prend en compte les interactions au moment du calcul, il va considérer le système au tout début de son évolution, là où les interactions ne l'ont pas encore structuré.

Recalculer une nouvelle solution depuis le début implique un coût de calcul supplémentaire et il est donc nécessaire de jouer entre la fréquence du calcul et la validité de la solution de répartition : plus le calcul sera fréquent, meilleure sera la solution, mais plus importante sera la charge de calcul dédiée uniquement à la répartition.

Une répartition statique, outre le fait qu'elle ne permet pas de prendre en compte la dynamique des entités autrement que par un nouveau calcul, ne permet pas non plus de

prendre en compte la dynamique des machines.

10.1.2 Organisations et répartition

Dans notre contexte, les entités du système interagissent permettant l'émergence d'auto-organisation. La quantité d'interactions entre les membres d'une même organisation étant plus importante que celle entre les membres d'organisations différentes, il devient possible, pour peu que l'on soit capable de détecter ces organisations, de répartir les dites organisations sur l'ensemble des machines de façon à ce que les membres d'une même organisation soient placés sur une même machine. On obtient ainsi une répartition de la majeure partie des communications sur les machines, tandis que la part des communications entre machines, c'est à dire des communications entre les organisations, est faible.

10.1.3 Répartition dynamique

La nature du système que nous distribuons implique que l'algorithme de répartition de charges soit en mesure de prendre en compte la dynamique et soit capable de s'adapter aux perturbations induites par l'ouverture du système.

La prise en compte de la dynamique va au delà d'un nouveau calcul d'une solution de répartition lorsqu'une modification significative est détectée. Du fait que l'on cherche à optimiser la charge de calcul, il ne faut pas que la charge dédiée au répartiteur de charges soit supérieure au gain apporté, et doit donc être minimisée. Une caractéristique importante de l'algorithme de répartition est donc d'avoir la capacité de mettre à jour une solution existante afin de ne pas avoir à recalculer une solution à partir du début.

10.1.4 Qui répartit le répartiteur ?

La localisation du répartiteur de charges dans le système distribué est un problème qui s'impose aussi à nous. Dans [Dutot, Olivier et Savin 2010b], nous présentions trois scénarios envisageables concernant les différentes localisations possibles, résumés dans la figure 10.1.

La première solution consiste à centraliser la réification du graphe dynamique modélisant la totalité de l'application sur une machine dédiée et d'exécuter le répartiteur de charges sur ce graphe. Les notifications de migrations des entités sont ensuite envoyées aux machines correspondantes. Cette solution possède plusieurs désavantages. Tout d'abord, elle ne tolère pas les pannes : si une panne se produit sur la machine dédiée au calcul de la répartition, cela désactive entièrement le calcul sur la totalité de l'application distribuée. De plus, la mise à jour du graphe et l'envoi des notifications de migration tendent à augmenter la charge du réseau. Enfin, cette solution, dans notre contexte, abouti à un non-sens : nous considérons en effet une forte dynamique, autant au niveau de l'application que des machines permettant la distribution ; or cette solution implique de rendre une partie des machines, la partie centrale dédiée à la répartition, statique.

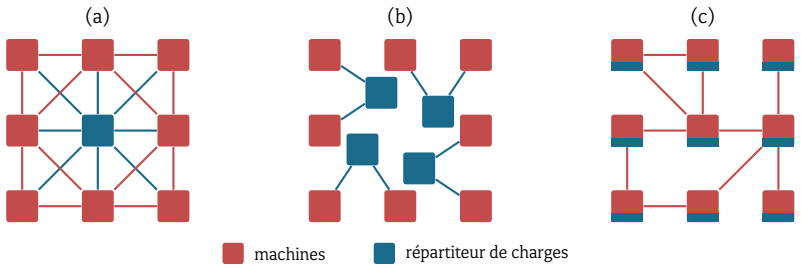


FIGURE 10.1 : Différents scénarios d'utilisation du répartiteur de charges : (a) une machine dédiée, (b) un ensemble de machines dédiées, (c) chaque machine participe à la répartition

La seconde solution est similaire à la précédente : la réification du graphe est toujours centralisée mais cette fois-ci sur un ensemble de machines, chaque machine hébergeant une partie du graphe. Cette solution permet de tolérer des pannes qui surviendraient sur des machines dédiées au calcul de la répartition mais implique à nouveau une augmentation de la charge du réseau.

La dernière solution consiste à inclure une instance du répartiteur de charges sur chacune des machines. Chacune de ces instances travaille sur la partie locale du graphe modélisant l'exécution de l'application distribuée, c'est à dire le sous-graphe composé des entités s'exécutant sur la machine courante. Les instances partageant des interactions entre entités s'échangent des informations afin de maintenir un graphe global décentralisé qui soit cohérent. Cette solution apporte plusieurs avantages :

1. la solution devient intégralement tolérante aux pannes des machines : de par la vision locale que chaque instance a du graphe, une panne survenant sur une machine n'a pas de répercutions sur le reste des machines ;
2. contrairement aux deux premières solutions, seules les informations concernant les intersections entre les différents sous-graphes sont échangées au travers du réseau. De plus, les échanges d'informations se font de façon pair à pair ce qui permet de mieux répartir ces échanges sur le réseau.

Dans ce dernier scénario, la charge de calcul de chaque instance du répartiteur est dépendante de la quantité d'entités hébergées sur la machine. Si les entités sont réparties de manière équilibrée sur l'ensemble des machines, alors la charge de calcul dédiée au répartiteur est elle aussi répartie de manière équilibrée ! On peut alors dire d'un tel répartiteur de charge qu'il est *auto-réparti*.

Ces différents scénarios ne sont malheureusement pas possibles pour n'importe quel algorithme de répartition de charges du fait que l'algorithme doit lui même être distribuable pour les deux derniers scénarios.

Le premier scénario est difficilement envisageable dans notre contexte du fait que nous avons comme contrainte un nombre d'entités considéré comme trop important pour qu'une

structure de données dont la taille serait proportionnelle à ce nombre d'entités puisse être utilisée. Or il est peu probable qu'une telle structure ne soit pas nécessaire lorsque le répartiteur ne fonctionne que sur une machine.

10.2 AntCo^2

Afin de répondre aux problèmes liés à la dynamique, Antoine DUTOT propose dans sa thèse [DUTOT 2005] un algorithme nommé AntCo^2 permettant la détection d'organisations dans un graphe dynamique. Cet algorithme repose sur des colonies de fourmis dont les mécanismes de rétroaction positive et de rétroaction négative vont permettre à l'algorithme de s'adapter à la dynamique.

Nous allons voir dans un premier temps le fonctionnement de cet algorithme, puis nous verrons un problème lié à l'oscillation éventuelle de la solution qui aboutit à un surplus de migrations des entités. Pour terminer, nous développerons une contribution permettant de lisser le résultat fourni par AntCo^2 afin de limiter l'oscillation.

10.2.1 Présentation

L'idée principale sur laquelle repose AntCo^2 consiste à utiliser la structure du réseau d'interaction sous-jacent à la simulation, les organisations, pour mieux distribuer les entités qui la composent sur les ressources de calcul à disposition. Le rôle d'AntCo^2 consiste à détecter ces organisations au sein du réseau d'interaction, représenté par un graphe dynamique. L'algorithme cherche alors à placer, si possible, les entités d'une même organisation sur la même ressource de calcul. L'objectif est de minimiser les coûts de communication entre entités fortement liées par leur liens préférentiels d'interaction, en distribuant la charge de calcul par organisation sur les ressources disponibles.

Cependant, la détection d'organisation dans un graphe dynamique pose, comme nous l'avons vu, le problème du maintien de ces organisations à travers les changements du graphe. Si une approche naïve consiste à recalculer les organisations à chaque changement significatif dans le graphe, cette approche, outre la charge considérable de calcul générée évoquée plus haut (cf. 10.1.3), pose la question importante de la mise en concordance des organisations détectées entre deux étapes. AntCo^2 est un algorithme *dynamique* capable de recycler les calculs des étapes précédentes pour s'adapter à ces changements. Il est similaire en cela aux algorithmes de *ré-optimisation*.

AntCo^2 utilise des colonies de *fourmis numériques* pour détecter et suivre les organisations dans le graphe dynamique représentant le réseau d'interaction de la simulation. Les fourmis vont classifier les nœuds du graphe en les associant à une colonie de fourmis ou une autre. Chaque colonie représente une ressource de calcul et, pour les distinguer, est associée à une couleur. Les fourmis vont coloniser des parties du graphes pour se les approprier et ainsi indiquer sur quelle ressource de calcul les entités doivent s'exécuter. Chaque sous-graphe connexe dont les nœuds sont d'une même couleur forme alors une organisation au

sens de la répartition sur les ressources de calcul. Cependant chaque couleur peut éventuellement représenter plusieurs organisations. En effet le nombre d'organisations ne correspond pas au nombre de ressources de calcul. De plus, le nombre de fourmis de chaque colonie dépend de la puissance relative des ressources de calcul et peut donc capturer un nombre d'organisations petites ou grandes différent.

Un algorithme fourmi est constitué d'un ensemble de fourmis qui vont parcourir un environnement, en l'occurrence le graphe dynamique, en déposant de l'information sur leur chemin, des *phéromones* numériques par analogie aux molécules chimiques déposées par les fourmis naturelles. Ces phéromones vont agir comme une boucle de rétroaction positive qui va permettre de renforcer un chemin ou une zone en particulier. L'homéostasie du système est possible grâce à un phénomène d'évaporation des phéromones qui permet d'une part d'éviter une sur-concentration de phéromones et d'autre part d'adapter la solution aux modifications de l'environnement.

L'algorithme repose sur un mécanisme de *collaboration* ainsi qu'un mécanisme de *compétition*. La collaboration existe entre les fourmis d'une même colonie et permet aux fourmis de détecter une structure en étant attirées par les phéromones de la colonie dont elles font partie. Ceci est complété par la compétition qui se produit entre les colonies et qui se manifeste par une répulsion des fourmis envers les phéromones des autres colonies. Ce second mécanisme va permettre de maintenir une certaine pression sur la membrane des organisations détectées et de maintenir un nombre équilibré d'organisations par colonie.

Nous allons maintenant décrire le fonctionnement de l'algorithme. Chaque fourmi traverse le graphe dynamique, passant de nœud en nœud en suivant les arêtes. À chaque passage sur une arête, les fourmis déposent une quantité de phéromones *colorées* spécifique à leur colonie, un indicateur que les autres fourmis pourront lire. Quand un nœud est entouré d'arêtes d'une couleur dominante, on dit que le nœud est conquis par la colonie de cette couleur. L'algorithme en déduit donc que l'entité représentée par ce nœud doit s'exécuter sur la ressource de calcul correspondante. Cette communication indirecte par le biais de phéromones est dite *stigmergique* (cf. 6.2.1).

Les fourmis choisissent la prochaine arête à traverser en utilisant un choix *aléatoire biaisé*. Elles ont tendance à favoriser les arêtes qui représentent un fort niveau d'interaction ainsi que celles ayant déjà une quantité importante de phéromones de leur couleur. Le premier critère permet de favoriser la colonisation de zones de forte interaction, les organisations. Le second permet aux fourmis, par rétroaction positive, de se regrouper sur une même zone au sein de ces organisations.

Les phéromones s'évaporent à chaque étape de l'algorithme, en conséquence elles doivent être maintenues par le passage constant des fourmis. C'est ce mécanisme qui permet à la fois la colonisation des organisations par une colonie à l'exclusion des autres, et la gestion de la dynamique du graphe. En effet, Les nouvelles organisations seront colonisées en raison de leur forte interaction. Les organisations détectées seront entretenues par la forte interaction et une quantité importante de phéromones d'une couleur sur leurs arêtes. Les organisation qui disparaissent seront progressivement abandonnées par les fourmis en raison de plus faibles interactions et du mécanisme de rétroaction négative dû à l'évaporation des phé-

romones non renouvelées. Ainsi les solutions de distribution déterminées par AntCo^2 sont continuellement entretenues et évoluent en accord avec l'évolution du réseau d'interaction sous-jacent lui-même.

La communication indirecte entre les fourmis est aussi un atout pour la distribution même de l'algorithme. En effet les fourmis étant des processus indépendants utilisant les informations déposées dans leur environnement, elles sont implicitement distribuables. Ceci permet d'utiliser AntCo^2 avec les scénarios b et c de distribution évoqués dans la figure 10.1, en particulier le scénario c qui implique une forme d'auto-répartition de l'algorithme d'AntCo^2 lui-même (cf. 10.1.4).

Le modèle de l'algorithme est le suivant. Nous considérons un graphe $G(t) = (\mathcal{V}(t), \mathcal{E}(t), \mathcal{C}(t))$ représentant l'ensemble des entités à distribuer et leurs interactions. Le temps t permet de gérer la dynamique de l'ensemble de nœuds $\mathcal{V}(t)$, arêtes $\mathcal{E}(t)$ et couleurs $\mathcal{C}(t)$. Chaque couleur est une colonie et représente une ressource de calcul. Les arêtes sont pondérées par un poids $w^{(t)}$ indiquant l'importance de leur interaction au niveau de la simulation.

À cette description, nous ajoutons une information de quantité de phéromones déposée par les fourmis sur les arêtes. Ainsi ces dernières contiennent un ensemble de messages déposés par les fourmis de chaque couleur : $card(\mathcal{C}(t))$ messages au temps t. Ces messages représentent la quantité de phéromones de chaque couleur présente sur les arêtes au temps t.

On note $\mathcal{F}(t)$ l'ensemble des fourmis au temps t, et $\mathcal{F}_c(t)$ l'ensemble des fourmis de couleur c au temps t.

Une fourmi k de couleur c traversant une arête e entre les itérations t et $t+1$ dépose une quantité de phéromones de couleur c :

$$\Delta_k^{(t)}(e, c) \tag{10.1}$$

La somme totale de phéromones de couleur c déposée sur l'arête e par toutes les fourmis de cette couleur entre t et $t+1$ est :

$$\Delta^{(t)}(e, c) = \sum_{k \in \mathcal{F}_c(t)} \Delta_k^{(t)}(e, c) \tag{10.2}$$

Ainsi, la quantité totale de phéromones sur une arête, quelle que soit la couleur est :

$$\Delta^{(t)}(e) = \sum_{c \in \mathcal{C}(t)} \Delta^{(t)}(e, c) \tag{10.3}$$

Quand $\Delta^{(t)}(e) \neq 0$, la proportion de phéromones de la couleur c sur e par rapport aux autres couleurs entre les temps t et $t+1$ est :

$$K_c^{(t)}(e) = \frac{\Delta^{(t)}(e, c)}{\Delta^{(t)}(e)} \tag{10.4}$$

avec $K_c^{(t)}(e) \in [0, 1]$.

La quantité totale de phéromones de couleur c présente sur l'arête e entre les temps t et $t+1$ est :

$$\tau^{(t)}(e,c) \tag{10.5}$$

et la quantité totale, en considérant toutes les couleurs est :

$$\tau^{(t)}(e) \tag{10.6}$$

Au début, cette valeur est fixée à :

$$\tau^{(0)}(e) = \epsilon \tag{10.7}$$

Nous verrons par la suite que cette quantité ne peut être négative.

Comme nous l'avons dit plus haut, les phéromones s'évaporent à chaque étape de l'algorithme. Ainsi, la quantité totale de phéromones de couleur c sur l'arête e au temps t est définie par la récurrence :

$$\tau^{(t)}(e,c) = \rho \cdot \tau^{(t-1)}(e,c) + \Delta^{(t-1)}(e,c) \tag{10.8}$$

Où $\rho \in]0,1]$ est le facteur de persistance des phéromones (l'évaporation est $1-\rho$).

La couleur $\xi^{(t)}(u)$ d'un nœud u du graphe est ensuite définie par la couleur prédominante sur toutes ses arêtes adjacentes :

$$\xi^{(t)}(u) = \arg\max_{c \in \mathcal{C}(t)} \sum_{e \in \mathcal{E}_u(t)} \tau^{(t)}(e,c) \tag{10.9}$$

avec $\mathcal{E}_u(t)$ l'ensemble des arêtes adjacentes au nœud u au temps t.

Les fourmis utilisent le graphe $G(t)$ comme environnement et le traversent en allant de nœud en nœud. Ainsi, à chaque pas de temps, chaque fourmi est associée à un nœud (qui peut donc accueillir plusieurs fourmis). À chaque étape, les fourmis traversent une arête.

La quantité de phéromones perçue par une fourmi de couleur c est $\tau^{(t)}(e,c)$, mais pondérée par la présence d'autres couleurs de phéromones sur l'arête. Grâce à ce mécanisme, les fourmis sont attirées par leur couleur et repoussées par les autres couleurs. On utilise un facteur $K_c^{(t)}(e)$ pour définir les phéromones perçues sur une arête e par une fourmi de couleur c :

$$\Omega^{(t)}(e,c) = K_c^{(t)}(e)\tau^{(t)}(e,c) \tag{10.10}$$

Ainsi à chaque étape de temps, une fourmi k de couleur c sur un nœud u choisit de traverser une arête ou une autre en fonction des phéromones de sa couleur et des autres couleurs sur cette arête. Nous notons $p^{(t)}(e,c)$ la probabilité que cette fourmi traverse l'arête $e=(u,v)$ entre t et $t+1$:

$$\begin{cases} p^{(t)}(e,c) = \dfrac{w^{(0)}(e)}{\sum\limits_{e_i \in \mathcal{E}_u(t)} w^{(0)}(e_i)} & \text{if } t = 0 \\ p^{(t)}(e,c) = \dfrac{\left(\Omega^{(t)}(e,c)\right)^{\alpha} \cdot \left(w^{(t)}(e)\right)^{\beta}}{\sum\limits_{e_i \in \mathcal{E}_u(t)} \left(\Omega^{(t)}(e_i,c)\right)^{\alpha} \cdot \left(w^{(t)}(e_i)\right)^{\beta}} & \text{if } t \neq 0 \end{cases} \tag{10.11}$$

où les paramètres α et β (tous deux > 0), comme dans l'article original de Dorigo [Dorigo, Maniezzo et Colorni 1996], permettent de modifier l'importance relative des phéromones par rapport au poids des arêtes, et où $\mathcal{E}_u(t)$ est l'ensemble des arêtes adjacentes au nœud u.

Afin d'éviter les mouvements oscillatoires dans lesquels les fourmis vont et viennent entre les deux mêmes nœuds u et v, nous introduisons dans chaque fourmi une mémoire des derniers nœuds visités. Cela est très comparable à une liste tabou d'une taille donnée. Nous introduisons dans 10.11 un coefficient $\eta \in]0,1]$ qui diminue l'importance des arêtes déjà traversées. Ainsi chaque fourmi se souvient des M arêtes dernièrement traversées. La valeur η pour une fourmi k sur le nœud u, considérant l'arête $e = (u, v)$ est :

$$\eta_k(v) = \begin{cases} 1 & \text{si } v \notin \mathcal{W}_k \\ \eta & \text{si } v \in \mathcal{W}_k \end{cases} \tag{10.12}$$

Pour une fourmi de couleur c sur le nœud u, la nouvelle probabilité de choisir l'arête $e = (u, v)$ entre les temps t et $t+1$ est :

$$p^{(t)}(e,c) = \frac{\left(\Omega^{(t)}(e,c)\right)^{\alpha} \cdot \left(w^{(t)}(e)\right)^{\beta} \cdot \eta_k(u)}{\sum_{e_i \in \mathcal{E}_u(t)} \left(\Omega^{(t)}(e_i,c)\right)^{\alpha} \cdot \left(w^{(t)}(e_i)\right)^{\beta} \cdot \eta_k(v_i)} \tag{10.13}$$

Ce modèle présente le principe global de fonctionnement de l'algorithme. Il existe d'autres mécanismes qui sont utilisés dans les dernières versions de l'algorithme. Ils permettent de l'améliorer en évitant la sur- ou sous-population de fourmis dans certaines zones du graphe par exemple, en utilisant un paramètre d'agoraphobie ainsi qu'un autre de surpopulation. Il y a aussi des mécanismes permettant de gérer l'ajout ou le retrait de colonies durant l'exécution, fournissant ainsi la possibilité d'avoir une nombre de ressources de calcul variable durant la simulation. Ces mécanismes sont décrits plus en détail dans [Dutot 2005] et [Bertelle et al. 2007].

10.2.2 Extraction des organisations

AntCo^2 fournit une classification des nœuds du graphe qui permet de dire que tel nœud *appartient* à telle colonie. Le résultat ne fournit pas directement la structure des organisations, mais permet de l'extraire : si deux nœuds adjacents appartiennent à une même colonie alors ils sont dans la même organisation. On peut, à partir de cette assertion, extraire les organisations à un temps donné.

L'analyse que nous souhaitons faire des organisations implique cependant d'aller au delà de l'étude de leur structure à un temps donné, afin d'étudier l'évolution de cette structure. Il est donc nécessaire d'être en mesure de maintenir la structure extraite d'une organisation dans le temps.

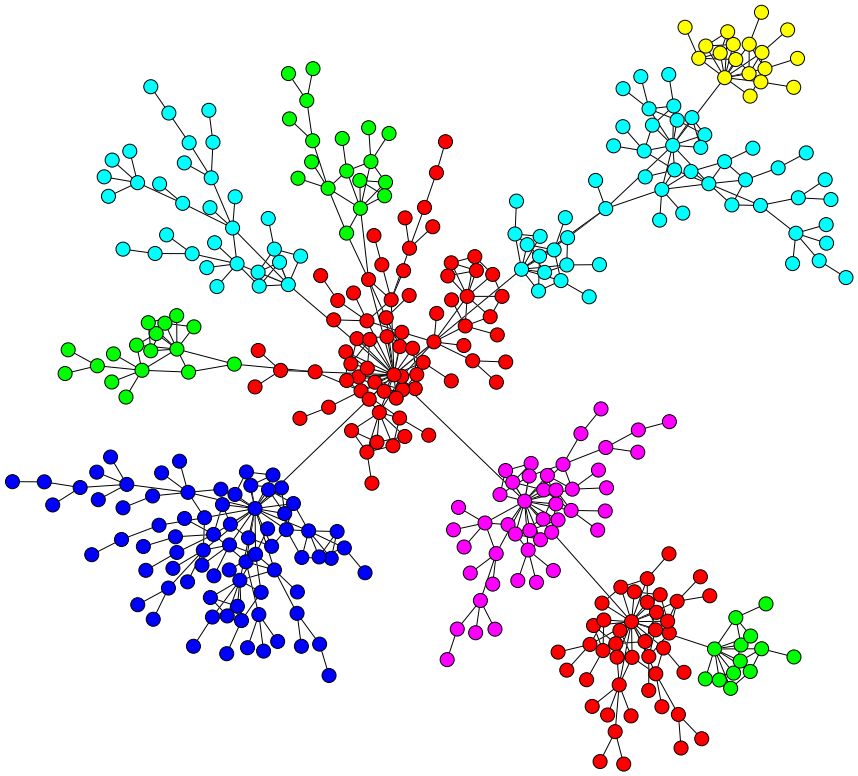


FIGURE 10.2 : Application de AntCo^2 à un graphe de 400 nœuds générés par attachement préférentiel. Six colonies de fourmis ont été utilisées ici, on compte cependant une dizaine d'organisations qu'il faut extraire.

10.2.3 Lissage

La membrane des organisations est une zone d'instabilité du fait que les nœuds de cette partie de l'organisation interagissent avec les nœuds d'autres organisations. En fonction de l'intensité de ces interactions, l'algorithme de détection d'organisations peut avoir quelques difficultés à déterminer à quelle organisation appartiennent ces nœuds. C'est par exemple le cas d'un nœud dans le graphe du *Zachary Karate Club* [ZACHARY 1977] dans lequel on observe deux factions, celle de Mr. HI et celle de JOHN telles que représentées sur la figure 10.3, mais où un nœud entretient des relations ambiguës avec les deux factions ce qui rend difficile de le classer dans l'une ou l'autre. Lorsque de tels nœuds existent dans un graphe dynamique, il y a une forte probabilité qu'ils oscillent d'une colonie à une autre. Dans le cadre de la répartition de charges, une colonie étant associée à une ressource de calcul, un changement de colonie va se traduire par la migration de l'entité correspondante au nœud vers sa nouvelle machine. Cette migration a un coût d'une part en terme de charge réseau du fait que les données liées à l'entité doivent être transférées, et d'autre part un coût temporel du fait que l'entité devient temporairement indisponible pendant sa migration.

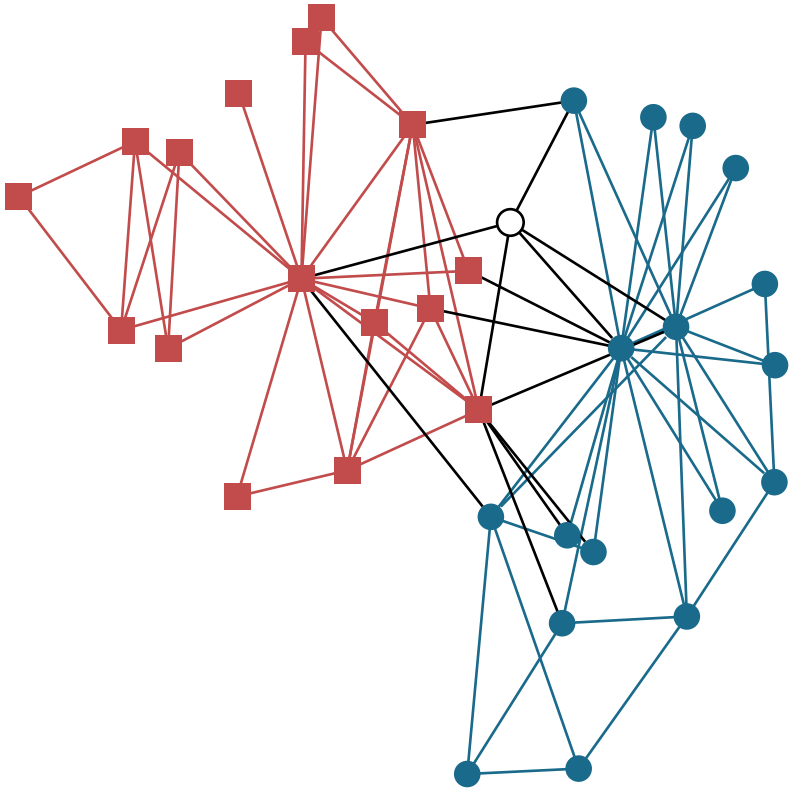


FIGURE 10.3 : Graphe du Zachary Karate Club. Les nœuds carrés représentent la faction de Mr. Hi, les ronds celle de John. Le nœud rond sans couleur pose problème pour déterminer la faction à laquelle il appartient.

Le problème vient du fait que nous devons être capables de distinguer une migration qui est le fruit d'un réel changement d'organisation d'un nœud et qui est donc nécessaire, du bruit qui se produit aux périphéries des organisations provoquant des migrations à répétition.

Ce qui suit décrit différentes techniques de *lissage* dont le but est de réduire ce bruit provoquant des migrations intempestives. Ces techniques prennent la forme d'un filtre qui se place entre l'algorithme AntCo^2 et la plate-forme dont le rôle est de procéder à une distribution effective.

Différents modes de lissage

Le but n'est pas ici de référencer toutes les techniques possibles et imaginables mais de présenter celles qui nous paraissent être les principales. Ces techniques peuvent être décrites sous la forme d'une fonction booléenne dont un résultat de 0 indique que la migration ne doit

pas être effective, et inversement lorsque le résultat est 1 :

$$l(n,o,m) \rightarrow b : (V,\mathbb{N},\mathbb{N}) \rightarrow \{0,1\} \tag{10.14}$$

où :

— n est le nœud pour lequel un changement de colonie est possible ;
— o correspond à la couleur de la colonie à laquelle appartient actuellement le nœud ;
— m est la couleur de la nouvelle colonie proposée par AntCo^2 pour le nœud.

Un lissage de base passif est le *lissage identité* qui se contente de transmettre les résultats de AntCo^2 sans les modifier :

$$l(n,o,m) = 1 \tag{10.15}$$

Une première technique consiste à utiliser le temps. On introduit alors un délai qu'il faut respecter avant de procéder à une nouvelle migration de l'entité :

$$l(n,o,m) = \begin{cases} 0 & \text{si } time() - last(n) < delai \\ 1 & \text{sinon} \end{cases} \tag{10.16}$$

où :

— $time()$ donne la date actuelle sous la forme d'une valeur entière ;
— $last(n)$ est une fonction donnant la dernière date de migration du nœud n ;
— $delai$ est une constante correspondant au délai minimum nécessaire entre deux migrations d'un nœud.

Cette technique possède l'avantage de sa simplicité de mise en œuvre et permet de réduire la fréquence des migrations d'une entité oscillante. Cependant, l'oscillation des migrations perdure bien qu'à une échelle de temps différente. De plus, si la fréquence de l'oscillation est proportionnelle au délai, cette technique devient quasiment inefficace.

Il est possible de pallier ce dernier problème en instaurant un délai aléatoire par entité à chaque nouvelle migration. Cette méthode comme la précédente reste une technique basique qui permet simplement de réduire la fréquence de l'oscillation des migrations sans toutefois la supprimer.

Lissage biomimétique

Une autre approche consiste à s'inspirer à nouveau de la nature, en l'occurrence du comportement de groupe que l'on retrouve chez certaines espèces d'oiseaux, de poissons ou encore d'insectes. C. REYNOLDS dans [REYNOLDS 1987] proposa un modèle permettant de reproduire le comportement de ces essaims sur la base de trois règles s'appliquant à chaque entité, nommée *boid*, de la simulation :

1. la cohésion, qui permet aux boids de se regrouper ;
2. la séparation, un mécanisme permettant de maintenir une certaine distance entre deux boids afin qu'ils n'occupent pas le même espace ;

3. l'alignement, qui permet à l'ensemble des boids du groupe d'avoir une direction similaire.

On peut facilement étendre ce modèle afin de faire cohabiter plusieurs *espèces* de boids qui se repousseraient mutuellement ou au contraire qui seraient régis par une relation de proie/prédateur.

L'intérêt de ce modèle dans le cadre du lissage des résultats de AntCo^2 vient du fait que lorsqu'une entité (respectant les règles des boids) fait partie d'un groupe, il est nécessaire d'appliquer une certaine force afin de séparer l'entité de son groupe ce qui donne une certaine forme d'élasticité à la structure qui lui permet de rester soudée tout en s'adaptant aux perturbations de l'environnement.

Or c'est cette élasticité que l'on recherche dans le lissage : une force qui maintienne l'entité dans son organisation suffisamment pour résister aux migrations intempestives mais qui permette tout de même à l'entité de quitter l'organisation lorsque c'est nécessaire.

10.3 Résultats

10.3.1 Description des simulations

La première simulation, nommée $\mathcal{S}_1$, est composée d'un graphe statique de 10201 nœuds dont la topologie est une grille. Quatre colonies de fourmis sont utilisées pour répartir la charge. La table 10.1 récapitule les paramètres utilisés pour initialiser l'algorithme AntCo^2, qui est appliqué pendant 10000 étapes.

Nous nous basons sur le modèle des boids pour la seconde simulation, nommée $\mathcal{S}_2$, afin de générer un graphe aléatoire. Les nœuds du graphe restent les mêmes tout au long de la simulation mais les interactions évoluent. Le modèle des boids est idéal du fait qu'il amène à la création d'organisation : des groupes de boids fortement liés se créent au fur et a mesure. Pour éviter la création d'un seul groupe, nous utilisons deux espèces de boids : la première est attirée par la seconde, tandis que la seconde est repoussée par la première. Nous utiliserons de même quatre colonies de fourmis dans AntCo^2 qui sera initialisé avec les mêmes paramètres que la simulation précédente.

10.3.2 Méthodes existantes

Les résultats de AntCo^2 sont initialement mesurés au travers de deux mesures r_1 et r_2. La première permet de mesurer la proportion de communications effectives, établies entre des entités hébergées sur différentes machines, par rapport aux communications locales établies entre des entités hébergées sur une même machine. Cette mesure, dont le résultat est compris dans l'intervalle $[0;1]$, permet d'évaluer la charge réseau : plus r_1 est proche de zéro, plus la charge réseau est faible. On considère un graphe $G(t) = (\mathcal{V}(t), \mathcal{E}(t))$ au temps t. Pour chaque arête $e \in \mathcal{E}(t)$, $w^{(t)}(e)$ indique le poids de e. $\mathcal{A}(t)$ englobe les arêtes correspondant

graine aléatoire :	29136152552356L
α **:**	1.0
β **:**	3.0
fourmi par sommet :	4
ρ **:**	0.8
dépôt phéromone :	0.1
mémoire :	9
agoraphobie :	0.3
surpopulation :	10

TABLE 10.1 : Paramètres.

aux communications effectives, c'est à dire les arêtes dont les nœuds sont contrôlés par des colonies différentes de fourmis. r_1 peut alors s'écrire :

$$r_1 = \frac{\sum_{a \in \mathcal{A}(t)} w^{(t)}(a)}{\sum_{e \in \mathcal{E}(t)} w^{(t)}(e)} \tag{10.17}$$

Cette mesure seule ne suffit pas : si toutes les entités sont placées sur une seule et même machine alors l'on obtient une valeur de r_1 optimale, or la charge de calcul n'est pas équilibrée ! r_1 est donc associée à la mesure r_2 qui permet de mesurer la qualité de la répartition de la charge de calcul par un ratio entre la taille (en nombre de nœuds acquis) de la plus petite colonie et la taille de la plus grande. Ainsi, plus r_2 se rapproche de 1, plus la différence entre les quantités d'entités des ressources de calcul diminue et donc meilleure est la répartition de la charge. En considérant les sous-ensembles de nœuds V_i correspondant aux nœuds contrôlés par chaque colonie de fourmis i, on peut écrire r_2 sous la forme suivante :

$$r_2 = \frac{\min(\text{card}(\mathcal{V}_1), \ldots, \text{card}(\mathcal{V}_k))}{\max(\text{card}(\mathcal{V}_1), \ldots, \text{card}(\mathcal{V}_k))} \tag{10.18}$$

Si l'on considère r_1^i, r_1^t etr_1^c comme correspondant respectivement aux valeurs de r_1 pour la méthode sans lissage, avec lissage temporisé, et avec le lissage par cohésion, on peut alors considérer les différences $\Delta^{ti} r_1 = r_1^t - r_1^i$ et $\Delta^{ci} r_1 = r_1^c - r_1^i$. De manière similaire pour r_2, on peut considérer les différences $\Delta^{ti} r_2 = r_2^t - r_2^i$ et $\Delta^{ci} r_2 = r_2^c - r_2^i$. Ces différences nous permettent de mieux évaluer la divergence des méthodes de lissage par rapport au résultat initial.

Simulation $\mathcal{S}_1$

On peut observer sur les figures 10.4 et 10.5 l'évolution des paramètres r_1 et r_2 pour la simulation $\mathcal{S}_1$ pour les différentes méthodes de lissage. On constate une période d'instabilité sur les premières 1000 itérations, commune aux trois modes de lissage comparés, qui correspond au temps nécessaire à AntCo^2 pour stabiliser la solution. Puis on observe que la mesure r_1 se stabilise rapidement vers une valeur seuil proche de 0. La valeur 0 implique qu'il n'y ait pas d'interactions entre les organisations, ce qui n'aurait aucun intérêt du fait que dans cas, il n'y aurait pas de structuration à l'échelle globale du graphe.

La méthode de lissage utilisée affecte peu l'évolution des mesures r_1 et r_2. On constate une valeur moyenne pour $\Delta^{ti} r_1$ de $3.87 \cdot 10^{-3}$ avec un écart type de $2.81 \cdot 10^{-2}$, et pour $\Delta^{ci} r_1$ de $1.65 \cdot 10^{-4}$ avec un écart type de $1.88 \cdot 10^{-3}$. La mesure r_1 pour les lissages temporisé et par cohésion est donc très proche de la mesure d'origine, avec un écartement plus faible pour le lissage par cohésion. En ce qui concerne la mesure r_2, la valeur moyenne de $\Delta^{ti} r_2$ est égale à $1.20 \cdot 10^{-3}$ et celle de $\Delta^{ci} r_2$ vaut $1.72 \cdot 10^{-4}$ avec un écart type respectif de $1.05 \cdot 10^{-2}$ et de $2.07 \cdot 10^{-3}$. On constate que la mesure de r_2 pour les résultats avec lissage par cohésion est très proche du résultat d'origine. C'est aussi le cas, bien que de manière plus fluctuante, pour le lissage temporisé.

Un décalage se fait ressentir pour le lissage temporisé par rapport aux résultats d'origine et cela principalement lors d'une évolution significative de la mesure. Ce décalage n'est cependant que le résultat de la temporisation du fait que les valeurs dont dépendent les mesures r_1 et r_2 ne varient qu'en présence de migrations.

Modifier le délai dans le cas du lissage temporisé n'a que peu d'impact sur les mesures r_1 et r_2 : on constate simplement que le décalage relatif à l'évolution des mesures est proportionnel au délai de la méthode de lissage (cf. les figures 10.6 et 10.7 pour la simulation $\mathcal{S}_1$, et les figures 10.8 et 10.9 pour $\mathcal{S}_2$).

Simulation $\mathcal{S}_2$

L'ajout de la dynamique entraine des modifications significatives dans la structure du graphe qui va nous permettre d'étudier l'adaptation dont est capable de faire preuve l'algorithme. L'aspect général du tracé de r_1 présenté dans la figure 10.10 montre que le lissage a peu d'incidence sur la valeur de cette mesure. Le tracé de $\Delta^{ti} r_1$ et $\Delta^{ci} r_1$ indique que le lissage temporisé améliore la qualité de r_1, et que la valeur de r_1 pour le lissage par cohésion est confondue avec la valeur d'origine. La valeur moyenne de $\Delta^{ti} r_1$ est $6.38 \cdot 10^{-3}$ pour un écart type de $2.71 \cdot 10^{-2}$. Celle de $\Delta^{ci} r_1$ est $2.54 \cdot 10^{-4}$ avec un écart type de $1.51 \cdot 10^{-3}$.

L'observation de la mesure r_2 (cf. la figure 10.11) ne nous permet pas une conclusion tranchée sur l'impact du lissage. L'observation de $\Delta^{ti} r_2$ et $\Delta^{ci} r_2$ montre que le lissage par cohésion entraine une différence moins importante avec le résultat d'origine que le lissage temporisé, mais que pour ces deux lissages, la différence est fluctuante.

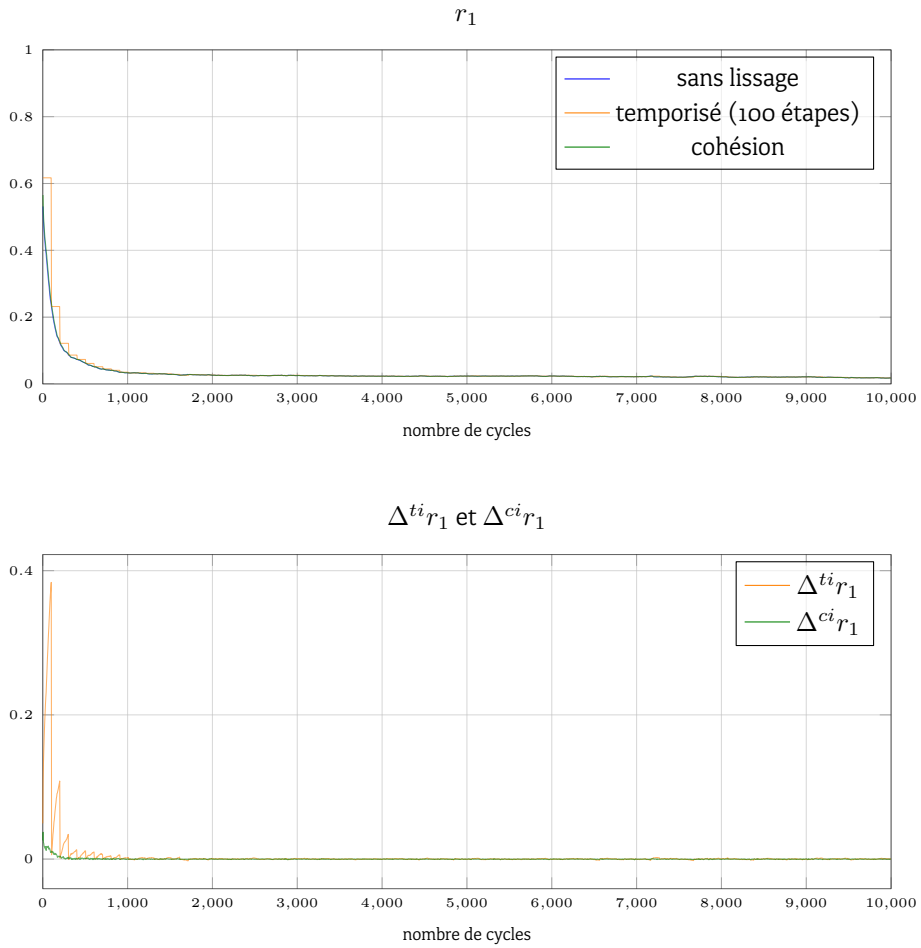


FIGURE 10.4 : Tracé de la mesure r_1 pour la simulation $\mathcal{S}_1$. La différence moyenne entre la mesure de r_1 pour le lissage temporisé et celle de la méthode sans lissage est $3.87 \cdot 10^{-3}$ avec un écart type de $2.81 \cdot 10^{-2}$. Pour le lissage par cohésion, cette différence moyenne est de $1.65 \cdot 10^{-4}$ avec un écart type de $1.88 \cdot 10^{-3}$

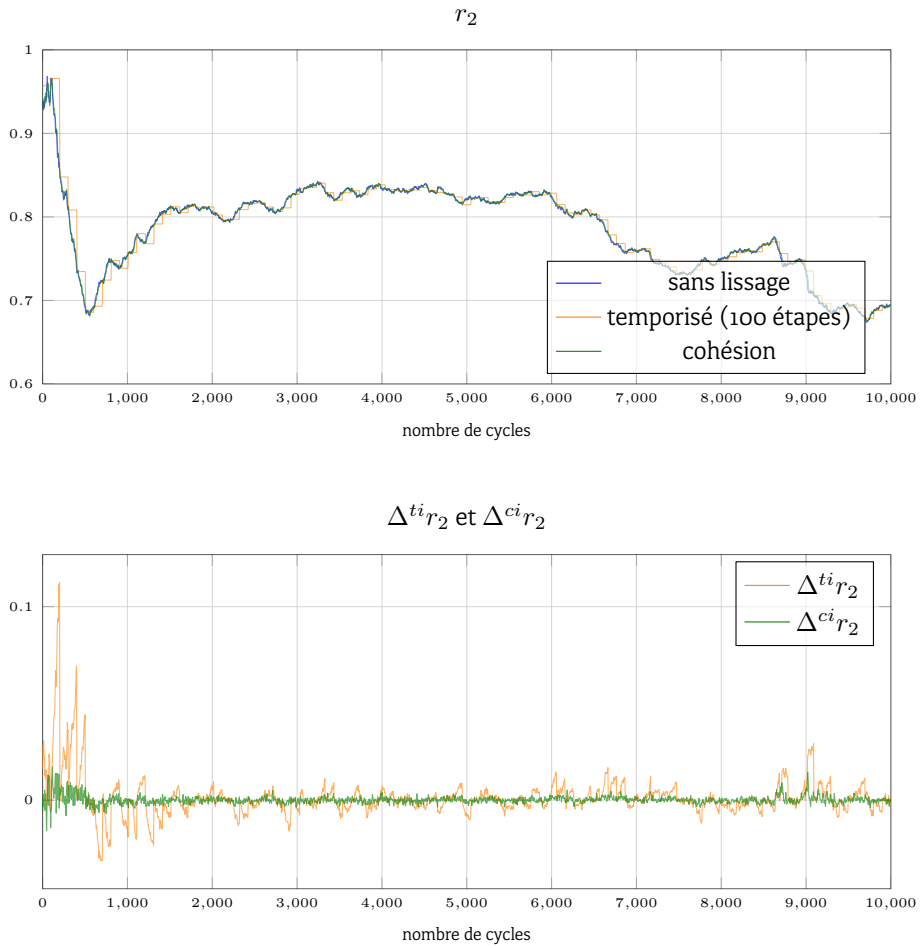


FIGURE 10.5 : Tracé de la mesure r_2 pour la simulation $\mathcal{S}_1$. La différence moyenne entre la mesure de r_2 pour le lissage temporisé et celle de la méthode sans lissage est $1.20 \cdot 10^{-3}$ avec un écart type de $1.05 \cdot 10^{-2}$. Pour le lissage par cohésion, cette différence moyenne est de $1.72 \cdot 10^{-4}$ avec un écart type de $2.07 \cdot 10^{-3}$

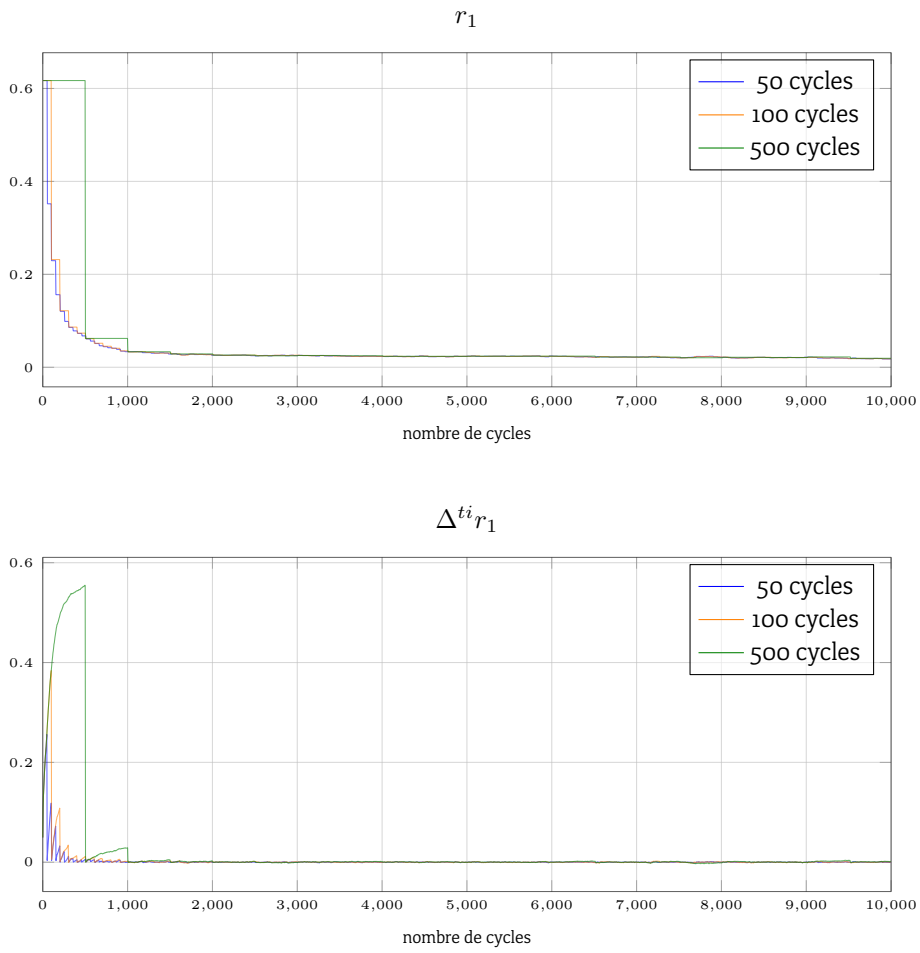


FIGURE 10.6 : Impact de la valeur du délai sur l'évolution de r_1 pour la simulation $\mathcal{S}_1$.

Synthèse pour r_1 et r_2

Cette première série de résultats nous permet d'observer l'impact de la méthode de lissage sur les mesures r_1 et r_2. Le lissage a pour but de réduire la quantité de migrations en tentant de supprimer les migrations qui sont le résultat de l'oscillation d'une entité entre plusieurs colonies. Les mesures r_1 et r_2 ne tenant pas compte des migrations, on ne s'attend pas ici à une amélioration du résultat de ces mesures mais l'on souhaite en revanche que le lissage ne les dégrade pas. D'après les observations que l'on peut faire sur les résultats, le lissage a un impact faible sur les mesures, ce qui nous permet de le considérer comme peu significatif.

Le lissage utilisant la cohésion est plus proche des résultats d'origine que la méthode temporisée. Bien que les différences soient peu significatives, on peut toutefois noter que

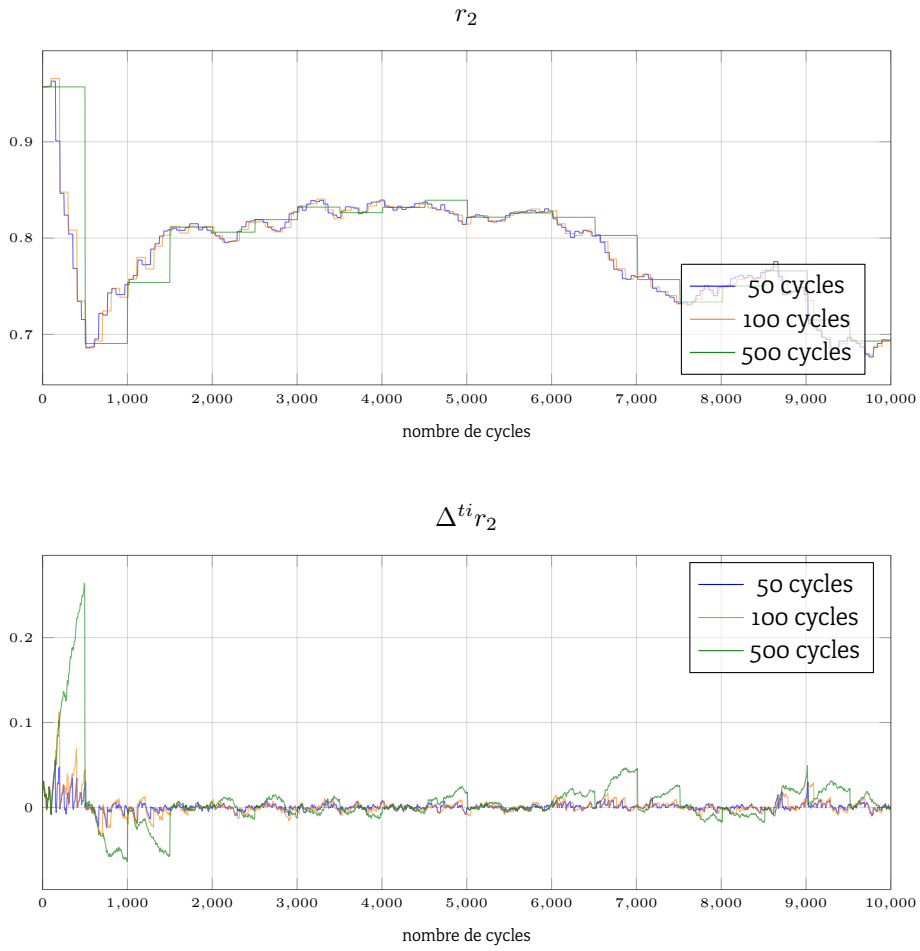


FIGURE 10.7 : Impact de la valeur du délai sur l'évolution de r_2 pour la simulation $\mathcal{S}_1$.

la mesure r_1 semble être la moins affectée par le lissage. D'après ces premiers résultats, l'objectif est atteint : l'ajout du lissage n'a pas dégradé la qualité de la répartition.

10.3.3 Intégration du nombre de migrations

L'objectif que nous cherchons à atteindre au travers des méthodes de lissage est la réduction de la quantité de migrations. En effet, une migration rend temporairement indisponible l'entité migrante et implique des communications effectives pour transporter l'entité d'une machine à une autre.

Nous allons observer désormais l'évolution d'une part du nombre total de migrations survenant pendant la simulation, et d'autre part de celui des communications effectives. Ces

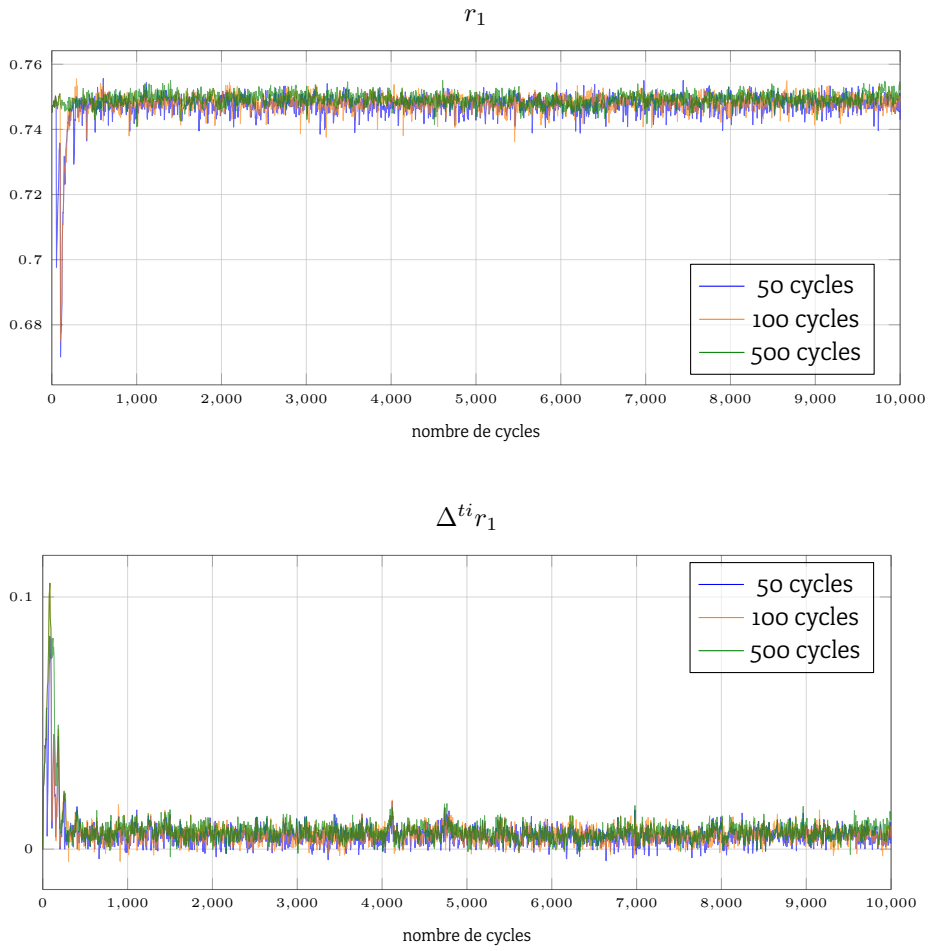


FIGURE 10.8 : Impact de la valeur du délai sur l'évolution de r_1 pour la simulation $\mathcal{S}_2$.

informations sont représentées sur la figure 10.12 pour la première simulation et sur la figure 10.13 pour la seconde.

Simulation $\mathcal{S}_1$

On constate que la méthode temporisée a permis de réduire de près de deux tiers la quantité totale de migrations pendant la simulation $\mathcal{S}_1$. On note toutefois une augmentation du nombre total de communications, mais cette augmentation a eu lieu durant la période d'instabilité des 1000 premiers cycles. Cette méthode a permis en moyenne de diminuer le nombre de migrations par cycle de 3.55, pour un écart type de 70.71. La quantité de communications a, quant à elle, augmenté en moyenne de 78.09 par cycle, pour un écart type

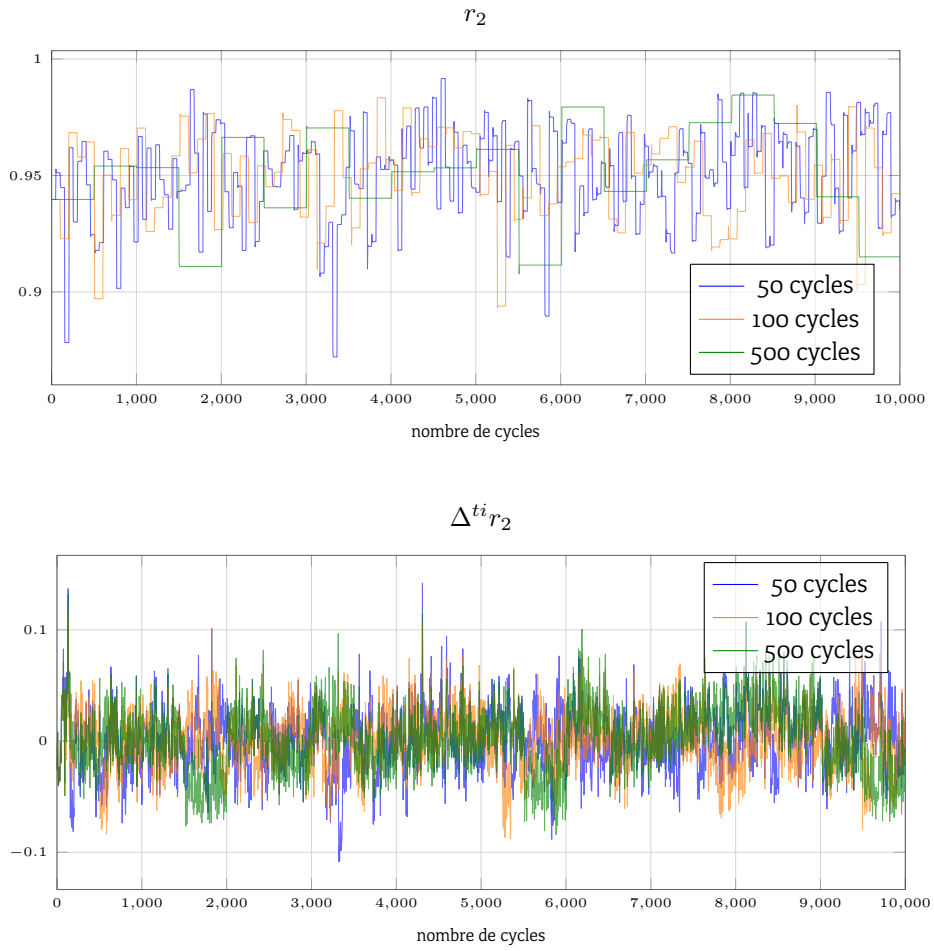


FIGURE 10.9 : Impact de la valeur du délai sur l'évolution de r_2 pour la simulation $\mathcal{S}_2$.

de 568.16.

La réduction du nombre de migrations entrainée par la méthode par cohésion est moins importante que celle de la méthode précédente mais a permis, tout de même, de réduire d'un peu moins d'un tiers la quantité de migrations pour une augmentation des communications faible. Le tracé de la quantité totale de communications montre que l'impact sur les communications est quasiment nul. On observe une réduction moyenne de 1.48 migrations par cycle, pour un écart type de 6.80 ; ainsi qu'une augmentation moyenne de 3.33 communications par cycle, pour un écart type de 38.00.

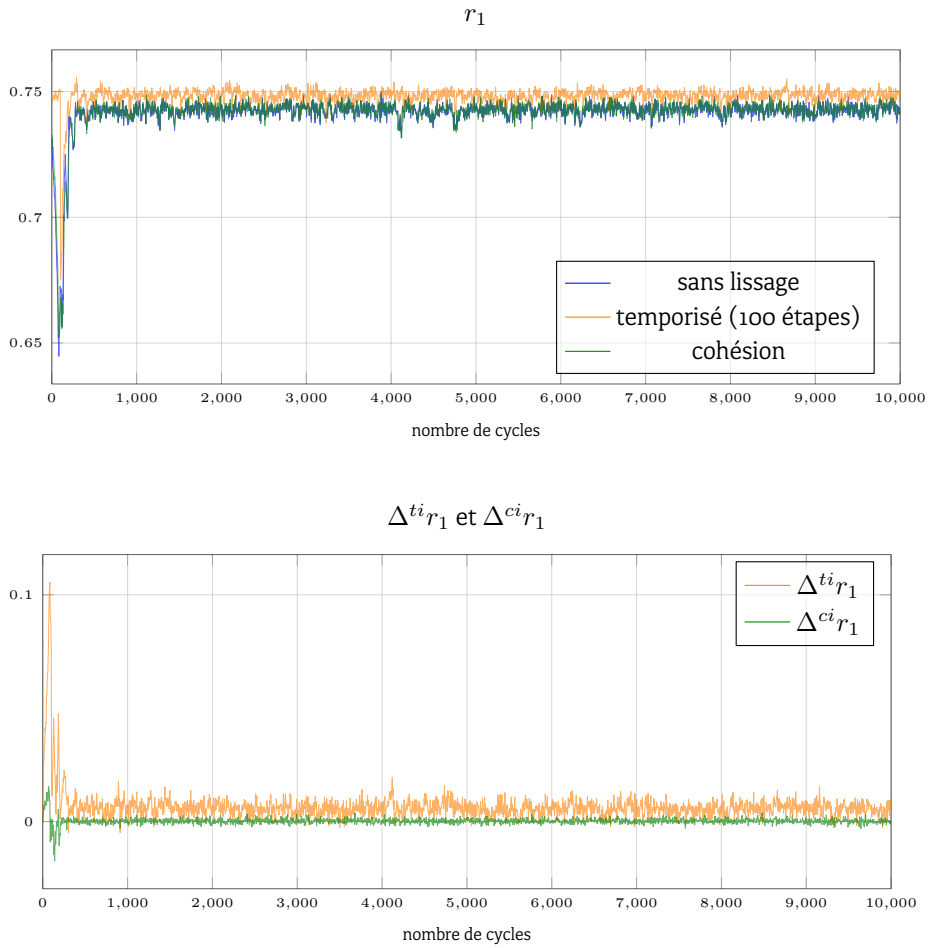


FIGURE 10.10 : Tracé de la mesure r_1 pour la simulation $\mathcal{S}_2$. La différence moyenne entre la mesure de r_2 pour le lissage temporisé et celle de la méthode sans lissage est $6.38 \cdot 10^{-3}$ avec un écart type de $7.08 \cdot 10^{-3}$. Pour le lissage par cohésion, cette différence moyenne est de $2.54 \cdot 10^{-4}$ avec un écart type de $1.51 \cdot 10^{-3}$

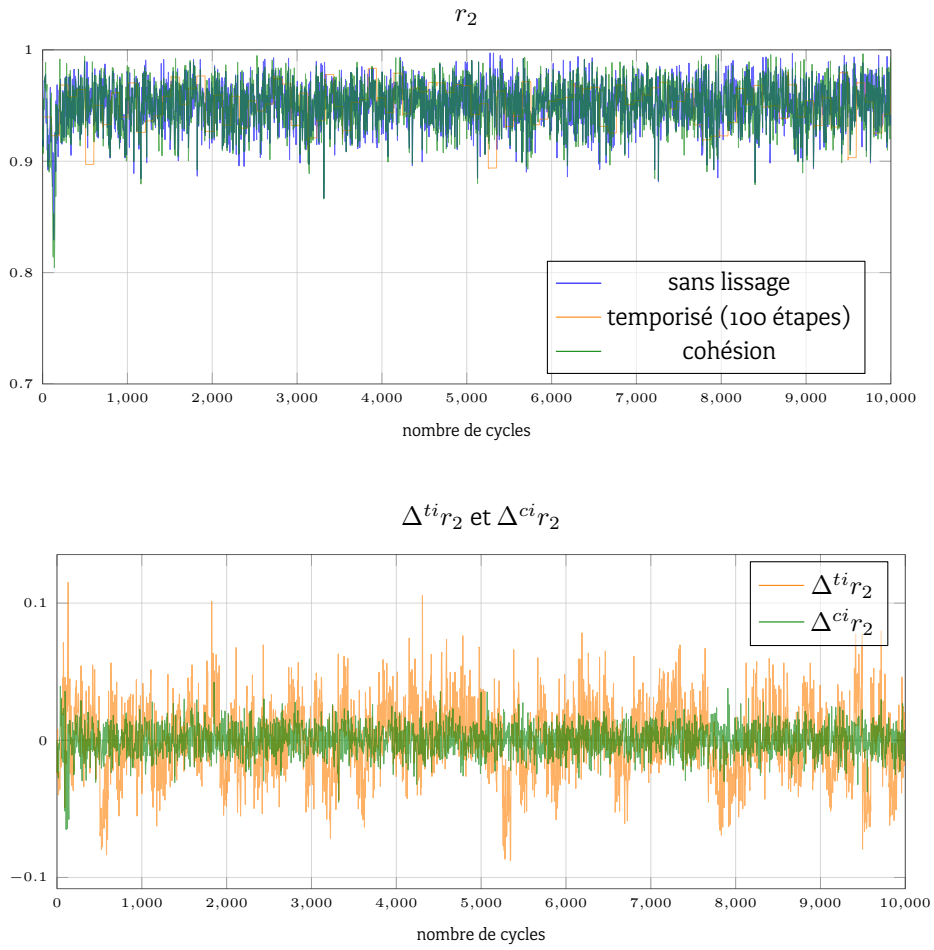


FIGURE 10.11 : Tracé de la mesure r_2 pour la simulation $\mathcal{S}_2$. La différence moyenne entre la mesure de r_2 pour le lissage temporisé et celle de la méthode sans lissage est $-5.21 \cdot 10^{-4}$ avec un écart type de $2.71 \cdot 10^{-2}$. Pour le lissage par cohésion, cette différence moyenne est de $-9.47 \cdot 10^{-5}$ avec un écart type de $1.08 \cdot 10^{-2}$

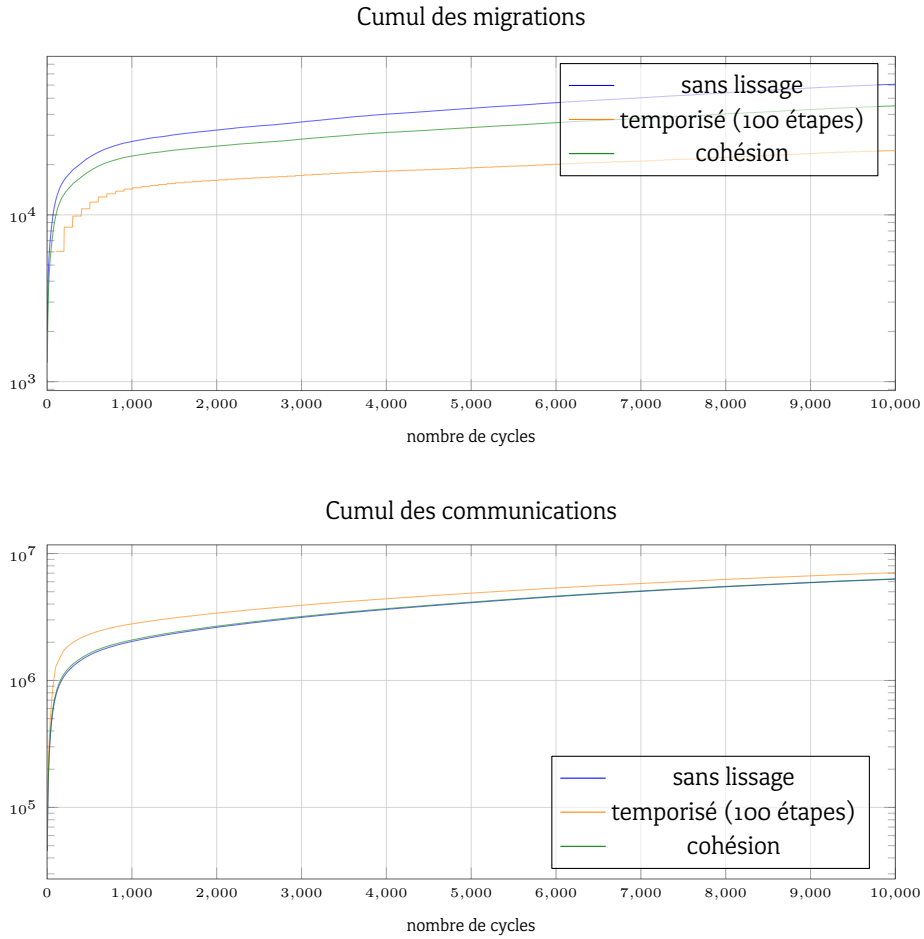


FIGURE 10.12 : Évolution du nombre total de communications effectives et de migrations pour la simulation $\mathcal{S}_1$ (échelle logarithmique)

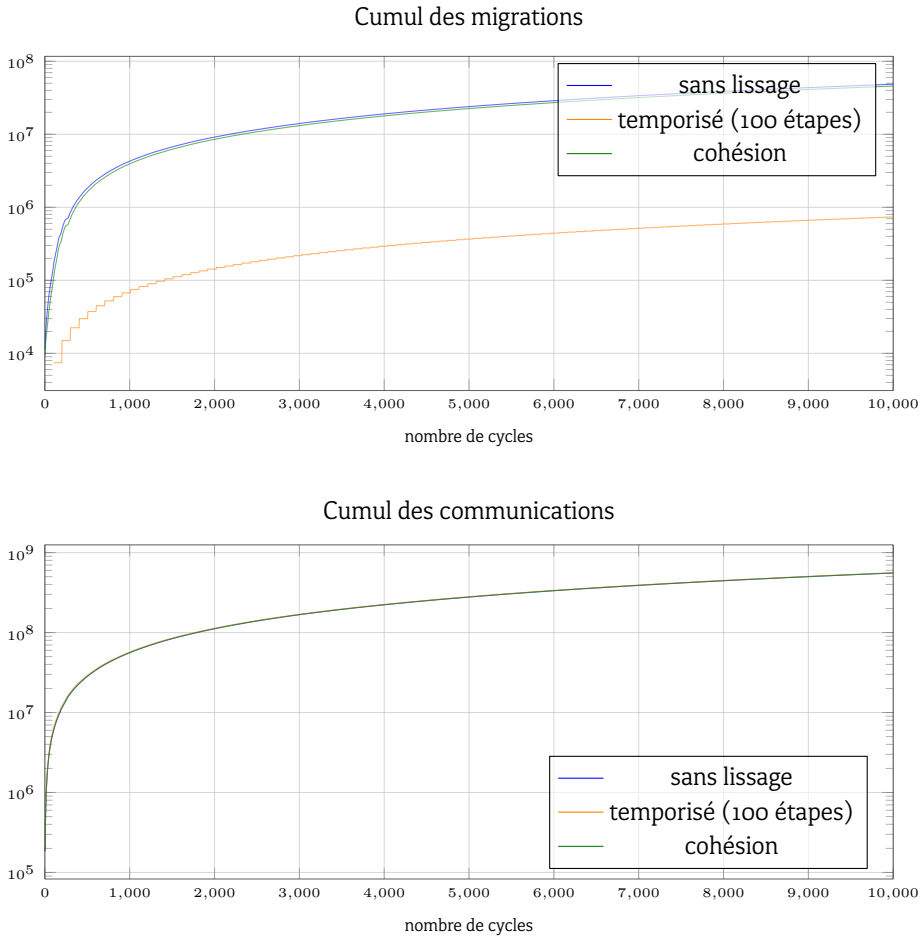


FIGURE 10.13 : Évolution du nombre total de communications effectives et de migrations pour la simulation $\mathcal{S}_2$ (échelle logarithmique)

Simulation $\mathcal{S}_2$

Les résultats des mesures effectuées sur la simulation $\mathcal{S}_2$ montrent une très forte réduction du nombre de migrations grâce au lissage temporisé, cette méthode ayant permis de réduire le nombre total de migrations d'un facteur 65.37. La quantité de migrations par cycle a été réduite de 4778.06, pour un écart type de 933.86. L'impact sur les communications se traduit par une augmentation moyenne de 483.23 communications par cycle, pour un écart type de 586.94.

Relativement aux résultats de la méthode temporisée, le lissage par cohésion n'a eu que peu d'impact sur le nombre total de migrations. On observe toutefois une réduction non-négligeable de 279.22 migrations par cycle, pour un écart type de 97.42. La quantité de communications, quant à elle, n'a augmenté en moyenne que de 19.56 par cycle, pour un écart type de 119.63.

Synthèse pour le nombre de migrations et de communications

On remarque ici que la méthode de lissage implique un compromis entre la réduction du nombre de migrations et l'augmentation des communications effectives. Ceci s'explique du fait que le lissage augmente la difficulté pour une entité de quitter une organisation pour une autre qui lui serait plus appropriée. Si la nouvelle organisation est plus appropriée, on peut supposer que l'entité interagit fortement avec les membres de cette nouvelle organisation, ce qui se traduit par des communications effectives tant que l'entité n'aura pas rejoint le groupe.

Pour évaluer ce compromis, on considère qu'une migration possède un coût c_m et qu'une communication possède un coût c_c. Comme nous connaissons le nombre total de migrations n_m et celui de communications n_c, nous pouvons calculer le coût total C de la simulation sous la forme :

$$C = n_c \times c_c + n_m \times c_m$$

Une migration impliquant au minimum une communication, on peut exprimer c_m en fonction de c_c tel que $c_m = k \times c_c$ pour $k \geq 1$. On obtient alors :

$$C = c_c \times (n_c + k \times n_m)$$

En se basant sur une valeur de référence $c_c = 1$, nous pouvons faire varier la valeur de k et observer les implications sur C. Les figures 10.14 et 10.15 illustrent ces variations pour les différents modes de lissage.

On peut alors observer que dans la cas de la simulation $\mathcal{S}_1$ le choix de la méthode de lissage la mieux adaptée est dépendant de k. Lorsque $k < 20$, on constate que le lissage temporisé a un apport négatif sur le coût global de la simulation. Passé ce seuil, plus la valeur de k augmente, plus le lissage temporisé permet d'améliorer de façon significative le coût. Le lissage par cohésion améliore le coût global, même pour des faibles valeurs de k. Passé le seuil de $k = 35$, cette méthode est moins performante que le lissage temporisé.

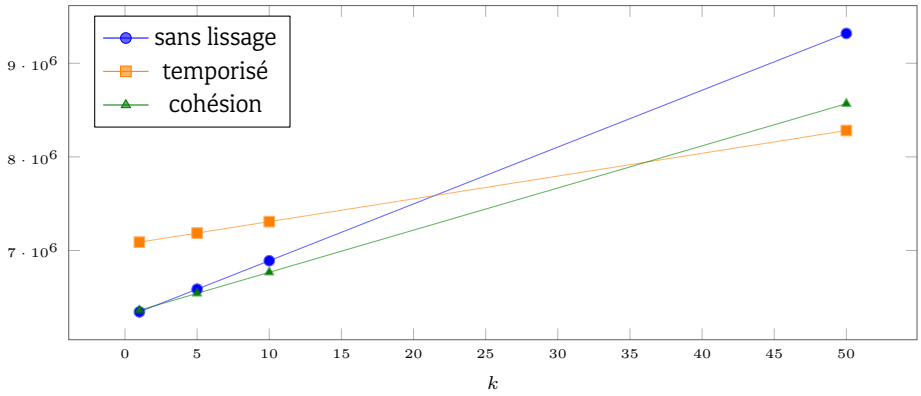


FIGURE 10.14 : Évolution de C en fonction de k pour la simulation $\mathcal{S}_1$

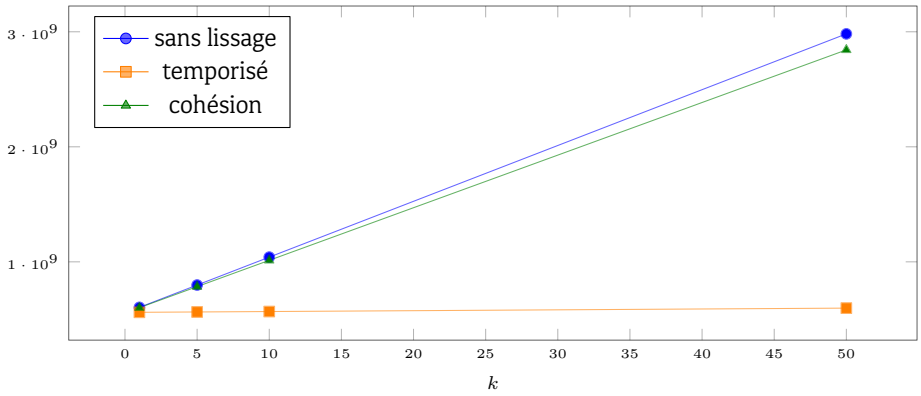


FIGURE 10.15 : Évolution de C en fonction de k pour la simulation $\mathcal{S}_2$

Contrairement à la simulation $\mathcal{S}_1$ où le choix de la méthode de lissage est dépendant de k, l'observation du coût pour $\mathcal{S}_2$ ne laisse aucune ambiguïté : le lissage temporisé est avantageux peu importe la valeur de k. Le lissage par cohésion apporte tout de même une amélioration du coût qui est cependant sans commune mesure avec l'amélioration apportée par la temporisation.

10.3.4 Synthèse générale sur les résultats

Nous avons présenté des résultats sur des méthodes de lissage des résultats de AntCo^2. Le but de ce lissage est de réduire la quantité de migrations sans détériorer la qualité de la répartition. Nous avons donc dans un premier temps observé l'impact du lissage sur les mesures r_1 et r_2 qui permettent d'évaluer respectivement la charge du réseau et la charge de calcul. Puis nous avons analysé l'évolution de la quantité totale de migrations ainsi que

de celle des communications entre machines.

Pour cela, nous avons considéré deux simulations : une première, $\mathcal{S}_1$, sur un graphe statique dont la topologie est une grille ; et une seconde, $\mathcal{S}_2$, sur un graphe dynamique généré grâce au modèle des boids (cf. l'annexe A). D'autres expérimentations devront être réalisées sur des topologies de graphes différentes afin d'affiner ces résultats.

L'étude de r_1 et r_2 nous a permis d'observer que le lissage a un faible impact sur la qualité de la solution. Cet impact est fluctuant, et peut en général être positif ou négatif. On distingue tout de même le cas de la mesure r_1 dans la simulation $\mathcal{S}_2$ pour le lissage temporisé pour laquelle l'impact est ici négatif, bien que dans une moindre mesure.

L'étude du nombre de migrations et de communications permet de démarquer la méthode de lissage temporisé, en particulier dans le cas de la simulation $\mathcal{S}_2$ qui correspond au cas de figure le plus proche de la pratique. La méthode par cohésion remplit tout de même notre objectif d'améliorer la répartition en réduisant la quantité de migrations. Cependant, le lissage temporisé possède l'avantage de la simplicité de sa mise en pratique.

CHAPITRE 11

INTERGICIEL

Nous avons vu jusqu'à présent que nous pouvons modéliser l'application que l'on souhaite distribuer sous la forme d'un graphe dynamique. Puis, nous avons présenté comment, en utilisant un algorithme dynamique de détection d'organisations, nous pouvions fournir les informations nécessaires à la répartition des entités de l'application sur l'ensemble des machines à disposition de manière à équilibrer la charge de calcul et à réduire la charge du réseau.

Afin de parvenir à une distribution de l'application, nous avons besoin de fournir à cette application un environnement qui va permettre de gérer la répartition effective des entités sur l'ensemble des machines. Cet environnement, que nous avons décrit dans la partie 5.1.4, prend la forme d'une couche qui va venir se placer entre l'application et l'ensemble des

machines. Nous présentons ici notre plateforme de distribution qui a été développée durant cette thèse, qui permet d'aborder la distribution de manière dynamique et décentralisée. Elle s'appuie sur le modèle d'acteur que nous décrivons dans la partie 11.2.

11.1 Vue d'ensemble

L'intergiciel est une couche qui va venir se placer entre les ressources de calcul et l'application que l'on souhaite distribuer. Il va permettre à l'application de s'abstraire de la gestion des ressources de l'une des façons suivantes :

1. soit en rendant la distribution complètement opaque pour l'application qui n'a alors pas connaissance de sa distribution ;
2. soit en distribuant de manière transparente, de ce fait l'application a alors connaissance de sa distribution et peut éventuellement agir dessus ; l'intergiciel permet dans ce cas de fournir la gestion et les fonctionnalités de base pour permettre cette distribution.

Nous voyons l'intergiciel comme faisant partie de l'écosystème computationnel dans lequel va évoluer l'application et qui va lui permettre d'être distribuée. Une particularité de l'ensemble des machines qui sont à disposition est sa dynamique : de nouvelles machines peuvent être ajoutées au cours de la distribution tandis que certaines peuvent être retirées. Le retrait de machines peut être d'une part intentionnel, c'est à dire que nous avons un certain contrôle sur ce retrait, mais il peut être aussi inopiné, suite à une défaillance par exemple. Ces ajouts et suppressions de machines peuvent aussi être liés à des perturbations de la connexion comme ça peut être le cas de connexion sans fil pour des machines mobiles. L'objectif de l'intergiciel vis-à-vis de cette dynamique est d'être capable de s'adapter aux différents cas de figure, particulièrement les cas liés à l'imprévisible, et cela afin de garantir la robustesse du système.

Le rôle de l'intergiciel est de gérer les migrations des entités sur les machines disponibles en utilisant les informations fournies par l'algorithme de répartition de charges. Il doit aussi fournir l'outillage nécessaire pour permettre aux entités d'interagir entre elles en particulier lorsque ces entités sont hébergées sur des machines différentes. Cette partie de l'écosystème computationnel doit à son tour subir les contraintes liées aux simulations que l'on souhaite distribuer c'est à dire une ouverture du système, une quantité d'entités et d'interactions entre ces entités suffisamment importante pour empêcher l'utilisation de structures de données globales.

11.2 L'acteur, un objet actif

Du fait du nombre important d'entités que la plateforme doit être capable de supporter, et de la quantité d'interactions entre ces entités, nous avons besoin d'un modèle qui soit adapté à ces contraintes. Des communications directes entres les entités ne sont pas envisageables

du fait qu'il en résulterait un blocage systématique, voire définitif, de l'application. Nous optons donc pour des communications indirectes prenant la forme de requêtes envoyées par une entité à une autre. Nous utilisons, de ce fait, le concept d'*objet actif*, aussi nommé *acteur*, dont l'architecture a été initialement définie par C. HEWITT, P. BISHOP et R. STEIGER dans [HEWITT, BISHOP et STEIGER 1973].

Un objet actif se distingue d'un objet classique par le fait qu'il possède son propre fil d'exécution qui lui permet une gestion désynchronisée des appels aux méthodes. La figure 11.1 illustre cette différence. Dans le cas d'objets classiques, A doit attendre que B termine l'exécution de la méthode avant de pouvoir effectuer d'autres tâches. A est alors dans un état bloquant tant que B n'a pas terminé son exécution. Au contraire, dans le cas d'objets actifs, l'appel d'une méthode prend la forme d'une requête qui est envoyée à la cible et dont on peut traiter le résultat que plus tard, lorsqu'il sera disponible. A peut alors effectuer d'autres tâches en attendant que l'exécution de B soit terminée.

11.2.1 Architecture

Un acteur est formé de différents composants. Tout d'abord, le noyau même de l'acteur qui est le moteur étant capable d'exécuter les requêtes. Le noyau est composé d'un thread, d'une liste de requêtes à exécuter, et d'un objet classique qui contient les méthodes disponibles. Les requêtes sont exécutées à l'intérieur du thread selon une politique FIFO [1] qui peut cependant être personnalisée pour répondre aux besoins de l'application. On peut par exemple définir des requêtes prioritaires qui seraient exécutées en premier lieu.

Le second composant d'un objet actif est le *proxy* qui permet de recevoir les requêtes et donc d'alimenter la liste du noyau. Les requêtes peuvent être envoyées au proxy au travers d'un réseau ou d'une boucle locale ce qui permet de distribuer les objets actifs sur un ensemble de machines. L'envoi des requêtes est réalisé grâce à un objet *talon* qui symbolise l'objet que l'on souhaite contacter, ce qui permet de le manipuler de manière transparente.

Afin de gérer l'asynchronisme des appels aux méthodes, l'objet talon procède à deux opérations :

1. il envoie une requête, décrivant la méthode invoquée, au proxy de l'objet actif qu'il représente ;
2. il retourne directement à l'objet appelant un *futur* qui représente le résultat de la requête qui sera transmis par la suite.

Le futur permet de fournir les fonctionnalités nécessaires pour connaître l'état de la requête. Ainsi l'objet appelant peut savoir, grâce au futur qui lui a été retourné, si le résultat est disponible, ou encore entrer dans un état bloquant dont il ne pourra sortir qu'une fois le résultat mis à disposition.

La figure 11.2 résume l'ensemble de l'architecture d'un acteur.

1. First In First Out

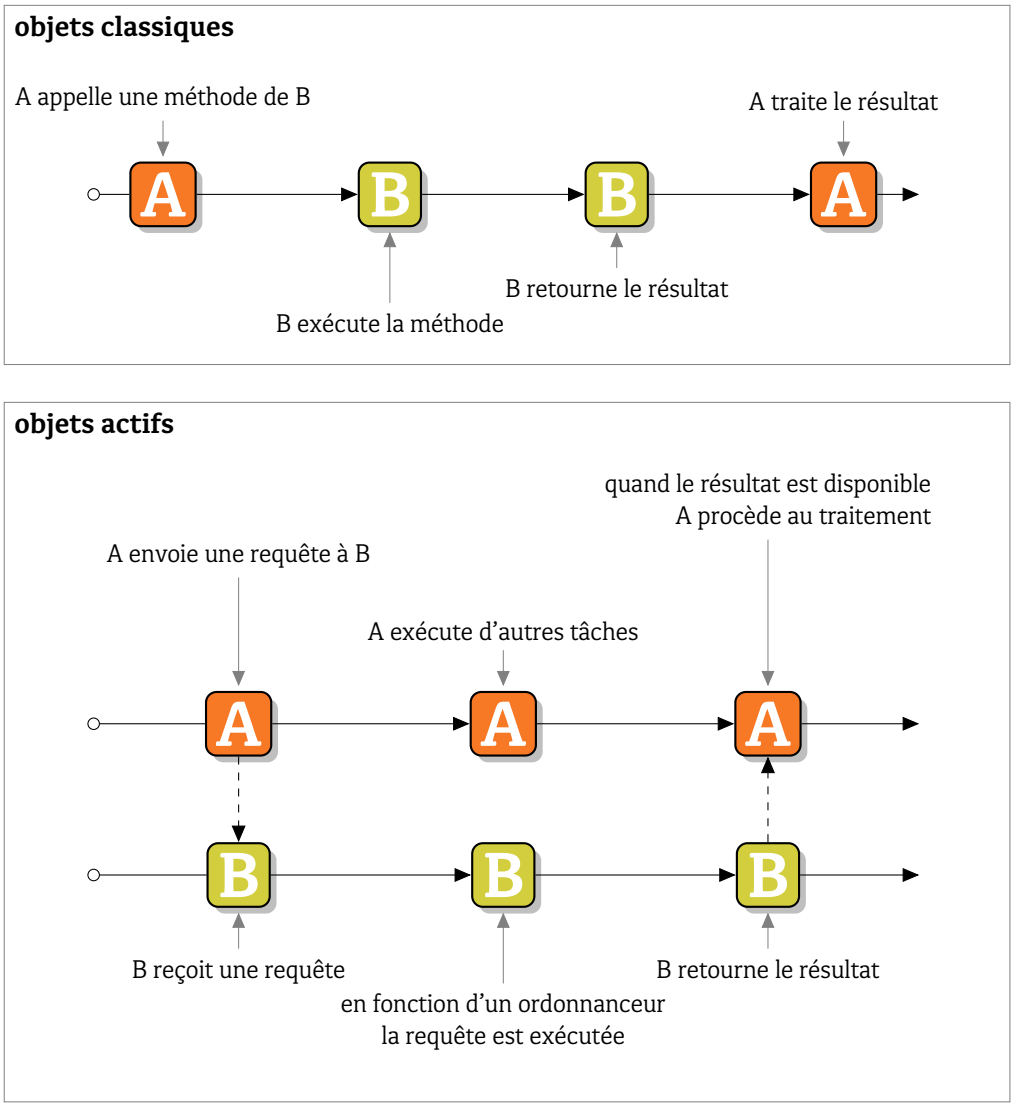


FIGURE 11.1 : Appel de méthode entre objets classiques et objets actifs. Un objet A appelle une méthode d'un objet B. B exécute alors cette méthode et retourne un résultat à A qui effectue alors un traitement.

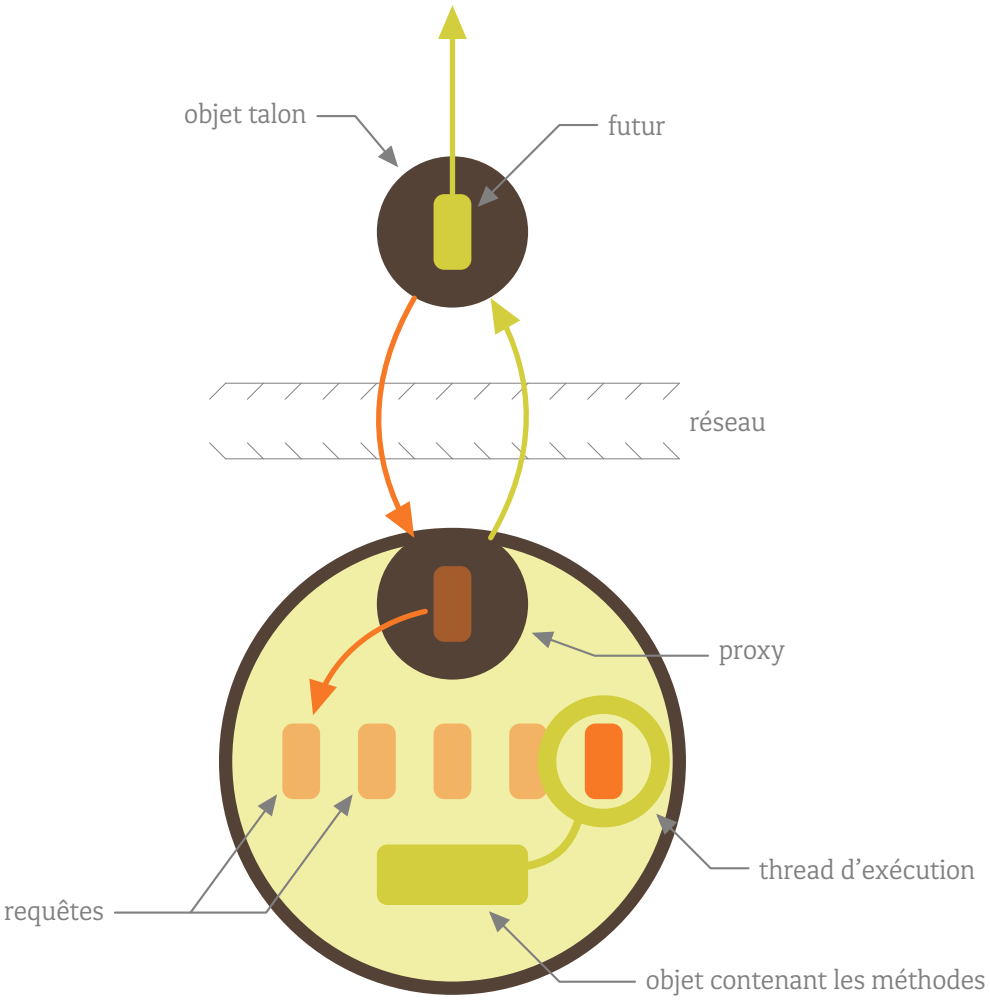


FIGURE 11.2 : Architecture d'un acteur.

11.2.2 Le paradigme acteur

Dans [AGHA et al. 1997], Gul AGHA *et al.* résument les principes fondamentaux qui servent de base au paradigme acteur :

1. tout objet est un acteur, les auteurs incluent aussi les messages et les nombres dans cette propriété ; bien que nous considérons nous aussi un système où tout est acteur, comme nous le verrons dans 11.3.2, certains objets restent des objets classiques (en particulier les types primitifs comme les entiers et les réels) pour des raisons de performances ainsi que pour simplifier la plateforme de distribution ;
2. on ne peut agir sur un acteur, l'observer, ou le modifier autrement que par l'envoi d'une requête à cet acteur, qu'il exécutera lui-même ;
3. les seules choses qui se produisent dans un système composé d'acteurs sont des évènements, qui consistent à l'envoi/réception d'un message entre acteurs ;
4. un acteur peut envoyer plusieurs messages au même moment mais les reçoit de manière linéaire et ne les traite donc qu'un par un ;
5. chaque acteur a connaissance d'un ensemble d'autres acteurs ; cet ensemble peut évoluer au fil du temps grâce à l'échange de messages qui permet de rencontrer indirectement de nouveaux acteurs.

Il ressort de ces principes que la transmission des requêtes est au cœur d'un système composé d'acteurs.

11.2.3 Gestion des différents mécanismes

Nous avons vu l'architecture générale d'un acteur. Dans la pratique, il est nécessaire de gérer différents problèmes liés à ce modèle, ainsi qu'à la décentralisation de la plateforme. En particulier, sur l'identification des objets, le référencement des futurs, ainsi que sur le mécanisme de migration d'un objet.

Gestion des objets

La plateforme de distribution est constituée d'un nombre important d'acteurs qui sont pour certains des composants de l'application distribuée, et pour d'autres des composants de la plateforme elle-même. Afin de contacter chacun de ces acteurs, pour leur transmettre des requêtes, il est nécessaire d'être en mesure de les identifier de manière unique.

L'identification d'un acteur est faite grâce à une chaîne de caractères correspondant à une URI [2] telle que définie dans [MASINTER, BERNERS-LEE et FIELDING 2005]. L'identifiant a alors la forme suivante :

```
protocole://hote/chemin/id
```

2. Uniform Resource Identifier

Cet identifiant peut être divisé en deux parties. D'une part, une partie fixe composée du chemin et de l'id qui sont fixés définitivement à la création de l'acteur. Et d'autre part, une partie variable, composée du protocole et de l'hôte, qui elle varie en fonction de la machine qui héberge l'objet actif. L'*id* est une chaîne de caractères unique pour l'ensemble des machines. Elle est créée à partir de l'adresse de la machine et d'une estampille temporelle afin d'assurer cette unicité. Le *chemin* permet de regrouper des objets entre eux lorsque ceux-ci partagent une même fonction par exemple, ou sont du même type. Enfin, le *protocole* définit la méthode à utiliser afin de communiquer avec le proxy de l'objet actif désigné. La seconde partie peut être omise, dans ce cas on considère que l'objet actif cible se situe sur la même machine que la source.

Chaque acteur possède un certain nombre de méthodes qui peuvent être appelées par d'autres acteurs. Le nom de la fonction qu'un acteur source souhaite appeler est alors ajouté à l'identifiant de la cible pour former une requête :

```
protocole://hote/chemin/id?callable=nom
```

Gestion des futurs

Les futurs sont aussi des composants qui ont besoin d'être identifiés de manière unique. La portée de cette unicité peut toutefois être limitée à celle de la machine qui héberge l'acteur à la source de la requête. La référence du futur peut alors être ajoutée à la requête lorsque l'on en attend un résultat :

```
protocole://hote/chemin/id?callable=nom&future=id_futur
```

Le futur est alors référencé sur la machine source en attendant le résultat de l'acteur cible. Celui-ci, une fois l'exécution de la requête terminée, peut retourner le résultat, accompagné de la référence du futur, à l'acteur source.

Gestion des migrations

La migration consiste à déplacer un acteur de la machine qui l'héberge à une autre. Il est donc nécessaire pour cela de transférer la liste des requêtes courantes ainsi que l'objet utilisé par l'acteur pour exécuter les requêtes.

Le protocole de migration consiste à :

1. demander l'accord de migration à la machine de destination ;
2. transférer les données de l'acteur si l'autorisation est accordée ;
3. reprendre l'exécution des requêtes depuis le nouvel hôte.

Lorsqu'une demande de migration est effectuée, l'acteur entre dans un état de transition lorsque l'exécution de la requête courante est terminée. La demande d'hébergement est alors envoyée à la machine de destination. Celle-ci donne alors son autorisation pour continuer la migration, ou au contraire, refuse que la migration soit effectuée si, par exemple, elle ne

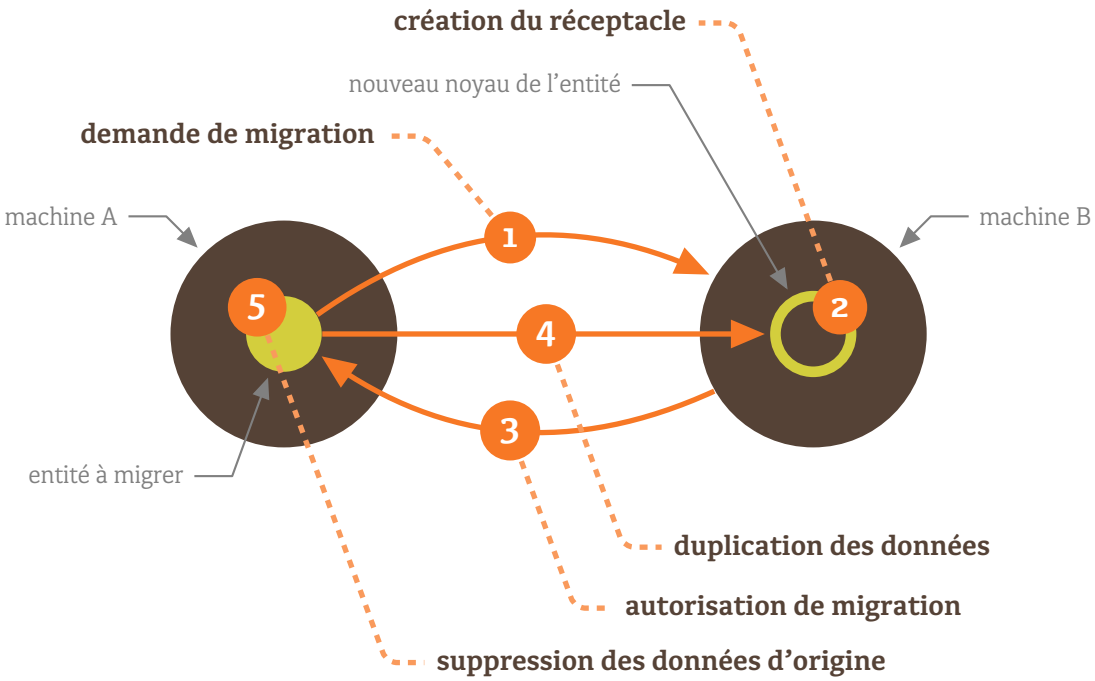


FIGURE 11.3 : Protocole de migration

possède pas les ressources nécessaires pour accueillir le nouvel acteur. Si la migration est refusée, l'acteur retourne à l'exécution de ses requêtes : la migration est annulée. Si la machine accepte, elle initialise une nouvelle liste de requêtes qui servira à accueillir les requêtes reçues pendant la migration. L'acteur duplique ensuite ses requêtes vers cette nouvelle liste ainsi que les autres données qui lui sont associées. Si aucune erreur n'est rencontrée pendant la migration, l'acteur peut alors continuer son exécution sur son nouvel hôte, et les anciennes données peuvent être supprimées.

Gestion des threads

L'architecture d'un acteur implique que chaque acteur possède son propre fil d'exécution. En pratique, et particulièrement dans notre cas où nous manipulons un nombre important d'acteurs, il n'est pas possible d'affecter à chaque acteur un thread dédié. La plateforme de distribution que nous proposons est prévue pour accueillir des machines « standards ». Ces machines permettent à ce jour d'exécuter 4 à 8 threads [3] simultanément dans des conditions optimales. Nous serions donc fortement limités sur la quantité d'acteurs qui pourraient s'exécuter sur chaque agence.

3. en considérant 2 à 4 cœurs hyperthreadés par machine

C'est pourquoi nous définissons un ensemble de threads qui sera partagé par l'ensemble des acteurs. Afin d'exécuter une requête, un acteur doit donc attendre qu'un thread soit disponible, puis prendre la main sur ce thread. L'acteur exécute sa requête puis rend la main qu'il a prise sur le thread.

11.3 Une architecture décentralisée

Le modèle d'intergiciel que nous proposons repose sur une architecture *décentralisée* dans le sens où la hiérarchie entre les machines est inexistante. Chaque machine participe donc à la formation de l'écosystème computationnel au même titre que les autres. L'objectif est d'augmenter la robustesse du système. En effet, lorsqu'un système est centralisé, il devient dépendant de la partie centrale, et si cette dernière venait à être inaccessible, le système entier en deviendrait figé.

11.3.1 Agence

L'agence est la réification locale de la plateforme distribuée sur chacune des machines y participant. Une machine qui souhaite rejoindre la distribution doit donc lancer sa propre agence qui va ensuite se connecter au réseau d'agences.

Une agence est un acteur particulier qui a la possibilité d'en créer d'autres. Elle permet de fournir les fonctionnalités de base de la plateforme.

11.3.2 Le tout acteur

Chaque composant de la plateforme de distribution est un acteur. Ainsi on trouve l'acteur principal qui est l'agence, puis les fonctionnalités de cette agence, les entités, *etc*... De même, les protocoles de communication, qui permettent l'échange de requêtes entre agences, sont aussi des acteurs. On peut classer ces acteurs en plusieurs catégories :

l'agence elle est unique et nécessaire pour une instance locale de la plateforme ;

les fonctionnalités acteurs modulaires qui sont chargés par l'agence pour étendre ses fonctionnalités ; ils n'ont pas la possibilité de migrer ;

les protocoles ce sont les acteurs dédiés à l'échange de requêtes entre les acteurs situés sur des agences différentes ;

les entités il s'agit des acteurs qui vont composer l'application distribuée ; ils ont la possibilité de migrer d'une agence à une autre.

11.4 Services des agences

11.4.1 Découverte d'autres agences

La plateforme distribuée repose sur une architecture décentralisée, formée par l'interconnexion d'agences. De ce fait, elle ne possède pas d'annuaire centralisé auquel une agence pourrait se connecter pour connaître la liste des agences actuellement connectées à la plateforme. Une agence a cependant besoin d'informations afin de pouvoir se connecter à d'autres, c'est pourquoi il est nécessaire d'introduire une fonctionnalité permettant à l'agence d'obtenir ces informations, sans avoir recours à une structure centrale.

La fonctionnalité « Discovery » utilise une diffusion *multicast* sur le protocole IPv6, telle que définie dans [HINDEN et DEERING 2006], afin de transmettre les informations relatives. Le multicast est une technique de routage de paquets ip permettant de transmettre les paquets à un ensemble de machines plutôt qu'à une machine en particulier (*unicast*). Contrairement au *broadcast* qui diffuse les paquets à toutes les machines sans distinction, le multicast permet de créer une adresse ip virtuelle qui est partagée par les machines qui souhaitent recevoir ces informations. Ces trois modes de diffusion de paquets ip sont illustrés par la figure 11.4. La diffusion par multicast est particulièrement intéressante par exemple dans le cas de diffusion d'un flux vidéo : plutôt que de maintenir autant de connexions que d'utilisateurs, on peut alors créer une connexion unique qui diffusera la vidéo à tous les utilisateurs qui le souhaitent. Le multicast sur le protocole IPv6 est particulièrement intéressant (comparé à celui sur IPv4) du fait qu'il permet un contrôle plus fin sur la diffusion en permettant, par exemple, de spécifier la portée qu'auront les paquets diffusés.

On définit une adresse de multicast qui est commune à toutes les agences. Chaque module Discovery diffuse donc les informations de base propres à son agence en utilisant ce protocole. Ces informations incluent l'adresse réseau de l'agence, les protocoles qu'elle est capable de manipuler, ainsi qu'une information de hachage symbolisant l'état actuel de l'agence. De plus, Discovery est à l'écoute d'éventuels messages en provenance d'autres agences. Grâce aux informations contenues dans les messages reçus, il est alors possible de tenir un annuaire local des agences disponibles. Le hachage permet de déterminer si des modifications significatives sont survenues dans l'agence, qui nécessiteraient de contacter cette agence afin de mettre à jour des informations. Par exemple, si l'on souhaite suivre l'évolution des entités présentes sur une agence tout en réduisant la quantité de requêtes envoyées à cette agence, il suffit de redemander la liste des entités uniquement lorsque le hachage est modifié.

11.4.2 Lucioles et horloge globale décentralisée

La décentralisation implique l'absence de contrôle central et global sur les entités. Par conséquent, si l'on souhaite créer une application dans laquelle une même étape algorithmique est exécutée simultanément pour l'ensemble des entités, alors la présence d'une horloge globale, permettant aux agences de se synchroniser, est une nécessité. Dans le cas d'un

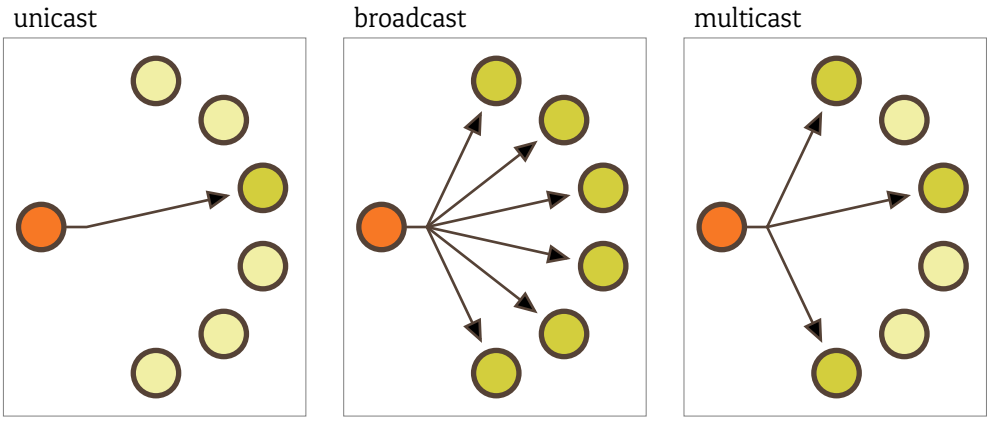


FIGURE 11.4 : Différences entre unicast, broadcast et multicast. Une source diffuse de l'information aux cibles.

système centralisé, un « maître », la partie centrale, peut être utilisée afin de fournir ce synchronisme. Or notre système est décentralisé, car nous souhaitons éviter par exemple, que l'indisponibilité du maître fournissant le synchronisme paralyse l'application entière.

À nouveau, la nature regorge de mécanismes qui sont des sources d'inspiration pour nous aider à trouver des solutions aux problèmes que nous rencontrons. En l'occurrence, l'inspiration est puisée dans la lumière émise par les lucioles, et plus particulièrement dans le mécanisme de synchronisation qui apparaît chez certaines espèces. Les mâles produisent des flashs pour attirer les femelles, et on peut observer la synchronisation des flashs et de ce fait l'émergence d'un rythme global. La photographie présentée dans la figure 11.5 est une longue exposition prise à l'est des États-Unis où l'on peut observer ces lucioles. Ce phénomène est le sujet de différents ouvrages scientifique : John BUCK en fait un présentation détaillé dans [BUCK 1988], il est la source d'inspiration de Steven H. STROGATZ dans un chapitre de son livre [STROGATZ 2003], mais on le retrouve aussi plus récemment dans [LEWIS et CRATSLEY 2008] par Sara M. LEWIS et Christopher K. CRATSLEY.

Nous nous sommes inspirés de ce phénomène afin de simuler une synchronisation globale qui ne nécessite pas de connaissance globale. La fonctionnalité « Luciole » permet de répéter une même étape algorithmique en tentant de se synchroniser avec les autres lucioles connues. Chaque luciole exécute dans un premier temps le code qui lui est associé. Il s'agit d'envoyer une requête aux acteurs liés à la luciole et d'attendre que la requête ait été exécutée. La luciole émet ensuite un signal vers les lucioles dont elle a connaissance, c'est à dire qu'elle envoie une requête dont le but est d'indiquer qu'elle a terminé son exécution et qu'elle se prépare pour un nouveau cycle. Elle indique dans cette requête le temps t qui lui a été nécessaire afin de réaliser un cycle. Elle tente ensuite, pendant un temps maximal Δt de recevoir les signaux de ses voisines. Δt est égal à la différence entre le temps maximum

FIGURE 11.5 : Les lucioles « synchrones » à Elkmont dans le parc national des Great Smoky Mountains

t_M mis par les lucioles voisines, et t :

$$\Delta t = \begin{cases} \delta & \text{si } t > t_M \\ t_M - t + \delta & \text{sinon} \end{cases}$$

La constante δ définit un délai minimal qui permet d'absorber d'éventuelles fluctuations dans le temps nécessaire à l'exécution d'un cycle. Lorsque tous les signaux ont été reçus, ou si le délai est écoulé, la luciole met à jour la valeur de t_M en fonction des données reçues puis redémarre un nouveau cycle. La présence d'un temps d'expiration permet de ne pas bloquer définitivement le système dans l'attente d'une hypothétique réponse : la luciole tente d'attendre ses voisines, mais continue son chemin en cas d'absence de réponse.

Nous avons développé un prototype permettant de tester ce concept. Pour cela, nous simulons des lucioles de façon à ce que chaque luciole possède son temps d'exécution, qui va évoluer au fur et à mesure de la simulation. Le comportement d'une luciole de ce prototype est donné par l'algorithme 8.

Afin de mesurer le synchronisme du réseau de lucioles, nous utilisons la mesure $\psi(v)$, définie par l'équation 11.1, qui permet de mesurer le décalage du début de l'exécution d'une luciole v avec ses voisines. La fonction $\omega(v)$ permet de définir la dernière date d'exécution de la luciole v. On peut alors observer la synchronisation globale en analysant la valeur moyenne $\overline{\psi}$ de $\psi(v)$ pour l'ensemble $\mathcal{L}$ des lucioles (cf. équation 11.2). La figure 11.6 présente le tracé de cette mesure. On y voit que suite à une période de désynchronisation correspondant aux 80 premières secondes, le mesure atteint une valeur très proche de 0 ce qui nous permet de considérer que les lucioles se sont synchronisées. On observe ensuite une désynchronisation

Algorithme 8 : Cycle d'une luciole

```
1  début
2      tant que en vie faire
           /* La fonction random() retourne une valeur dans [−1;1]. Le
              paramètre δ_delai définit l'évolution maximale du délai par
              cycle */
3          délai ← délai + random() × δ_delai ;
4          flash(max) ;
           /* Nous demandons simplement à la luciole de faire une pause
              dans son exécution ce qui permet de simuler le temps qui
              serait nécessaire dans la pratique pour effectuer le
              traitement des actions */
5          dormir(délai) ;
6          max ← délai ;
           /* La fonction horloge() donne la date courante */
7          d ← horloge() + (max − délai) + δ ;
           /* Le nombre de messages attendus dépend de la topologie du
              réseau d'interconnexion des lucioles. Pour simuler ce
              réseau, nous générons un graphe dans lequel chaque nœud
              représente une luciole */
8          tant que d > horloge() et qu'il manque des messages faire
9              dormir(d-horloge()) ;
10         fin
11         pour chaque m ∈ messages faire
12             max ← max(max, m) ;
13         fin
14     fin
15 fin
```

autour de la $140^{\text{ème}}$ seconde, puis à nouveau une synchronisation. On peut considérer le réseau synchronisé lorsque la mesure $\overline{\psi}$ passe en dessous d'un certain seuil ϵ.

$$\psi(v) = \frac{1}{\text{card}(\mathcal{N}(v))} \times \sum_{u \in \mathcal{N}(v)} |\omega(v) - \omega(u)| \tag{11.1}$$

$$\overline{\psi} = \frac{1}{\text{card}(\mathcal{L})} \times \sum_{v \in \mathcal{L}} \psi(v) \tag{11.2}$$

Ce type de synchronisation ne permet pas d'assurer la régularité des cycles. Elle permet en revanche de s'adapter complètement à la dynamique de l'infrastructure et de ne dépendre d'aucune centralisation, ce qui est en parfaite adéquation avec les objectifs que nous

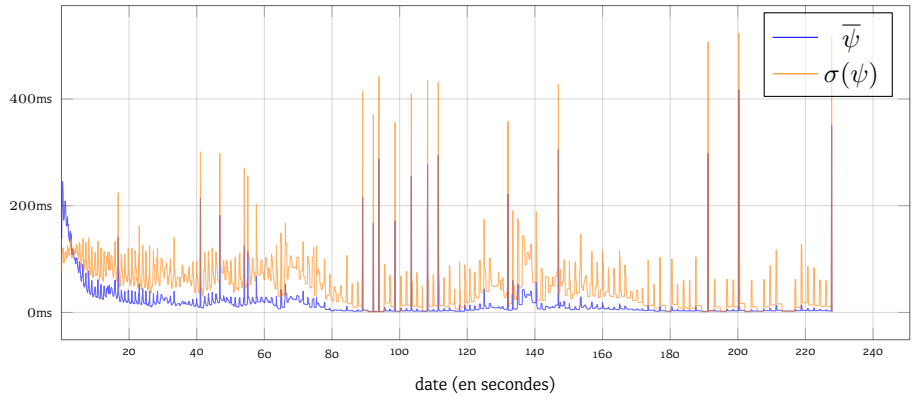


FIGURE 11.6 : Synchronisation des lucioles. La valeur moyenne $\overline{\psi}$ de la mesure ψ, ainsi que son écart type $\sigma(\psi)$.

recherchons. Les premiers tests réalisés grâce au prototype indiquent que ce concept est fonctionnel. Il est nécessaire d'élargir les expérimentations et de réaliser des tests sur une implantation en pratique.

Références

AGHA, Gul et al. (1997). "A foundation for actor computation". In : *Journal of Functional Programming* 7 (1), p. 1–72. ISSN : 0956-7968.

BUCK, John (1988). "Synchronous rhythmic flashing of fireflies. II." In : *Quarterly Review of Biology*, p. 265–289. ISSN : 0033-5770.

HEWITT, Carl, Peter BISHOP et Richard STEIGER (1973). "A universal modular actor formalism for artificial intelligence". In : Proceedings of the 3rd international joint conference on Artificial intelligence. Morgan Kaufmann Publishers Inc., p. 235–245.

HINDEN, Robert M. et Stephen E. DEERING (2006). *IP Version 6 Addressing Architecture*. URL : http://tools.ietf.org/html/rfc4291 (visité le 24/11/2013).

LEWIS, Sara M et Christopher K CRATSLEY (2008). "Flash signal evolution, mate choice, and predation in fireflies". In : *Annu. Rev. Entomol.* 53, p. 293–321. ISSN : 0066-4170.

MASINTER, Larry, Tim BERNERS-LEE et Roy T. FIELDING (2005). *Uniform Resource Identifier (URI) : Generic Syntax*. URL : https://tools.ietf.org/html/rfc3986 (visité le 22/11/2013).

STROGATZ, Steven H (2003). *Sync : The emerging science of spontaneous order*. Hyperion. ISBN : 0786868449.

Conclusion & Perspectives

Nous avons présenté, dans ces travaux, différents éléments permettant de contribuer à la modélisation d'un système distribué en tant qu'écosystème computationnel. Il s'agit d'un vaste domaine, et nous n'avons pas la prétention d'avoir apporté autre chose qu'un simple pas, sur la longue distance qu'il reste encore à parcourir avant d'atteindre un écosystème fonctionnel. Les systèmes complexes, et plus particulièrement leur simulation, ont permis de fixer le cadre et de définir les contraintes concernant les applications que l'on souhaite distribuer.

Une première partie des travaux est concentrée sur les graphes dynamiques qui permettent de modéliser le réseau d'interactions sous-jacent à un système complexe. La difficulté réside dans la dynamique, dont nous avons présenté différentes approches, et tenté d'apporter une esquisse de formalisme. Nous exploitons le graphe afin d'en dégager de l'organisation. Nous avons repris pour cela l'algorithme AntCo^2, un algorithme fourmis permettant de détecter des organisations dans un graphe dynamique, et nous avons proposé des améliorations visant à réduire l'oscillation des nœuds entre plusieurs organisations. De plus, nous proposons d'utiliser la centralité, et plus particulièrement les centroïdes, afin de détecter des organisations fragiles et ainsi augmenter la robustesse du système. La manipulation du graphe est possible grâce à un outil de modélisation que nous présentons : GraphStream. Il permet de fournir les bases d'une architecture permettant de gérer la dynamique d'un graphe.

Les travaux se concentrent ensuite sur le système distribué. La détection d'organisations est utilisée afin de répartir la charge du système sur un ensemble dynamique de machines. Cela permet de plus de réduire la charge du réseau. Nous avons présenté une plateforme qui permet d'aborder la distribution de manière décentralisée. L'objectif est de former un système qui soit robuste et résilient, et pour cela il doit être en mesure de s'adapter à la dynamique et aux différentes perturbations provoquant des évènements imprévisibles. Nous utilisons pour cela le modèle d'acteur, ainsi qu'un système où tout est acteur. L'aspect décentralisé apportant son lot de contraintes, nous avons fait certaines propositions afin de permettre une première dans la résolution de certains de ces problèmes : la découverte des machines voisines participant à la distribution, et la création d'un synchronisme global ne faisant appel qu'à des mécanismes locaux.

Notre problématique possède de nombreux aspects : de la modélisation des graphes dynamiques, à la distribution en pratique d'une application, en passant par la détection de com-

munauté, la répartition de charges, ou encore la centralité d'un graphe. De plus, beaucoup de temps a été consacré au développement durant cette thèse : d'une part de l'outil de modélisation de graphes dynamique GRAPHSTREAM ainsi que de ces multiples extensions ; et d'autre part de la plateforme de distribution. De ce fait, une seule thèse ne suffit pas à approfondir sérieusement tous les aspects du travail, et certaines choses mériteraient donc d'être validées de façon plus importante. Les perspectives, qui vont définir la suite de ces travaux, sont donc multiples. Nous avons ouvert une nouvelle piste permettant d'augmenter la résilience du système en détectant les centroïdes des organisations. Il est nécessaire par la suite d'approfondir les expérimentations de l'algorithme proposé, en particuliers en élargissant les topologies de graphe considérées. Puis de continuer dans cette voie en appliquant l'algorithme de détection de centroïde à l'affinage des résultats de la détection d'organisations afin d'augmenter la robustesse de ces organisations. Il devra s'en suivre des expérimentations afin d'évaluer le gain apporté.

La bibliothèque de modélisation GRAPHSTREAM, qui a été développée au cours de ces travaux, possède désormais une base solide. Il s'agit d'un outil qui est déjà utilisé en pratique et qui possède sa propre communauté. Elle fait régulièrement l'objet d'atelier au sein de la conférence ECCS [4] [DUTOT et PIGNÉ 2010 ; DUTOT, PIGNÉ et SAVIN 2011] et de tutoriels au cours l'école thématique CSSS [5] en 2012 et 2013. Du temps supplémentaire devra lui être accordé par la suite pour continuer son amélioration et ses possibilités afin de fournir un outil qui soit le plus complet possible.

Nous avons par exemple commencé à nous intéresser à l'interconnexion de GRAPHSTREAM avec une base de donnée de type NoSQL [6]. Différents outils existent déjà, des serveurs de base de données comme par exemple Neo4J [7], Titan [8], ou encore InfiniteGraph [9] ; mais aussi des interfaces de programmation comme par exemple Cypher [10], Gremlin [11] ou Blueprints [12]. Pour plus d'informations sur ce sujet, la page WIKIPÉDIA [13] propose une liste plus exhaustive de ces différents outils. Le but ne serait pas ici d'apporter un n-ième outil qui apporterait des fonctionnalités déjà existantes mais :

1. de permettre l'interaction de GRAPHSTREAM avec les outils existants ;
2. étudier de quelle façon notre bibliothèque pourrait contribuer à ce domaine en apportant son savoir-faire en terme de dynamique de graphes.

4. European Conference on Complex System
5. Complex Systems Summer School
6. Not Only SQL
7. http://www.neo4j.org/
8. http://thinkaurelius.github.com/titan/
9. http://infinitegraph.com/
10. http://www.neo4j.org/learn/cypher
11. http://gremlin.tinkerpop.com/
12. http://blueprints.tinkerpop.com/
13. https://en.wikipedia.org/wiki/Graph_database

Cette approche fait partie des voies que nous avons à explorer dans le cadre de la mise en œuvre de graphes distribués qui est une perspective importante de notre problématique.

Les travaux commencé au cours de cette thèse sur le changement d'échelle n'en sont qu'à leurs balbutiements. Le projet « organic », librement consultable sur la forge logicielle Github [14] vise à produire de manière automatique un graphe qui soit une vue macroscopique d'un autre. Il est basé sur GraphStream et correspond aux travaux qui ont été réalisés sur la réification des organisations. Plus de temps sont nécessaires pour approfondir ce projet, améliorer ses performances et son ergonomie, ce qui rentre pleinement dans les perspectives de nouveaux travaux qui prendront la suite de ceux-ci.

Nous avons aussi commencé à envisager l'étude des organisations en nous focalisant sur les membranes de ces organisations, c'est à dire la frontière délimitant l'organisation et dont les nœuds sont en contact avec d'autres organisations. Cette membrane semble jouer un rôle important dans la représentation des organisations et une discussion importante est nécessaire afin d'explorer ce sujet en profondeur.

14. https://github.com/gsavin/organic

Annexes

ANNEXE A

UN MODÈLE DE COMPORTEMENT DE GROUPE : LES BOIDS

Nous utilisons régulièrement dans ces travaux le modèle comportemental des boids, introduit dans [REYNOLDS 1987]. Craig REYNOLDS souhaitait pouvoir reproduire dans une simulation le comportement de groupe que l'on retrouve dans les nuées d'oiseaux, les troupeaux d'animaux terrestres, ou encore les bancs de poissons. Ce modèle a été utilisé dans des jeux vidéos (Half-Life par exemple) ainsi que dans des films (Batman Returns de Tim BURTON entre autres).

Nous lui portons un intérêt particulier du fait qu'il permet de créer un réseau d'interactions dans lequel on observe des propriétés émergentes. L'interaction est définie par la proximité entre les individus, qui fait que la présence d'un boid va influencer les propriétés de ses voisins. Les propriétés émergentes du système sont les organisations qui se créent. Ce modèle est donc particulièrement adapté pour nos expérimentations.

A.1 Règles de base

Ce modèle définit le comportement d'un individu à partir de trois règles :

1. la séparation ;

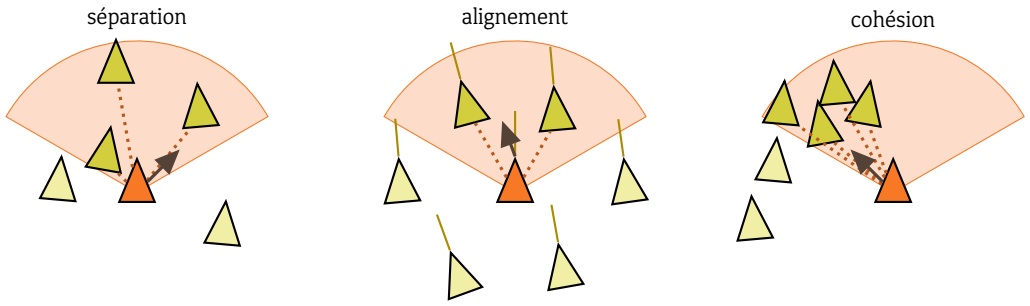


FIGURE A.1 : Règles appliquées aux boids telles que définies par Craig REYNOLDS.

2. l'alignement ;
3. la cohésion.

La séparation est un mécanisme qui permet d'éviter aux boids les collisions avec leurs voisins. Lorsqu'un boid est trop proche d'un de ses voisins, il est alors repoussé par ce dernier. L'alignement, quant à lui, fait que chaque individu tente d'aligner le vecteur définissant sa direction et sa vitesse sur celui de ces voisins. Enfin, la cohésion défini l'attraction exercée sur un boid par ses voisins proches.

A.2 Voisinage

Le voisinage d'un boid regroupe ses voisins les plus proches. Il est donc nécessaire de définir une distance maximale qui permet d'expliciter cette proximité : pour un boid donné, son voisinage correspond à l'ensemble des boids se trouvant à une distance inférieure à la distance maximale définie.

Cette distance maximale seule ne permet pas de définir un voisinage correct du fait qu'il inclut à la fois les individus qui se trouvent devant et derrière le boid considéré. Ceci peut mener à la stagnation des groupes de boids. C'est pourquoi nous introduisons l'angle correspondant à un champ de vision qui permet de définir en fonction de la direction d'un boid les individus qu'il peut "voir" et qui vont donc influencer sa trajectoire.

Nous limitons de plus le nombre d'individus qui peuvent être contenu dans un même voisinage. La raison est d'éviter l'explosion exponentielle du nombre d'interactions lors de la création de groupes et empêcher ainsi la création d'amas de boids, qui d'une part sont peu représentatif de la réalité, et d'autre part, apporte une contrainte majeure en terme de complexité lors de l'étude du graphe modélisant le réseau d'interactions. Les individus les plus proches sont sélectionnés en priorité.

Le voisinage peut être calculé de manière gloutonne, ce qui implique pour chaque individu de considérer l'ensemble de la population. Ainsi, la complexité d'une étape est $O(n^2)$ où n représente la taille de la population. Cette méthode a l'avantage d'être facilement mise en place, et peut être efficace pour des tailles de population faibles. En revanche, elle s'avère

inutilisable lorsque la taille de la population est importante. C'est pourquoi nous ajoutons une seconde méthode utilisant un n-tree, qui permet de diviser l'espace en cellules. Chaque individus n'a alors à considérer que la cellule dont il fait partie et éventuellement les cellules avoisinantes. On réduit ainsi la complexité en limitant les individus considérés à ceux suffisamment proches.

A.3 Intégration dans GraphStream

Notre implantation du modèle de boids est réalisée en utilisant la bibliothèque logicielle GraphStream. Chaque boid est directement modélisé par un nœud, et le réseau d'interactions par un graphe. Il est possible de définir plusieurs espèces de boids avec ses propres caractéristiques. La configuration est effectuée via les attributs du graphe et peut donc être stockée facilement dans un fichier DGS.

La simulation est disponible à la fois en tant que graphe mais aussi en tant que générateur de graphe, ce qui permet de l'intégrer facilement dans des applications existantes.

ANNEXE B

SIMULATIONS DE TESTS

Cette annexe décrit les graphes qui ont été utilisés par les simulations $\mathcal{S}_1$ et $\mathcal{S}_2$ lors de l'étude du lissage des résultats de l'algorithme AntCo^2 dans la partie 10.3.

B.1 Simulation statique, $\mathcal{S}_1$

La simulation $\mathcal{S}_1$ est effectuée sur un graphe statique dont la topologie est une grille. On considère un espace en deux dimensions divisé à intervalles réguliers. Chaque case ainsi formée représente un nœud qui est connecté aux quatre cases adjacentes : nord, sud, ouest et est. Un exemple d'une telle grille est donné dans la figure B.1. On peut étendre le voisinage d'un nœud, bien que ça ne soit pas le cas pour cette simulation, aux quatre cases accessibles en diagonale : nord-ouest, nord-est, sud-ouest et sud-est. Chaque nœud est donc de degré 4, excepté les nœuds de la bordure de degré 3 et les quatre nœuds des coins de degré 2. Il est possible d'obtenir une structure entièrement régulière en plaquant la grille sur un *tore* comme illustré dans la figure B.2.

Dans le cadre de la simulation $\mathcal{S}_1$, on considère une grille de dimension 101×101, composée de 10201 nœuds pour 20200 arêtes. La génération du graphe est faite avant l'exécution de l'algorithme AntCo^2. Ce dernier est ensuite exécuté pendant 10000 cycles.

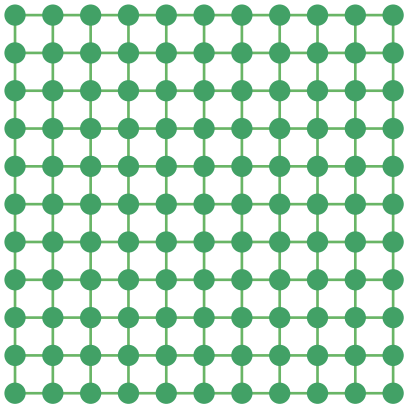


Figure B.1 : Une grille de dimension 11×11

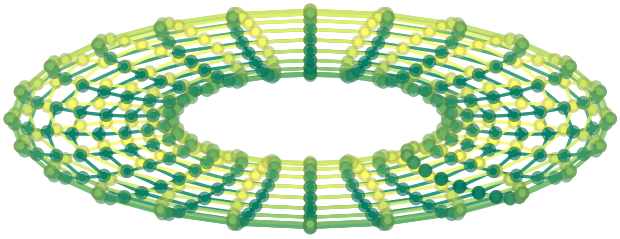


Figure B.2 : Un exemple de grille plaquée sur un tore

B.2 Simulation dynamique, $\mathcal{S}_2$

La simulation $\mathcal{S}_2$ est effectuée, quant à elle, sur un graphe dynamique. La dynamique porte sur les interactions, l'ensemble des nœuds restant fixe. Nous utilisons le modèle des boids, décrit dans l'annexe A, afin de générer les interactions entre les entités.

Deux espèces de boids sont utilisés : les moustiques et les abeilles. Les paramètres de ces espèces sont donnés dans la table B.1. La seconde espèce permet d'ajouter de la perturbation dans le graphe en repoussant les boids première espèce. Au total, 10000 nœuds sont créés, la quantité d'arêtes varie selon les cycles pour une moyenne d'environ 50000 nouvelles arêtes par cycle.

Pour chacun des 10000 cycles, on fait donc évoluer la structure du graphe grâce aux boids, puis on effectue un cycle de l'algorithme AntCo^2.

Moustiques		Abeilles
0	angle de vue	0.25
0.15	distance de visibilité	0.15
0.3	facteur de vitesse	0.3
0.9	vitesse max	1.0
0.04	vitesse min	0.1
0.1	facteur de direction	0.2
0.5	facteur d'attraction	0.3
0.001	facteur de répulsion	0.005
1.1	inertie	1.1
10	taille max du voisinage	10
9000	quantité	1000

TABLE B.1 : Paramètres du modèle de boids utilisé pour la simulation $\mathcal{S}_2$

ANNEXE C

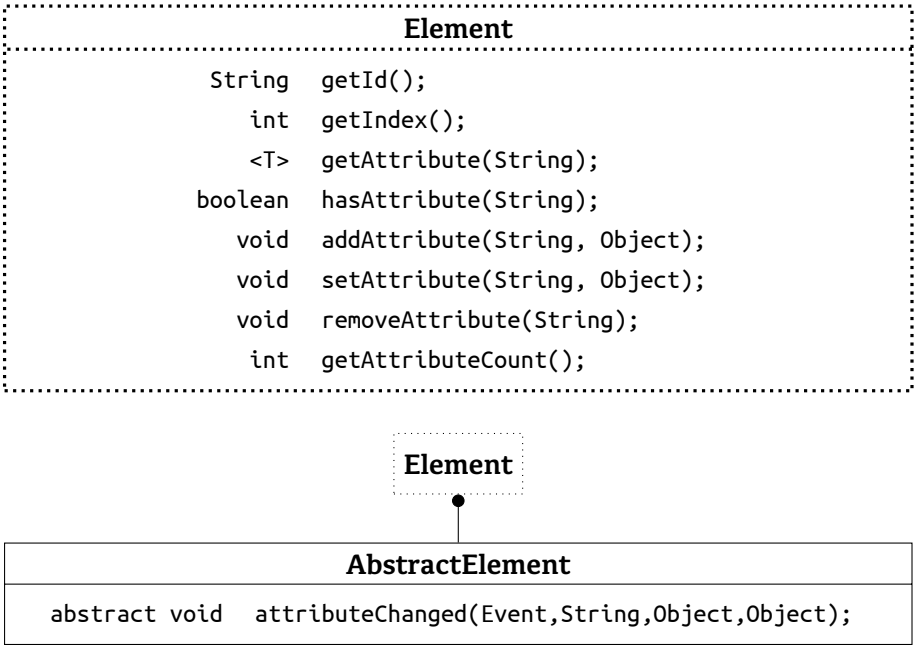
GRAPHSTREAM

C.1 Diagrammes de classes

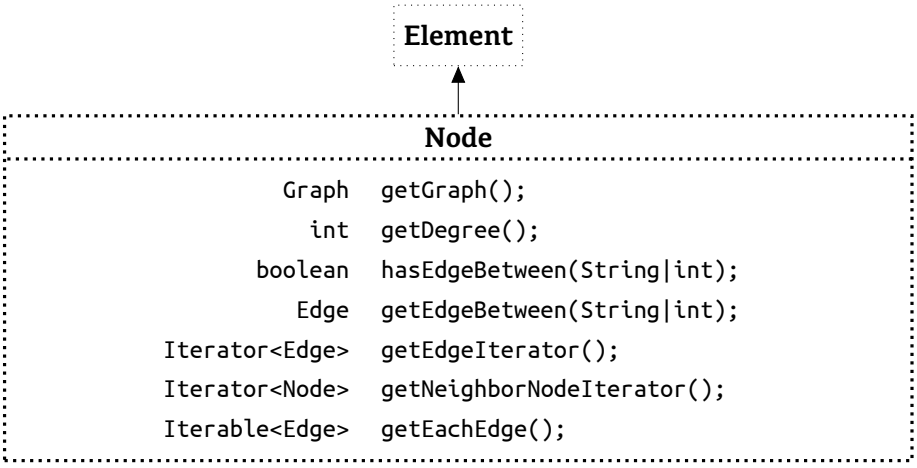
Nous exposons, dans cette partie de l'annexe, certaines classes de la bibliothèque GRAPHSTREAM.

C.1.1 Éléments

Il s'agit de la structure de base qui définit un objet pouvant contenir des attributs. Les méthodes `getAttribute(String)` et `hasAttribute(String)` possèdent différentes déclinaisons (qui ne sont pas présentées dans le schéma) permettant d'apporter des précisions sur le type de l'attribut, par exemple `getNumber(String)` ou `hasLabel(String)`.

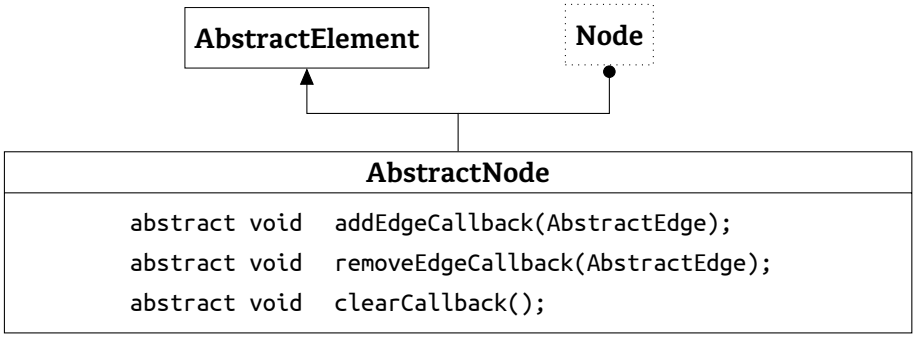


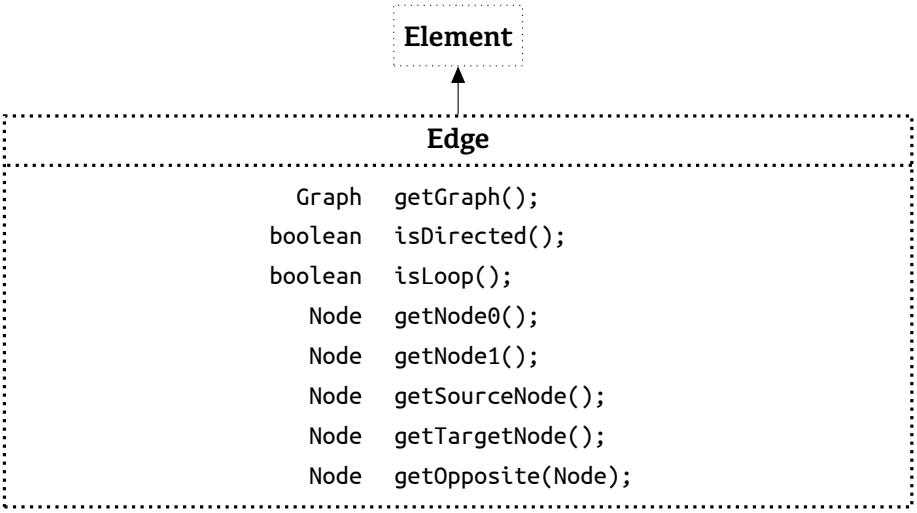
La classe `AbstractElement` permet de regrouper les parties de code communes aux nœuds, arêtes et aux graphes, concernant la gestion des attributs. De même, les classes `AbstractNode`, `AbstractEdge` et `AbstractGraph` regroupent respectivement le code commun aux différentes implantations de nœuds, d'arêtes et de graphes.

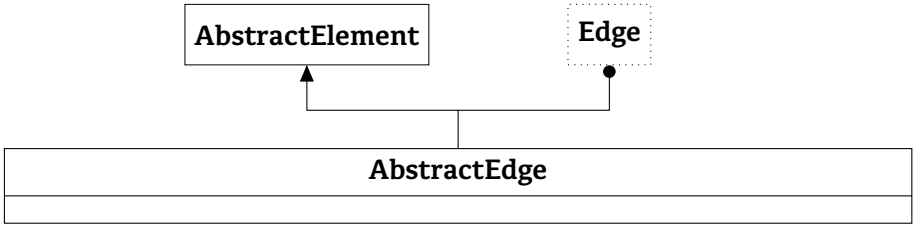


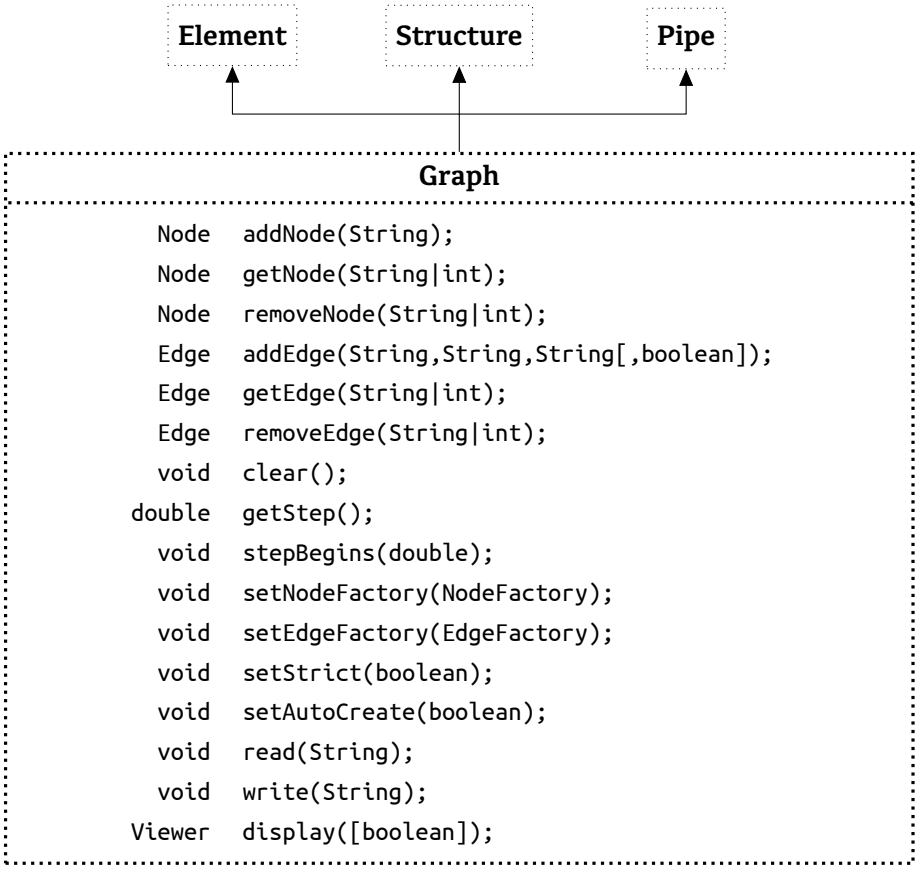
Certaines variations des méthodes ne sont pas présentées dans ce schéma. Il est par exemple possible de faire la distinction entre arêtes sortantes et entrantes. La méthode `getDegree()`, par exemple, est alors déclinée en `getInDegree()` et `getOutDegree()`, de même que la méthode `getEdgeBetween(String)` possède ses variations `getEdgeToward(String)`

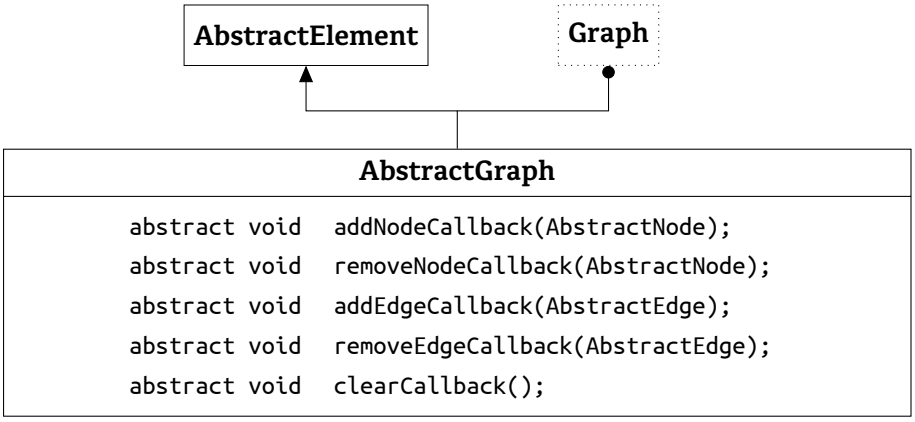
et `getEdgeFrom(String)`. Les itérateurs possèdent aussi leurs variantes qui permettent par exemple d'utiliser un parcours en largeur ou en profondeur.








Element
Structure
Pipe
Graph
Node addNode(String);
Node getNode(String|int);
Node removeNode(String|int);
Edge addEdge(String,String,String[,boolean]);
Edge getEdge(String|int);
Edge removeEdge(String|int);
void clear();
double getStep();
void stepBegins(double);
void setNodeFactory(NodeFactory);
void setEdgeFactory(EdgeFactory);
void setStrict(boolean);
void setAutoCreate(boolean);
void read(String);
void write(String);
Viewer display([boolean]);


AbstractElement
Graph
AbstractGraph
abstract void addNodeCallback(AbstractNode);
abstract void removeNodeCallback(AbstractNode);
abstract void addEdgeCallback(AbstractEdge);
abstract void removeEdgeCallback(AbstractEdge);
abstract void clearCallback();

C.1.2 Structure

Structure

```
                        int   getNodeCount();
                        int   getEdgeCount();
              Iterator<Node>  getNodeIterator();
              Iterator<Edge>  getEdgeIterator();
              Iterable<Node>  getEachNode();
              Iterable<Edge>  getEachEdge();
            Collection<Node>  getNodeSet();
            Collection<Edge>  getEdgeSet();
```

La structure permet de définir un objet qui va contenir des nœuds et des arêtes. L'objectif est de fournir une façon générique de modéliser un tel objet afin de permettre un développement plus large. Par exemple, en considérant une classe `Organisation` qui implanterait cette interface, il serait possible d'y appliquer un algorithme qui manipule la structure, plutôt que de limiter la portée de cet algorithme aux graphes.

C.1.3 Source et puits

Source

```
          void  addSink(Sink);
          void  addElementSink(ElementSink);
          void  addAttributeSink(AttributeSink);
          void  removeSink(Sink);
          void  removeElementSink(ElementSink);
          void  removeAttributeSink(AttributeSink);
          void  clearSinks();
          void  clearElementSinks();
          void  clearAttributeSinks();
```

ElementSink

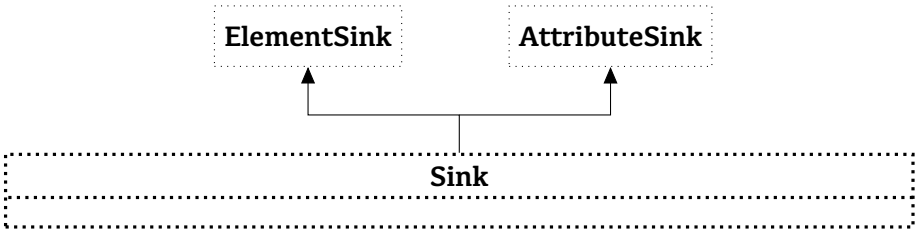
```
   void  nodeAdded(String,long,String);
   void  nodeRemoved(String,long,String);
   void  edgeAdded(String,long,String,String,String,boolean);
   void  edgeRemoved(String,long,String);
   void  graphCleared(String,long);
   void  stepBegins(String,long,double);
```

L'interface `ElementSink` définit un objet qui est capable de recevoir les événements liés aux modifications de la structure du graphe.

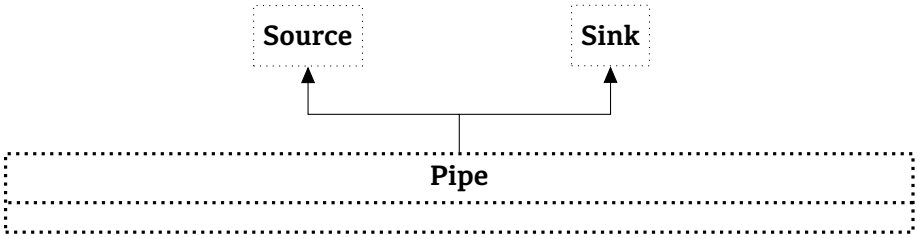
AttributeSink

```
void   {node|edge}AttributeAdded(String,long,String,String,Object);
void   graphAttributeAdded(String,long,String,Object);
void   {node|edge}AttributeChanged(String,long,String,String,Object,Object);
void   graphAttributeChanged(String,long,String,Object,Object);
void   {node|edge}AttributeRemoved(String,long,String,String);
void   graphAttributeRemoved(String,long,String);
```

En complément, l'interface `AttributeSink` définit un objet qui est capable de recevoir les événements liés aux modifications des attributs des différents éléments d'un graphe, ainsi que du graphe lui-même.



L'interface `Sink` définit un objet qui est à la fois un `ElementSink` et `AttributeSink`.



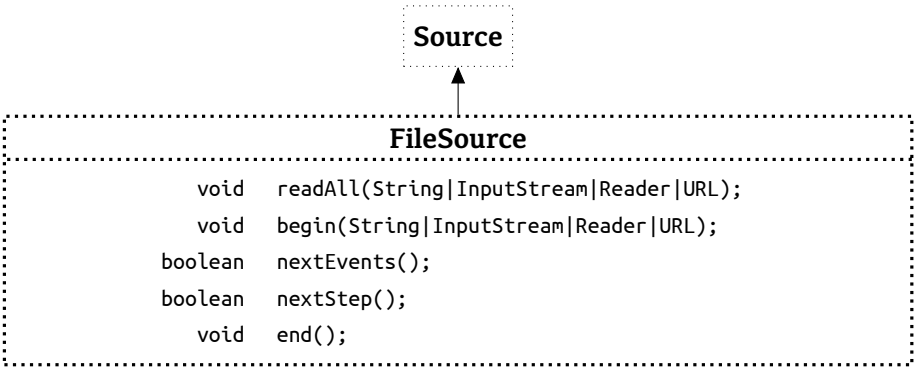
La classe `Pipe` représente un canal. Il s'agit simplement d'un objet qui est à la fois une source et un puits. L'intérêt des canaux est de permettre de définir la façon dont les évènements sont transmis entre une source et un puits. On peut alors créer des proxys qui permettent de faire transiter les évènements entre une source et un puits situés dans différents threads d'exécution, ou sur différentes machines.

Replayable

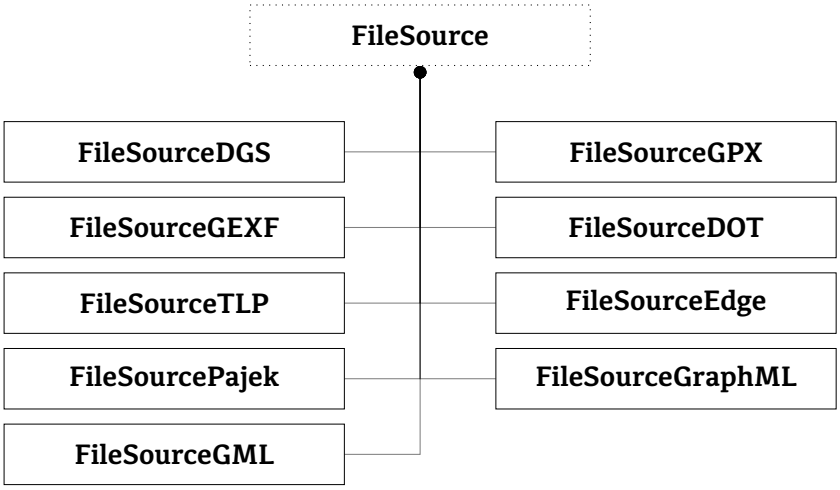
```
Controller   getReplayController();
```

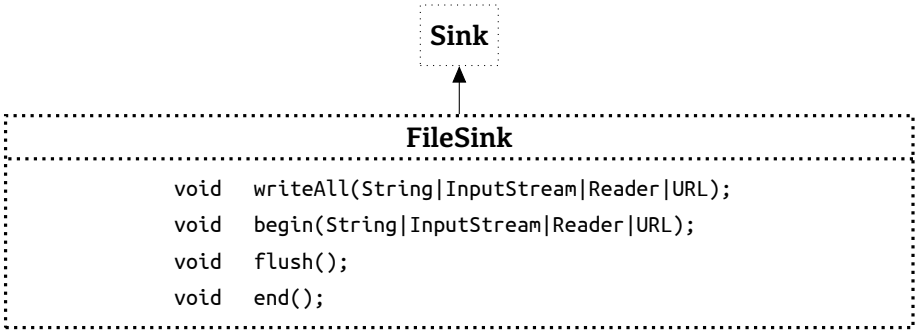
L'interface `Replayable` permet de caractériser une source qui peut être rejouée, c'est à dire qui possède une certaine mémoire. Un graphe par exemple, possède la mémoire de son état courant, mais pas de sa dynamique, il peut donc être en partie « rejoué ».

C.1.4 Fichiers

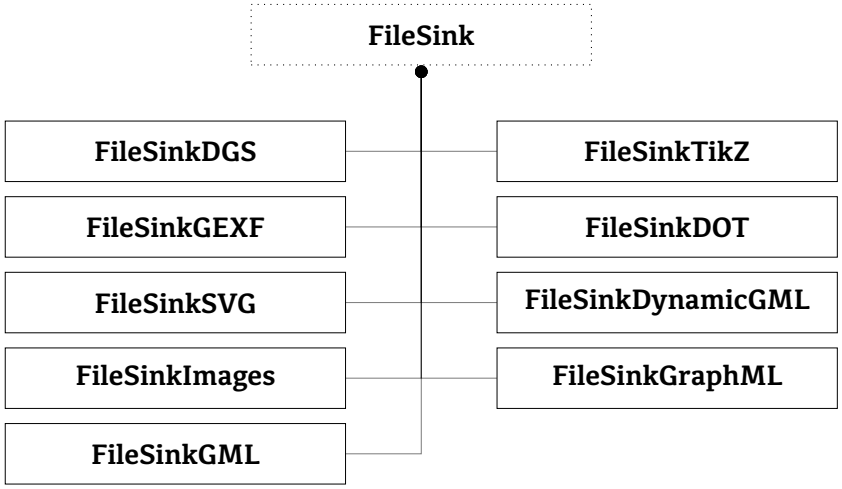


Un objet de type `FileSource` permet de définir une source qui va lire les événements dans un fichier. L'intégralité du fichier peut être lue à l'aide de la méthode `readAll()`, ou la lecture peut se faire événement par événement, ou encore par bloc d'événements.

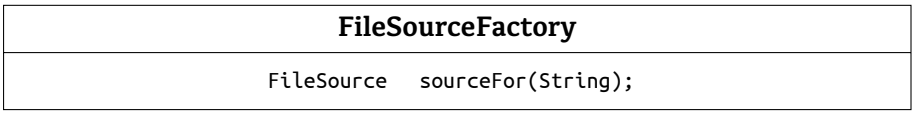


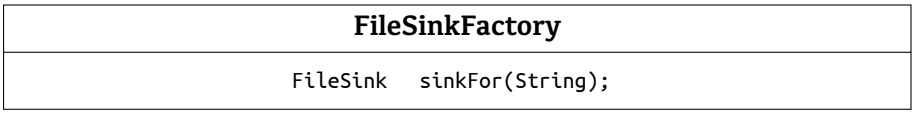


Les objets `FileSink` permettent quant à eux d'écrire le graphe dans un fichier. En fonction du format, la dynamique pourra être exportée ou non.



Afin de faciliter la gestion des différentes sources et puits fichiers, il existe des « fabriques » qui permettent de créer une source ou un puits correspondant à un nom de fichier, en fonction de l'extension de celui-ci.

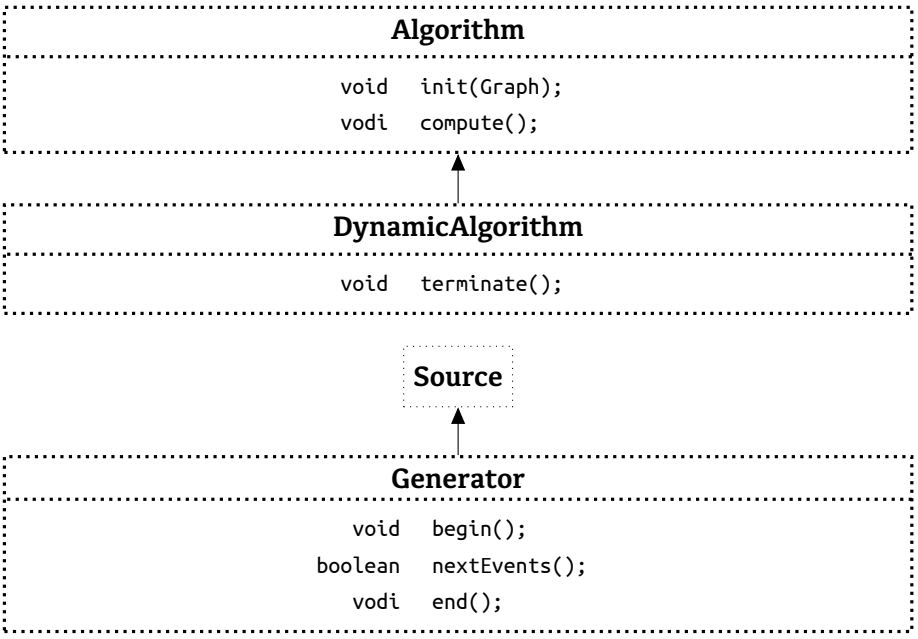




Ces fabriques sont utilisées par exemple dans l'implantation des méthodes `read(..)` et `write(..)` de `Graph`.

C.1.5 Algorithmes et générateurs

Les interfaces `Algorithm` et `DynamicAlgorithm` décrivent la forme générique des algorithmes dans GRAPHSTREAM. La première permet de définir un algorithme visant à s'exécuter ponctuellement sur un graphe. Il est alors initialiser, puis calculer au travers de la méthode `compute()`. Un algorithme dynamique est, quant à lui, régulièrement exécuté, par exemple à chaque évènement `stepBegins`. Il se différencie d'un algorithme statique par la méthode `terminate()` qui marque la fin de l'exécution de l'algorithme.



Les objets de type `Generator` permettent de générer de manière algorithmique des graphes.

C.2 Spécifications de DGS au format BNF

GRAPHSTREAM introduit son propre format de fichier, le format DGS pour « Dynamic Graph Stream ». Ce format, présenté dans la partie 9.3.1, permet de stocker un flux d'événements décrivant la dynamique d'un graphe. Chaque ligne décrit alors un événement.

La grammaire de ce format de fichier est donnée dans ce qui suit, sous la forme de Backus-Naur.

```
    <DGS>  ::= <header> ( <event> | <comment> | <EOL> )*
 <header>  ::= <magic> <EOL> <id> <int> <int> <EOL>
  <magic>  ::= "DGS004" | "DGS003"
  <event>  ::= ( <an> | <cn> | <dn> | <ae> | <ce> | <de> | <cg> | <st>
             | <cl> ) ( <comment> | <EOL> )
     <an>  ::= "an" <id> <attributes>
```

```
          <cn>  ::= “cn” <id> <attributes>
          <dn>  ::= “dn” <id>
          <ae>  ::= “ae” <id> <id> ( <direction> )? <id> <attributes>
          <ce>  ::= “ce” <id> <attributes>
          <de>  ::= “de” <id>
          <cg>  ::= “cg” <attributes>
          <st>  ::= “st” <real>
          <cl>  ::= “cl”
<attributes>  ::= ( <attribute> )*
 <attribute>  ::= ( ‘+’ | ‘-’ )? <id>
                  ( <assign> <value> ( ‘,’ <value> )* )?
     <value>  ::= <string> | <real> | ‘’ | <array> | <map>
     <array>  ::= ‘{’ ( <value> ( ‘,’ <value> )* )? ‘}’
       <map>  ::= ‘[’ ( <mapping> ( ‘,’ <mapping> )* )? ‘]’
   <mapping>  ::= <id> <assign> <value>
 <direction>  ::= ‘<’ | ‘>’ | ‘’
    <assign>  ::= ‘=’ | ‘:’
        <id>  ::= <string> | <int> | <word> ( ‘.’ <word> )*
   <comment>  ::= ‘#’ ( . )* <EOL>
       <int>  ::= ‘0’ | ( ‘1’ .. ‘9’ ) ( ‘0’ .. ‘9’ )*
      <real>  ::= <int> ( ‘.’ ( ‘0’ )* <int> )?
      <word>  ::= ( ‘a’ .. ‘z’ | ‘A’ .. ‘Z’ )
                  ( ‘a’ .. ‘z’ | ‘A’ .. ‘Z’ | ‘0’ .. ‘9’ | ‘-’ | ‘_’ )*
    <string>  ::= ‘”’ ( [^‘”’] | ‘”’ )* ‘”’
```

ANNEXE D

PLATEFORME D^3

La plateforme d^3, pour « dynamic and decentralized distribution », est l'intergiciel qui a été développé pendant cette thèse. Le code est hébergé sur la forge Github à l'adresse :

`https://github.com/gsavin/d3`

et une description est disponible sur le site qui lui est dédié :

`http://d3-project.org/`

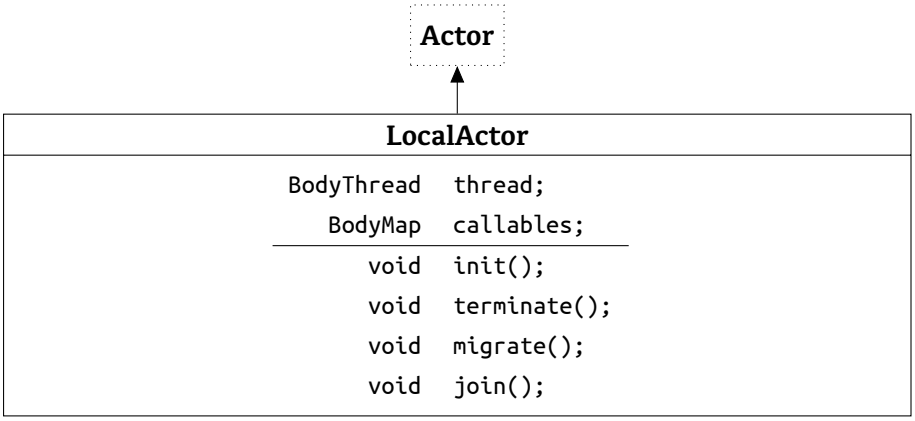
D.1 Diagramme de classes

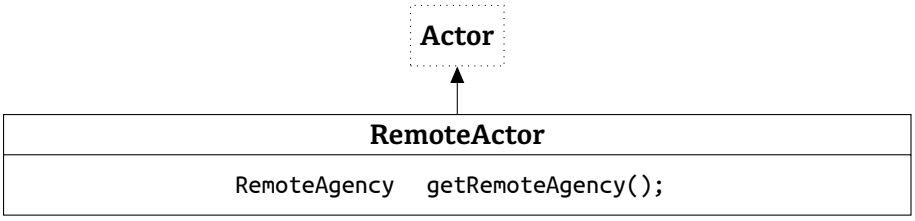
Nous présentons ici une partie des classes composant la plateforme de distribution, en particulier la partie correspondant au modèle d'acteur.

La classe abstraite `Actor` donne la base pour l'ensemble des acteurs. Elle regroupe les composants permettant de définir son URI. L'acteur qui étend cette classe doit définir les méthodes `call(..)` permettant d'effectuer une requête sur cet acteur.

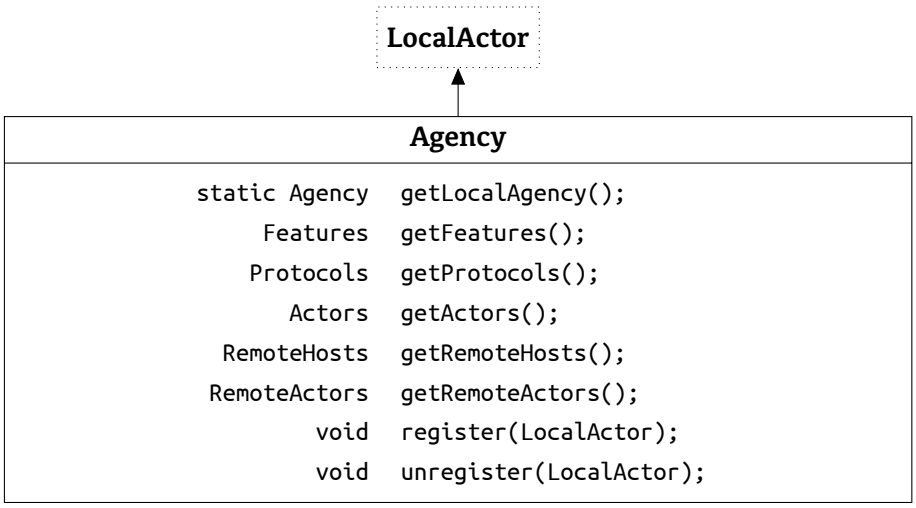
abstract **Actor**	
String	id;
HostAddresse	address;
String	agencyId;
String	path;
IdentifiableType	getType();
URI	getURI();
abstract boolean	isRemote();
abstract Object	call(String name, Object... args);
abstract void	call(String name, Future future, Object ... args);

On distingue, tout d'abord, les acteurs locaux `LocalActor` correspondant aux acteurs situés sur l'agence courante, des acteurs distants `RemoteActor`.

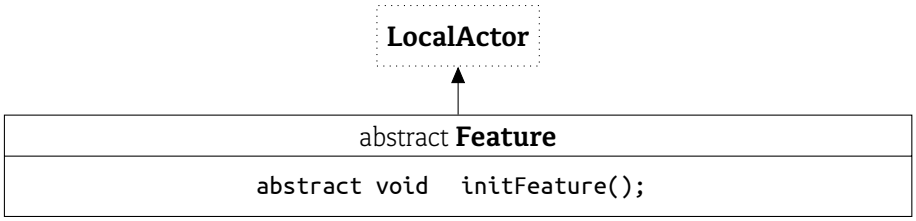




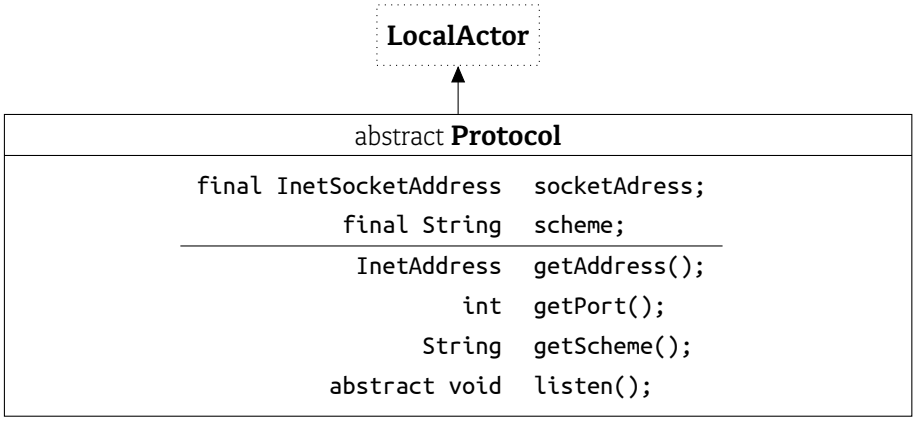
L'acteur principal de la plateforme est l'agence défini par la classe `Agency`. Elle permet de donner un point d'ancrage aux acteurs locaux et de regrouper les acteurs distants.

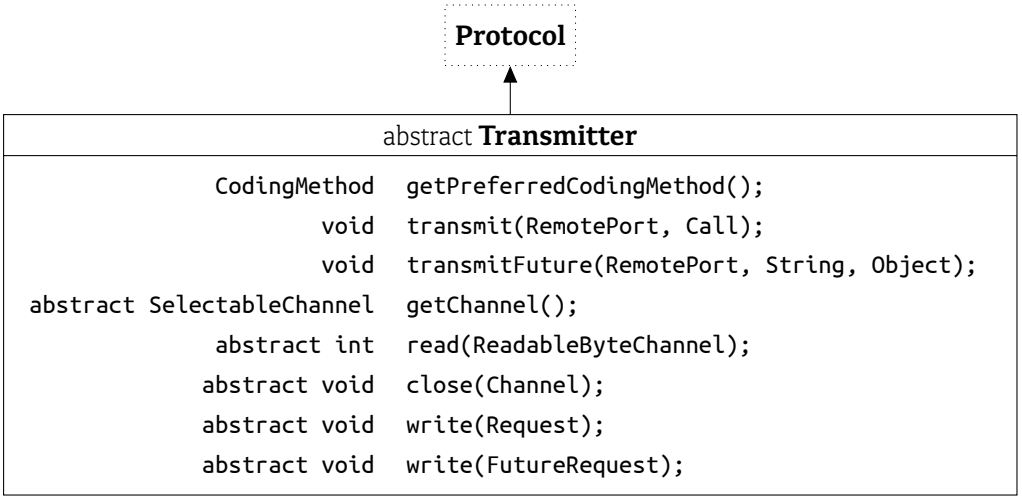


Les fonctionnalités de l'agence peuvent être étendues grâce aux acteurs `Feature`.

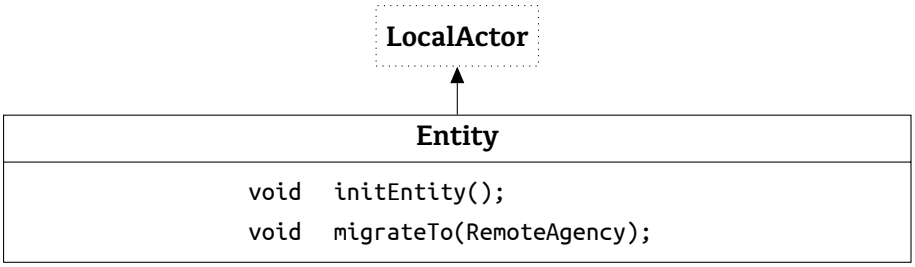


Certains acteurs ont la permission d'effectuer des échanges réseaux, en particulier d'ouvrir un port d'écoute de connexions extérieures. Ces acteurs sont nommés protocoles, et représentés par la classe `Protocol`. Une sous catégorie de ces protocoles, les transmetteurs, permettent de transmettre une requête d'une agence à l'autre. Ils sont définis par la classe `Transmitter`.

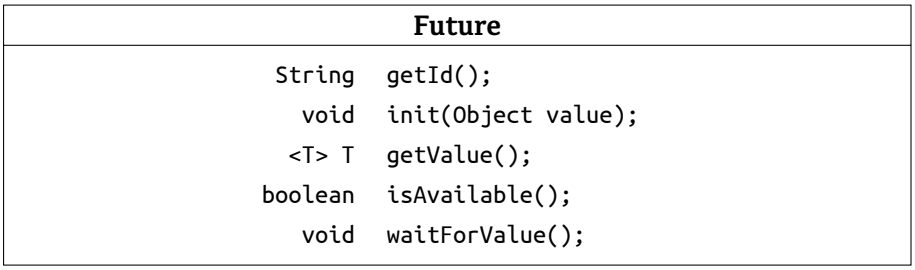




Les entités sont des acteurs qui ont la possibilité de migrer d'une agence à une autre. Elles sont définies par la classe `Entity`.



Le résultat d'une requête envoyée à un acteur est retourné sous la forme d'un `Future`. Il permet de gérer la mise à disposition asynchrone de ce résultat.



D.2 Requêtes aux acteurs

Une classe définissant un acteur doit indiquer quelles méthodes peuvent être invoquées par d'autres acteurs. Cela est possible grâce à l'annotation **`@Callable`**. La figure D.1 montre la définition d'une telle méthode.

```
public class MonEntite extends Entity {
    ...
    @Callable(”nom.de.la.requete”)
    public void uneMethodeInvocable() {
        ...
    }
    ...
}
```

FIGURE D.1 : Exemple de méthode invocable dans d^3

Lorsqu'une nouvelle classe d'acteur est détectée par l'agence, d^3 construit une « carte » de l'anatomie de l'acteur, c'est à dire qu'il détecte les méthodes qui peuvent être appelées grâce aux requêtes, et fait une association entre le nom de la requête et un objet qui permet d'appeler la méthode correspondante. Afin d'optimiser l'exécution, cet objet est compilé *à la volée* une seule fois. Dans le cas où un problème empêcherait cette compilation, la plateforme se rabat sur la « réflection ».

Bibliographie et index

Bibliographie

Agha, Gul et al. (1997). "A foundation for actor computation". In : *Journal of Functional Programming* 7 (1), p. 1–72. ISSN : 0956-7968.

Aldecoa, Rodrigo et Ignacio Marín (2011). "Deciphering network community structure by surprise". In : *PloS one* 6 (9), e24195. ISSN : 1932-6203.

Anderson, D. P. (nov. 2003). "Public Computing : Reconnecting People to Science". In : *Presented at the Conference on Shared Knowledge and the Web, Residencia de Estudiantes, Madrid, Spain*. URL : http://boinc.berkeley.edu/madrid.html.

Auber, David et al. (2003). "Multiscale Visualization of Small World Networks." In : INFOVIS. T. 3, p. 75–81.

Audi, Robert (1995). *The Cambridge Dictionary of Philosophy*. en. Cambridge University Press. 2390 p. ISBN : 9781107268616.

Awerbuch, Baruch et R Gallager (1987). "A new distributed algorithm to find breadth first search trees". In : *Information Theory, IEEE Transactions on* 33 (3), p. 315–322. ISSN : 0018-9448.

Bäck, Thomas (1994). *Evolutionary algorithms in theory and practice*. T. Bäck.

Baduel, Laurent et al. (2006). "Programming, Composing, Deploying for the Grid". Anglais. In : *Grid Computing : Software Environments and Tools*. Sous la dir. d'Omer F. Cunha Jose C. ; Rana. Springer, p. 205–229. ISBN : 978-1-85233-998-2. URL : http://hal.inria.fr/inria-00486114/PDF/ProgrammingDeployingComposingForTheGrid.pdf.

Baker, Henry et Carl Hewitt (1977). "Laws for communicating parallel processes". In :

Baldoni, Roberto et al. (2007). "Looking for a definition of dynamic distributed systems". In : *Parallel Computing Technologies*. Springer, p. 1–14. ISBN : 3540739394.

Barabasi, A. et E. Bonabeau (2003). "Scale-Free Networks". In : *Scientific American*, p. 50–59.

Barabási, Albert-László et Réka Albert (1999). "Emergence of Scaling in Random Networks". In : *Science* 286 (5439), p. 509–512. DOI : 10.1126/science.286.5439.509. URL : http://www.sciencemag.org/content/286/5439/509.abstract.

Batagelj, Vladimir et Andrej Mrvar (2004). *Pajek—analysis and visualization of large networks*. Springer. ISBN : 3642622143.

Bedau, M. A. (1999). "Weak Emergence". In : Tomberlin, James E. *Philosophical Perspectives, Mind, Causation And World*. T. 11, p. 375–399. ISBN : 9780631207931.

BELLMAN, Richard (1956). *On a routing problem*. DTIC Document.

BENDER-DEMOLL, Skye et Daniel A MCFARLAND (2006). "The art and science of dynamic network visualization". In : *Journal of Social Structure* 7 (2), p. 1–38.

BERTELLE, Cyrille et al. (2006). "Organization Detection Using Emergent Computing". In : *International Transactions on Systems Science and Applications* 2 (1), p. 61–70.

— (2007). "Organization Detection for Dynamic Load Balancing in Individual-Based Simulations". In : *Multi-Agent and Grid Systems* 3 (1), p. 42. URL : http://litis.univ-lehavre.fr/%20dutot/biblio/MAGS2007.pdf.

BISHOP, Toni A et Ramesh K KARNE (2003). "A survey of middleware". In : Computers and Their Applications, p. 254–258.

BOCCALETTI, S. et al. (2006). "Complex Networks : Structure and Dynamics". In : *Phys. Rep.* 424 (4-5), p. 175–308.

BORGATTI, Stephen P. et Martin G. EVERETT (2006). "A Graph-theoretic perspective on centrality". In : *Social Networks* 28 (4), p. 466–484. ISSN : 0378-8733. DOI : DOI:10.1016/j.socnet.2005.11.005. URL : http://www.sciencedirect.com/science/article/B6VD1-4J32JGJ-1/2/c8bf3911c0e5bb0e701c0a541e380475.

BORŮVKA, Otakar (1926). "O jistém problému minimálním (About a certain minimal problem)". In : *Práce mor. přírodověd. spol. v Brně* III (3), p. 37–58.

BRANDES, Ulrik (2001). "A Faster Algorithm for Betweenness Centrality". In : *Journal of Mathematical Sociology* 25, p. 163–177.

BRIN, Sergey et Lawrence PAGE (1998). "The anatomy of a large-scale hypertextual Web search engine". In : *Computer networks and ISDN systems* 30 (1), p. 107–117. ISSN : 0169-7552.

BUCK, John (1988). "Synchronous rhythmic flashing of fireflies. II." In : *Quarterly Review of Biology*, p. 265–289. ISSN : 0033-5770.

BUI-XUAN, B., A. FERREIRA et A. JARRY (avr. 2003). "Computing shortest, fastest, and foremost journeys in dynamic networks". In : *International Journal of Foundations of Computer Science* 14 (2), p. 267–285.

CARDON, Alain et al. (2006). "Emergent Properties in Natural and Artificial Dynamical Systems". In : sous la dir. de Moulay Aziz ALAOUI et Cyrille BERTELLE. Springer complexity : Understanding Complex Systems. Springer Verlag, p. 25–52. URL : http://litis.univ-lehavre.fr/%20dutot/biblio/EPNADS2006.pdf.

CASTEIGTS, A. et al. (2012). "Time-Varying Graphs and Dynamic Networks". In : *International Journal of Parallel, Emergent and Distributed Systems*. In press. DOI : 10.1080/17445760.2012.668546.

ČERNÝ, V. (1985). "Thermodynamical approach to the traveling salesman problem : An efficient simulation algorithm". In : *Journal of Optimization Theory and Applications* 45, p. 41–51.

CHALMERS, David J. (1997). *The Conscious Mind : In Search of a Fundamental Theory*. ISBN : 0195117891.

— (2006). "Strong and weak emergence". In : *The reemergence of emergence*, p. 244–256.

CHELLAPILLA, Kumar et David B FOGEL (1999). "Fitness distributions in evolutionary computation : motivation and examples in the continuous domain". In : *BioSystems* 54 (1), p. 15–29. ISSN : 0303-2647.

CHIKHI, Nacim Fateh (2010). "Calcul de centralité et identification de structures de communautés dans les graphes de documents." Université de Toulouse III. URL : http://thesesups.ups-tlse.fr/1364/.

CHUNG, Fan RK (1997). *Spectral graph theory*. T. 92. AMS Bookstore. ISBN : 0821803158.

CLAUSET, Aaron, Mark NEWMAN et Cristopher MOORE (2004). "Finding community structure in very large networks". In : *Physical review E*. APS 70 (6), p. 066111.

COQUILLARD, P. et D. R. C. HILL (1997). *Modélisation et Simulation des Écosystèmes*. Masson.

CORMEN, Thomas H. et al. (2009). "Introduction to Algorithms". In : MIT Press, p. 558–565. ISBN : 978-0-262-03384-8.

CORTES, Corina, Daryl PREGIBON et Chris VOLINSKY (déc. 2003). "Computational Methods for Dynamic Graphs". In : *Journal of Computational & Graphical Statistics* 12 (4). ISSN 1061-8600, Online ISSN : 1537-2715, 950–970(21).

COULOURIS, George, Jean DOLLIMORE et Tim KINDBERG (2012). *Distributed Systems : Concepts and Design Edition 5*. Addison-Wesley. ISBN : 978-0-13-214301-1.

CRAMER, Nichael Lynn (1985). "A Representation for the Adaptive Generation of Simple Sequential Programs". In : *ICGA*, p. 183–187.

CURRY, Edward (2004). "Message-oriented middleware". In : *Middleware for communications*, p. 1–28.

DEMETRESCU, Camil, David EPPSTEIN, Zvi GALIL et Giuseppe F. ITALIANO (2010). "Algorithms and theory of computation handbook". In : sous la dir. de Mikhail J. ATALLAH et Marina BLANTON. Chapman & Hall/CRC, p. 9–9. ISBN : 978-1-58488-822-2. URL : http://portal.acm.org/citation.cfm?id=1882757.1882766.

DEMETRESCU, Camil, David EPPSTEIN, Zvi GALIL et Giuseppe F ITALIANO (2010). "Dynamic graph algorithms". In : Algorithms and theory of computation handbook. Chapman & Hall/CRC, p. 9–9. ISBN : 1584888229.

DEMETRESCU, Camil et Giuseppe F. ITALIANO (2006). "Fully dynamic all pairs shortest paths with real edge weights". In : *J. Comput. Syst. Sci.* 72 (5), p. 813–837. ISSN : 0022-0000. DOI : 10.1016/j.jcss.2005.05.005.

DIETRICH, Johannes, Gabi LANDGRAFE et Elisavet H FOTIADOU (2012). "TSH and Thyrotropic Agonists : Key Actors in Thyroid Homeostasis". In : *Journal of thyroid research*.

DIJKSTRA, E. W. (1959). "A note on two problems in connexion with graphs". In : *Numerische Mathematik* 1, p. 269–271.

DORIGO, Marco (1992). "Optimization, learning and natural algorithms". In : *Ph. D. Thesis, Politecnico di Milano, Italy*.

DORIGO, Marco, Gianni DI CARO et Luca M GAMBARDELLA (1999). "Ant algorithms for discrete optimization". In : *Artificial life* 5 (2), p. 137–172.

DORIGO, Marco, Vittorio MANIEZZO et Alberto COLORNI (1996). "The Ant System : Optimization by a colony of cooperating agents". In : *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS-PART B* 26 (1), p. 29–41.

DOROGOVTSEV, S. N. et J. F. F. MENDES (2002). "Evolution of networks". In : *Adv. Phys*, p. 1079–1187.

— (2003). *Evolution of Networks : From Biological Nets to the Internet and WWW*. Oxford University Press.

DREYFUS, Stuart (2002). "Richard Bellman on the Birth of Dynamic Programming". In : *Oper. Res.* 50 (1), p. 48–51. ISSN : 0030-364X. DOI : 10.1287/opre.50.1.48.17791. URL : http://portal.acm.org/citation.cfm?id=767815.769382.

DUTOT, Antoine (2005). "Distribution dynamique adaptative par des mécanismes d'intelligence collective, détection d'organisations par des techniques de collaboration et de compétition". FST, Université du Havre : LITIS, Laboratoire d'Informatique, de Traitement de l'Information et des Systèmes. 206 p. URL : http://litis.univ-lehavre.fr/%20dutot/biblio/PhdThesisDutot.pdf.

DUTOT, Antoine, Frédéric GUINAND et al. (2007). "GraphStream : A Tool for bridging the gap between Complex Systems and Dynamic Graphs". In : *EPNACS : Emergent Properties in Natural and Artificial Complex Systems*.

DUTOT, Antoine et Yoann PIGNÉ (2009). "Tour d'horizon des problèmes combinatoires traités par les fourmis artificielles". In : *Fourmis artificielles 1, Des bases de l'optimisation aux applications industrielles*, p. 71–100.

— (sept. 2010). "GraphStream Workshop". In : *Emergent Properties in Natural and Artificial Complex Systems (EPNACS) in European Conference on Complex System (ECCS) 2010*.

DUTOT, Antoine et al. (oct. 2006a). "Pyocyanic Bacillus Propagation Simulation". In : *CoSSom workshop in European Simulation and Modeling conference (ESM)*. Toulouse, France. URL : http://litis.univ-lehavre.fr/%20dutot/biblio/COSSOM2006.pdf.

EDDY, S. R. (2004). "What is dynamic programming ?" In : *Nat Biotech* 22, p. 909–910.

EMMERICH, Wolfgang (2000). *Engineering distributed objects*. Wiley.com.

EPPSTEIN, D. (déc. 1995). "Dynamic Euclidean minimum spanning trees and extrema of binary functions". In : *Discrete and Computational Geometry* 13 (1), p. 111–122. DOI : 10.1007/BF02574030. URL : http://dx.doi.org/10.1007/BF02574030.

EULER, Leonhard (1741). "Solutio problematis ad geometriam situs pertinentis". In : *Commentarii academiae scientiarum Petropolitanae* 8, p. 128–140.

FALL, Kevin (2003). "A delay-tolerant network architecture for challenged internets". In : Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications. ACM, p. 27–34. ISBN : 1581137354.

FERREIRA, Afonso (mai 2002). "On models and algorithms for dynamic communication networks : The case for evolving graphs". In : *4e rencontres francophones sur les Aspects Algorithmiques des Telecommunications (ALGOTEL'2002)*. Mèze, France.

FIDUCCIA, C. M. et R. M. MATTHEYSES (1982). "A linear time heuristic for improving network partitions". In : *ACM IEEE Design Automation Confe- rence, pages*, p. 175–181.

FLOYD, Robert W. (1962). "Algorithm 97 : Shortest Path". In : *Communications of the ACM* 5 (6), p. 345.

FLYNN, Michael J (1966). "Very high-speed computing systems". In : *Proceedings of the IEEE* 54 (12), p. 1901–1909. ISSN : 0018-9219.

FOGEL, David B. (2006). *Evolutionary Computation : Toward a New Philosophy of Machine Intelligence*. en. John Wiley & Sons. 295 p. ISBN : 9780471749202.

FOGEL, Lawrence J. (1962). "Autonomous automata". In : *Industrial Research* 4 (2), p. 14–19.

FORD, Lester Randolph (1956). *Network Flow Theory*.

FORD, Lester Randolph et D. R. FULKERSON (1956). "Maximal Flow through a Network". In : *Canadian Journal of Mathematics*, p. 399–404.

FOSTER, Ian (2002). "What is the grid ?-a three point checklist". In : *GRIDtoday* 1 (6).

FOSTER, Ian et Carl KESSELMAN (2003). *The Grid 2 : Blueprint for a new computing infrastructure*. Access Online via Elsevier. ISBN : 0080521533.

FOSTER, Ian, Yong ZHAO et al. (2008). "Cloud computing and grid computing 360-degree compared". In : Grid Computing Environments Workshop, 2008. GCE'08. Ieee, p. 1–10. ISBN : 1424428602.

FRAIGNIAUD, Pierre et al. (2009). "Distributed computing with advice : information sensitivity of graph coloring". Anglais. In : *Distributed Computing* 21 (6). See paper for details., p. 395–403. DOI : 10.1007/s00446-008-0076-y. URL : http://hal.archives-ouvertes.fr/hal-00395775/PDF/DisComp-ICALP07.pdf.

FRANKS, Nigel R. et A. B. SENDOVA-FRANKS (1992). "Brood sorting by ants : distributing the workload over the work-surface". In : *Behavioral Ecology and Sociobiology*. Springer 30 (2), p. 109–123.

FREEMAN, Linton C. (1977). "A set of measures of centrality based upon betweenness". In : *Sociometry* 40, p. 35–41.

— (1978). "Centrality in social networks conceptual clarification". In : *Social Networks* 1 (3), p. 215–239. ISSN : 0378-8733. DOI : 10.1016/0378-8733(78)90021-7. URL : http://www.sciencedirect.com/science/article/pii/0378873378900217.

FUGGETTA, Alfonso, Gian Pietro PICCO et Giovanni VIGNA (1998). "Understanding Code Mobility". In : *IEEE Transactions on Software Engineering* 24, p. 342–361. ISSN : 0098-5589. DOI : http://doi.ieeecomputersociety.org/10.1109/32.685258.

GALLAGER, R. G., P. A. HUMBLET et P. M. SPIRA (1983). "A Distributed Algorithm for Minimum-Weight Spanning Trees". In : *ACM Trans. Program. Lang. Syst.* 5 (1), p. 66–77. ISSN : 0164-0925. DOI : 10.1145/357195.357200. URL : http://doi.acm.org/10.1145/357195.357200 (visité le 27/10/2013).

GAMBARDELLA, Luca Maria et Marco DORIGO (1995). "Ant-Q : A reinforcement learning approach to the traveling salesman problem". In : ICML, p. 252–260.

GAREY, Michael R et David S JOHNSON (1979). *Computers and intractability*. T. 174. Freeman New York.

GEPHI (2009). *GEXF File Format*. URL : http://gexf.net/format/ (visité le 04/11/2013).

GIRVAN, M. et M. E. J. NEWMAN (2002). "Community structure in social and biological networks". In : *Proc. Natl. Acad. Sci. USA* 99, p. 8271–8276.

GOSS, S. et al. (déc. 1989). "Self-organized shortcuts in the Argentine ant". In : *Naturwissenschaften* 76 (12), p. 579–581. DOI : 10.1007/BF00462870. URL : http://dx.doi.org/10.1007/BF00462870.

GRAPHSTREAM (2013). *GraphStream - The DGS File Format*. URL : http://graphstream-project.org/doc/Advanced-Concepts/The-DGS-File-Format_1.1/ (visité le 04/11/2013).

GRAPHVIZ (2013). *The DOT Language | Graphviz - Graph Visualization Software*. URL : http://www.graphviz.org/content/dot-language (visité le 04/11/2013).

GRASSÉ, Pierre-Paul (1959). "La Reconstruction du nid et les coordinations interindividuelles chez Bellicositermes natalensis et Cubitermes sp. La théorie de la stigmergie : essai d'interprétation du comportement des Termites constructeurs". In : *Insectes sociaux*.

GRINSTEAD, Charles M. et J. Laurie SNELL (1997). *Introduction to Probability*. ISBN : 978-0-8218-9414-9. URL : http://www.dartmouth.edu/~chance/teaching_aids/books_articles/probability_book/book.html.

GROGINSKY, H. L. et G. A. WORKS (nov. 1970). "A Pipeline Fast Fourier Transform". In : *IEEE Trans. Comput.* 19 (11), p. 1015–1019. ISSN : 0018-9340. DOI : 10.1109/T-C.1970.222826. URL : http://portal.acm.org/citation.cfm?id=1311954.1312605.

HALF-LIFE (2013). *Boid*. Half-Life Wiki. URL : http://half-life.wikia.com/wiki/Boid (visité le 14/11/2013).

HARARY, F. et G. GUPTA (1997). "Dynamic Graph Models". In : *Mathematical and Computer Modelling* 25 (7), p. 79–87.

HARARY, Frank (oct. 1969). *Graph Theory*. Published : Paperback. Westview Press. ISBN : 0201410338. URL : http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0201410338.

HART, Peter E, Nils J NILSSON et Bertram RAPHAEL (1968). "A formal basis for the heuristic determination of minimum cost paths". In : *Systems Science and Cybernetics, IEEE Transactions on* 4 (2), p. 100–107. ISSN : 0536-1567.

HEISS, Hans-Ulrich et Michael SCHMITZ (1995). "Decentralized dynamic load balancing : The particles approach". In : *Information Sciences*. Elsevier 84 (1), p. 115–128.

HERMAN, Ivan, Guy MELANÇON et M. Scott MARSHALL (2000). "Graph visualization and navigation in information visualization : A survey". In : *Visualization and Computer Graphics, IEEE Transactions on* 6 (1), p. 24–43. ISSN : 1077-2626.

HEWITT, Carl, Peter BISHOP et Richard STEIGER (1973). "A universal modular actor formalism for artificial intelligence". In : Proceedings of the 3rd international joint conference on Artificial intelligence. Morgan Kaufmann Publishers Inc., p. 235–245.

HINDEN, Robert M. et Stephen E. DEERING (2006). *IP Version 6 Addressing Architecture*. URL : http://tools.ietf.org/html/rfc4291 (visité le 24/11/2013).

HOLLAND, John H. (1975). *Adaptation in Natural and Artificial Systems*. University Michigan Press.

Holling, C. S (1973). “Resilience and stability of ecological systems”. In : *Annual review of ecology and systematics* 4, p. 1–23.

Holling, C S. (1996). “Engineering resilience vs. ecological resilience”. In : *Engineering Within Ecological Constraints*. The National Academies Press, p. 31–44. ISBN : 9780309051989.

Huberman, Bernado A (1988). *The ecology of computation*. Elsevier Science Inc.

Jacyno, Mariusz et Seth Bullock (2008). “Energy, entropy and work in computational ecosystems : A thermodynamic account”. In :

Johnson, Donald B. (1977). “Efficient Algorithms for Shortest Paths in Sparse Networks”. In : *J. ACM* 24 (1), p. 1–13. ISSN : 0004-5411. DOI : 10.1145/321992.321993. URL : http://doi.acm.org/10.1145/321992.321993 (visité le 17/10/2013).

Johnson, Eric E (1988). “Completing an MIMD multiprocessor taxonomy”. In : *ACM SIGARCH Computer Architecture News* 16 (3), p. 44–47. ISSN : 0163-5964.

Jordan, Camille (1869). “Sur les assemblages des lignes”. In : *J. Reine Angew. Math* 70, p. 185–190.

Junginger, Markus Oliver et Yugyung Lee (2004). “Peer-to-peer middleware”. In : *Middleware for Communications*, p. 81–107. ISSN : 0470862084.

Karypis, George et Vipin Kumar (1999). “A fast and high quality multilevel scheme for partitioning irregular graphs”. In : *SIAM, Journal on Scientific Computing* 20 (1), p. 359–392.

Katz, Leo (1953). “A New Status Index Derived from Sociometric Index”. In : *Psychometrika*, p. 39–43.

Kennedy, J et R Eberhart (1995). “Particle Swarm Optimization”. In : *IEEE International of first Conference on Neural Networks*.

Kephart, Jeffrey O, Tad Hogg et Bernado A Huberman (1989). “Dynamics of computational ecosystems”. In : *Physical Review A*.

Kernighan, B. W. et S. Lin (1970). “An Efficient Heuristic Procedure for Partitioning Graphs”. In : *The Bell system technical journal* 49 (1), p. 291–307.

Kirkpatrick, Scott, Mario P Vecchi et D. Gelatt (1983). “Optimization by simulated annealing”. In : *science* 220 (4598).

Kleinberg, Jon M. (août 2000). “Navigation in a small world”. In : *Nature* 406 (6798), p. 845. ISSN : 0028-0836. DOI : 10.1038/35022643. URL : http://dx.doi.org/10.1038/35022643.

Knabe, Frederick (1997). “An Overview of Mobile Agent Programming”. In : *Selected papers from the 5th LOMAPS Workshop on Analysis and Verification of Multiple-Agent Languages*. London, UK : Springer-Verlag, p. 100–115. ISBN : 3-540-62503-8. URL : http://portal.acm.org/citation.cfm?id=646912.710959.

Kolmogorov, Andreï (1965). “Three Approaches for Defining the Concept of Information Quantity”. In : *Problems of information transmission* 1 (1), p. 1–7.

Krakowiak, S. (2008). *Intergiciel et Construction d’Applications Réparties*.

Kruskal, Joseph B (1956). “On the shortest spanning subtree of a graph and the traveling salesman problem”. In : *Proceedings of the American Mathematical society* 7 (1), p. 48–50. ISSN : 0002-9939.

Kuntz, Pascale, Paul Layzell et Dominique Snyers (1997). “A colony of ant-like agents for partitioning in VLSI technology”. In : *Proceedings of the Fourth European Conference on Artificial Life*. MIT, p. 417–424.

Kuntz, Pascale et Dominique Snyers (1994). “Emergent colonization and graph partitioning”. In : *Proceedings of the third international conference on Simulation of adaptive behavior : from animals to animats 3*, p. 494–500.

Lamport, Leslie (18 mai 1987). *distribution*. English. E-mail. URL : https://research.microsoft.com/en-us/um/people/lamport/pubs/distributed-system.txt (visité le 08/08/2013).

Laszewski, Gregor von et Kaizar Amin (2004). “Grid middleware”. In : *Middleware for Communications*, p. 109.

Lavender, R. Greg et Douglas C. Schmidt (1996). *Active Object – An Object Behavioral Pattern for Concurrent Programming*.

Le Moigne, Jean-Louis (1994). *La théorie du système général, théorie de la modélisation*. Les classiques du réseau intelligence de la complexité.

Leung, Ian XY et al. (2009). “Towards real-time community detection in large networks”. In : *Physical Review E*. APS 79 (6).

Lewis, Sara M et Christopher K Cratsley (2008). “Flash signal evolution, mate choice, and predation in fireflies”. In : *Annu. Rev. Entomol.* 53, p. 293–321. ISSN : 0066-4170.

Lin, H.-C. et Cauligi S. Raghavendra (1992). “A dynamic load-balancing policy with a central job dispatcher (LBC)”. In : *Software Engineering, IEEE Transactions on* 18 (2), p. 148–158.

Lloyd, Stuart (1982). “Least squares quantization in PCM”. In : *Information Theory, IEEE Transactions on* 28 (2), p. 129–137. ISSN : 0018-9448.

Lovelock, James (1993). *La terre est un être vivant : l’hypothèse Gaïa*. Champs sciences. Flammarion. ISBN : 978-2-0812-4481-8.

Lynch, Nancy A (1996). *Distributed algorithms*. Morgan Kaufmann. ISBN : 0080504701.

Lynch, Nancy A et Mark R Tuttle (1987). “Hierarchical correctness proofs for distributed algorithms”. In : Proceedings of the sixth annual ACM Symposium on Principles of distributed computing. ACM, p. 137–151. ISBN : 089791239X.

Mahmoud, Qusay H et John Wiley (2004). *Middleware for communications*. Wiley Online Library. ISBN : 0470862068.

Masinter, Larry, Tim Berners-Lee et Roy T. Fielding (2005). *Uniform Resource Identifier (URI) : Generic Syntax*. URL : https://tools.ietf.org/html/rfc3986 (visité le 22/11/2013).

Mell, Peter et Timothy Grance (2011). “The NIST definition of cloud computing (draft)”. In : *NIST special publication* 800 (145), p. 7.

Miller, Brad L. et David E. Goldberg (1995). “Genetic Algorithms, Tournament Selection, and the Effects of Noise”. In : *Complex Systems* 9, p. 193–212.

Morrison, Richard S. (2003). *Cluster Computing : Architectures, Operating Systems, Parallel Processing & Programming Languages*.

Mortveit, Henning S. et Christian M. Reidys (2008). *An Introduction to Sequential Dynamical Systems*. Springer.

Nabaa, Michel et al. (2009). "Exploitation of a displacement survey to detect road network use vulnerability". In : ICCSA 2009 Conference Proceedings Special Sessions, p. 151–155.

Newman, M. E. J. (2003). "The Structure and Function of Complex Networks". In : *SIAM Review* 45, p. 167–256. DOI : 10.1137/S003614450342480.

Newman, Mark (2010). *Networks : An Introduction*. New York, NY, USA : Oxford University Press, Inc. ISBN : 0199206651, 9780199206650.

Newman, Mark E. J. (2004). "Fast algorithm for detecting community structure in networks". In : *Physical review E*. APS 69 (6), p. 066133.

— (2006). "Finding community structure in networks using the eigenvectors of matrices". In : *Physical review E*. APS 74 (3), p. 036104.

Olivier, Damien (2006). *Modélisation informatique de systèmes à base d'interactions et détection d'organisations. Modèles du vivant*. Habilitation à Diriger des Recherches de l'Université du Havre.

OMG, Committee (1991). *The Common Object Request Broker : Architecture and Specification*. OMG Document Revision 1.

Opsahl, Tore, Filip Agneessens et John Skvoretz (2010). "Node centrality in weighted networks : Generalizing degree and shortest paths". In : *Social Networks* 32 (3), p. 245–251. ISSN : 0378-8733. DOI : 10.1016/j.socnet.2010.03.006. URL : http://www.sciencedirect.com/science/article/pii/S0378873310000183.

Ore, Oystein (1962). *Theory of graphs*. English. American Mathematical Society, Providence. v. P.

O'Sullivan, David (2001). "Graph-cellular automata : a generalised discrete urban and regional model". In : *Environment and Planning B : Planning and Design* 28, p. 687–705.

Palla, Gergely et al. (2005). "Uncovering the overlapping community structure of complex networks in nature and society". In : *Nature* 435 (7043), p. 814–818. ISSN : 0028-0836.

Passau, Universität (2011). *Graph Modelling Language*. Projects. URL : http://www.fim.uni-passau.de/en/fim/faculty/chairs/theoretische-informatik/projects.html (visité le 11/03/2013).

Pigné, Yoann (2008). "Modélisation et traitement décentralisé des graphes dynamiques : Application aux réseaux mobiles ad hoc". French. Université du Havre. URL : http://tel.archives-ouvertes.fr/tel-00371962/PDF/these.pdf.

Pigné, Yoann et Frédéric Guinand (2010). "Short and robust communication paths in dynamic wireless networks". In : *Proceedings of the 7th international conference on Swarm intelligence*. ANTS'10. Berlin, Heidelberg : Springer-Verlag, p. 520–527. ISBN : 3-642-15460-3, 978-3-642-15460-7. URL : http://portal.acm.org/citation.cfm?id=1884958.1885012.

Pimm, Stuart L (1984). "The complexity and stability of ecosystems". In : *Nature* 307 (5949), p. 321–326.

PIOTROWSKI, W. et M. M. SYSŁO (1991). "Some properties of graph centroids". In : *Annals of Operations Research* 33 (3). 10.1007/BF02115757, p. 227–236. ISSN : 0254-5330. URL : http://dx.doi.org/10.1007/BF02115757.

PONS, Pascal et Matthieu LATAPY (2005). "Computing communities in large networks using random walks". In : *Computer and Information Sciences-ISCIS 2005*. Springer, p. 284–293. ISBN : 3540294147.

POTHEN, Alex, Horst D. SIMON et Kang-Pu LIOU (1990). "Partitioning sparse matrices with eigenvectors of graphs". In : *SIAM, Journal on Matrix Analysis and Applications* 11 (3), p. 430–452.

PRIGOGINE, Ilya (2008). *Les lois du chaos*. Flammarion. ISBN : 978-2-0812-1487-3.

PRIGOGINE, Ilya et Isabelle STENGERS (1996). *La fin des certitudes : temps, chaos et les lois de la nature*. Odile Jacob. ISBN : 2738103308.

PRIM, Robert Clay (1957). "Shortest connection networks and some generalizations". In : *Bell system technical journal* 36 (6), p. 1389–1401.

RABERTO, Marco, Fabio RAPALLO et Enrico SCALAS (2011). "Semi-Markov Graph Dynamics". In : *PLoS ONE* 6 (8).

RAGHAVAN, Usha Nandini, Réka ALBERT et Soundar KUMARA (2007). "Near linear time algorithm to detect community structures in large-scale networks". In : *Physical Review E*. APS 76 (3), p. 036106.

REDMOND, Frank E (1997). *Dcom : Microsoft Distributed Component Object Model with Cdrom*. IDG Books Worldwide, Inc. ISBN : 0764580442.

REYNOLDS, Craig W. (août 1987). "Flocks, herds and schools : A distributed behavioral model". In : *SIGGRAPH Comput. Graph.* 21 (4), p. 25–34. ISSN : 0097-8930. DOI : http://doi.acm.org/10.1145/37402.37406. URL : http://doi.acm.org/10.1145/37402.37406.

SADASHIV, Naidila et SM Dilip KUMAR (2011). "Cluster, grid and cloud computing : A detailed comparison". In : Computer Science & Education (ICCSE), 2011 6th International Conference on. IEEE, p. 477–482. ISBN : 1424497175.

SANDERS, Peter et Christian SCHULZ (2013). "Think Locally, Act Globally : Highly Balanced Graph Partitioning". In : *Experimental Algorithms*. Springer, p. 164–175. ISBN : 3642385265.

SATOH, Ichiro (2001). "Adaptive Protocols for Agent Migration". In : *Proceedings of the The 21st International Conference on Distributed Computing Systems*. ICDCS '01. Washington, DC, USA : IEEE Computer Society, p. 711–. URL : http://portal.acm.org/citation.cfm?id=876878.879272.

SAUSSURE, Ferdinand de (1931). *Cours de linguistique générale*. Payot. Geneva.

SCHAEFFER, S. (août 2007). "Graph clustering". In : *Computer Science Review* 1 (1), p. 27–64. ISSN : 15740137. DOI : 10.1016/j.cosrev.2007.05.001. URL : http://dx.doi.org/10.1016/j.cosrev.2007.05.001.

SCHMITZ, Michael (2002). *UNICORE Plus Final Report*. Published : Joint Project Report for the BMBF Project UNICORE Plus. URL : http://www.unicore.org/documents/UNICOREPlus-Final-Report.pdf.

SCHOLLMEIER, Rüdiger (2001). "A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications". In : Peer-to-Peer Computing, 2001. Proceedings. First International Conference on. IEEE, p. 101–102. ISBN : 0769515037.

SCHOONDERWOERD, Ruud et al. (1997). "Ant-based load balancing in telecommunications networks". In : *Adaptive behavior* 5 (2), p. 169–207. ISSN : 1059-7123.

SLATER, Peter J. (1978). "Centers to centroids in graphs". In : *Journal of Graph Theory* 2 (3), p. 209–222.

— (1983). "Some definitions of central structures". In : *Lecture Notes in Mathematics* 1073/1984, p. 169–178.

STRAßER, Markus et M SCHWEHM (1997). "A performance model for mobile agent systems". In : *International Conference on Parallel and Distributed Processing Techniques and Applications*. Sous la dir. d'HEditor ARABNIA. T. 2. Citeseer, p. 1132–1140. URL : http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.39.30&rep=rep1&type=pdf.

STROGATZ, S. H. et D. J. WATTS (1998). "Collective dynamics of 'small-world' networks". In : *Nature* 393, p. 440–442.

STROGATZ, Steven H (2003). *Sync : The emerging science of spontaneous order*. Hyperion. ISBN : 0786868449.

STÜTZLE, Thomas et Holger H HOOS (2000). "MAX–MIN ant system". In : *Future generation computer systems* 16 (8), p. 889–914. ISSN : 0167-739X.

TEL, Gerard (2000). *Introduction to distributed algorithms*. Cambridge university press. ISBN : 0521794838.

TERESCO, J. D., K. D. DEVINE et J. E. FLAHERTY (2006). "Partitioning and Dynamic Load Balancing for the Numerical Solution of Partial Differential Equations". In : *Numerical solution of partial differential equations on parallel computers*. Bruaset, Are Magnus. Springer.

THOMSEN, Bent, Lone LETH et Tsung-Min KUO (1996). "A Facile Tutorial". In : *Proceedings of the 7th International Conference on Concurrency Theory*. CONCUR '96. London, UK : Springer-Verlag, p. 278–298. ISBN : 3-540-61604-7. URL : http://portal.acm.org/citation.cfm?id=646731.703703.

TRUSZCZYNSKI, Miroslaw (1985). "Centers and Centroids of Unicyclic Graphs". In : *Mathematica Slovaca* 35 (3), p. 223–228.

TULIP (2013). *Tulip software graph format (TLP) | Tulip*. URL : http://tulip.labri.fr/TulipDrupal/?q=tlp-file-format (visité le 04/11/2013).

TYRRELL, Alexander, Gunther AUER et Christian BETTSTETTER (2006). "Fireflies as role models for synchronization in ad hoc networks". In : *Proceedings of the 1st international conference on Bio inspired models of network, information and computing systems*. BIONETICS '06. New York, NY, USA : ACM. ISBN : 1-4244-0463-0. DOI : http://doi.acm.org/10.1145/1315843.1315848. URL : http://doi.acm.org/10.1145/1315843.1315848.

VLISSIDES, John et al. (1995). "Design patterns : Elements of reusable object-oriented software". In : *Reading : Addison-Wesley* 49, p. 120.

Von Luxburg, Ulrike (2007). "A tutorial on spectral clustering". In : *Statistics and computing* 17 (4), p. 395–416. ISSN : 0960-3174.

Von Neumann, John (1993). "First Draft of a Report on the EDVAC". In : *Annals of the History of Computing, IEEE* 15 (4), p. 27–75. ISSN : 1058-6180.

Walker, Brian et al. (2004). "Resilience, Adaptability and Transformability in Social– ecological Systems". In : *Ecology and Society* 9 (2).

Warshall, Stephen (1962). "A theorem on boolean matrices". In : *Journal of the ACM (JACM)* 9 (1), p. 11–12. ISSN : 0004-5411.

Wasserman, Stanley et Katherine Faust (1994). *Social Networks Analysis*. Cambridge University Press.

Watts, Duncan J et Steven H Strogatz (1998). "Collective dynamics of 'small-world'networks". In : *nature* 393 (6684), p. 440–442. ISSN : 0028-0836.

Weiser, Mark (1991). "The computer for the 21st century". In : *Scientific american* 265 (3), p. 94–104. ISSN : 0036-8733.

— (1993). "Some computer science issues in ubiquitous computing". In : *Communications of the ACM* 36 (7), p. 75–84. ISSN : 0001-0782.

Williams, Roy D. (1991). "Performance of dynamic load balancing algorithms for unstructured mesh calculations". In : *Concurrency : Practice and experience* 3 (5), p. 457–481.

Wollrath, Ann, Roger Riggs et Jim Waldo (1996). "A Distributed Object Model for the Java TM System". In : *Computing Systems* 9, p. 265–290. ISSN : 0895-6340.

Xie, Jierui, Stephen Kelley et Boleslaw K Szymanski (2011). "Overlapping community detection in networks : the state of the art and comparative study". In : *arXiv preprint arXiv :1110.5813*.

Zachary, W. W. (1977). "An information flow model for conflict and fission in small groups". In : *Journal of Anthropological Research* 33, p. 452–473.

Zambonelli, Franco et Mirko Viroli (2011). "A survey on nature-inspired metaphors for pervasive service ecosystems". In : *International Journal of Pervasive Computing and Communications* 7 (3), p. 186–204.

Zeigler, Bernard P., Herbert Praehofer et Tag Gon Kim (2000). *Theory of Modeling and Simulation [Second Edition] : Integrating Discrete Event and Continuous Complex Dynamic Systems*.

Zhang, Shuai et al. (2010). "The comparison between cloud computing and grid computing". In : Computer Application and System Modeling (ICCASM), 2010 International Conference on. T. 11. IEEE, p. V11–72. ISBN : 1424472350.

Publications

Dutot, Antoine, Damien Olivier et Guilhelm Savin (29 june 2009). "Dagda, a load-balanced middleware to distribute Complex Systems simulations". In : *3rd International Conference on Complex Systems and Applications (ICCSA'09)*. France, p. 171–174. URL : http://litis.univ-lehavre.fr/~savin/bib/ICCSA2009.pdf.

— (juil. 2010a). "Collaboration and Competition in Boids Simulations with Predation". In : *Emergent Properties in Natural and Artificial Complex Systems (EPNACS) in European Conference on Complex System (ECCS) 2010*.

— (29th March - 1st April 2010b). "Swarm Intelligence to Distribute Simulations in Computational Ecosystems". In : *Swarm Intelligence Algorithms and Applications Symposium (SIAAS) in the Thirty Sixth Annual Convention of the Society for the Study of Artificial Intelligence and Simulation of Behaviour (AISB'10)*. De Montfort University, Leicester. URL : http://litis.univ-lehavre.fr/~savin/bib/AISB2010.pdf.

— (sept. 2011). "Centroids : a decentralized approach". In : *Emergent Properties in Natural and Artificial Complex Systems (EPNACS) in European Conference on Complex System (ECCS) 2011*.

Dutot, Antoine, Yoann Pigné et Guilhelm Savin (sept. 2011). "GraphStream Workshop". In : *Emergent Properties in Natural and Artificial Complex Systems (EPNACS) in European Conference on Complex System (ECCS) 2011*.

Dutot, Antoine et al. (oct. 2006b). "Pyocyanic Bacillus Propagation Simulation". In : *CoSSom workshop in European Simulation and Modeling conference (ESM)*. Toulouse, France. URL : http://litis.univ-lehavre.fr/~dutot/biblio/COSSOM2006.pdf.

Savin, Guilhelm (16-18 novembre 2009). "Dagda, un intergiciel pour la distribution dynamique de simulations de système complexe". In : *majecSTIC 2009*. France. URL : http://litis.univ-lehavre.fr/~savin/bib/majecSTIC2009.pdf.

Index

Résumé

Nous présentons dans ces travaux une contribution concernant la distribution de simulations de système complexe dans des environnements distribués ouverts. Nous considérons ces environnements comme des écosystèmes computationnels, dont nous décrivons les propriétés et les caractéristiques, dans lesquels évoluent, de par leur exécution, les simulations. Elles sont modélisées sous la forme d'un réseau d'interactions représenté à l'aide d'un graphe dynamique. En considérant les différentes dynamiques possibles, nous proposons un formalisme général représentant ces graphes, ainsi qu'une interface de programmation, GraphStream, permettant de les manipuler et de les étudier.

Le graphe dynamique est alors un sujet d'étude dans lequel nous recherchons des organisations, que nous suivons dans le temps, afin de minimiser les coûts de communication entre les machines et d'équilibrer la charge de calcul. Nous apportons une amélioration visant à réduire les oscillations des résultats de l'algorithme AntCo^2 utilisant des colonies de fourmis numériques qui, grâce à des mécanismes de compétition et de collaboration, détecte des organisations. La stabilité de ces dernières est déterminée par l'intermédiaire d'une heuristique de recherche distribuée et dynamique de centroïdes.

Un intergiciel est proposé permettant de distribuer de manière décentralisée et dynamique les simulations dans un écosystème computationnel en favorisant les organisations et en respectant l'équilibrage de charge.

Abstract

This work presents our contribution about the distribution of complex system simulations in open distributed environments. We consider these environments as computational ecosystems, of whose we describe properties and characteristics, wherein simulations will evolve, by their execution. An interaction network models these simulations, that we represent as a dynamic graph. Considering the different kinds of possible dynamics, we propose a global formalism that can be used to describe these graphs, along with a software framework, GraphStream, allowing their manipulation and their study.

The dynamic graph is then the object of a study wherein we are looking for organisations, following them through time, in order to minimise communications's costs between machines, and to balance the computation load. We bring an improvement aiming to reduce results's oscillations of AntCo^2, which is algorithm using digital ants colonies to detect organisations, through competition and collaboration mechanisms. Stability of these organisations is determined by a dynamic and distributed heuristic to find centroid of graph.

We propose a middleware allowing a dynamic and decentralized distribution of simulations in a computational ecosystem, by favouring organisations and respecting the load balancing.