Distribution dynamique adaptative à l'aide de mécanismes d'intelligence collective

Antoine Dutot

Université du Havre
LIH – Laboratoire Informatique du Havre

THÈSE

en vue de l'obtention du
Doctorat de l'Université du Havre

Spécialité
Informatique

Antoine DUTOT
9 Décembre 2005

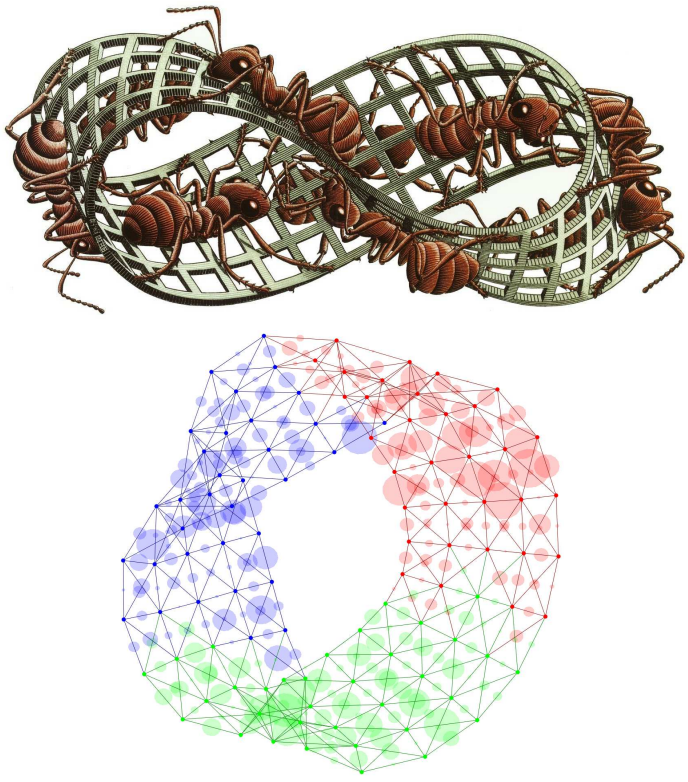
DISTRIBUTION DYNAMIQUE ADAPTATIVE À L'AIDE DE MÉCANISMES D'INTELLIGENCE COLLECTIVE

DÉTECTION D'ORGANISATIONS PAR DES TECHNIQUES DE COLLABORATION ET DE COMPÉTITION

Luca Maria GAMBARDELLA	IDSIA, Lugano	Rapporteur
Mohamed SLIMANE	LI, Tours	Rapporteur
Gérard DUCHAMP	LIPN, Paris	Examinateur
Habib ABDULRAB	PSI, Rouen	Examinateur
Frédéric GUINAND	LIH, Le Havre	Examinateur
Alain CARDON	LIP6, Paris	Directeur de thèse
Damien OLIVIER	LIH, Le Havre	Co-Directeur

DISTRIBUTION DYNAMIQUE ADAPTATIVE À L'AIDE DE MÉCANISMES D'INTELLIGENCE COLLECTIVE

DÉTECTION D'ORGANISATIONS PAR DES TECHNIQUES DE COLLABORATION ET DE COMPÉTITION



ANTOINE DUTOT

THÈSE DIRIGÉE PAR ALAIN CARDON, CO-ENCADRÉE PAR DAMIEN OLIVIER

Remerciements

Quand j'étais petit, durant ma licence puis ma maîtrise, quelques professeurs ont éveillé en moi l'envie de faire de la recherche. En premier lieu je tiens à remercier mon directeur de thèse, Alain Cardon.

Et puisque j'ai parcouru le chemin de cette thèse avec ceux qui étaient mes anciens professeurs, je tiens à remercier chaleureusement celui qui avec ses idées iconoclastes de fourmis (et d'autres encore plus passionantes) m'a réellement donné envie de faire de la recherche. Damien Olivier a co-encadré cette thèse, et l'a alimenté continuellement en perspectives motivantes. C'est «à cause» de lui que j'ai décidé de me lancer dans cette aventure, et c'est grâce à lui qu'elle à abouti.

Je tiens à remercier Luca Maria Gambardella et Mohamed Slimane pour avoir accepté d'être rapporteurs sur ma thèse. Voir sa thèse relue par un chercheur comme Luca Maria Gambardella qui est à la base des travaux sur les algorithmes fourmis est une chance que j'apprécie beaucoup. Les articles qu'il a écrit en collaboration avec Marco Dorigo et Gianni Di Caro sont la fondation du travail présenté dans ce mémoire. Merci à Mohamed Slimane pour sa relecture très attentive de mon mémoire et ses nombreuses remarques qui m'ont permis de l'améliorer. Merci très sincèrement à mes examinateurs pour le temps qu'ils ont bien voulu me consacrer, Habib Abdulrab et Gérard Duchamp.

Je voudrais aussi remercier Frédéric Guinand, pour m'avoir supporté durant toutes ces conférences et fait découvrir un mode de recherche non autiste.

Je suis aussi heureux d'avoir ici l'occasion de remercier Cyrille Bertelle, pour son aide, et pour m'avoir apris ce qu'est le travail 24 heures sur 24 (et aussi pour avoir choisi l'hôtel le plus éloigné qu'il se puisse d'une conférence).

Les thésards travaillant sur le même thème se sont regroupés en une équipe nommée GASP, dont je suis le «A». Je voudrais d'abord les remercier pour leurs prénoms, parce que GASP c'est joli. Ensuite il me faut avouer que la vie aurait été bien triste si Guillaume n'avait pas animé ce bureau que nous partageons à six. Avec qui aurais-je pu parler de choses puériles si tu n'étais pas là ? Il faut beaucoup de professionnalisme pour savoir en parler correctement. Merci Sylvain, pour les câlins que tu as prodigué à toute l'équipe et, malgré mes réticences initiales, à moi aussi. Il y a un énorme vide sur le côté gauche de mon bureau. Enfin merci Pierrick, parce que geek c'est plus drôle à plusieurs, mais des amis geek c'est encore mieux.

Sortons de l'équipe GASP, mais restons entre thésards. Merci Jérôme, pour ta mémoire extraordinaire, sans toi j'aurais manqué de nombreuses choses (et puis merci pour le LaTeX). Merci Yoann, parce que tu t'en défend, mais tu es quand même aussi un peu geek, et ca fait du bien d'avoir un geek dans son bureau. Merci Denis pour la folie de bonne humeur. Merci Emna pour le rayon de soleil, et Majed pour le coup de trompette tous les matin. Merci Samy parce que tu es bizarre, mais sous le bizarre il y a un type sympa. Merci Rolland d'avoir adopté et entretenu mon ancien bureau.

Véronique, merci d'avoir relu cette thèse, et merci pour ces discussions où maître Cappelo en aurait appris.

Merci Claire, vous êtes tout simplement indispensable.

Merci Jean-Luc pour la classe. Tu es le prochain James Bond. Merci Laurent, pour tes goûts en portables.

Merci Nadine, pour tous les «tracas» que je vous ai posé avec votre imprimante, et pour m'avoir aidé si gentiment.

Merci Mina et Claude pour les soirées crêpe et ces premières années de thèse.

Merci à tous les gentils et gentilles matheux, biologistes et mécaniciens que l'on croise constamment dans nos pérégrination de laboratoires à deux étages. Ça crée des liens.

Merci à Nicole, André, Raphaël, parce que je vous aime.

Table des matières

Table des figures

Liste des algorithmes

Première partie

Concepts

Chapitre 1

Introduction

Sommaire

1.1 Le problème

La figure 1.1 montre le calvaire de Sisyphe. Sisyphe était le fils d'Éole et d'Énarété. Il fonda Corinthe et les jeux Isthmiques. Il fut condamné, pour avoir trompé les dieux et en particulier Thanatos, à hisser éternellement une pierre jusqu'en haut d'une colline alors qu'elle redescendait chaque fois avant de parvenir à son sommet.

FIG. 1.1: Sisyphe. Titien (1548-49), Musée du Prado.

Le problème auquel nous nous intéressons peut sembler similaire au destin de Sisyphe. Il consiste à distribuer de manière dynamique les constituants en perpétuelle réorganisation d'une application informatique, tout en s'adaptant aux reconfigurations continues de cette dernière et de son support d'exécution. Le problème posé tient dans la dynamique de l'application à distribuer. Alors que le calcul de distribution est quasi terminé, que se passe-t-il si les données du problème changent ?

Commençons par définir le type d'applications auquel nous nous intéressons par un exemple concret. Plusieurs ressources de calcul, des micro-ordinateurs, des supercalculateurs, des PDAs, des ordinateurs portables équipés de Wi-Fi, peut-être un téléphone portable, participent à l'exécution d'une application. Il est en général nécessaire d'utiliser plus d'une machine pour exécuter cette application parce qu'elle est trop difficile à mettre en œuvre pour une unique machine. Cette application est prévue pour

être exécutée de manière distribuée, et est découpée en de multiples éléments distincts capables de communiquer entre-eux que nous appellerons *entités*. Sur la figure 1.2 ces entités sont représentées par des cercles dans la partie «représentation logicielle». Cette vue d'une application distribuée est celle que nous adopterons par la suite, c'est-à-dire un ensemble de ressources de calcul contenant des entités en interaction. En parallèle, la figure montre la vue physique de l'environnement d'exécution de l'application, constitué de machines de types divers reliées par un réseau de communication.

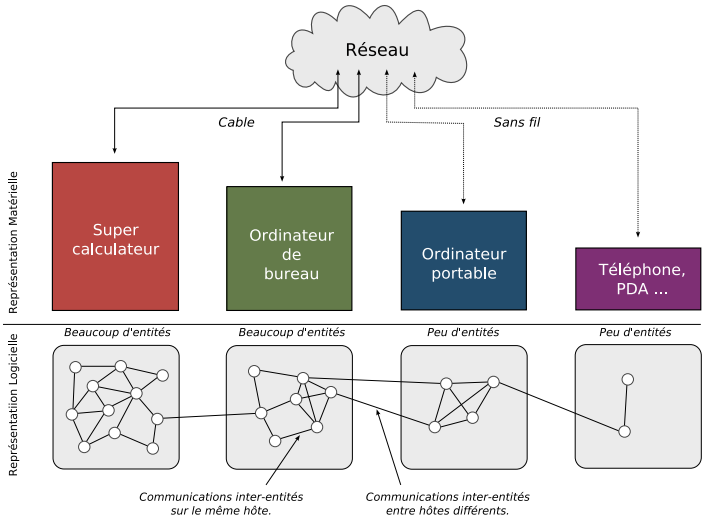


FIG. 1.2: Exemple d'application répartie nécessitant une distribution dynamique adaptative.

L'application démarre et ses entités apparaissent, puis par la suite disparaissent et apparaissent continuellement sur diverses machines, suivant les besoins de l'application. Ces entités communiquent entre elles, souvent en privilégiant les échanges avec un groupe d'entités plutôt qu'avec un autre, et ceci de manière imprévisible. Sur la figure, ces communications sont présentées par des arcs reliant les cercles.

Cette application est tout à fait en état de fonctionner ainsi sans autre intervention, en plaçant par exemple les entités sur une machine libre dès qu'elles apparaissent. Cependant deux problèmes majeurs empêchent son exécution optimale :

1. Les communications passent indistinctement par le réseau entre les ressources de calcul ou directement d'entité à entité si ces dernières se trouvent sur le même hôte. Or rien n'est fait pour prendre en considération les préférences d'entités à communiquer avec d'autres entités. Lorsque la communication passe par le réseau, l'information peut traverser de nombreux liens physiques, réseau filaire cuivré, backbone en fibre optique, voies hertziennes, liens Wi-Fi, *etc*. Cela peut introduire des latences considérables par rapport à une communication directe inter-entités situées sur la même ressource de calcul. C'est le problème de la *charge réseau*. La minimiser consiste à faire en sorte que des entités qui communiquent beaucoup soient regroupées sur un hôte.
2. Les entités n'apparaissent et ne disparaissent pas de manière équilibrée sur toutes les ressources de calcul. De même, les hôtes qui deviennent disponibles au cours de l'exécution ne sont pas forcément directement utilisés. Il est donc possible

qu'une ressource soit surchargée alors qu'une autre est sous employée. C'est le problème de la *charge machine*. L'équilibrer consiste à tenter de placer une charge proportionnellement à la capacité de calcul de l'hôte.

Une première amélioration est donc de trouver une méthode permettant de distribuer au mieux les entités de l'application sur toutes les ressources de calcul à disposition. Mais si distribuer de manière équitable les entités sur chaque hôte est une amélioration importante, elle peut être contrebalancée par les latences introduites par la communication réseau. Il devient donc nécessaire de tenir compte des «préférences d'une entité à communiquer avec un groupe d'entités» afin d'apporter une seconde amélioration permettant de minimiser la charge réseau.

Pour optimiser le temps d'exécution de notre application, nous devons donc travailler sur deux critères, la charge machine et la charge réseau, qui sont malheureusement antagonistes. En effet, placer toutes les entités sur une ressource de calcul minimise totalement la charge réseau mais maximise complètement la charge machine, alors que distribuer toutes les entités de manière parfaitement équitable sur les ressources minimise la charge machine mais peut très probablement augmenter la charge réseau. Nous cherchons donc un compromis entre ces deux critères. La figure 1.2 montre une telle répartition, pour laquelle chaque machine est chargée suivant ses possibilités avec un minimum de liens de communications inter-entités se faisant par le réseau de communication.

Comme nous le verrons dans la partie II page 87, ce problème a déjà été abordé de manière importante pour des entités non communiquantes en se préoccupant uniquement de la charge machine, et largement, bien que de façon moins conséquente, lorsque la charge réseau entre en compte. Par contre, peu de méthodes savent gérer la dynamique de l'application. En effet, dans l'exemple ci-dessus, l'environnement d'exécution correspond à une grille de calcul sur laquelle des machines (à ne pas confondre avec les entités décrites précédemment) deviennent disponibles ou disparaissent, et ceci de manière imprévisible. De plus, l'application elle même est source de dynamique, en générant et retirant constamment des entités. Enfin, les préférences de communications dont nous devons tenir compte évoluent au cours de l'exécution. Pour résumer :

1. l'environnement de calcul est dynamique ;
   – les ressources de calcul apparaissent et disparaissent ;
2. l'application est dynamique :
   – ses composants apparaissent, évoluent et disparaissent ;
   – les interactions entre ces composants font de même ;
3. les deux dynamiques évoluent de manière non déterministe, la première parce qu'elle est plongée dans le monde réel, la seconde parce que nous n'avons aucune information sur elle.

Face à ce dernier problème, la tâche peut effectivement sembler une gageure comparable à la punition de Sisyphe : une approche naïve devra en effet calculer une distribution pour une application dont l'état change constamment, et pour laquelle la distribution calculée, arrivé en haut de la colline, n'est probablement plus adéquate dès la fin du calcul. Il faudra alors recommencer du début, le rocher ayant dévalé la pente.

La méthode proposée dans cette thèse, nommée *AntCO*^{2}, s'inspire du comportement collectif d'insectes sociaux — les fourmis — pour résoudre le problème de la

distribution dynamique adaptative. Ce travail prend racine dans les travaux sur l'ACO (*Ant Colony Optimization*) de [Dorigo, 1996] et plus largement dans le domaine de l'intelligence collective.

Le nom *AntCO*^{2} est l'acronyme de «Ant Competitive Colonies», c'est-à-dire «colonies de fourmis en compétition». L'exposant signifie «deux fois CO» pour «COmpetitive» et «COlonies», et met en avant l'un des points originaux de la méthode proposée : les fourmis ne font pas que coopérer, elles entrent aussi en compétition. Nous verrons par la suite en quoi cela peut apporter un avantage lors de l'équilibrage de charge.

L'intérêt que nous portons aux fourmis tient dans la constatation que ces dernières, malgré un environnement en perpétuel réorganisation, et des besoins changeants, sont en mesure de maintenir l'existence d'une véritable société, et d'entretenir des constructions qui leur sont pérennes. De plus la relative simplicité de leur comportement[1], met leur intelligence à notre portée[2].

1.2 Organisation du document

La première partie s'intéresse aux concepts dont nous aurons besoin tout au long de ce manuscrit. La distribution d'applications est d'abord abordée avant de décrire un certain type d'applications que sont les simulations. Être en mesure de distribuer ces dernières est l'objectif principal de notre travail. Ces élément posés, nous nous intéresserons à un domaine qui peut sembler totalement déconnecté des précédents, les approches collectives. Nous verrons cependant que ces dernières sont souvent implicitement distribuées, et permettent de résoudre des problèmes similaires au nôtre. Enfin nous décrirons plus en détail les approches collectives à base de fourmis numériques, thème principal de ce manuscrit. Nous étudierons les mécanismes qu'elles utilisent pour mener à bien leurs tâches, et détaillerons quelques expériences et modèles significatifs à leur sujet.

La second partie fait écho à la problématique de distribution des applications informatiques en la complétant par un état de l'art des grandes tendances dans ce domaine (p. 87). Cette partie est majoritairement composée par la description des méthodes de distribution dites *statiques*, c'est-à-dire pour lesquelles les entités, les parties de l'application, n'évoluent pas, ou le font de manière déterminée. Ces dernières sont peu adaptées au problème qui nous occupe, mais sont majoritairement abordées dans la littérature. Nous distinguerons les approches pouvant gérer la charge réseau en même temps que la charge machine, de celles ne traitant que le second point. Les approches moins nombreuses de distribution dynamique sont ensuite énumérées et détaillées.

Les deux parties précédentes fournissent les concepts nécessaires à la présentation de notre modèle. La troisième partie présente donc ce dernier, d'abord informellement en précisant ses ressemblances et différences avec les modèles précédents l'ayant inspiré puis plus précisément en décrivant son comportement. Il est ensuite détaillé du point de vue de son utilisation, avant d'aborder sur ce sujet quelques approches complémentaires.

[1] Bien qu'un organisme tel que celui de la fourmi soit déjà d'une complexité stupéfiante.

[2] Cependant là aussi il est permis de s'interroger.

La quatrième partie décrit les expérimentations menées sur ce modèle. Viennent en premier des tests simples sur des applications statiques afin de cerner le modèle. Des applications plus complexes sont ensuite étudiées, toujours en statique. Enfin plusieurs séries de tests sur des applications dynamiques sont données. Cette partie présente aussi quelques pistes et travaux complémentaires menés à bien durant cette thèse. Ce sont principalement des modifications du modèle pour d'autres utilisations.

1.3 Cadre général

Ce travail prend place dans plusieurs contextes. D'une part il est en contact avec les simulations qu'il se propose de distribuer qui sont utilisées dans des domaines très divers, d'autre part la méthode qu'il décrit prend son inspiration en biologie, enfin il s'insère dans un projet à plus large échelle mené par le thème Modèles Informatiques du Vivant (MIV) au sein du Laboratoire d'Informatique du Havre (LIH).

MIV peut se voir suivant deux angles :

1. L'inspiration à partir du vivant pour produire des modèles et des outils informatiques utiles à d'autres tâches. Il s'agit ici d'extraire des modèles de l'observation de la nature pour les apliquer à des tâches typiquement informatiques (qu'elles soient ou non du domaine informatique).
2. La création d'outils et de simulation du vivant pour mieux le comprendre.

Ces deux points s'auto-référencent, l'un ayant besoin de l'autre, les avancées de l'un étant utiles à l'autre (figure 1.3). *AntCO*^{2} en est un exemple. L'algorithme s'inspire du comportement de fourmis naturelles, et définit un modèle utilisant des fourmis numériques pour une tâche informatique : la distribution dynamique d'applications réparties. En retour *AntCO*^{2} est utilisé pour aider l'exécution de simulations d'écosystèmes qui permettront de mieux comprendre leur contrepartie naturelle.

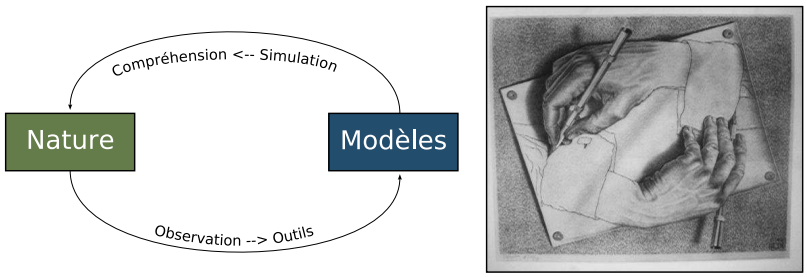


FIG. 1.3: Modèles Informatiques du Vivant (MIV) : S'inspirer du vivant pour l'informatique et utiliser l'informatique pour simuler le vivant (figure de droite, M.C. Escher «Drawing Hands», Lithographie, 1948).

1.4 Une autre vue du manuscrit

Les plans d'une fourmilière sont très variables d'un espèce de fourmi à l'autre. Certaines espèces n'en construisent d'ailleurs pas, leur préférant des nids temporaires

installés dans les arbres comme les fourmis *Tisserandes*, ou même des abris très impressionnants formés par leur propre corps, comme les fourmis *Magnans*, créant ainsi une fourmilière équipée d'une véritable frontière défensive. En règle générale, cependant, la colonie élit domicile sous terre, en construisant un dôme permettant la régulation de la température au sein du nid.

C'est le cas de la fourmi rousse commune. Le dôme est en général construit autour d'une vieille souche d'arbre, dans l'argile ou le sable, et est constitué de nombreux niveaux dont l'humidité, la luminosité et la température varient en fonction de leur profondeur. Les niveaux les plus profonds abritent la chambre royale, où la reine, captive, pond. Cependant le domaine des fourmis s'étend bien au delà de la fourmilière. De nombreuses pistes sont lancées aux alentours afin de rechercher la nourriture, du matériel, parfois aussi pour mener des guerres ou sceller des alliances avec des colonies voisines.

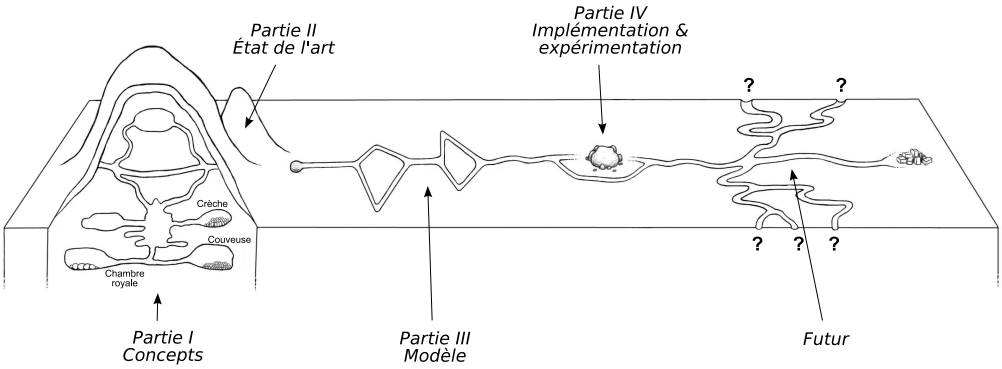


FIG. 1.4: Plan du manuscrit.

La figure 1.4 est un schéma explicatif du parcours de la fourmi dans un environnement tel que ce manuscrit. C'est la vue du myrmécologue. Cependant, ce document présentant une espèce de fourmi numérique colorée, il est intéressant de se baisser quelques instants pour comprendre la vue qu'ont ces dernières du document lorsqu'elles le parcourent. La figure 1.5 page suivante esquisse ce que les *FormicaNumerica* en perçoivent.

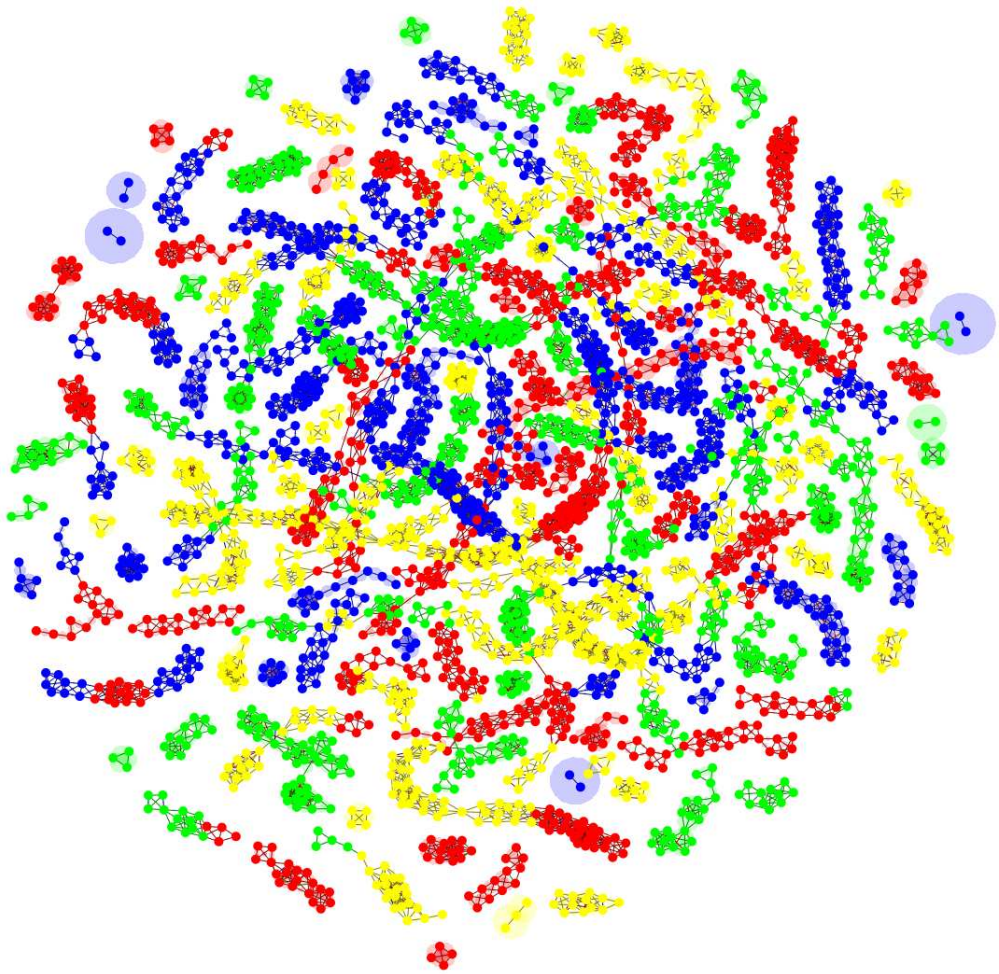


FIG. 1.5: Plan du manuscrit coloré par des fourmis numériques (pour information $r_1 = 0.06$ et $r_2 = 0.915$).

Chapitre 2

Distribution

Sommaire

Si un homme peut creuser un trou en une minute, alors soixante hommes peuvent creuser ce trou en une seconde ?

Anonyme

Le but, lorsque l'on distribue une application est de gagner du temps. Certains calculs ne peuvent pas être entrepris pratiquement sur une unique machine et la difficulté des tâches demandées s'accroît en parallèle de la puissance disponible. Dans cette optique, on démultiplie donc le nombre de processeurs, le nombre de machines, afin de multiplier en conséquence, on l'espère, la vitesse d'exécution de l'application.

Nous cherchons à mettre au point une méthode permettant la distribution dynamique adaptative, et ceci sur des systèmes distribués. Les deux premières sections de ce chapitre décrivent donc la partie physique, les machines, sur lesquelles on peut distribuer une application ainsi que la partie logicielle de la distribution, c'est-à-dire le

support matériel et logiciel, ainsi que sa formalisation, que nous utiliserons par la suite dans la description de notre méthode.

Enfin, nous nous arrêterons sur la question suivante : peut-on réellement augmenter la vitesse de calcul en conséquence du nombre d'unités de calcul ? La dernière section reprend l'aphorisme anonyme de ce début de chapitre, et explique qu'il n'est en pratique quasiment jamais réalisable. On aborde un problème d'organisation : comment les hommes vont-ils se coordonner pour creuser le trou ? Ne vont-ils pas par exemple se gêner ? Ce sont les questions majeures que nous pouvons transposer aux systèmes distribués : il existe une relation conflictuelle entre le nombre de processeurs et les communications que cette démultiplication induit. Cette problématique est essentielle en parallélisme et dans les systèmes distribués, et constitue le fil conducteur de la méthode présentée dans cette thèse.

2.1 La partie physique d'un système distribué

Un système distribué est un environnement où différents processus s'exécutent, se partagent des ressources, et collaborent (ou entrent en compétition) pour réaliser un objectif éventuellement commun. Nous allons maintenant décrire cet environnement dans un premier temps dans sa partie physique.

2.1.1 Parallélisme

On touche en fait à deux domaines, très proches l'un de l'autre, le *parallélisme* (PRAM) et les *systèmes répartis* (grappes de machines).

On peut placer le second comme une extension du premier, ce qui nous amène à nous demander ce qu'est le parallélisme. Suivant la Wikipedia[1] :

> **Parallélisme** :
> Parallel computing is the simultaneous execution of the *same task* (split up and specially adapted) on multiple processors in order to obtain faster results. *Le calcul parallèle consiste en l'exécution simultanée de la même tâche (scindée et adaptée spécifiquement) sur de multiples processeurs de façon à obtenir des résultats plus rapidement.*

Les points importants de cette définition sont les multiples processeurs, la simultanéité, ainsi que le but à atteindre : accélérer les calculs.

Le parallélisme, en tant que discipline, est en étroite relation avec la création de machines parallèles, c'est-à-dire d'ordinateurs uniques possédant plus d'un processeur d'instructions, ainsi que des interconnexions entre ces processeurs effectuant le partage des ressources disponibles sur la machine, telles que la mémoire ou l'espace disque. Le parallélisme a donc trait aux techniques de création de *machines parallèles*. Mais

[1] Selon ses propres termes, la «Wikipedia est un projet d'encyclopédie gratuite, écrite coopérativement et dont le contenu est réutilisable selon les conditions de la Licence de documentation libre GNU». Ce projet est particulièrement critiquable sur ce qui fait aussi son point fort : la liberté totale d'édition des articles, un bon article pouvant être réécrit par quelqu'un de moins informé ou compétent. L'article cité est celui de la version anglaise, tel qu'on pouvait le trouver à l'adresse `http://en.wikipedia.org/wiki/Parallel_computing` le 17/05/2005.

c'est aussi et surtout la recherche et l'exécution d'algorithmes pouvant s'exécuter sur ces machines. Ce dernier point concerne à la fois les méthodes pour paralléliser un algorithme existant ainsi que la création d'algorithmes parallèles dédiés.

Historiquement, le parallélisme se cantonne aux machines parallèles, c'est-à-dire à ces machines spécifiques vues comme un tout possédant une partie partagée (mémoire, disques, *etc*.) et une partie multiple (plusieurs processeurs d'instructions). Bien entendu la conception de telles machines est elle-même un domaine de recherche, et de nouveaux modes d'interconnexion, de répartition des ressources, partagées ou non entre les processeurs, ainsi que beaucoup d'autres points d'architecture sont constamment développés.

Cependant, il n'y a pas de raison dans la définition donnée ci-dessus de s'arrêter aux machines uniques possédant plusieurs processeurs, une interconnexion spécifique, et une répartition des ressources spécifiques. Les processeurs peuvent se trouver dans une machine propre, connectée elle-même à d'autres sur un réseau (ou une interconnexion de réseaux). En d'autres termes, peu importe le réseau d'interconnexion entre les processeurs d'instructions et la façon dont les ressources sont partagées. Un cluster[2], un réseau d'ordinateurs, un super-calculateur, tant que l'on reste au niveau de la définition donnée ci-dessus, et bien qu'ayant des architectures différentes, peuvent se simplifier en *un ensemble de ressources de calcul interconnectées entre elles, ayant accès à des ressources de stockage.*

Cette simplification a un prix : on ne prend pas en compte les caractéristiques spécifiques de certaines architectures (quand ces caractéristiques ne sont pas transparentes). Cependant elle nous permet de faire le pont avec le domaine du calcul distribué, qui lui trouve ses racines dans le développement des réseaux de communication.

2.1.2 Calcul distribué

On peut définir le calcul distribué (distributed computing)[3] :

> **Calcul distribué (Wikipedia)** :
> Distributed computing is the process of agregating the power of several computing entities to collaboratively run a single computational task in a transparent and coherent way, so that they appear as a single, centralized system. *Le calcul distribué est l'agrégation de la puissance de calcul de plusieurs unités de calcul pour exécuter une tâche unique de manière transparente et cohérente, de façon à ce qu'elles apparaissent comme un unique système centralisé.*

On cherche toujours à exécuter une unique tâche coupée en parties, mais on ne précise plus si ces dernières s'exécutent sur des processeurs ou des machines à part entière, on ne fait plus référence, en fait, à une architecture (bien que dans le domaine des systèmes distribués on sous-entend quasiment toujours l'existence d'un réseau de machines), le terme *computing entity* pouvant identifier un processeur comme une ma-

[2] Réseau de machines dédiées, contrairement au réseaux en général où les machines peuvent être utilisées à toutes fins.

[3] Version anglaise de la page `http://en.wikipedia.org/wiki/Distributed_system` à la date du 17/05/05.

chine.

La méthode développée dans cette thèse se veut indépendante d'une architecture donnée, c'est pourquoi nous parlerons par la suite de *ressource de calcul* pour toute entité capable d'exécuter les instructions d'un programme séquentiellement[4], et ceci, que la ressource de calcul soit au sein d'un supercalculateur, d'un cluster, ou autres. C'est pourquoi on ne prendra pas en compte les architectures d'interconnexion de ces ressources de calcul, ainsi que les interconnexions avec d'autres types de ressources (mémoire, stockage, *etc.*).

D'ailleurs, les systèmes distribués se détachent le plus souvent de cette architecture d'interconnexion, et évoluent de plus en plus vers une configuration en *grille*. Ce terme désigne des systèmes hétérogènes, dans lesquels on utilise diverses ressources de calcul et de stockage en fonction de leur disponibilité. Parfois les grilles sont dédiées, mais souvent il s'agit de machines dont on a décidé qu'elles pouvaient faire partie d'une grille et qui fournissent des cycles de calcul aux applications lorsqu'elles ne sont pas chargées. Le terme anglo-saxon *grid computing* est principalement utilisé, et n'a pas encore de traduction généralisée en France.

Il existe plusieurs acceptions pour le terme grid-computing. Voici la plus commune :

> **Grid computing** :
> Modèle permettant de résoudre des problèmes de calcul massifs en utilisant les ressources disponibles (cycles CPU et stockage disque) d'un nombre important de ressources de calcul vues comme un cluster virtuel compris dans une infrastructure de télécommunications distribuée.

Ce qui rend le grid computing différent d'un simple cluster réside dans l'hétérogénéité des domaines administratifs géographiques que la grille couvre. On le voit dans cette définition, le réseau est «virtualisé», sa topologie inconnue. Ces systèmes introduisent aussi une autre nouveauté dans les réseaux de ressources de calcul : la dynamique. Des ressources peuvent apparaître et disparaître continuellement sur la grille et l'application qui s'y exécute doit être en mesure de prendre en compte ces événements.

Auparavant, seuls les problèmes de tolérance aux pannes pouvaient introduire de tels événements. En effet même dans un cluster où les ressources de calcul sont clairement identifiées, il est possible qu'une de ces ressources disparaisse pour une cause inconnue. Dans certains cas, il suffit d'arrêter l'application pour la relancer en prenant en compte les modifications du réseau, cependant, souvent, il est primordial que l'application continue de s'exécuter. Un tel service peut être implémenté de manière logicielle, grâce à de la duplication de code [Marin, 2003]. C'est cependant un domaine qui sort du sujet exposé ici, mais nous considérerons que sur les systèmes distribués où les simulations s'exécutent un tel service est disponible.

[4]On ne prendra pas en compte non plus le micro-parallélisme qui introduit divers cœurs dans un processeur, nous les considérerons comme plusieurs ressources de calcul.

2.1.3 Le système distribué matériel

La figure 2.1 propose une classification des approches évoquées ci-dessus. Le parallélisme concerne toutes les approches visant à accélérer un calcul en le coupant en parties pour l'exécuter sur diverses ressources de calcul. Ceci peut être fait sur des machines *centralisées* tels des supercalculateurs massivement parallèles, ou sur des systèmes répartis. Ces derniers peuvent alors se scinder en deux groupes, les clusters dédiés, physiquement localisés en des points précis, connus et administrés de manière globale, et les grilles qui représentent un cluster virtuel, physiquement réparti couvrant plusieurs domaines administratifs.

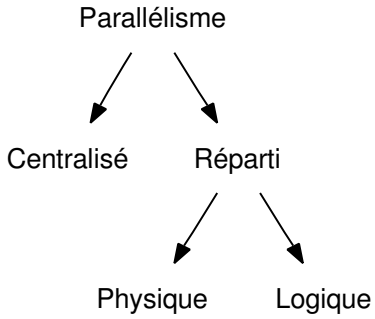


FIG. 2.1: Classement des approches.

En ne prenant pas explicitement en compte l'architecture du système informatique sur lequel nos applications seront distribuées, nous pouvons donc considérer un modèle physique très simple :

1. chaque ressource de calcul peut communiquer avec n'importe quelle autre via un réseau dont la topologie est inconnue, en un mot le réseau de ressource de calcul physique est connexe ;
2. les ressources de calcul peuvent apparaître ou disparaître naturellement au sein de ce réseau pendant que l'application s'exécute ;
3. si une ressource disparaît à la suite d'une panne, on se repose sur un mécanisme de duplication des parties logicielles s'y exécutant, ceci afin de maintenir la connexité du réseau ;
4. les ressources de calcul sont hétérogènes, de puissance différente, et ayant accès à des ressources différentes.

Voici donc les définitions que nous utiliserons pour la partie matérielle du système distribué :

> **Ressource de calcul** :
> Tout élément informatique permettant d'exécuter des instructions qu'il soit au sein d'une machine parallèle, dans un processeur multicœur, ou dans une machine unique.

> **Éléments physiques du système distribué** :
> Réseau dynamique connexe de ressources de calcul hétérogènes.

2.2 La partie logicielle d'un système distribué

Nous avons décrit l'*exécutant* du système distribué, et l'avons réduit aux seuls éléments significatifs pour notre travail, permettant d'appliquer notre méthode à un grand nombre d'architectures. Il nous reste à définir ce qui y est *exécuté*.

2.2.1 Composants d'un système distribué

Dans le chapitre 1 de *Distributed Systems* [Mullender, 1993], les systèmes distribués sont définis ainsi :

> **Système distribué (1993)** :
> Several computers doing something together. *Plusieurs ordinateurs faisant quelque chose ensemble.*

Cette définition succincte définit une granularité pour l'application qui se restreint aux ressources de calcul (ici décrites comme des ordinateurs). On ne connaît pas les détails d'organisation au delà du seuil de l'ordinateur. Beaucoup d'approches en parallélisme considèrent que la création d'un algorithme parallèle revient à le découper en autant de parties qu'il y a de processeurs. Cette approche prévaut dans une vision procédurale, où l'on considère les données et l'algorithme comme des entités séparées.

La Wikipedia[5] donne une autre définition :

> **Application distribuée (2005)** :
> An application that consists of components running on different computers concurrently. These components must be able to communicate and be designed to operate separately. *Une application constituée de composants s'exécutant sur différents ordinateurs de manière concurrente. Ces composants doivent être capables de communiquer et opérer séparément.*

La notion de composant est introduite. De fait, il peut y avoir plusieurs composants sur une même ressource de calcul. Il s'agit d'une vision plus fine que celle décrite plus haut, l'application à paralléliser est non seulement découpée en parties correspondant à chaque ressource de calcul, mais de plus ces parties sont elles mêmes scindables en composants.

On peut en fait rapprocher cette définition de la notion d'*objet* dans les langages de programmation et de *Système distribué à objets*, de plus en plus répandue, où les composants sont bien identifiés et prennent place dans un paradigme de programmation éprouvé, qui de plus en plus remplace la vision procédurale. Dans ces systèmes, une découpe de l'application est souvent déjà introduite au niveau de ses composants de base. Cette découpe ne correspond pas forcément à une distribution possible, c'est-à-dire que l'étape de parallélisation reste à faire, mais généralement la granularité des composants permet un équilibrage de la charge.

[5] Version du 18/05/2005

Nous utiliserons le terme *entité* pour décrire une partie atomique, un composant, d'une application. Ce terme très générique pourrait être remplacé suivant les cas par *tâche*, *processus*, *objet*, *objet actif* ou encore *agent*, la liste n'est pas exhaustive. Il s'agit des parties que l'on ne veut pas scinder plus pour les distribuer. Nous nous placerons dans le cas où la granularité de l'application est suffisamment petite pour que plusieurs composants s'exécutent en parallèle sur une seule machine.

> **Entité** :
> Élément atomique distribuable, identifié et considéré indivisible au sein d'une application répartie.

Une application distribuée sera donc constituée de nombreux composants tous identifiés et adressables individuellement, capables d'être plusieurs sur une ressource de calcul unique. On se départit d'approches où les données et les algorithmes sont séparés, où l'algorithme est d'abord coupé en parties plus ou moins indépendantes chacune placée sur une ressource de calcul, puis les données partagées et envoyées en entrée pour obtenir des sorties. Ici le modèle couple les données aux traitements dans un concept objet, et c'est l'objet qui est distribué.

2.2.2 Adressage

Le fait que les composants soient adressables est particulièrement important. Les services d'annuaires, aussi appelés services de nommage, sont un point pivot d'un système distribué. Ces derniers sont parfois eux-mêmes distribués ou redondants, afin d'éviter un service global, véritable goulot d'étranglement pour une application faisant souvent référence à elle-même, et ne supportant bien entendu pas les pannes. Comme lors de la description de la partie physique du système distribué, nous prendrons le parti que chaque entité logicielle de l'application est accessible aux autres entités par le biais d'un service d'annuaire afin d'entrer en communication.

2.2.3 Migration

Certains systèmes distribués permettent la migration des entités d'une machine à l'autre. Une répartition de la charge nécessite souvent de la mobilité, dans le cas de composants qui s'exécutent en continu. Certains auteurs appellent souvent des entités dont la durée est limitée dans le temps des *tâches*. Nous ne différencions pas les tâches des entités, une entité pouvant avoir une durée limitée et connue, ou non. La caractéristique importante à considérer est qu'une entité peut avoir une durée bornée, connue précisément, alors que d'autres ne donnent aucune indication sur une fin possible.

Les entités dont la durée est connue n'ont généralement pas besoin de migrer, et en effet dans ce cas, un bon placement peut être suffisant. Les entités dont la durée est inconnue, en revanche, auront nécessité de migrer si l'on veut être en mesure d'effectuer une répartition de la charge sur les diverses ressources de calcul à disposition, car elles peuvent s'exécuter potentiellement longtemps. Le modèle que nous présentons requiert donc de la mobilité afin de gérer l'imprévisibilité dans les durées d'exécution des divers éléments.

Il existe plusieurs formes de migration qui dépendent souvent du modèle de programmation utilisé, procédural ou objet. Nous nous plaçons dans une approche objet,

où la migration se fait sur ces derniers, et peut être de deux types, faible ou forte :

faible L'entité à migrer incorpore dans son comportement des mécanismes de migration explicites. Elle s'arrête en un point donné de son code et reprend au même point. Elle nécessite un effort particulier de programmation dans l'application elle-même pour être utilisée. C'est une méthode de migration que l'on peut qualifier de *coopérative*, car il faut que l'entité donne son accord pour migrer, et on peut attendre cet accord indéfiniment en cas de problème.

forte L'entité à migrer n'a pas trace explicite dans son comportement et son code de mécanismes de migration. Le système d'exécution de l'entité est capable de la faire migrer «à son insu» de manière transparente pour cette dernière. On peut qualifier un tel système de *préemptif*, car bien que la décision de migration puisse venir de l'entité elle-même, cette dernière ne peut l'empêcher. Qui plus est, elle peut se trouver à n'importe quelle position dans son code et dans son comportement, et elle reprendra son exécution sur l'hôte distant exactement à ce point arbitraire.

De plus, les migrations peuvent entraîner le déplacement de ressources associées à des entités, ou utilisées par ces entités.

La méthode que nous présentons supporte les deux formes de migration, faible et forte, ainsi que les migrations de ressources associées, car en pratique seule l'application est responsable, en dernier lieu, de la migration de ses entités. La méthode de distribution n'aura à se soucier que du coût éventuel d'une telle migration.

2.2.4 Interactions

Nous utilisons le terme *interaction* pour décrire une relation entre deux ou plusieurs entités. Elles se traduisent souvent par des communications, mais pas uniquement. Par exemple il y a interaction lorsqu'un animal d'une simulation d'écosystème entre dans le champ de vision d'un autre.

Certaines interactions sont plus importantes que d'autres (une fois encore en terme de communication, d'âge ou simplement pour le modèle utilisé), une interaction peut donc être (et sera souvent) valuée. Par exemple s'il y a communication entre deux entités, cette valuation représentera le débit de communication, le nombre de messages, la bande passante utilisée, *etc*. toute mesure faisant sens pour l'application. Pour une entité entrant dans le champ de vision d'une autre, la valeur pourra être l'âge du lien.

> **Interaction** :
> Relation quantifiée entre deux entités.

Dans un système à entités (objets, objets actifs, processus, acteurs, agents, *etc*.), il y a forcément interaction, que les entités soient sur la même ressource de calcul ou non, il est probablement utile de faire une distinction dans les modes de communications.

> **Communication effective** :
> On appelle communication effective, toute communication qui utilise effectivement le réseau d'interconnexion entre les ressources de calcul.

En effet, de telles communications sont bien plus coûteuses en temps que les

simples passages de messages (comme par exemple des appels de méthodes) entre objets sur la même ressource de calcul. Qui plus est, si on peut approximativement borner une communication en interne, il est quasi impossible de donner une date d'échéance pour une communication sur un réseau, à plus forte raison s'il est hétérogène. Des latences de toutes sortes peuvent survenir.

Enfin, nous utilisons le terme *réseau d'interaction* pour définir l'ensemble des relations entre les entités d'une application, ce réseau pouvant souvent être modélisé comme un graphe que nous noterons G ou $G(t)$ s'il varie avec le temps.

> **Réseau d'interactions** :
> Ensemble des relations entre toutes les entités d'une application représenté par un graphe G s'il est statique ou $G(t)$ s'il est dynamique.

Notre vision de l'application répartie sera donc un tel réseau d'interactions modélisé par un graphe possiblement dynamique. Ce graphe sera décrit en détail à la section 9.2.1 page 119. Il faut noter que ce graphe est complètement dé-corrélé du réseau de ressources de calcul.

2.2.5 Parallélisme et réseaux d'interaction

La modélisation par graphe d'entités en interaction est commune. Le parallélisme utilise des graphes d'entités afin de réaliser :

1. des *ordonnancements*, si le graphe est orienté, c'est-à-dire si des tâches dépendent d'autres ;
2. des *placements* si les tâches sont indépendantes.

Cependant, ici, nous nous intéresserons uniquement au placement en considérant des entités dont la durée d'exécution n'est pas connue à l'avance. Nous verrons que les interactions sont importantes et considérées, non pas sous l'angle direct de la dépendance entre tâches avec un algorithme prenant des entrées et ayant des sorties, mais plutôt sous l'angle d'entités en communication, c'est-à-dire avec de très nombreuses entrées et sorties, au début de leur exécution, à la fin, mais aussi durant toute leur durée de vie.

Cette différence importante est explicitée par la figure 2.2 page 39. D'un côté (2.2(a)) on modélise l'application par un graphe de tâches dont certaines dépendent d'autres. On ne peut placer en parallèle que les tâches indépendantes les unes des autres. L'approche est souvent fonctionnelle, avec une tâche par processeur, chaque tâche étant une fonction, et les données sont communiquées d'une tâche à l'autre au début et à la fin de leur exécution. La figure montre à la fois le graphe de dépendance et un diagramme temporel de communication où chaque tâche est représentée par un rectangle dont la hauteur correspond à sa durée.

D'un autre côté (2.2(b)) on modélise l'application par un graphe de tâches communicant continuellement, sans dépendance. L'approche est souvent objet, avec les données encapsulées avec le traitement, et les objets évoluant souvent en parallèle dans des threads[6], à plusieurs sur une ressource de calcul, lorsqu'il y a besoin d'information

[6] Fils d'exécution.

(une autre forme de dépendance), les objets communiquent, et ceci peut se produire plusieurs fois au cours de leur exécution.

Il faut noter que le second modèle contient le premier qui en est une spécialisation. Ainsi une méthode capable de prendre en compte le second modèle sera à même de s'appliquer au premier, même s'il est possible qu'elle soit légèrement moins efficace qu'une méthode dédiée.

2.2.6 Parallélisme et mode d'exécution

Continuons sur l'idée énoncée à la section précédente et essayons de l'étendre à l'exécution même du programme.

Nous avons jusqu'à présent décrit un système distribué orienté objet. En effet nous sommes partis du principe que nous allions paralléliser une application dans son entier, et non pas un algorithme dont la tâche est unique et bien délimitée. En d'autres termes nous nous plaçons à un niveau de description plus élevé que celui de l'algorithme et ses données : nous décrivons un ensemble d'algorithmes avec leurs données.

Dans cette optique, l'approche consistant à essayer de scinder cette application au mieux, et dans notre cas en encapsulant les données avec les algorithmes dans une structure nommée objet fait sens.

Une autre approche consisterait à oublier ce découpage sémantique, et à essayer de paralléliser directement le code de l'application. Considérons par exemple le cas de l'addition de n vecteurs de dimension 4 dans une boucle. Clairement on sera très vite amené à écrire un algorithme ressemblant à celui de la figure 2.1.

2.1 : Addition de vecteurs en séquentiel.

$s1 = 0, s2 = 0, s3 = 0, s4 = 0$ des entiers;
pour i *de* 1 *à* n **faire**
    $s1 \leftarrow s1 + v_{i,1}$;
    $s2 \leftarrow s2 + v_{i,2}$;
    $s3 \leftarrow s3 + v_{i,3}$;
    $s4 \leftarrow s4 + v_{i,4}$;
return *vector(*$s1, s2, s3, s4$*)*;

On constate que si l'on avait 4 processeurs, on pourrait diviser le temps d'exécution directement par 4. Ici on ne s'occupe pas d'un quelconque découpage sémantique entre les parties de l'application, on scinde directement l'algorithme. Il s'agit de l'approche décrite par la figure 2.2(a) page 39.

Un tel découpage, bien qu'applicable à notre approche comme nous l'avons vu précédemment, peut poser plusieurs problèmes :

- les relations entre les parties sont des relations de dépendance uniquement, durant le calcul, les tâches ne communiquent pas ;
- ces relations sont difficiles à détecter et à pondérer, les données sont partagées par différentes parties de l'algorithme ;
- elles nécessitent une autre application dédiée qui est en mesure de repérer les parties parallélisables et les dépendances ;

Encore une fois, le modèle orienté objet contient le second fonctionnel. C'est une des raisons majeures pour laquelle nous nous sommes dirigés vers l'approche objet. Un découpage sémantique de l'application est fait, les données sont bien situées dans les objets, et si des données sont partagées entre deux objets, ces dernières sont aussi encapsulées dans une structure à part et les liens facilement identifiables. De plus, comme nous le verrons par la suite, cette démarche facilite aussi l'activité voire la proactivité des entités.

2.2.7 Le système distribué logiciel

Le système distribué logiciel sera donc composé d'entités capables d'interagir par communication. Ces entités seront situées chacune sur une ressource de calcul, à plusieurs sur ces ressources, et seront en mesure de migrer d'une ressource à une autre. Il sera possible que des entités apparaissent ou disparaissent en cours d'exécution. De plus chaque entité sera en mesure d'entrer en communication avec n'importe quelle autre, et ces interactions pourront évoluer au cours du temps.

2.3 Le système distribué dans son ensemble

Ainsi nous pouvons désormais décrire le système distribué tel que nous le verrons par la suite en termes de :

- ressources de calcul,
- autres ressources diverses (base de données, matériel dédié, *etc*.),
- entités,
- interactions.

La partie physique est un réseau dynamique de ressources de calcul dont on ne connaît pas la topologie et qui reste connexe.

La partie logicielle est un réseau dynamique d'entités logicielles, et ce réseau est formé par les interactions entre les entités logicielles. Ces interactions peuvent être effectives, c'est à dire passer par le réseau entre deux ressources de calcul.

Le réseau d'entités logicielles est dé-corrélé du réseau de ressources de calcul.

Nous verrons dans la partie II page 87 que de nombreuses méthodes de distribution ne prennent en compte qu'une partie du modèle de système distribué que nous avons décrit.

2.4 Le coût de la distribution

Comme on l'a vu, le but lorsque l'on distribue une application est avant tout de gagner en vitesse (et parfois en espace de stockage) afin de pouvoir mener à terme un calcul de large envergure. La loi d'Amdahl (de Gene myron Amdahl [Amdahl, 1967]), appliquée au domaine du parallélisme dit que si $F \in [0, 1]$ est la fraction du calcul menée en séquentiel, $(1 - F)$ la partie parallèle, et N le nombre de ressources de calcul, alors l'accélération maximale que l'on peut atteindre est :

$$\frac{1}{F + (1 - F)/N} \tag{2.1}$$

Mais ceci est une accélération théorique, car toute distribution a un coût. En effet, distribuer c'est :

- communiquer plus ;
- passer un certain temps à attendre l'arrivée de messages distants (sans parfois pouvoir effectuer une autre tâche en attendant) ;
- prendre en compte les délais de communication (empaquetage et dépaquetage des messages) ;
- et aussi et avant tout se synchroniser, souvent.

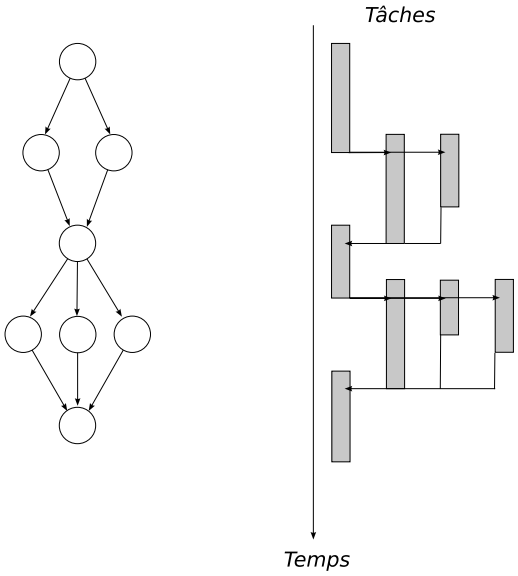
Il faut en conséquence veiller à ce que ce coût ne surpasse pas le gain possible apporté par la distribution. Si l'idée intuitive que la multiplication des ressources de calcul et de l'espace mémoire va accélérer une application est souvent fondée, il faut néanmoins prendre en compte le mécanisme propre de distribution qui doit se greffer sur l'application. Ce dernier n'est pas nécessairement léger.

Ce surcoût étant souvent lié à la communication induite par la distribution, on opposera souvent deux facteurs qui vont de paire :

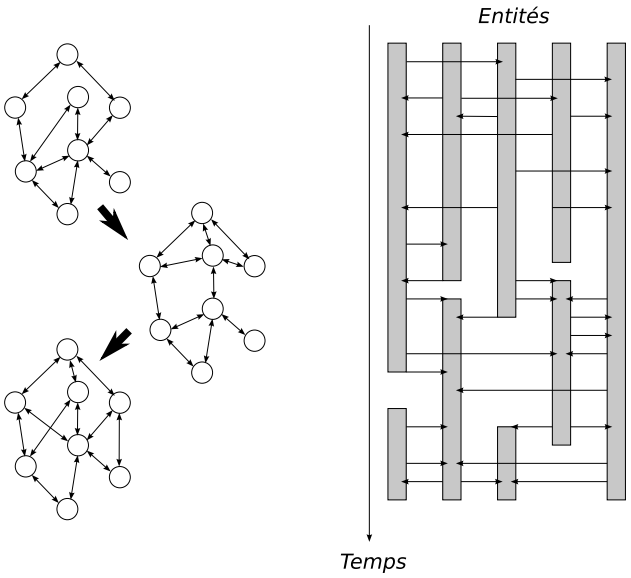
- l'augmentation des ressources de calcul fait gagner du temps ;
- l'augmentation des communications fait perdre du temps.

En général, plus on augmente le nombre de ressources de calcul, plus on augmente les communications. Plus ces communications sont effectives, plus on perdra de temps à communiquer, ceci pouvant à terme annuler les bénéfices de la multiplication des ressources de calcul. En conséquence un bon algorithme de placement visera à minimiser les communications effectives.

Un algorithme qui distribue la charge de calcul, ne peut que chercher *un compromis* entre la distribution du calcul et la minimisation des communications introduites par la distribution.



(a) Les données parcourent un flot de tâches dépendantes les unes de autres ; les tâches dépendantes s'exécutent séquentiellement.



(b) Les données sont encapsulées avec le traitement dans un objet ; toutes les entités tournent en parallèle.

FIG. 2.2: De l'ordonnancement au placement, et de l'approche fonctionnelle à l'approche objet.

Chapitre 3

Simulations

Sommaire

Après avoir défini les systèmes informatiques physiques avec lesquels nous allons travailler, après avoir décrit et nommé les composants essentiels de la distribution d'applications sur ces systèmes, nous allons décrire le calcul qui sera mené à terme sur ces systèmes : les applications elles-mêmes.

Nous nous concentrerons en majorité sur les applications de simulation, et détaillerons pour cela l'élément principal d'une telle application : son modèle. En particulier nous nous intéresserons aux modèles dits «computables», que l'on peut simuler. Nous étudierons ensuite les modèles dits «individu centrés». Cette partie nous sera aussi utile vis-à-vis de la présentation formelle que nous ferons d'*AntCO*^{2} à la partie III page 113.

Nous passerons ensuite en revue les réseaux d'interactions qui sous-tendent notre modèle. Nous étudierons comment l'application que nous cherchons à distribuer peut être vue comme un réseau d'interaction, et ferons le lien avec une modélisation par graphe. Pour finir, nous nous intéresserons à la structure et à l'organisation au sein des réseaux d'interaction.

3.1 Modèles et simulations

Roger Bacon (1214-1294), a été un des premiers à formaliser les étapes de la démarche scientifique. Elle peut être décomposée ainsi[1] :

1. observation du phénomène et *caractérisation* de ce dernier ;
2. formulation d'*hypothèses* sur le fonctionnement de ce dernier souvent au travers d'un ou plusieurs *modèles* ;
3. prédictions, fondées sur ces hypothèses ;
4. vérification par l'expérience ;
5. évaluation, et remise en cause en fonction des résultats ;
6. communication à la communauté ;

Suivant cette démarche on crée un *modèle* du phénomène observé sur lequel on fait ensuite des expériences. L'un des moyens de mener à bien ces expériences est la *simulation*. On utilise la simulation parce que parfois :

- il n'est pas possible de réaliser l'expérience sur le phénomène lui-même en situation pour des raisons pratiques ;
- l'observateur perturberait l'expérience ;
- l'expérience serait contraire à l'éthique.

On peut donc voir deux raisons majeures à l'utilisation de simulations dans les sciences :

1. comprendre et expliquer un phénomène réel ou abstrait, parce que l'on ne peut pas pour des raisons pratiques expérimenter sur le modèle lui-même ;
2. prévoir un phénomène réel ou abstrait afin d'en connaître le comportement à venir en fonction de différentes conditions initiales, afin d'en prédire l'évolution.

Les simulations sont donc le processus de calcul que l'on applique à un modèle. On utilise le modèle pour reproduire des phénomènes réels observés à travers ce dernier. Nous donnerons une définition plus précise des simulations par la suite, mais avant il est nécessaire de décrire plus précisément ce que l'on entend et désigne par le terme *modèle*.

3.1.1 Modèles

On trouve dans *Le trésor de la langue française* (TLF) plusieurs définitions pour le terme *modèle*. La première est une définition générale mais qui cadre bien avec la notion de modèle telle que nous l'utilisons :

> **Modèle, 1ère définition, générale** :
> Chose ou personne qui, grâce à ses caractéristiques, à ses qualités, peut servir de référence à l'imitation ou à la reproduction.

[1] Mais les racines de cet esprit rationnel sont bien plus anciennes, on les retrouve chez les grecs en particulier chez les philosophes sceptiques. Certains voient dans le papyrus d'Edwin Smith datant de 1600 avant J.-C. une approche de cette méthode. Ce papyrus détaille en effet, des examens, des diagnostics, des traitements et des pronostics médicaux pour de nombreux aliments. Mais c'est le frère franciscain Roger Bacon qui a en premier formulé explicitement les étapes de la méthode scientifique.

Un modèle imite et représente donc un phénomène. Il permet de faire «comme» le phénomène pour les points qui nous intéressent, et d'en avoir une «vue» par rapport à ce qui nous intéresse. Mais cette définition n'est bien entendu pas suffisante. La seconde définition s'adapte plus à la vision que l'on peut avoir d'un modèle en informatique :

> **Modèle, 2nde définition, en sciences** :
> Système physique, mathématique ou logique représentant les structures essentielles d'une réalité et capable à son niveau d'en expliquer ou d'en reproduire dynamiquement le fonctionnement.

Le point important souligné par ces définitions est que le modèle *n'est pas* la réalité qu'il représente. Cette réalité reste inaccessible, hors de portée. Il est seulement, «capable à son niveau, d'expliquer ou de reproduire» le fonctionnement du phénomène. D'ailleurs, tout modèle explicatif ne saura pas reproduire le fonctionnement du phénomène et inversement tout modèle reproductif ne saura pas expliquer un phénomène. Comme nous le verrons par la suite, certains modèles peuvent reproduire la complexité d'un phénomène sans pour autant le rendre compréhensible.

Le terme *à son niveau* signifie que le modèle dépend des observations faites à l'étape 1 de la démarche scientifique : le modèle n'est qu'une représentation de la réalité filtrée par l'observateur, soit parce que l'observateur a *sélectionné* les points qu'il jugeait importants (les structures essentielles d'une réalité), soit parce que l'observateur n'a pas été en mesure de percevoir les points importants.

Un modèle est donc une *conceptualisation* de phénomènes observés. C'est pourquoi on ne peut jamais avoir la certitude que le modèle et son comportement correspondent à la réalité. C'est aussi la raison pour laquelle un modèle est toujours perfectible, il s'agit de l'étape 5 de la démarche scientifique.

Alain Cardon dans [Cardon, 2005] donne la définition suivante :

> **Modèle** :
> Un modèle est une fonction de connaissabilité, une forme intermédiaire explicite permettant de passer du phénomène que l'on observe à une forme conceptuelle permettant de le comprendre, de le connaître, de savoir ce qu'il est, de prévoir son comportement et évidemment de transmettre cette connaissance à d'autres observateurs s'il en est besoin.

Cette définition, plus ancrée dans notre domaine, ajoute la notion importante de communication du modèle : le modèle, au contraire du phénomène réel est communicable. Cela n'implique encore une fois pas que le modèle est compréhensible : simplement, sa définition formelle peut être communiquée. Le modèle, tout comme la maquette du bâtiment[2], permet d'observer un phénomène sous divers angles, et d'en parler.

Et c'est pour cela que l'on modélise. Jean-Louis Lemoigne dans [Le Moigne, 1990] définit l'acte de modélisation :

[2] Puisque le terme modèle vient de là.

> **Modélisation** :
> Action d'élaboration et de construction intentionnelle, par composition de symboles, de modèles susceptibles de rendre intelligible un phénomène perçu complexe, et d'amplifier le raisonnement de l'acteur projetant une intervention délibérée au sein du phénomène ; raisonnement visant notamment à anticiper les conséquences de ces projets d'actions possibles.

Il insiste ainsi sur le fait que le modèle rend le phénomène *intelligible*. Comme nous l'avons dit, ceci n'est pas toujours vrai. Nous dirons plutôt que le modèle est formellement communicable, mais pas toujours compréhensible : il peut être simplement reproductif.

Nous pouvons désormais résumer les traits saillants d'un modèle en quelques points. Un modèle :

- représente une réalité cachée mais n'est pas cette réalité inaccessible ;
- est donc toujours perfectible, et est rendu obsolète par un meilleur modèle ;
- explique, pour certains, ou traduit pour d'autres une réalité *observée* ;
- ne donne aucune certitude quant à sa correspondance effective à la réalité.

Mais le modèle nous permet aussi, comme le disent ces définitions, «d'amplifier le raisonnement de l'acteur projetant une intervention délibérée», c'est-à-dire de prévoir les conséquences sur le modèle, ainsi que son comportement en fonction de «projets d'action».

3.1.2 Simulations

Mais quels peuvent être ces projets d'action ? Nous les avons déjà rencontrés :

- comprendre, expliquer ;
- reproduire, prévoir.

Le modèle nous permet en effet de comprendre et de communiquer sur le phénomène. Et parfois, le modèle n'est que cela, une représentation pratique nous permettant de mieux expliquer un phénomène car il nous permet de l'observer de tous côtés. C'est un outil pour l'esprit, communicable. C'est ce que la seconde définition du TLF décrit par simplement «comprendre» le phénomène.

Mais un modèle peut aussi nous permettre de *prévoir* ou de tester le *comportement* du système dans telle ou telle circonstance, c'est-à-dire, comme le dit la seconde définition du TLF, d'en «reproduire *dynamiquement* le fonctionnement». En d'autres termes le modèle peut être *opérationnel*. Nous pouvons par un moyen ou un autre *calculer* son comportement au cours du temps.

Par calculer, nous entendons «pouvoir obtenir les divers états dans le temps à partir de conditions initiales données». Cette notion est bien sûr différente de la notion de calculabilité en mathématiques. Nous utiliserons par la suite, pour marquer cette distinction, le néologisme «computable»[3].

[3]Le Français n'accepte que le terme «computation» (du latin *computus* signifiant calcul) provenant du terme initialement français *comput* (détermination du jour ou de l'époque d'une fête religieuse). Ce terme est passé tel quel à l'anglais avec «computation», mais aussi avec le verbe to «compute», où il est synonyme de calculer. Cependant il est plus orienté vers l'informatique par le nom que l'on donne aux ordinateurs : «Computers».

Certains modèles seront donc *computables* et se prêteront à la *simulation*. Une simulation sera alors le calcul d'un modèle. Plus précisément, pour rendre la dynamique du modèle, il sera nécessaire de définir :

- des conditions initiales, la forme première, l'état de départ du modèle, représentant le phénomène à un temps $t = 0$;
- des paramètres qui détermineront une partie du *comportement* du modèle ;
- des entrées, des valeurs fournies au modèle, *simulant* son *environnement* ;

On dit alors qu'une simulation est «un modèle plongé dans le temps». En effet le modèle va être calculé, et va donc *évoluer*.

3.1.3 Modèles continus et modèles discrets

On peut ensuite *faire évoluer le modèle* de deux façons, en fonction du mode de calcul et de la représentation du temps, continue ou discrète.

La première se distingue de la seconde par le fait qu'il est possible d'exprimer l'état du modèle à tout instant t réel, alors que dans la seconde, le temps est discrétisé et on passe d'un état à un autre par pas de temps. Ces pas ne sont d'ailleurs pas forcément réguliers, puisque ce sont les états du modèle qui nous intéressent, il est possible que *le temps ne soit pas homogène*.

La figure 3.1 page suivante présente les deux types de simulations : continues (3.1(a)) et discrètes (3.1(b)). Ces deux types de simulations sont paramétrées et apparaissent dans un état initial donné. Ensuite on fait évoluer le modèle avec le temps continûment ou discrètement (ce qui comprend les simulations par événement). Chaque évaluation du modèle à un temps t (réel et arbitraire pour les simulations continues, découpé en pas de temps entiers pour les simulations discrètes), peut prendre des entrées et déjà fournir des sorties. Ces dernières correspondent à des flux d'informations qui transitent à travers le modèle, et peuvent correspondre à son environnement. Le résultat final est disponible lorsque l'on a atteint un état ou un temps donné.

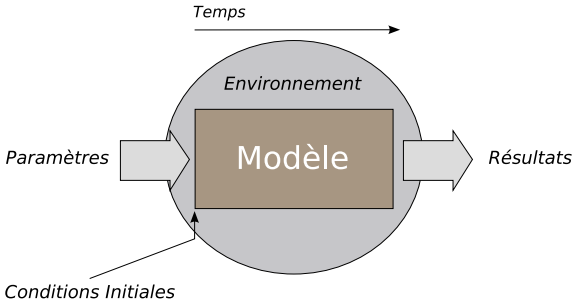
Le niveau d'échelle auquel on considère une simulation et les divers modèles qui la sous-tendent, peut modifier notre vision continue ou discrète de la simulation. La simulation peut être constituée de multiples processus dont l'évolution est discrète, mais vus dans l'ensemble, ils constituent un ensemble continu dont les pas discrets se chevauchent.

3.1.4 La flèche du temps

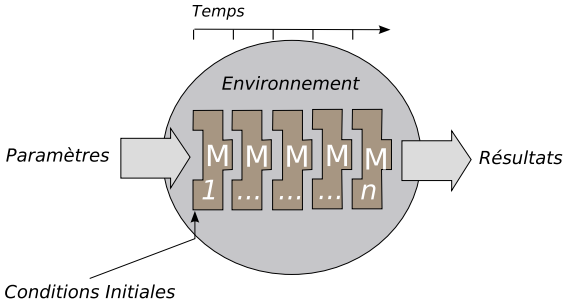
On peut aussi distinguer dans les simulations, outre l'aspect continu ou discret, la façon dont le temps s'écoule :

- Les simulations dont la flèche du temps est réversible, c'est-à-dire les simulations pour lesquelles il est possible de calculer les états dans un ordre quelconque ;
- Les simulations dont la flèche du temps est irréversible, c'est-à-dire les simulations dont on doit calculer les $n - 1$ premiers états pour calculer l'état n.

Les simulations dont la flèche du temps est irréversible sont souvent liées à des entrées externes que la simulation ne contrôle pas. Ce peut être des entrées de l'expérimentateur (rarement), imprévisibles (bien que cette notion soit confondue avec l'aléatoire, nous en reparlerons ci-dessous), des entrées faites par d'autres programmes



(a) Simulation continue.



(b) Simulation discrète.

FIG. 3.1: Simulations continues et discrètes.

dont la simulation dépend mais sur lesquels elle n'a aucun contrôle, *etc*. Dans tous ces exemples, la simulation ne peut pas rejouer les entrées aux étapes de temps t passées et donc ne peut revenir à ces étapes.

3.1.5 Aléatoire et imprévisibilité

Nous citons *l'imprévisibilité* comme facteur rendant une simulation orientée dans le temps, en faisant référence à un aléatoire naturel, en contact avec le monde réel. Prenons l'exemple d'un programme distribué, sensible à l'ordre dans lequel arrivent des requêtes de deux hôtes sur le réseau. Si la requête R_1 (toujours envoyée par l'hôte H_1) arrive avant la requête R_2 (toujours envoyée par l'hôte H_2) le résultat sera différent de la combinaison R_2 puis R_1. Or le système distribué utilise un réseau publique, lui-même utilisé par des utilisateurs humains qui influent sur sa charge (et pourquoi pas sur la charge machine de H_1 et H_2). Il devient alors impossible de prédire l'ordre dans lequel les deux requêtes vont arriver, puisque il est impossible de prédire la charge.

Il est important de différencier les notions d'aléatoire et d'imprévisible. L'imprévisible, tel que nous l'avons décrit ci-dessus rend la simulation informatique dirigée dans le temps. L'aléatoire peut, en revanche, être pris en compte (et est parfois une nécessité dans une simulation), et n'empêche en aucun cas de prédire statistiquement un résultat. On dira que la simulation est *stochastique*.

Il faut noter que bien qu'une simulation informatique soit obligée de prendre des entrées dites *imprévisibles*, elle n'est pas pour autant elle-même imprévisible, ce qui lui enlèverait dans la majeure partie des cas, toute utilité.

3.1.6 Distribution

L'un des points les moins souvent développés sur un modèle computable est sa capacité à être distribué. Probablement tous les modèles computable peuvent être simulés sur un système distribué, mais pour certains le coût de la distribution sera prohibitif (comme nous l'avons vu à la section 2.4 page 37). Tout comme il est souvent important de créer un modèle pour qu'il soit calculable, il est important de créer un modèle pour qu'il soit distribuable. Nous verrons dans la section suivante sur les modèles individu centrés, que certains se prêtent plus facilement à la distribution que d'autres.

Un modèle opérationnel computable sera distribuable s'il est possible de le décomposer en parties peu ou très peu interconnectées. Cela est à la fois vrai pour le service dont nous décrirons le modèle à la partie III page 113, mais bien entendu aussi des simulations que nous devrons distribuer. Le degré d'interconnexion est primordial. Si une application est composée de multiples entités mais que toutes ces entités communiquent souvent avec toutes les autres alors l'application est peu distribuable.

La loi d'Amdahl ne modélise pas tout à fait complètement ce fait. Elle considère l'application à distribuer comme constituée de calculs parallèles et de calculs séquentiels. Même s'il est vrai que l'on peut considérer qu'une communication entre deux parties distribuées est un point de synchronisation, et donc est une forme de calcul effectué en séquentiel, ceci n'est pas toujours le cas. Certaines communications peuvent être implémentées en parallèle : c'est le cas de tous les messages que l'on peut qualifier d'«optionnels», c'est-à-dire dont aucune partie de l'application ne dépend, les messages que l'application n'attend pas[4]. Pourtant, ces messages ont une importance sur la charge réseau. Leur envoi et leur réception prend du temps. En un mot, il ralentissent quand même l'application. La loi d'Amdahl ne prend pas directement en compte la charge réseau. La charge communicative n'est prise en compte que du point de vue du temps pris à empaqueter et à dépaqueter les messages. La loi d'Amdahl est axée sur le calcul, elle ne se préoccupe que de la charge machine. Pour elle un modèle est distribuable si on peut couper l'application en parties parallèles.

Bien sûr on s'attache avant tout à ce que le modèle soit représentatif, ou à ce qu'il se comporte le plus fidèlement possible au phénomène observé. Cependant pour les modèles opérationnels, computables, ceux pour lesquels on fera un calcul, ceux dont le but est une simulation, si plusieurs possibilités de représentation sont disponibles, il sera judicieux d'utiliser celle qui est la plus distribuable au sens que nous avons esquissé plus haut :

> **Facilité de distribution (vision communicative)** :
> Plus il est aisé de décomposer le modèle en parties, en entités, peu intercommunicantes, plus le modèle est distribuable.

Ce critère s'ajoutant à la loi d'Amdhal qui ne le prend en compte qu'implicitement et partiellement.

[4] Au sens où nulle partie de l'application ne doit se bloquer en attente de réception du message.

3.2 Modèles computables

Nous avons déjà déterminé plusieurs propriétés des modèles et des simulations. Ils peuvent être :

- stochastiques ou déterministes ;
- continus ou discrets ;
- locaux ou distribués ;
- orientés ou non dans le temps.

Nous allons maintenant nous concentrer sur les modèles opérationnels, computables et nous pencher sur les méthodes utilisées pour réaliser ces calculs. Nous discernons deux grandes classes de modèles computables :

1. les modèles analytiques ;
2. les modèles individu-centrés.

Il n'y a pas, bien sûr, de modèle général permettant de décrire tous les phénomènes, mais bien plusieurs classes de phénomènes et différents modèles les décrivant, parfois plusieurs types de modèles pour un phénomène, parfois aucun. Les deux grandes approches que nous avons distinguées ne sont pas les seules mais sont à l'heure actuelle les plus utilisées, avec une très large prépondérance pour les modèles analytiques.

3.2.1 Les modèles analytiques

Dans le modèle analytique, le phénomène est exprimé sous la forme d'une équation ou d'un système d'équations, dont les variables représentent les points saillants, mesurables, du phénomène. Ces modèles sont en général continus, les variables étant très souvent définies par des nombres réels[5]. On peut donc définir un état du modèle à tout temps t donné.

Ces modèles sont aussi dits «à base de lois» car la vision que l'on a du phénomène est *globale*, *générale*. On modélise le phénomène dans son ensemble, ses variables, les traits saillants qui nous intéressent, et leurs évolutions et effets, tels qu'observés. Pour cela il faut déterminer les lois générales, les forces qui s'exercent et les relations entre ces forces. On les représente alors par des équations. Ces dernières sont souvent des équations différentielles ou aux dérivées partielles.

La notion de système dans ce cadre correspond au processus que l'on cherche à modéliser. Il évolue dans le temps. Le système est alors décrit par des variables qui changent en fonction du temps. Le système est ouvert s'il y a des échanges (matière/énergie) avec l'extérieur. Les échanges s'effectuent par l'intermédiaire des entrées et des sorties du système, c'est-à-dire les échanges avec le milieu extérieur, ainsi qu'avec les variables modifiées. Les paramètres d'un modèle interviennent dans les équations du modèle et sont des grandeurs fixées. Les fonctions utilisées sont fréquemment non-linéaires soit par rapport aux paramètres, soit par rapport aux variables.

Effectuer une simulation se traduit alors par le calcul de la *trajectoire* du modèle, c'est-à-dire le calcul des valeurs ou états du modèle à tous les points qui nous intéressent, en fonction de conditions initiales, de la forme :

$$X_0 = A_0, \quad X_1 = A_1, \quad \dots \quad X_n = A_n \tag{3.1}$$

[5] Ou d'autres valeurs continues, complexes, *etc*.

et de valeurs prédéterminées pour les variables du modèle, et d'un système d'équations de la forme [Bertalanffy, 1968] :

$$\begin{cases} \frac{dX_0}{dt} &= f_0(X_0, X_1, \ldots, X_n) \\ \frac{dX_1}{dt} &= f_1(X_0, X_1, \ldots, X_n) \\ \ldots & \ldots \\ \frac{dX_n}{dt} &= f_n(X_0, X_1, \ldots, X_n) \end{cases} \tag{3.2}$$

Bien entendu, dans un tel système, certaines variables s'écrivent $0X_0$, et toutes les fonctions f_i ne dépendent pas de toutes les variables.

Les modèles analytiques ont longtemps été quasiment l'unique méthode de modélisation des phénomènes observés. Tant et si bien que certains auteurs l'intègrent directement dans la démarche scientifique décrite plus haut. On l'utilise en physique, en chimie, en biologie, en économie, en sociologie, *etc*. Si bien que l'on pourrait penser «que ce formalisme s'applique à tout et qu'il est *le* langage de description de tous les phénomènes [...] Mais il n'y a pas de miracle, et il n'y a sans doute pas de langage universel pour décrire tous les phénomènes». [Cardon, 2005].

Ainsi, un tel formalisme :

- s'applique à des phénomènes qui peuvent cependant être modélisés autrement, auquel cas il est peut-être judicieux d'étudier :
  - s'il est possible de rendre la simulation du modèle plus aisée à calculer : en rapidité, en facilité de mise en œuvre ;
  - si dans cette optique il est possible de rendre le modèle plus distribuable ;
  - si un autre modèle n'est pas plus informatif ;
- ne décrit pas le déploiement organisationnel du modèle, se concentre uniquement sur les effets du modèles ;
- la liste n'est pas exhaustive...

Les modèles individu-centrés présentent un formalisme alternatif qui résout certains des problèmes énoncés ci-dessus[6].

3.2.2 Les modèles individu-centrés (IBM)

L'acronyme anglo-saxon IBM signifie Individual-Based Models, que l'on traduit le plus souvent en français par modèles individu-centrés. Ces modèles, comme leur nom l'indique, reposent sur la modélisation explicite de certaines parties du phénomène, nommés individus, et l'expression elle aussi explicite de leurs interactions.

Au lieu de modéliser *globalement* le phénomène en représentant la dynamique générale de son évolution, on représente *localement* certaines parties significatives du phénomène en modélisant leur comportement propre. Ces parties peuvent être des éléments déjà perçus comme distincts dans le phénomène (des fourmis dans une fourmilière) ou bien des éléments abstraits qui n'ont d'utilité que par rapport au modèle (ensemble de particules numériques formant des méta-particules dans un flux, aucune meta-particule n'existe, ni même les particules numériques de base qui sont l'information numérique modélisant le flux).

On dit aussi de ces modèles qu'ils sont «à base de règles», les règles définissant le comportement des différentes parties du phénomène modélisé. Ce type de modèle met

[6] Mais peut en créer d'autres...

en avant *les interactions* entre les parties du modèle. En effet, au contraire des modèles analytiques où les équations du système dépendent des mêmes variables, les modèles IBM définissent explicitement les relations entre les éléments du modèle sous forme d'interactions.

Le modèle est donc basé sur les conséquences globales des interactions des éléments qui le constituent, et on peut suivre dans le modèle l'évolution de ces éléments. Cette caractéristique donne un certain pouvoir d'expression à ces modèles. Dans [Cardon, 2005] (p.29), Alain Cardon écrit :

> « Dans ces systèmes d'équations, toujours, les variables sont les caractères du phénomène considérés comme pertinents : le phénomène n'est appréhendé que par des caractères définis auparavant et aucunement par son état organisationnel lui-même : il n'est pas décrit tel il se déploie, se réorganise et évolue, mais il est représenté selon ses caractères les plus pertinents et selon ses effets. Un système d'équations décrit mais ne représente pas de déploiement organisationnel ni la génération d'objets nouveaux dans le phénomène. En effet, les systèmes d'équations produisent des résultats qui ne sont pas de nouveaux systèmes d'équations à d'autres échelles interférant avec les premières, tel le vivant cellulaire produit du vivant organisé et modifie son substrat par exemple. Et c'est sur ce point qu'est la limitation de l'approche équationnelle. Si l'on s'intéresse à l'état organisé d'un phénomène parce que celui-ci est organisation et n'est en fait rien d'autre, si l'on s'intéresse à la façon dont sa structure change aux différentes échelles et à différents moments selon son auto-adaptation, si l'on souhaite connaître ce qui conduit ses réorganisations, une approche basée sur des systèmes d'équations sera inadaptée. »

De tels modèles ont la particularité d'être potentiellement *spatialement explicites*, c'est-à-dire qu'il est possible de positionner dans un environnement spatialisé chaque entité et de suivre l'évolution de sa position au cours du temps, même s'il est bien sûr possible de créer des modèles IBM non spatialisés.

Les modèles IBM sont typiquement informatiques. Bien que le modèle de comportement des entités puisse être quelconque, leur mise en interaction, deux-à-deux et avec leur environnement est algorithmique. Le plus souvent, le comportement même des entités est défini par un algorithme, et leur développement dans le temps ne peut être simulé que sur un ordinateur, en raison du nombre rapidement trop important d'entités et d'interactions : un voire plusieurs ordinateurs sont nécessaires à leur simulation.

Les modèles IBM sont parfois nommés «modèles à base d'entités» ou «modèles agent». Nous ne garderons que le terme «entité», le domaine des systèmes multiagents (SMA) donne plusieurs définitions précises de ce qu'est un agent et comment il interagit avec les autres, et peut donc être considéré comme un domaine en intersection avec les IBM. Par la suite, nous nommerons donc *entités* les éléments d'un modèle IBM, ceci en rapport avec la dénomination que nous avons utilisée précédemment pour décrire les systèmes distribués (section 2.2 page 32).

3.3 Réseaux d'interaction

Dans les modèles IBM, les trois éléments suivants sont mis en avant :
– un grand nombre d'entités, modélisées explicitement ;

– des interactions elles aussi explicites et nombreuses entre ces entités ;
– un flux d'information ou d'énergie traversant ces entités et les structurant.

Notre objectif est de détecter les organisations au sein d'un tel système. Détecter de telles structures signifie en effet délimiter les zones de forte interaction et donc être capable de placer les éléments les plus liés par leurs interactions ensembles sur la même ressource de calcul, ceci afin de limiter la charge réseau.

En effet, dans un modèle IBM, nous modélisons les éléments du système par des entités, et les interactions par des communications. Il s'agit bien entendu d'un parti pris concernant la modélisation IBM, mais ce dernier est de plus en plus employé en informatique, en particulier avec le développement des approches orientées agent.

L'ensemble des entités et leurs interactions forme ce que nous nommerons un *réseau d'interaction*. On retrouve ces réseaux dans de très nombreux domaines : en télécommunication, en biologie, en sociologie, en économie, *etc*. Nous avons d'ailleurs déjà utilisé ce terme à la section 2.2 page 32 sur les systèmes distribués logiciels.

Il est aisé, et quasiment direct, de représenter un tel réseau par un graphe. La transition est triviale. Cependant, nous parlerons de réseau d'interaction quand il s'agira du modèle représentant un phénomène, avec ses entités et leur comportement, en un mot lorsqu'il s'agira du modèle et de sa simulation en action, avec une sémantique bien définie. Nous parlerons d'un graphe lorsque nous nous concentrerons sur la structure et les organisations extraites du réseau d'interaction. Ces structures, ces organisations seront en quelque sorte une esquisse du réseau d'interactions réel. En effet elles ne traduiront plus directement le comportement des entités du modèle, elles ne contiendront plus la sémantique du modèle.

Il faut donc faire la distinction entre :
– structures et organisations,
– le modèle simulé, le réseau d'interactions et sa représentation sous forme de graphe.

3.3.1 Structures et organisations

Dans un modèle IBM fait d'un grand nombre d'entités en interaction, on peut s'attendre à voir apparaître des relations privilégiées entres certaines entités, formant ainsi des groupes d'entités. Ces relations privilégiées peuvent être dues à une proximité géographique (modèles spatialisés) ou temporelle, à une prédilection dans le comportement de l'individu pour un certain type de communication, ou même un effet de bord de son comportement. Parfois ces interactions privilégiées sont d'ailleurs voulues par le concepteur du modèle.

De tels groupes d'interaction apparaissent et disparaissent au cours de l'évolution de la simulation du modèle. Nous parlerons d'*organisations* pour décrire de tels groupes. Il ne s'agit pas encore là d'auto-organisation, car nous n'avons pas décrit la façon dont ces groupes se forment. Nous verrons ceci par la suite. Le point saillant qui nous intéresse réside dans le fait que ces groupes existent au sein d'un système constitué d'éléments en interaction.

Bien entendu il est possible de trouver un système dans lequel tous les éléments communiquent :
– tous les uns avec les autres,
– à tout moment,

– dans les mêmes proportions à tout moment.

Si nous devions modéliser un tel système par un graphe, nous obtiendrions un graphe complet tel que celui de la figure 3.2. Un tel graphe est totalement régulier, et ne contient aucune organisation (ou une organisation globale à ce niveau de description).

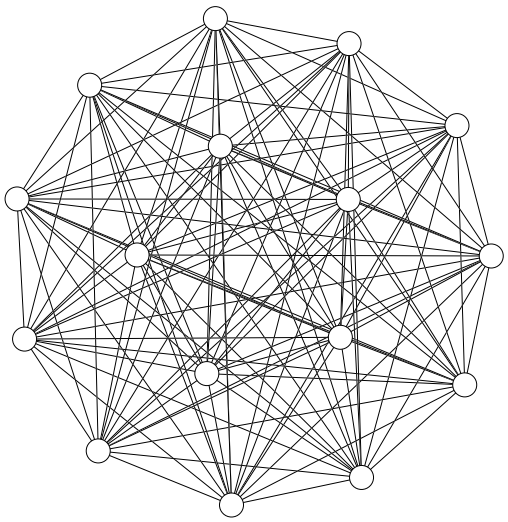


FIG. 3.2: Un graphe complet de 16 nœuds.

Cependant, de tels cas sont très rares, car peu intéressants du point de vue des modèles IBM. En général, si l'on utilise une approche IBM c'est avant tout pour modéliser le déploiement organisationnel d'un phénomène.

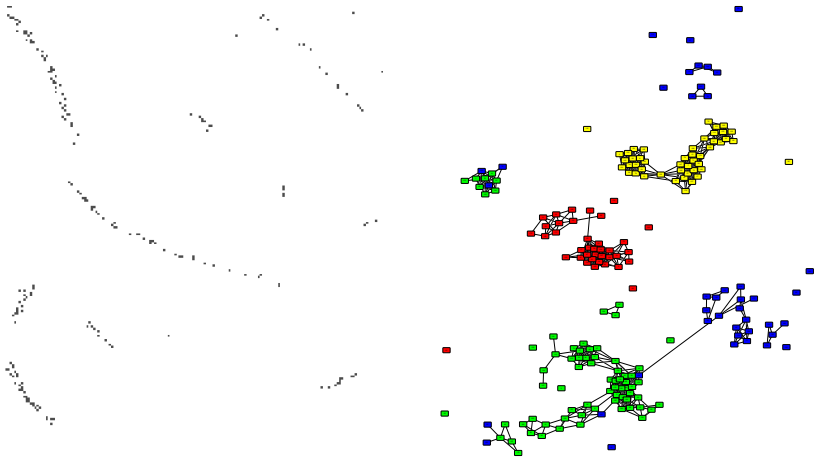
Le graphe de la figure 3.3 page suivante représente une simulation simple de *Boids* similaire à celle décrite section 4.5.1 page 68 et section 12.3 page 166. Les Boids [Reynolds, 1987] sont des entités au comportement extrêmement simple, dont le modèle, IBM, est facile à mettre en œuvre. On voit sur ce graphe apparaître divers groupes de nœuds très reliés, correspondant aux groupes de Boids dans la simulation. C'est un exemple typique des simulations auxquelles nous serons confrontés.

Ainsi, en ce qui concerne le travail décrit ici, une organisation sera :

> **Organisation** :
> Groupe d'entités en très forte interaction.

Cette définition est volontairement simple. En effet, qu'est-ce qu'une forte interaction ? Où se situe la frontière de l'organisation ? Y a-t'il même une frontière ? Nous définirons ces notions par la suite en fonction de nos attentes dans le chapitre 4 page 55 sur les approches collectives, et plus spécifiquement encore dans le chapitre 9 page 115 dédié à la méthode *AntCO*^{2}, sujet de cette thèse.

En effet, diverses interprétations définiront diverses organisations à divers niveaux d'échelle pour le même modèle. Toutes ces organisations auront un sens particulier en fonction des attentes du simulateur. Cependant, en ce qui nous concerne, nous ne sommes intéressés que par la détection de groupes *communicant* le plus, puisque nous avons pris le parti de traduire les interactions par des communications. Ainsi nous



(a) La simulation de Boids. (b) Le graphe associé.

FIG. 3.3: Simulation de Boids et le graphe associé.

pourrions même réécrire la définition précédente :

> **Organisation** :
> Groupe d'entités très inter-communicantes.

Dans [Le Moigne, 1990], l'auteur distingue les termes *structure* et *organisation* qui sont en général utilisés comme synonymes dans le cadre qui nous intéresse. Nous utiliserons cette distinction ainsi :

- une structure décrit une organisation statique, dont les interactions et les entités sont fixes dans le temps ;
- une organisation décrit un ensemble dont les entités et les interactions varient et se réorganisent dans le temps.

Ainsi, une organisation en constante évolution aura une durée de vie différente du groupe d'entités et des interactions qui la composent durant le déroulement de la simulation. Il est possible que des interactions, ou même des entités, apparaissent ou disparaissent au sein d'une organisation sans pour autant la briser. Une organisation ne sera donc pas inerte, mais en constante évolution interne, tout en pouvant être considérée comme un tout.

Si nous restreignons volontairement la définition d'organisation, c'est parce que nous cherchons uniquement à distribuer la charge des ressources de calcul (nombre d'entités) et la charge réseau (nombre de communications coûteuses). Ainsi définir une organisation comme un ensemble d'entités très communicantes nous permet d'identifier les groupes que nous cherchons à distribuer.

3.3.2 Réseaux d'interaction, sémantique et graphes

Comme nous l'avons vu, le réseau d'interaction d'une simulation et le graphe qui en est extrait sont découplés. Nous considérerons le réseau d'interaction comme représentant la sémantique de la simulation, avec une représentation propre à la simulation

et ainsi au phénomène simulé. Le réseau d'interaction fait partie du modèle.

Le graphe en revanche sera pour nous une représentation du réseau d'interaction ne contenant que les informations nécessaires pour la distribution de la simulation. Il sera dépourvu de toute sémantique liée à la simulation, si bien que toute simulation sera vue sous la forme de graphes de même type. Le graphe sera suffisant pour détecter par exemple les organisations de communication telles que définies plus haut.

On peut comparer cette distinction à celle que l'on fait entre le sens d'une phrase et sa construction lexicale et grammaticale. Le réseau d'interaction contient le sens de la phrase, tandis que le graphe n'est que l'expression du lexique et de la grammaire.

Les graphes que nous utiliserons seront notés $G = (\mathcal{E}, \mathcal{V})$ avec $\mathcal{E}$ l'ensemble des arcs et $\mathcal{V}$ l'ensemble des sommets. Cependant, puisque la simulation évolue au cours du temps, le graphe la représentant sera lui aussi dynamique et sera noté, lorsque le temps importe, $G(t) = (\mathcal{E}(t), \mathcal{V}(t))$ avec $t \in \mathbb{N}^+$ le temps et $\mathcal{E}(t)$ un ensemble dynamique d'arcs et $\mathcal{V}(t)$ un ensemble dynamique de sommets.

Pour indiquer l'importance des interactions, donc des communications, les arcs seront valués. À chaque arc $e \in \mathcal{E}(t)$ sera associé un poids noté $w^{(t)}(e) \in \mathbb{R}^+$. Les graphes que nous utiliserons seront donc dynamiques et pondérés.

3.3.3 Dynamique

Le réseau d'interaction sous-tendu par l'application évolue. Bien entendu, les interactions au sein de l'application ne sont pas statiques. C'est par ces interactions que l'application s'exécute et accomplit sa tâche. Conséquemment, le graphe modélisant ce réseau d'interaction doit lui aussi évoluer. Nous nommons ce type de graphes des «graphes dynamiques».

Il est ainsi possible de représenter l'évolution du réseau d'interaction par une série de graphes à chaque instant discret t, représentant l'état de ce réseau durant un intervalle de temps donné. En fait nous considérons cette famille de graphes comme un unique et même graphe évoluant. Cette notion est plus proche de la vision informatique d'un graphe que de sa vision mathématique en tant qu'instance figée et invariante. Le graphe sera pour nous une structure de données dynamique contenant des ensembles de sommets $\mathcal{V}(t)$, d'arcs $\mathcal{E}(t)$, et d'informations sur chacun de ces ensembles, dont l'état varie au cours du temps.

Chapitre 4

Approches Collectives

Sommaire

Dans les sociétés d'insectes, le «projet» global n'est pas programmé explicitement chez les individus, mais émerge de l'enchaînement d'un grand nombre d'interactions élémentaires entre individus, ou entre individus et environnement. Il y a en fait intelligence collective construite à partir de nombreuses simplicités individuelles.

Jean-Louis Deneubourg

4.1 Intelligence en Essaim

Le terme anglo-saxon «Swarm Intelligence» traduit directement en français par «Intelligence en Essaim» à été introduit en 1989 par G. Beni et J. Wang dans le contexte des systèmes robotiques cellulaires [Beni, 1989], c'est-à-dire constitués d'une

multitude de robots. Pour qualifier les approches collectives, on parle en effet souvent d'intelligence collective, ou intelligence en essaim, sous-domaine de l'intelligence artificielle distribuée. Si l'on utilise le terme «intelligence», c'est que les systèmes collectifs sont souvent capables, non pas au niveau de l'individu mais bien au niveau du groupe, de comportements d'apprentissage, d'adaptation aux changements, de robustesse vis-à-vis des cas non prévus, *etc*. qui sont des qualités que l'on rapporte usuellement à l'intelligence. Cependant, comme nous le verrons par la suite, ce qui fait la particularité de ces approches réside dans le fait que les entités formant l'essaim ont un comportement en général très simple, généralement réactif, que l'on ne peut pas qualifier d'intelligent.

L'intelligence en essaim s'inspire en large partie du comportement de groupes ou de sociétés d'animaux dans la nature. Comment, par exemple, les individus d'une colonie de fourmis qui nous semblent si frustres, peuvent ils bâtir des nids dont la taille est plusieurs centaines de fois celle de leur bâtisseurs (figure 4.1), organisés, structurés pour maintenir la vie en collectivité d'un nombre très important d'individus, comparable à des villes, et parfois même des réseaux de villes sur plusieurs kilomètres, avec leurs réserves, leurs crèches, leurs chambres, leurs cimetières, une régulation de la température, un approvisionnement en nourriture, des groupes à but ouvrier, guerrier, *etc*. ? La recherche des mécanismes sous-jacents et des moyens de les simuler pour les réutiliser, ou en inventer de nouveaux, est l'axe de développement majeur de l'intelligence en essaim.

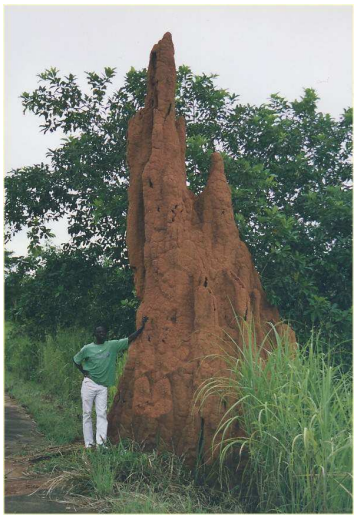


FIG. 4.1: Une termitière.

Le comble de cette organisation est l'*eusocialité*, où un groupe restreint (souvent à un unique individu) s'occupe de la reproduction, les autres participant aux tâches de maintien de la colonie. On parle d'eusocialité (ou parfois d'*éosocialité*) si :

1. le groupe est constitué d'au moins deux générations ;
2. il y a cohésion entre les membres ;
3. l'élevage des jeunes est coopératif ;

4. certains individus sont spécialisés.

On constate un comportement eusocial chez les hyménoptères[1], mais aussi chez certains mammifères tels les rats-taupes vivant en Ethiopie. Ce mécanisme semble aller à l'encontre de la génétique selon Darwin, mais chez des individus haplo-diploïdes, la favorisation d'individus ayant le même patrimoine (altruisme) permet de contrecarrer le fait que certains individus malgré leurs performances personnelles (et parfois leur comportement de sacrifice altruiste) ne se reproduisent pas [Rennard, 2002] (p. 244).

Les modèles déjà mis au point par les biologistes et les informaticiens permettent de simuler une partie du comportement de certains animaux sociaux ou eusociaux. Par exemple le comportement bâtisseur de termites, d'abeilles ou de guêpes, le comportement de chasse des fourmis, ou encore le comportement de bancs de poissons, de vols d'oiseaux, de hordes ou de hardes. Ces modèles sont basés sur une formalisation, souvent assez simple, des comportements individuels au sein d'un groupe, plutôt que sur la description de l'évolution générale du groupe. On retrouve ici les modèles individu centrés (IBM) que nous avons vu au chapitre 3.2.2 page 49.

Bien que les modèles globaux et les modèles IBM soient tous deux utiles à différents niveaux, les modèles IBM représentent le phénomène à un niveau qui permet d'observer la formation d'organisations, une création de formes, au sein du groupe. Les modèles globaux permettent d'étudier uniquement l'évolution de paramètres généraux au cours de la vie du groupe tels que croissance de la population, densité, natalité, mortalité, et tout au plus des compartiments. En un mot ils permettent seulement l'observation de caractères du groupe en son entier. Les modèles IBM quant à eux autorisent l'observation de ces mêmes paramètres (même si leur mise en œuvre serait plus lourde uniquement pour de telles observations), mais en sus, permettent l'observation des interactions des individus les uns avec les autres ainsi qu'avec l'environnement. La modélisation explicite de ces interactions, comme nous le verrons par la suite, rend possible diverses observations impossibles à réaliser avec les modèles globaux :

- la formation de sous-groupes d'individus au sein de la population ;
- la construction d'éléments dans l'environnement ;
- le fonctionnement de castes ;
- la rétroaction de ces éléments sur les sous-groupes et la population.

Cette liste n'est bien entendu pas exhaustive, cependant elle montre bien que certains éléments essentiels sont fournis par une modélisation basée sur les individus, éléments que nous pouvons utiliser, en les adaptant, pour mener à bien d'autres tâches, d'où l'intérêt porté à ces méthodes par les informaticiens. En particulier nous verrons avec le chapitre consacré au fourmis numériques (p. 73), puis dans celui dédié à la méthode que nous exposons (p. 115), que la capacité à construire des éléments dans l'environnement est d'une grande richesse applicative.

4.2 Définition

Mais plus précisément, avant tout, qu'est-ce qu'une approche collective ? Toutes les approches collectives ont en commun la définition suivante :

[1] Dont font partie les fourmis et les abeilles. Hyménoptère signifie «aux ailes en membrane». En effet, les fourmis, lors de la reproduction développent des ailes.

> **Approche collective** :
> Comportement d'une population d'individus interagissant *localement* les uns avec les autres, ainsi qu'avec leur environnement.

Cette définition est bien sûr très générale, et donc imprécise. Elle reprend la citation de Jean-Louis Deneubourg que l'on retrouve au début de ce chapitre, plus spécifiquement ciblée sur les sociétés d'insectes. Ce dernier cependant ajoute la notion de «projet global», non programmé explicitement dans le comportement des individus, et parle ainsi d'«intelligence collective». Ainsi une définition d'intelligence collective serait :

> **Intelligence collective** :
> Comportement d'une population d'individus aux capacités simples dont les interactions *locales* inter-individus et avec l'environnement permettent l'émergence et la réalisation d'un projet global non programmé explicitement.

Dans ces deux définitions, on insiste sur le caractère local des interactions entre individus. Ces modèles sont en effet caractérisés par une décentralisation totale. Bien que, par exemple, de nombreuses colonies d'insectes aient ce que l'on nomme souvent une «reine», ces individus que l'observateur humain a d'abord privilégié par anthropomorphisme sont cantonnés à la reproduction et en aucun cas n'ont de vision globale de l'état de la colonie, et ne distribuent d'ordres ou de lois sur le comportement général[2].

Cette dernière définition peut aussi paraître insuffisante car elle ne semble pas dire comment ce projet global va être mené à bien. Elle parle de comportement simple des individus, n'ayant aucune vision globale, n'ayant pas connaissance du projet global.

Nous allons donner un exemple d'un tel projet global, puis par la suite, nous verrons par quels moyens il est mis en œuvre, dans le cadre de la construction d'une loge royale chez les termites. Cet exemple est tiré des travaux de Pierre-Paul Grassé puis de Eric Bonabeau et Guy Théraulaz [Théraulaz, 1997]. Les nids sont formés de voûtes, soutenues par des piliers assez régulièrement espacés. Le mécanisme mis en œuvre utilise de la *phéromone*. La phéromone (ou parfois *phérormone*) est une substance chimique odoriférante que les insectes sont en mesure de produire et de détecter. Le comportement d'une termite lors de la construction est le suivant :

- Les termites se déplacent au hasard.
- Si la termite constructeur, portant une petite quantité de terre, détecte une plus forte valeur de phéromone alentour il la suit jusqu'à trouver un point où la phéromone (on dira par la suite le «gradient de phéromone») est la plus forte.
- Lorsque il trouve ce point il dépose la terre en y ajoutant une quantité donnée de phéromone.
- La phéromone s'évapore.

Ce comportement est particulièrement simple et peut très facilement être mis en œuvre. Mais la question que l'on peut se poser cependant est la suivante : comment, à partir d'un comportement aussi simple, le nid de termites est-il construit ? Comment avec un système basé sur un ensemble d'individus dont le comportement entièrement spécifié à l'avance ne comporte aucune trace d'un plan de la loge, du projet global à at-

[2]Cependant, par son unicité, l'individu reproducteur unique influence bien plus la colonie, et peut avoir un impact sur tous les individus. C'est le cas de toutes les populations eusociales.

teindre, peut-on obtenir ce projet ? Comment le nid, au départ complètement aléatoire, peut-il s'organiser en un ensemble de piliers et de voûtes ?

La réponse a trait aux interactions très nombreuses entre les individus et avec l'environnement.

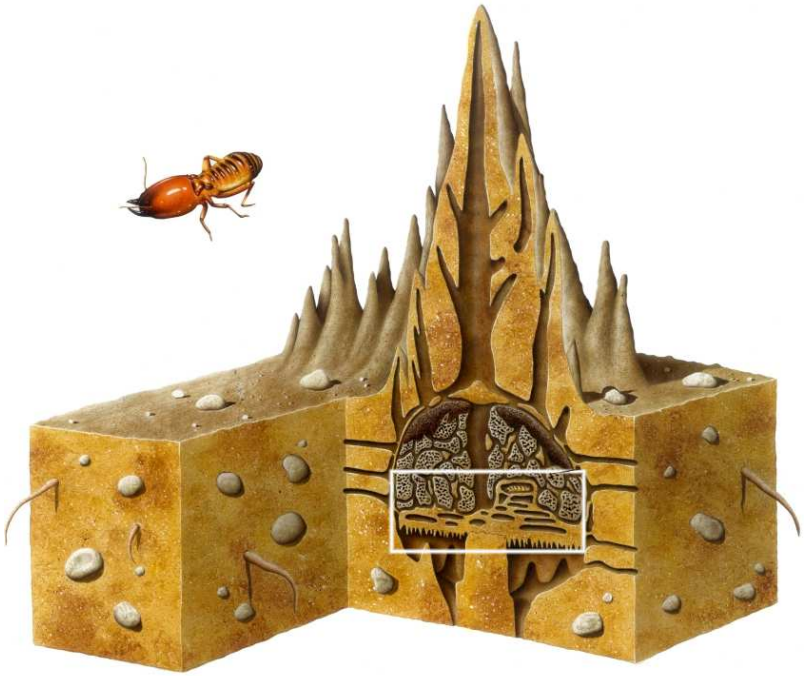


FIG. 4.2: Coupe d'une termitière (la partie encadrée correspond à la zone décrite).

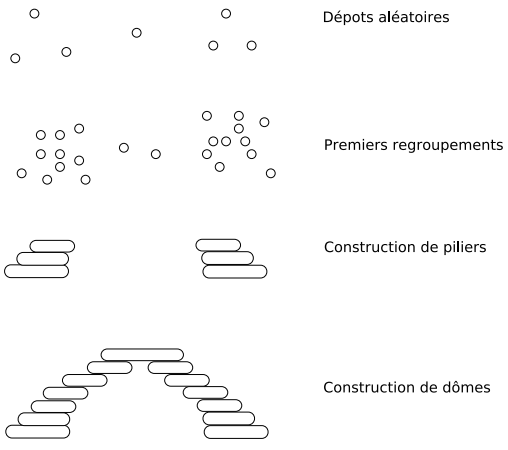


FIG. 4.3: La formation des piliers dans la construction de la termitière.

4.3 Organisation et Auto-Organisation

En effet, on dit que ce processus est un exemple d'*auto-organisation*. On assiste bien à une *organisation* de la loge en piliers et en voûtes, mais cette organisation n'est à aucun moment dirigée hiérarchiquement par un individu principal qui délègue à d'autres, *etc*. L'organisation en piliers puis en voûtes, apparaît d'elle-même à partir des interactions, nombreuses, des termites. C'est pourquoi on la qualifie d'*auto*-organisation, et c'est aussi pour cela que l'on dit qu'elle *émerge*.

L'auto-organisation n'est pas observée que dans les systèmes biologiques. Elle a en tout premier lieu été décrite formellement dans les systèmes physiques, en particulier avec les cellules de Bénard, des exemples de structures dissipatives, et il en existe des exemples en chimie (réseaux auto-catalytiques, systèmes réaction-diffusion), en économie (les marchés), en sociologie (la création d'une ville [de Rosnay, 1975]), ainsi qu'en mathématiques (automates cellulaires).

4.3.1 Organisations

L'auto-organisation est donc un processus par lequel des organisations se créent et se maintiennent d'elles-mêmes, mais qu'est-ce d'abord qu'une organisation ? Nous avons déjà abordé le sujet des organisations au sein des réseaux d'interaction au chapitre 3.3 page 50, et effectivement la majorité des systèmes dans lesquels on retrouve de l'auto-organisation peuvent être vus à un certain niveau de description comme un réseau d'interaction.

Il est difficile de définir le terme *organisation*. Plusieurs types d'organisations peuvent être présents dans un même système, en fonction des besoins et de l'observateur ou de leur action. On peut cependant déjà en dégager deux types :

- les organisations définies uniquement par l'observateur ;
- les organisations qui en tant que telles rétroagissent sur le système dont elles font partie.

Le premier type se scinde en deux. Une première forme d'organisation est de type hiérarchique, comme par exemple une entreprise. Ce type d'organisation auquel nous ne nous intéresserons quasiment pas, est plutôt statique. Ces organisations sont créées par un ou plusieurs «décideurs» qui gèrent centralement la forme de l'organisation.

Une seconde forme d'organisation est constituée par les groupes formés au sein d'un ensemble d'entités par un observateur extérieur[3]. De tels groupes sont arbitraires et dépendent des besoins et des attentes de l'observateur. Il s'agit de séparations suivant un critère particulier d'un ensemble d'éléments, par exemple suivant une relation d'ordre, ou en fonction d'un critère de ressemblance, de sémantique, d'éloignement ou de rapprochement, *etc*. Ce critère de séparation définit les *frontières* des organisations suivant une notion d'ordre.

Le second type d'organisation concerne celles qui peuvent non seulement être observées mais agissent en tant que tout sur le système dont elles font partie. Cette action est une rétroaction qui elle-même peut entraîner la définition de plus en plus précise de l'organisation. En d'autres termes, elle peut entraîner la formation d'une *frontière* délimitant l'organisation, et par là même une notion d'appartenance, donc de

[3] Il est donc possible de trouver de telles organisations au sein d'une organisation hiérarchique.

critère de sélection. On peut citer l'exemple des cellules au sein du corps humain, mais aussi de groupes sociaux humains qui lorsqu'ils atteignent une masse critique prennent conscience d'eux-mêmes en tant que groupe, en se nommant.

Nous nommerons ces trois cas :

1. organisations hiérarchiques ;
2. organisations observées ;
3. organisations rétroactives.

Le concept de *frontière* est particulièrement important, puisque c'est cette dernière qui délimite les organisations, donc qui les identifie au sein du groupe entier d'individus ou d'entités. Elle peut être virtuelle ou réelle, parfois engendrée par l'organisation elle-même. L'un des rôle majeurs de l'algorithme *AntCO*^{2} que nous décrivons est de détecter des organisations dans un graphe en fonction d'un critère de communication. L'algorithme détectera donc des organisations observées. Cependant ces organisations dans le réseau d'interaction sous-jacent au graphe seront peut-être des organisations rétroactives, faisant sens pour la simulation dans laquelle elles se développent.

Nous pouvons donc affiner la définition d'organisation esquissée à la section 3.3 sur les réseaux d'interaction, en y incorporant la notion de frontière :

> **Organisation** :
> Groupe d'entités en très forte interaction, délimité par une frontière définie par un critère d'appartenance à l'organisation.

Ensuite il est important d'insister sur le fait qu'une organisation, observée ou rétroactive, évolue avec les éléments qui la composent. Ces éléments peuvent avoir une durée de vie bien inférieure à l'organisation, ils peuvent apparaître, disparaître, changer, sans pour autant remettre en cause l'existence de l'organisation. En ce sens, l'organisation et sa structure sont pérennes alors que les éléments qui la composent ne le sont pas. Ainsi il existe des organisations non seulement dans les ensembles d'éléments stables, en nombre constant, mais bien entendu aussi dans les ensembles d'éléments en perpétuelle reconfiguration. Et c'est d'ailleurs la dynamique de ces ensembles qui mène au domaine de l'auto-organisation.

4.3.2 Auto-Organisation

Certaines des organisations que nous avons données en exemple à la section précédente, sont aussi des exemples d'auto-organisation. C'est d'ailleurs souvent le cas des organisations rétroactives mais pas toujours.

Dans [Théraulaz, 1997], les auteurs donnent la définition suivante de l'auto-organisation :

> **Auto-Organisation** :
> Tout processus au cours duquel des structures émergent au niveau collectif (ou plus généralement apparition d'une structure à l'échelle $N + 1$ à partir d'une dynamique définie à l'échelle N), à partir de la multitude des interactions entre individus, sans être codées explicitement au niveau individuel.

Nous reprendrons cette définition, tout en conservant la distinction que nous avons

faite entre structure et organisation au chapitre 3.3 page 50, en remplaçant donc le mot structure par organisation.

Mais comment ces structures émergent-elles ? Leur apparition est liée aux interactions très nombreuses se produisant au sein de la population. Ces interactions se font entre les individus eux-mêmes, mais aussi avec l'environnement. En général, on distingue quatre facteurs fondamentaux :

- des rétroactions positives ;
- des rétroactions négatives ;
- la présence d'un nombre critique d'entités *en interaction* ;
- l'amplification des fluctuations.

4.3.3 Les mécanismes de l'auto-organisation

On parle ainsi de rétroaction positive lorsqu'en réponse à un changement d'une variable d'un système, le système «répond» en changeant cette variable à nouveau *dans le même sens*. C'est le cas par exemple, si en réponse à l'augmentation de la taille d'une organisation, cette organisation grandit encore plus. On considère généralement les rétroactions positives comme un mécanisme «explosif» dans le sens où si rien ne le stoppe, le phénomène s'amplifie continuellement.

Un exemple simple de rétroaction positive peut être donné en sociologie : plus un groupe social est gros, plus il devient désirable ou puissant, et plus il attire d'individus en son sein. On le voit cependant, s'il n'est pas contrôlé, ce processus mène à l'explosion du groupe qui devient trop important.

Quant aux rétroactions négatives, comme leur nom l'indique, elles prennent le contre-pied des rétroactions positives. On parle de rétroaction négative lorsqu'en réponse à un changement d'une variable d'un système, le système répond en changeant cette variable *dans le sens contraire*. Les rétroactions négatives ont un effet stabilisateur, elles maintiennent les structures et organisations créées par rétroactions positives.

L'exemple le plus simple et connu de rétroaction négative est le thermostat. Lorsque la température d'un local atteint un seuil haut donné par chauffage, le thermostat arrête ce chauffage. Si le seuil bas donné de température est atteint, le thermostat réenclenche le chauffage. On pourrait penser ici à un exemple de rétroaction négative (éteindre le chauffage), suivi d'un exemple de rétroaction positive (allumer le chauffage). En fait il s'agit bien de deux cas de rétroaction négative. Les termes «négative» et «positive» n'ont pas le sens «bénéfique» ou «néfaste», simplement une rétroaction négative «stabilise», et une rétroaction positive «amplifie».

On parle de *morphogénèse* lorsque les rétroactions positives amplifient la création d'un groupe, d'une structure ou d'une organisation, en un mot d'une forme, morphogénèse signifiant littéralement création de forme. On parle aussi de processus *autocatalytique* en référence à la physique.

On parle de *morphostase* lorsque le processus de morphogénèse est contrôlé par des rétroactions négatives qui le ramènent sans cesse vers l'équilibre, limitant les amplifications, morphostase signifiant conservation de forme.

Ainsi, si les rétroactions positives créent des organisations et ont un facteur générateur nommé *morphogénèse*, les rétroactions négatives les maintiennent dans une «forme stable» en tant que telles nommée *morphostase*. Cependant, des interactions

extérieures (environnement, autres organisations) peuvent à tout moment déstabiliser les organisations de telle façon que :

- les rétroactions positives font exploser l'organisation ;
- les rétroactions négatives font disparaître l'organisation ;
- un facteur extérieur essentiel n'est plus disponible, faisant disparaître l'organisation.

L'auto-organisation crée donc des formes qui se développent en trois phases :

1. La phase *juvénile*, dans laquelle les interactions positives sont plus fortes, ou les interactions négatives n'ont pas lieu, car aucun seuil de déclenchement n'a été rencontré. C'est la morphogénèse, l'apparition de la forme.
2. La phase *adulte*, dans laquelle les interactions positives et négatives jouent de concert, maintenant implicitement la forme en état. C'est la morphostase.
3. La phase *sénescente*, dans laquelle un facteur déstabilisant détruit la forme.

Les rétroactions négatives et positives, bien que nécessaires à l'auto-organisation, et en étant en quelque sorte la signature, ne sont pas forcément signe d'auto-organisation. Il est bien entendu possible d'observer des systèmes dans lesquels il y a ces deux formes de rétroactions sans pour autant qu'il y ait auto-organisation (le thermostat, la régulation en automatique). En effet, comme nous l'avons dit plus haut, un autre critère est essentiel : un nombre critique d'entités en interaction.

Nous avons décrit les interactions positives et négatives sous leur forme générale, en parlant de variable d'un système, donc de manière globale. Nous avons utilisé le terme «forme» plutôt qu'organisation. Ceci puisque interactions positives et négatives ne signifient pas forcément auto-organisation. En effet, dans les processus d'auto-organisation, ce sont les interactions entre les entités et ces entités mêmes qui se développent en organisations et créent des formes. Pour que cela se produise, il est nécessaire qu'un nombre critique, suffisant, d'entités soit présent afin de créer les rétroactions négatives et positives.

4.3.4 La loge royale des termites (suite et fin)

Revenons sur l'exemple que nous avons développé précédemment. Les mécanismes à l'œuvre lors de la construction du nid de termites sont ceux définis précédemment : rétroactions positives, négatives, et grand nombre d'individus en interactions. Voici comment la loge est construite :

- Les termites portant des boules de terre, se déplacent aléatoirement.
- Au début elles déposent ces boulettes, préalablement enduites de phéromones tout aussi aléatoirement.
- Cependant, dès qu'un nombre d'individus critique est présent, l'uniformité des dépôts est rompue, par des individus attirés par d'autres dépôts. Il y a *amplification des fluctuations*. Dès que la densité de phéromone sur un point particulier est plus élevée, le comportement des termites les pousse à se diriger vers ce point pour déposer les boules de terre là où le gradient de phéromone est le plus élevé. Il s'agit bien d'un processus auto-catalytique, une rétroaction positive, puisque ces dépôts favorisent ensuite encore plus d'autres dépôts.
- Certains dépôts prennent beaucoup d'importance, et des piliers sont alors érigés en leur centre. Il y a alors morphogénèse.

– Le processus régulateur faisant que les piliers grandissent par le haut est l'évaporation de la phéromone. Celle-ci agit en rétroaction négative.
– Lorsque le nombre de piliers s'accroît, les zones d'attraction, couvertes de phéromones, commencent à se chevaucher, si bien que les zones de plus fort gradient se trouvent alors sur le côté des piliers, en direction des autres piliers, et en haut (puisque c'est là que la phéromone a été déposée en dernier). Les dépôts sur les côtés des piliers forment alors des arches les joignant.
– Ces arches sont alors étayées entre elles par d'autres, leur gradient de phéromone se recoupant aussi, si bien qu'un dôme est finalement construit.

Il y a dans cet exemple des rétroactions positives évidentes dans le phénomène auto-catalytique de formation des piliers. Les rétroactions négatives sont avant tout dues à l'évaporation des phéromones (ce qui fait que les piliers montent puisque les dépôts les plus récents sont à leur sommet), ainsi qu'au nombre d'individus.

Ce mode de communication indirecte par influence de phéromone a été nommé par Pierre-Paul Grassé *stigmergie* à partir des racines *stigma* qui signifie piqûre, et *ergon* qui veut dire travail ou œuvre. En d'autres termes, la stigmergie est un travail stimulant[4], et *auto*-stimulant. Une définition formelle de la stigmergie est donnée par [Kennedy, 2001] :

> **Stigmergie** :
> Mode de communication par la modification de l'état de l'environnement de telle manière que cela affecte le comportement des autres pour qui l'environnement est un stimulus.

On parle aussi de *recrutement*, puisque les phéromones attirent de nouveaux individus et qu'un nombre important de ces derniers est critique pour le bon fonctionnement des mécanismes de l'auto-organisation.

La stigmergie place en avant un acteur majeur : l'environnement. C'est à travers ce dernier que se font les communications. De nombreux exemples naturels montrent que l'environnement est un facteur de contrainte qui influence le développement des espèces. Cependant ici il y a un processus bouclé de modification de l'environnement pour contraindre la colonie et ainsi ses modifications futures.

À partir de règles très simples, on obtient un résultat particulièrement remarquable. Il est important de noter une certaine imprévisibilité dans le résultat dont nous reparlerons plus bas : dans le mécanisme esquissé ci-dessus, on ne sait pas où seront les piliers ni l'agencement précis de la loge. En revanche on peut être certain que la loge sera construite quel que soit le terrain et ses accidents, de manière *adaptée* à l'environnement préexistant, ce qui constitue l'une des grandes forces de ces méthodes.

4.3.5 Émergence

Jusqu'à présent nous avons parlé de l'auto-organisation comme faisant apparaître, *émerger*, des formes, structures ou organisations. On confond souvent d'ailleurs les mécanismes de l'auto-organisation avec ceux de l'*émergence*.

Bien que cette notion d'*émergence* soit encore aujourd'hui l'une des notions les plus floues et les plus discutées, nous pouvons en donner une définition satisfaisant

[4] On pourrait donc appliquer ce terme à bien d'autres tâches !

une majorité[5] :

> **Émergence (incomplète)** :
> Processus de formation de motifs complexes à partir de règles simples.

En effet, on a parlé d'apparition d'organisations au sein d'un ensemble d'entités en interaction, ces organisations en tant que tout n'étant pas programmées explicitement dans le comportement des entités en interaction.

En général on parle d'émergence lorsque le résultat des divers processus que l'on observe n'est pas prévisible à partir de la connaissance des éléments et des mécanismes de base du phénomène observé. Et effectivement l'auto-organisation telle que nous l'avons décrite est un exemple d'émergence de formes. Cependant auto-organisation et émergence sont deux concepts séparés, et nous n'utiliserons l'émergence que sous la forme citée plus haut : pour décrire les organisations qui apparaissent dans un système d'entités en interaction.

On parlera donc par la suite d'émergence pour des formes, des organisations, non prévisibles, souvent plus complexes que les éléments qui les composent, apparaissant sans que cette formation soit décrite explicitement. On le voit cette définition d'émergence concerne avant tout les simulations et les modèles créés par l'homme, donc artificiels. Nous utiliserons, plus précisément, la définition :

> **Émergence (restreinte)** :
> Apparition d'organisations à partir de règles d'interaction ne définissant pas explicitement l'apparition et le développement de ces organisations.

Cette émergence que nous utiliserons sera avant tout «dans l'œil de l'observateur», en tant qu'outil pour identifier la formation d'organisations qu'il serait difficile, voire impossible à prévoir en observant seulement les règles simples dans une simulation IBM[6].

4.3.6 Stochastique et auto-organisation

Nous avons cité trois points essentiels pour l'auto-organisation :
- les rétroactions positives ;
- les rétroactions négatives ;
- nécessité d'un nombre critique d'entités en interaction.

Cependant, un autre point s'avère souvent primordial : l'aléatoire[7]. Comme nous le verrons par la suite, et comme nous l'avons déjà vu dans l'exemple des termites, de nombreux comportements intègrent une notion d'aléatoire. Cela ne veut en aucun cas dire que la simulation en son entier devient imprévisible.

En effet, le fait que la simulation utilise des processus stochastiques à un niveau d'échelle (les entités en interaction qui la composent), ne signifie pas que le travail final fourni par la simulation soit aléatoire. Au contraire même parfois, cette part d'aléatoire peut être ce qui va rendre le modèle plus robuste aux changements, aux cas imprévus,

[5] Et qui en fait, reporte le problème sur la définition des termes «simple» et «complexe»...
[6] Par exemple, la position exacte des piliers dans la loge royale.
[7] À ne pas confondre avec l'imprévisible, voir le chapitre 3.1.5 page 46.

etc. Et c'est cette robustesse à l'imprévu qui fait d'ailleurs l'une des grandes forces des approches collectives : elles sont capables d'*adaptation*. Ici on oppose presque imprévisibilité et aléatoire, la stochastique permettant de résister aux imprévus.

Dans la construction de leur nid, les termites se déplacent d'abord aléatoirement avant que les gradients de phéromone ne les dirigent à former des piliers qui se termineront en voûtes. Ainsi lors d'une simulation de ce processus, on sait que les termites seront à peu près uniformément réparties avant le début de la construction, que les piliers seront à peu près uniformément écartés les uns des autres. On ne peut prédire exactement leur position, mais on sait que ces piliers et ces voûtes seront construits, quelque soit l'état précédent de l'environnement à cet endroit.

4.4 Vision Globale et Vision Décentralisée

Nous avons passé en revue deux des grandes forces des modèles IBM utilisant des approches collectives :

– ils permettent la création et l'observation d'organisations ;
– ils sont résistants et adaptatifs ;

Une autre de leurs forces est leur décentralisation implicite. En effet, nous l'avons vu, il n'y a pas d'individu dominant, «chef» ou «roi», qui possède une connaissance globale du projet à accomplir, et délègue ses ordres à des exécutants. Tous les individus ou entités d'un système utilisant les approches collectives ne fonctionnent qu'avec des *règles locales* et des *informations locales*. Bien entendu les entités peuvent utiliser une mémoire, mais en général leur vision du système n'est pas complète. Ce sont souvent des entités réactives en opposition au terme cognitif, où les entités seraient élaborées et capables de proactivité.

C'est cette localité dans le comportement qui rend les approches collectives *facilement distribuables* (section 3.1.6 page 47). En effet, si le but de l'approche que nous décrirons par la suite est de distribuer des simulations, il peut être particulièrement intéressant de distribuer aussi l'algorithme de distribution ! En effet, bien que l'algorithme soit plutôt léger, le comportement des fourmis que nous décrirons étant souvent très simple et rapidement exécuté par un ordinateur, il est important de considérer le nombre élevé d'informations qu'il y aura à traiter.

En effet, plus il y aura d'entités à distribuer, plus le graphe reproduisant le réseau d'interaction de la simulation sera large. Or le but est de pouvoir distribuer des applications faites d'un nombre très important d'entités. Il n'est donc pas inintéressant de penser directement à la distribution effective de l'algorithme de distribution lui-même.

Nous verrons par la suite (section 9.3 page 127) l'architecture utilisée pour distribuer l'application et l'algorithme *AntCO*^{2} de distribution sans pour autant tomber dans une récursion infinie : l'algorithme de distribution étant distribué par un autre algorithme de distribution, lui-même distribué...

4.5 Essaim de particules et Optimisation

Nous avons détaillé les mécanismes fondamentaux des approches collectives, décrit l'auto-organisation, et passé en revue leurs avantages. Nous avons vu que l'obser-

vation des animaux sociaux dans la nature est une source riche d'inspiration, cependant nous n'avons pas encore décrit de méthode s'en inspirant effectivement.

La majeure partie des applications utilisant les méthodes collectives qui ont été développées, l'ont été dans le domaine de l'optimisation[8]. Optimiser, c'est chercher à trouver la ou les meilleures solutions (maxima ou minima), d'une variable ou d'un ensemble de variables dans un espace de solutions, parfois en fonction de contraintes. Pour certains de ces problèmes[9], la recherche de la ou des solutions optimales est un problème NP-complet, et le recours à des heuristiques devient nécessaire, des algorithmes exacts n'étant pas en mesure de trouver les solutions en un temps acceptable. En conséquence il devient nécessaire de trouver un algorithme capable d'explorer l'espace de solutions de manière guidée en essayant d'éviter les zones les moins intéressantes. Or de nombreux exemples naturels sont d'une grande inspiration dans ce domaine.

Deux approches collectives maintenant utilisées majoritairement dans le cadre de l'optimisation sont les algorithmes fourmi que nous verrons dans le chapitre suivant, et l'optimisation par essaim de particules (le terme anglo-saxon consacré étant *Particle Swarm Optimization*, PSO, acronyme que nous utiliserons par la suite), que nous allons décrire ici. Cette technique a été utilisée au cours de cette thèse pour optimiser les paramètres d'*AntCO*^{2}, et sous cet angle un autre chapitre lui est consacré à la section 10.1 page 136.

FIG. 4.4: Nuée de Queleas, Delta de l'Okavango, Botswana.

PSO est une technique dérivée de l'observation des nuées d'oiseaux, mais modifiée pour que ces nuées soient dirigées vers un but particulier. On observe de tels comportements chez les oiseaux migrateurs, ou dans les nuées attirées par la nourriture (figure 4.4). Il est toujours étonnant d'observer de tels essaims. Par exemple des étourneaux, dans lesquels un ou plusieurs milliers d'oiseaux volent en une masse qui

[8] Outre le domaine de la vie artificielle, dont le but premier est la simulation de la vie et de ses mécanismes, ne cherchant pas à l'utiliser pour un projet externe.

[9] Optimisation combinatoire

semble compacte, se déformant, se divisant parfois en plusieurs essaims, puis se reformant. Le caractère d'unicité de la ou des masses d'oiseaux est le premier caractère qui frappe l'imagination. On peut aussi être étonné à l'idée du nombre important d'animaux au sein de la nuée, et de leur capacité, pourtant, à s'éviter les uns les autres. De plus, de tels nuages semblent «dirigés». En les observant, on a l'impression que les nuages volent effectivement dans une direction particulière, en tant qu'entité à part entière.

4.5.1 Boids

L'un des premiers chercheur à avoir étudié ce sujet sous l'angle de la modélisation des phénomènes de nuées est Craig Reynolds [Reynolds, 1987]. En créant les *Boids* son but était avant tout infographique, mais la solution proposée est étonnante de simplicité, et donne un modèle particulièrement fidèle du comportement des nuées d'oiseaux, des bancs de poissons, *etc*.

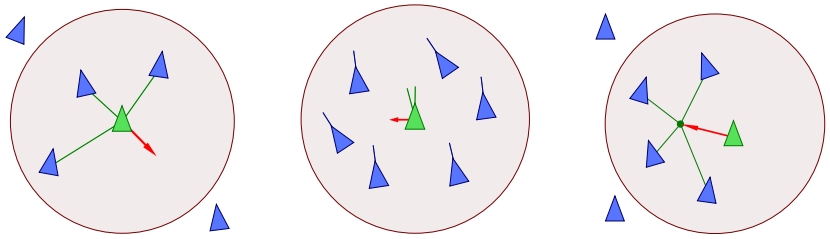
Le modèle est basé sur une représentation IBM, chaque individu au comportement très simple étant modélisé explicitement. Chaque Boid vole suivant un vecteur direction qui indique aussi sa vélocité. Durant ce déplacement, le Boid respecte trois contraintes :

Séparation Le Boid essaye d'éviter les collisions avec ses voisins (figure 4.5(a)).

Alignement Le Boid essaye de s'aligner son vecteur direction et sa vélocité avec ses voisins (figure 4.5(b)).

Cohésion Le Boid essaye de se diriger au barycentre de ses voisins (figure 4.5(c)).

Les voisins d'un Boid sont définis par une zone de vision telle que celle montrée figure 4.6 page ci-contre et qui est représentée par un rond sur les figures 4.5(a), 4.5(b) et 4.5(c) (l'angle est donné sur 180 degrés, de chaque côté du Boid).



(a) Règle de séparation. (b) Règle d'alignement. (c) Règle de cohésion.

FIG. 4.5: Les trois contraintes comportementales d'un Boid.

Tout comme pour les termites, ces règles extrêmement simples de comportement créent un comportement collectif ressemblant particulièrement au comportement de nuées d'oiseaux ou de bancs de poissons. Les figures 4.7, 4.8 et 4.9 montrent l'évolution d'un groupe de Boids au début, puis peu après le démarrage, et enfin beaucoup plus tard durant une simulation. Certaines règles ont été ajoutées pour qu'il existe différents types de Boids, ces types se repoussant avec des mécanismes d'attraction et de répulsion modélisant une représentation proies/prédateurs. On voit qu'au début,

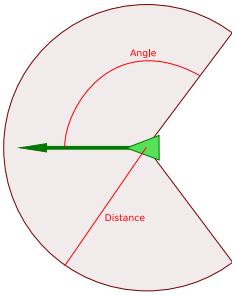


FIG. 4.6: Voisinage d'un Boid.

les Boids sont à peu près également dispersés dans l'espace. Très vite des groupes se forment. Par la suite les groupes évoluent, des individus s'échangent entre groupes lorsque ces derniers passent à proximité, ou les groupes sont dissous par le passage d'un groupe d'un autre type de boids. Cependant, la simulation présente ensuite toujours des ensembles de boids se déplaçant de concert.

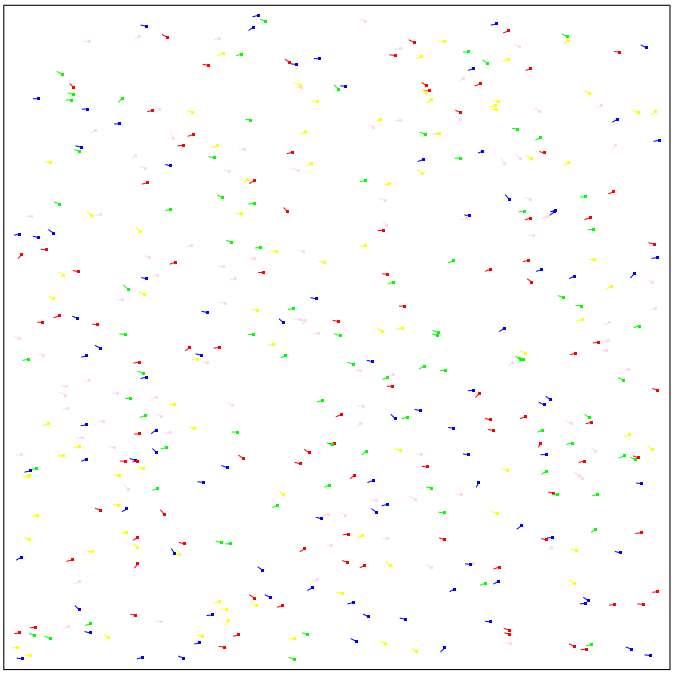


FIG. 4.7: Simulation de Boids au départ.

Lorsque l'on considère le comportement évoqué ci-dessus, on constate que bien qu'étant très simple, l'algorithme utilisé pour mettre en œuvre les Boids à une complexité en $O(n^2)$. En effet, chaque Boid est obligé de parcourir la liste de tous les autres Boids afin de savoir s'ils sont dans son champ de vision. Cependant, comme nous le verrons par la suite, il est possible avec des structures de stockage et un arrangement spatial approprié de réduire ce coût à pratiquement $O(n)$. Nous verrons ceci dans un autre chapitre consacré aux Boids, mais cette fois dans le cadre d'expérimentations sur

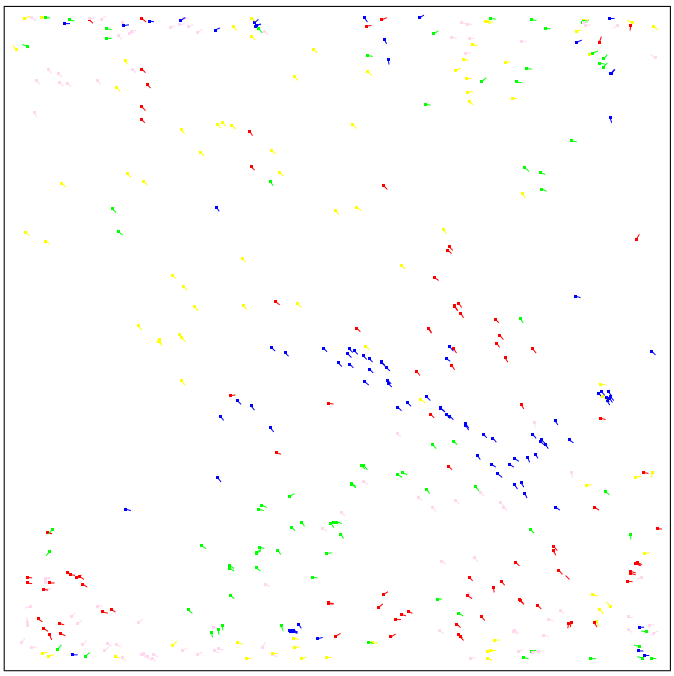


FIG. 4.8: Simulation de Boids peut de temps après le démarrage, les organisations sont formées.

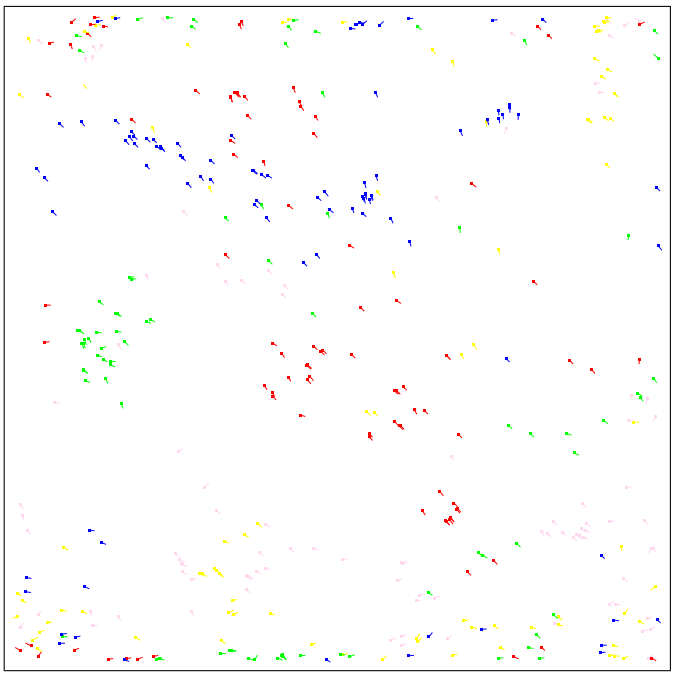


FIG. 4.9: Simulation de Boids en général, on observe le même niveau d'organisation.

l'algorithme *AntCO*^{2}, à la section 12.3 page 166.

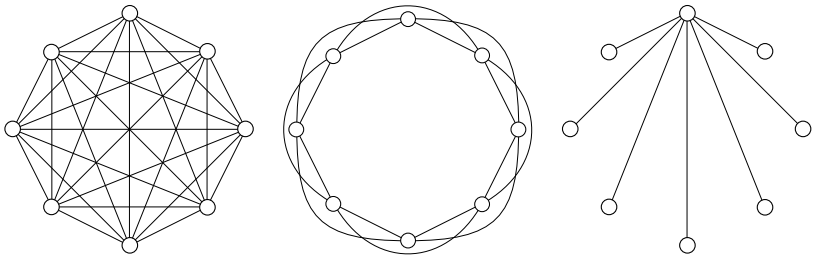
4.5.2 PSO

Les Boids n'ont d'autre but que de simuler le comportement de groupes animaliers en déplacement. PSO s'inspire des Boids mais ajoute en outre la notion d'objectif. Cette technique est inspirée des travaux de Frank Heppner et Ulf Grenader [Heppner, 1990]. Le principe de base en est simple : faire voler un essaim de particules dans un hyperespace représentant l'espace des solutions en s'inspirant du vol d'un essaim d'oiseaux, mais *en donnant aux individus un but* par l'intermédiaire d'une fonction de *fitness* $\mathcal{F}$. Ces travaux ont été développés par James Kennedy et Russel C. Eberhart [Kennedy, 1995] pour lesquels le partage d'information entre individus au sein d'une espèce peut offrir un avantage évolutif, tout comme la transmission de gènes en génétique[10].

Les règles de comportement locales sont comparables à celles de Boids dans le sens où les particules sont attirées les unes par les autres. Cependant elles différent sur les autres points :

- chaque particule i peut mémoriser la meilleure valeur rencontrée pour sa fonction de fitness $pbest_i$, ainsi que la position où cette valeur à été trouvée $\vec{x}_{pbest_i}$;
- chaque particule possède un *voisinage* dont la topologie est définie à l'avance, comme par exemple ceux présentés par la figure 4.10 ;
- dans le voisinage de la particule i la meilleure fitness rencontrée est $lbest_i$ et sa position est $\vec{x}_{lbest_i}$;
- chaque particule est attirée par $\vec{x}_{pbest_i}$;
- chaque particule est attirée par $\vec{x}_{lbest_i}$;

Si le voisinage est l'ensemble des autres particules, on note la meilleure fitness $gbest$ et sa position $\vec{x}_{gbest}$.



(a) Complet. (b) Anneau. (c) Rayon.

FIG. 4.10: Trois exemples de topologie de voisinage.

On note $\vec{x_i}(t)$ la position de la particule i au temps t et son déplacement est exprimé par :

$$\vec{x_i}(t) = \vec{x_i}(t-1) + \vec{v_i}(t-1) \tag{4.1}$$

Avec $\vec{v_i}(t)$ la vitesse de la particule i. L'algorithme 4.1 page suivante montre comment ce vecteur vitesse est calculé et utilisé. Cet algorithme utilise un voisinage global.

[10] En un mot, la culture aussi est importante !

4.1 : Optimisation par essaim de particules (PSO).

Entrées : n le nombre de particules, ρ_1 et ρ_2 sont des valeurs aléatoires positives.

Initialisations

Initialiser les particules de manière aléatoire;

Traitement

répéter

  pour i *de* 1 *à* n **faire**

    si $\mathcal{F}(\vec{x_i}) > pbest_i$ **alors**

      $pbest_i \leftarrow \mathcal{F}(\vec{x_i})$;

      $x_{pbest_i} \leftarrow \vec{x_i}$;

    si $\mathcal{F}(\vec{x_i}) > gbest_i$ **alors**

      $gbest_i \leftarrow \mathcal{F}(\vec{x_i})$;

      $x_{gbest} \leftarrow \vec{x_i}$;

  pour i *de* 1 *à* n **faire**

    $\vec{v_i} \leftarrow \vec{v_i} + \rho_1(x_{\vec{pbest_i}} - \vec{x_i}) + \rho_2(x_{\vec{gbest}} - \vec{x_i})$;

    $\vec{x_i} \leftarrow \vec{x_i} + \vec{v_i}$;

jusqu'à *convergence*;

Bien entendu cet algorithme de base peut être étendu de multiples façons :
- en jouant sur les nombres ρ_1 et ρ_2 qui influencent la vitesse ;
- en ajoutant à la particule une inertie ;
- en définissant de nouveaux voisinages, en particulier, le voisinage global n'est pas toujours indiqué car il peut mener plus facilement l'algorithme dans un optimal local ;
- *etc*.

Cette technique est particulièrement simple à mettre en œuvre. On le voit, l'espace des solutions est exploré en de multiples points, les meilleures zones découvertes par une particule étant communiquées à un voisinage donné afin de répercuter l'information. Cependant, en général le voisinage n'est pas complet, ce qui évite à l'algorithme de tomber dans des optima locaux.

C'est une technique d'optimisation particulièrement efficace, qui semble souvent mieux fonctionner que les algorithmes génétiques qui ont le défaut de ne pas «sembler suffisamment guidés». En effet le croisement de deux codes génétiques évalués comme «bons» ou «adaptés» n'apporte en aucun cas la certitude que le résultat sera lui aussi adapté. En d'autres termes, PSO utilise un vecteur vitesse que n'ont pas les algorithmes génétiques.

Chapitre 5

Fourmis numériques

Sommaire

5.1 Fourragement et Stigmergie

Lorsque nous avons introduit l'exemple de la construction d'une loge royale de termites, nous avons vu que ces dernières ne communiquaient jamais directement, tout au moins dans ce modèle de construction[1]. C'est par l'intermédiaire de «messages» déposés dans l'environnement, tels que les phéromones, que les termites se passent l'«information» de construction des piliers (bien que celle-ci soit, comme on l'a vu, quasi-implicite). Ce mode de fonctionnement est nommé *stigmergie*.

On distingue en général deux types de stigmergie :

1. la stigmergie qualitative ;
2. la stigmergie quantitative.

La première introduit la notion de sémantique attachée au message déposé dans l'environnement. La seconde introduit le concept d'attraction dans un processus auto-catalytique. En réalité, la majeure partie des stigmergies quantitatives sont aussi, bien entendu, qualitatives.

Les fourmis, très proches des termites, utilisent elles aussi la stigmergie pour mener à bien certaines tâches, et en particulier le fourragement, c'est-à-dire l'exploration d'un territoire à la recherche de nourriture, ainsi que son exploitation, une fois découverte.

[1] Dans la nature bien entendu, les termites, tout comme les fourmis, sont en mesure de communiquer de nombreuses façons, par sons (chocs sur le sol), par contact antennaire, stridulation, etc.

5.1.1 Dans la nature

Ainsi, certaines castes de fourmis, au sein d'une colonie, sont amenées par leur âge, par leur alimentation, ou simplement par leur génétique à chasser pour la colonie. Ces fourmis partent du nid et explorent les alentours de manière «aléatoire»[2].

Lorsque sa recherche est fructueuse, une fourmi revient au nid en suivant le même parcours, mais en déposant derrière elle une phéromone particulière. Elle marque ainsi son trajet et indique à ses congénères le chemin vers la source de nourriture ou tout autre matériel utile à la colonie. De tels chemins partent tous de la colonie et forment une arborescence, parfois enchevêtrée par la colonie elle même ou une autre, dont les branches ressemblent à la représentation donnée par la figure 5.1.

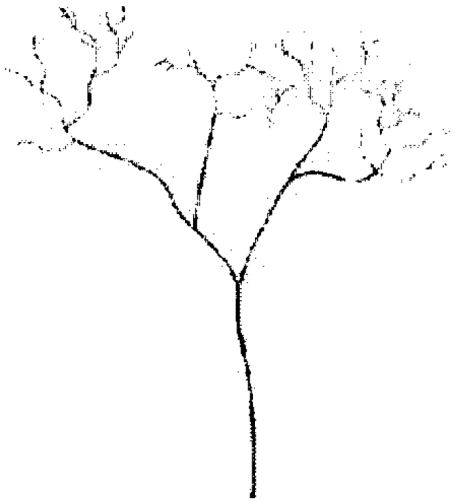


FIG. 5.1: Chemin formé par des traces de phéromones.

Les fourmis se déplacent aléatoirement si elles n'ont pas détecté de phéromones, afin d'explorer les alentours, mais sont attirées par ces dernières et le message qu'elles diffusent. Ainsi une autre fourmi en quête de nourriture rencontrant un chemin de phéromone, aura une forte inclinaison à suivre ce chemin. De plus si deux chemins se présentent à elle, il y a de grandes chances pour qu'elle choisisse celui ayant le plus fort taux de phéromones. Cependant, nous verrons par la suite que ce «choix» n'est pas totalement déterministe, ou que certains taux élevés de phéromones peuvent aussi être rédhibitoires.

Les fourmis ayant rencontré un chemin et ayant trouvé de la nourriture à son extrémité déposent à nouveau de la phéromone sur ce dernier, renforçant alors son importance, et multipliant ses chances d'être choisi par rapport à un autre. En revanche pour un chemin ne menant plus à une source intéressante, aucune phéromone ne sera déposée, et ce dernier disparaîtra, la phéromone s'évaporant au fil du temps. Les fourmis tissent usuellement autour de leur nid une véritable arborescence de tels chemins de phéromones.

[2]Même si on sait que ces fourmis sont en mesure de s'orienter par rapport à des repères tels que des accidents de terrain ou même le soleil, et que leur recherche est probablement plus guidée que cela.

Il faut noter qu'en général une fourmi sur un chemin donné n'a pas de moyen de détecter si elle va en direction du nid ou si elle s'en éloigne. Cependant, ces chemins sont «polarisés» aux embranchements. En effet, on a constaté que les embranchements sur les chemins se font toujours en forme de «Y», et que l'angle entre les deux branches est plus ou moins toujours le même. Ainsi la fourmi sait quelle direction prendre à chaque embranchement, et, revenant d'une source de nourriture, ne repart pas vers une autre, plutôt que de revenir au nid.

5.1.2 Expériences

Le pont à double voie

Jean-Louis Deneubourg dans [Deneubourg, 1989] présente une expérience mettant en exergue les mécanismes d'auto-organisation au sein d'une colonie de fourmis *linepithema humile* dans le cadre de leur recherche de nourriture. Cette expérience consiste à séparer artificiellement le nid de fourmis d'une unique source de nourriture en plaçant entre deux un pont se séparant en son milieu en deux branches identiques. La figure 5.2 montre une vue schématique de cet environnement.

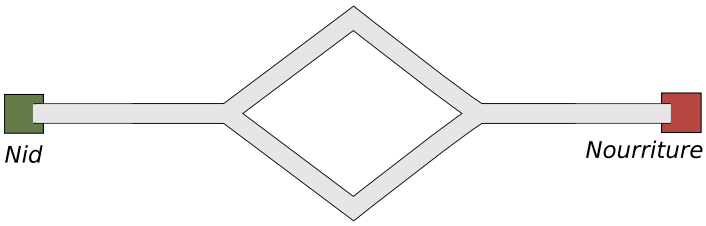


FIG. 5.2: Pont à deux voies entre un nid de fourmis et une source de nourriture.

Les fourmis ont au départ autant de chances d'emprunter les deux chemins, et au début de l'expérience, elles les traversent en nombre égal de chaque côté (photographie 5.3 page suivante). S'il n'y avait pas de dépôt de phéromone, les choix étant aléatoires, la distribution de fourmis entre chaque chemin resterait en moyenne identique. Cependant, très vite, en raison des légères fluctuations, et par le dépôt de phéromone qui en raison de ces écarts devient légèrement plus important d'un côté que de l'autre, une partie plus importante de la population est attirée sur une branche de préférence.

Ces légères fluctuations se trouvent amplifiées par le mécanisme d'attraction des phéromones, tant et si bien qu'un chemin est ensuite emprunté en priorité sur l'autre, cette tendance se fortifiant au fil du temps (photographie 5.4 page suivante).

Cependant, si la population de fourmis devient très importante, et qu'une branche du pont devient surchargée, un autre mécanisme se met en place. On peut l'observer dans la nature aussi lorsqu'un chemin de phéromone semble trop riche. Si les phéromones sont trop fortes, alors la fourmi dédaigne le chemin qu'elle juge déjà exploité suffisamment pour en choisir un autre. Ici le même processus se produit, et une branche embouteillée du pont peut être délaissée par des fourmis bien que très chargée en phéromone. Les fourmis empruntent alors les deux branches (photographie 5.5 page suivante).

Nous nous sommes inspirés de ce mécanisme dans l'algorithme *AntCO*^{2} pour ce

FIG. 5.3: Au départ les fourmis passent indifféremment par les deux voies du pont (© CNRS Photothèque VIDAL Gilles).

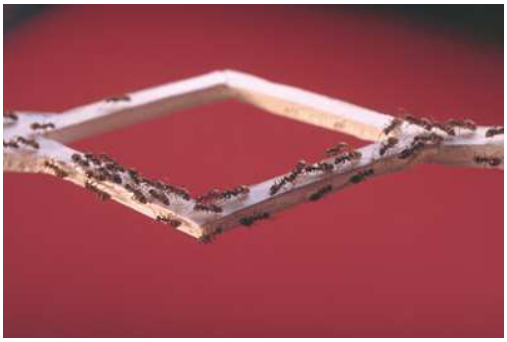


FIG. 5.4: Par amplification des fluctuations, un chemin est sélectionné (© CNRS Photothèque VIDAL Gilles).

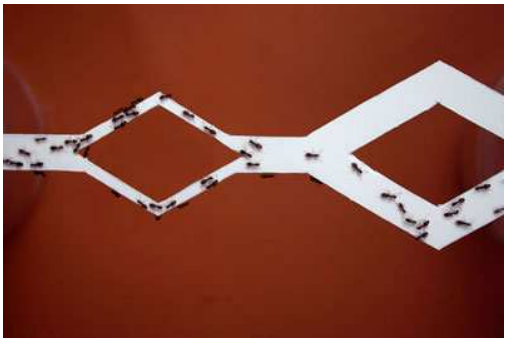


FIG. 5.5: Les embouteillages rétablissent l'usage de la seconde voie, ici démontré par l'utilisation de voies larges et fines (© CNRS Photothèque VIDAL Gilles).

que nous avons appelé la «pression démographique» et nous en reparlerons par la suite. Cependant il est important de noter les avantages qu'il peut apporter. D'une part les fourmis explorent d'autres pistes qui pourraient être devenues intéressantes, en termes d'optimisation, elles ne tombent pas dans un optimum local, mais en plus elles peuvent équilibrer leur exploitation des ressources.

Création de pistes

L'exemple suivant n'est pas une simple expérience, mais une simulation. Cependant celle-ci démontre bien de nombreux mécanismes. Elle s'inspire quasiment directement du comportement des fourmis tel qu'observé dans la nature. Ce modèle a été développé en premier par Mitchel Resnick puis décrit dans son livre [Resnick, 2000][3], ainsi que dans [Bonabeau, 1999].

Dans un environnement fait d'un nid central et de trois sources de nourriture tel que représenté à la figure 5.6(a) page 79, les fourmis auront le comportement suivant :

- Des fourmis sont lâchées dans l'environnement une par une dans l'état Recherche.
- Elles se déplacent aléatoirement jusqu'à trouver une source de nourriture.
- Si elles trouvent une source, elle changent d'état, passant à l'état Transport et retournent directement au nid.
- Sur le chemin de retour elles déposent des phéromones.
- La phéromone décroît avec la distance, mais est toujours au moins perceptible. Ainsi plus la phéromone est élevée plus on est près de la source de nourriture.
- Une fourmi en état Recherche et qui rencontre un chemin de phéromone, le suit, dans la direction où la phéromone s'intensifie.
- La phéromone, bien entendu, s'évapore, si bien qu'elle doit être renouvelée constamment par de nouvelles fourmis, sinon le chemin disparaît.
- Lorsqu'une fourmi dans l'état Transport rejoint le nid elle repasse dans l'état Recherche et repart.
- Un mécanisme spécifique est introduit pour que les fourmis dans l'état Transport rejoignent directement le nid : une phéromone spéciale est très largement diffusée par ce dernier pour leur indiquer sa position[4].

La figure 5.6 page 79 montre une telle simulation à divers instants (de gauche à droite, puis de haut en bas). La première illustre la situation initiale. Dès que les fourmis sont lâchées elles partent en toutes directions, aléatoirement. Très vite la source de nourriture la plus proche est trouvée (figure 5.6(b)). Du fait de sa proximité, un chemin de phéromone est facilement construit par plusieurs fourmis, ce qui recrute quasiment aussitôt toutes les autres (figure 5.6(c)) jusqu'à épuisement de la source (figure 5.6(d)).

Cette source épuisée, les fourmis reprennent un cheminement aléatoire (figure 5.6(e)) jusqu'à trouver la seconde source la plus proche (figure 5.6(f)) et l'épuiser à son tour (figure 5.6(g)). On voit sur cette avant-dernière figure que, bien entendu, certaines fourmis peuvent «perdre» le chemin de phéromone et continuer à chercher une source de nourriture. Certaines trouvent la troisième source de nourriture, la plus éloignée. Cependant les traces de phéromones pour une ou deux fourmis (alors que toutes les autres sont recrutées par la seconde source de nourriture) sont trop faibles pour qu'un chemin apparaisse. Enfin, après épuisement de la seconde source, la troisième est exploitée (figure 5.6(h)).

Cette dernière source est plus difficile à gérer pour les fourmis car le chemin de

[3] `http://zool33.uni-graz.at/schmickl/Self-organization/Collective_decisions/Ant_foraging/ant_foraging.html`.

[4] Le mécanisme de retour au nid implanté dans cette simulation peut sembler peu naturel. Cependant on sait que les fourmis utilisent de telles techniques pour s'orienter : utilisation de la position du soleil, mémoire de l'environnement, etc.

phéromones vers le nid est le plus long. Le nombre critique de fourmis pour maintenir le chemin est donc plus important. C'est pourquoi un chemin vers cette source de nourriture ne se forme qu'à la fin, lorsque toutes les fourmis sont disponibles pour entretenir le chemin.

On observe très facilement dans cette simulation simple les mécanismes de morphogénèse pour la création du chemin et de morphostase par évaporation pour son maintien et sa disparition. De plus l'importance cruciale du nombre d'individus est mise en avant, en particulier lors de l'exploitation de la troisième source de nourriture.

Pont à double voie de longueurs différentes

L'une des variations les plus intéressantes de l'expérience du pont à double voie de Jean-Louis Deneubourg consiste à rendre l'une des branches du pont plus longue que l'autre comme le montrent les figures 5.7 et 5.8 page 80.

Cette expérience amène des résultats étonnants : les fourmis choisissent toujours le chemin le plus court, tel qu'indiqué sur les figures 5.9 et 5.10 page 81.

Ceci est dû aux mêmes facteurs que pour l'expérience à branches identiques, cependant, ici, au lieu d'être dû aux fluctuations du nombre de fourmis choisissant une voie ou l'autre, le mécanisme de sélection de la voie la plus courte est dû à l'évaporation des phéromones. En effet, pour une fourmi il faut moins de temps pour parcourir l'un des deux chemins (les fourmis avançant approximativement toutes à la même vitesse), donc les fourmis qui choisissent la voie la plus rapide laissent plus de phéromones plus vite sur ce chemin. Ceci induit une légère préférence pour la voie la plus courte, qui est ensuite amplifiée de la même manière qu'une fluctuation aléatoire.

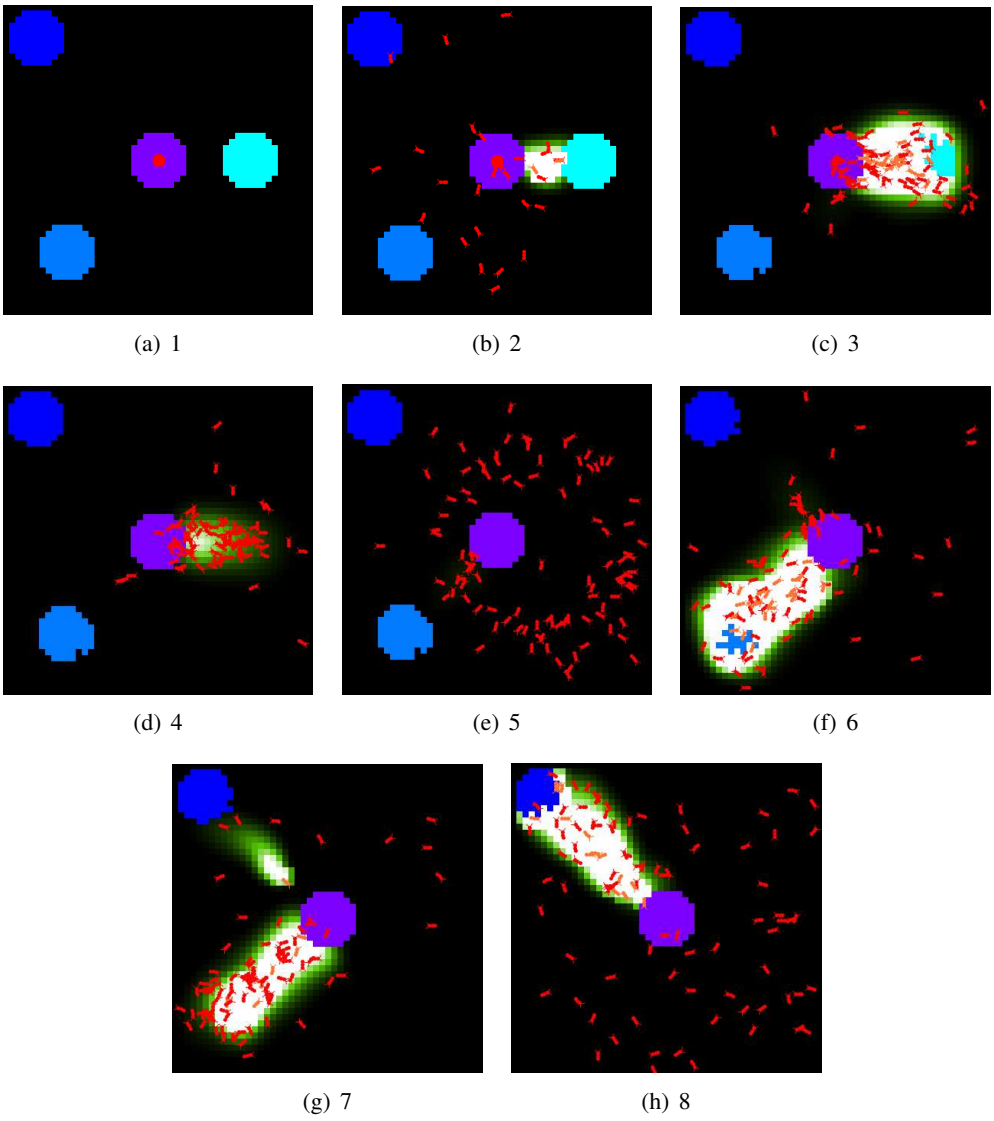
Outre mettre en évidence les mécanismes d'auto-organisation au sein d'une fourmilière, cette expérience apporte un mécanisme intéressant que l'on avait déjà vu s'esquisser avec la simulation précédente : les fourmis sont capables d'optimiser leur exploitation de la nourriture. En effet, elles sont capables de sélectionner le chemin le plus court entre plusieurs alternatives proposées comme dans l'expérience de Deneubourg. Cependant, même en terrain ouvert, elles sont en mesure d'exploiter d'abord les sources les plus proches, et donc les plus faciles.

5.2 Modèles de fourragement, et applications

Ces observations ont mené à diverses modélisations permettant d'extraire et reproduire certains comportements particuliers des fourmis. Mieux comprendre les techniques que ces dernières utilisent et savoir les reproduire permet d'utiliser ces mécanismes pour d'autres applications.

5.2.1 Simulation

Le modèle de ce comportement pour l'expérience du pont à double voie de même taille mis au point par Jean-Louis Deneubourg, S. Arom, S. Goss, J.-M. Pasteels [Goss, 1990] suppose un taux de phéromone sur chaque branche proportionnel au nombre de fourmis l'ayant traversé sans prendre en compte l'évaporation. En effet, la durée de l'expérience (d'une heure environ) n'est pas suffisante pour voir le taux de phéromone se dissiper significativement. Dans ce modèle, les branches du pont sont



(a) 1 (b) 2 (c) 3

(d) 4 (e) 5 (f) 6

(g) 7 (h) 8

FIG. 5.6: Simulation de fourragement.

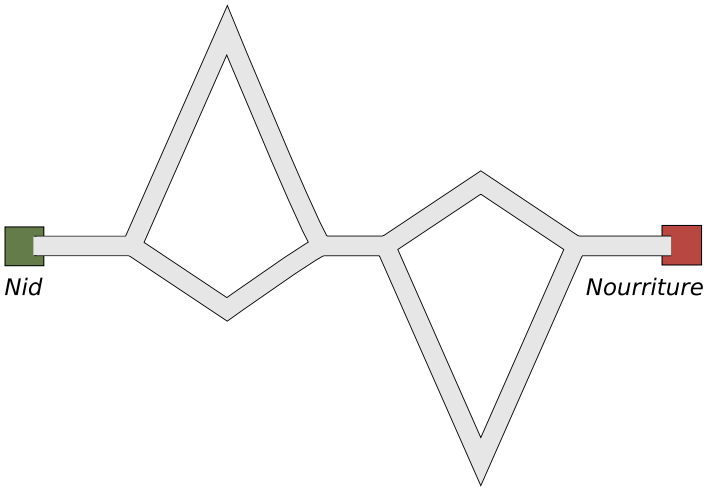


FIG. 5.7: Pont à plusieurs voies entre un nid de fourmis et une source de nourriture.

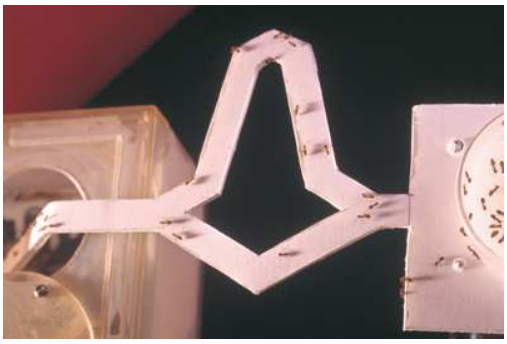


FIG. 5.8: Pont à plusieurs voies (© CNRS Photothèque VIDAL Gilles).

notées A et B. La probabilité p_A (resp. p_B) que la i-ème fourmi traverse la branche A (resp. B) est :

$$p_A = \frac{(k + A_i)^n}{(k + A_i)^n + (k + B_i)^n} = 1 - p_B \tag{5.1}$$

avec A_i et B_i le nombre de fourmis ayant traversé la branche A et respectivement la branche B. Le paramètre k permet de quantifier l'attraction d'une branche en l'absence de phéromone. Le paramètre n permet de régler l'importance des variations, en d'autres termes la non-linéarité de la réponse aux fluctuations.

Il faut noter pour l'expérience du pont à double voie de longueur différente, [Goss, 1989], [Bonabeau, 1999] que certaines fourmis peuvent ne pas sélectionner le plus court chemin, si trop de phéromones sont déposées trop vite sur le plus long. La fourmi ne connaissant pas à l'avance le plus court chemin choisira celui comportant le plus de phéromone, et le chemin le plus long restera le plus emprunté. En effet, dans la nature les phéromones ne s'évaporent pas forcément vite. Les vitesses d'évaporation varient de l'heure au mois en fonction de l'espèce ou des condition environnementales. Ainsi la sélection des chemins les plus courts ne se fait pas forcément rapidement, et cer-

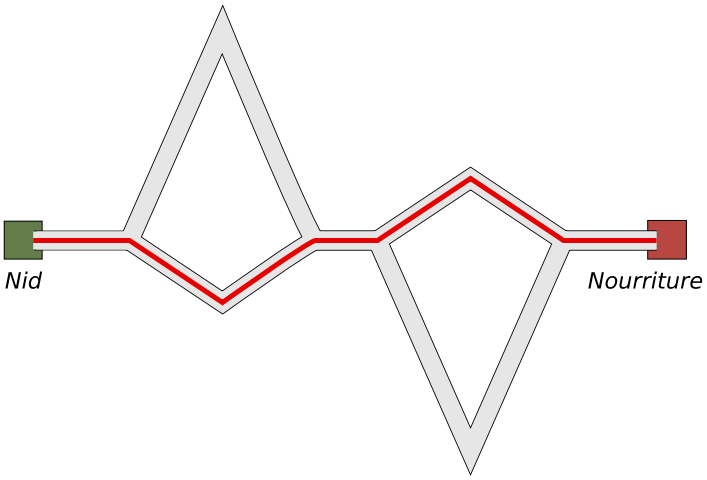


FIG. 5.9: Choix du chemin le plus court.

FIG. 5.10: Choix du chemin le plus court (© CNRS Photothèque VIDAL Gilles).

taines espèces développent en conséquence d'autres mécanismes telles les fourmis *Lasius Niger*, qui sont en mesure de détecter le fait qu'elles sont en train de s'éloigner de leur direction privilégiée vers le nid, et font demi-tour pour explorer l'autre voie.

Nous verrons dans la présentation de notre méthode qu'ainsi les mécanismes liés aux phéromones ne sont parfois pas suffisants, et que d'autres comportements sont à étudier pour arriver à un modèle satisfaisant, à la fois pour ne pas tomber dans des optima locaux, mais aussi pour sélectionner *dynamiquement* de nouvelles voies intéressantes.

Cependant, entre les modèles ayant des buts précis et la nature, l'utilisation de phéromones bien plus volatiles est possible, ce qui permet des méthodes d'optimisation particulièrement intéressantes.

5.2.2 Applications

Marco Dorigo dans sa thèse [Dorigo, 1992] a exploité cette capacité à déterminer le plus court chemin. Puisque les fourmis sont en mesure d'optimiser leur exploitation de la nourriture, ne serait-on pas en mesure d'appliquer ces techniques à d'autres pro-

blèmes ? Dans son manuscrit, il applique ainsi cette idée au problème du voyageur de commerce (ou en anglais TSP, Travelling Salesman Problem).

Ce problème consiste à trouver le chemin le plus court passant une fois, et une seule fois par toutes les villes d'un ensemble connectées par des chemins. On peut représenter ce problème par un graphe où les sommets sont les villes et les arcs valués les chemins et leur distance. Bien entendu, toutes les villes sont connectées, c'est-à-dire que le graphe est connexe. Ce problème est connu pour être NP-complet. On dit du TSP qu'il est euclidien, si la mesure de distance entre les villes est euclidienne, c'est-à-dire si entre deux villes $u = (x_u, y_u)$ et $v = (x_v, y_v)$ dans le plan reliées par un arc $e = (u, v)$, la distance est

$$d_e = \sqrt{(x_u - x_v)^2 + (y_u - y_v)^2} \tag{5.2}$$

On utilise ainsi un graphe dont les arcs sont valués par la distance séparant les sommets dans le plan.

L'algorithme introduit par Marco Dorigo [Dorigo, 1996] a été nommé Ant System ou AS[5]. Il consiste à utiliser un ensemble de m fourmis $\mathcal{F}(t)$ initialement déposées aléatoirement sur les sommets du graphe. Chacune de ces fourmis se déplace ensuite suivant un algorithme simple :

- elles choisissent la prochaine ville où se déplacer suivant une probabilité qui est fonction de la distance de la ville et de la quantité de phéromone présente sur l'arc y menant ;
- elles ont une mémoire permettant d'éviter qu'elles ne repassent par des villes déjà visitées, ceci afin de parcourir un tour valide par rapport au TSP ;
- après chaque tour terminé, une quantité de phéromone donnée est déposée sur chaque arc emprunté durant le trajet.

La quantité de phéromones sur un arc $e = (u, v)$ au temps t est notée $\tau^{(t)}(e)$. Le nombre de fourmis sur la ville v au temps t est noté $b_v(t)$. L'algorithme fonctionne par cycles. Un cycle est terminé quand toutes les fourmis ont achevé un tour complet, ainsi, chaque cycle dure n itérations s'il y a n villes dans le graphe. À la fin de chaque cycle, les quantités de phéromones sont mises à jour :

$$\tau^{(t+n)}(e) = \rho \cdot \tau^{(t)}(e) + \Delta^{(t+n)}(e) \tag{5.3}$$

où ρ représente la persistance de la phéromone (la quantité de phéromone conservée après évaporation) sur l'arc e entre les étapes t et $t + n$. De même :

$$\Delta^{(t+n)}(e) = \sum_{k=1}^{m} \Delta_k(e) \tag{5.4}$$

où $\Delta_k(e)$ est la quantité de phéromone par unité de distance déposée sur l'arc e par la k-ième fourmi entre les itérations t et $t + n$. Cette valeur est donnée par :

$$\Delta_k(e) = \begin{cases} \frac{Q}{L_k} & \text{Si la } k\text{-ième fourmi utilise l'arc } e \text{ dans son chemin ;} \\ 0 & \text{Sinon.} \end{cases} \tag{5.5}$$

[5]Les notations employées dans cet article ont été modifiées pour correspondre avec les notations que nous utiliserons par la suite. Un index des notations peut être trouvé à la section 8.2 page 195.

avec Q une constante et L_k la longueur du chemin de la k-ième fourmi.

Afin d'éviter de revenir sur une ville déjà parcourue, et ainsi de créer un chemin qui ne serait pas valide pour le TSP, chaque fourmi possède une mémoire qui lui permet de stocker un maximum de n villes déjà rencontrées. Cette mémoire possède un rôle double : empêcher le retour sur des villes visitées, mais aussi, sauvegarder le chemin parcouru par la fourmi afin de déposer les phéromones à la fin de chaque cycle. Cette mémoire est une liste notée $tabu_k$ évoluant dynamiquement avec l'avancement de la fourmi k.

La probabilité qu'une fourmi k passe de la ville u à la ville v par l'arc $e = (u, v)$ est définie par :

$$p_k^{(t)}(e) = \begin{cases} \dfrac{(\tau_e(t))^\alpha \cdot (\eta_e)^\beta}{\sum_{e \in \mathcal{E}_u} (\tau_e(t))^\alpha \cdot (\eta_e)^\beta} & \text{si } e \in \mathcal{E}_u \text{ ;} \\ 0 & \text{sinon.} \end{cases} \tag{5.6}$$

avec η_e un facteur de visibilité (représentant la valeur $1/d_e$), $\mathcal{E}_u$ l'ensemble des arcs incidents à u privé de ceux menant à une ville de $tabu_k$ et α et β des paramètres de calage permettant de régler l'importance relative des phéromones par rapport à la visibilité d'un sommet. L'algorithme 5.1 page suivante synthétise ce comportement.

Cet algorithme est à la base des travaux de Marco Dorigo sur l'Ant Colony Optimization (optimisation par colonie de fourmis, ACO) et d'autres sur la «méta-heurisitique fourmi», mais aussi de nombreuses dérivations, ainsi qu'un grand nombre de travaux sur d'autres problèmes d'optimisation, parmi lesquels on peut citer le problème de l'ordonnancement séquentiel [Dorigo, 1997], le problème de l'assignement quadratique [Gambardella, 1999], des applications au trafic routier [Bullnheimer, 1999] et [Bertelle, 2003b], au problème des scheduling [Forsyth, 1997], de coloration de graphe [Costa, 1997], de partitionnement de graphes [Kuntz, 1997], d'application aux télécommunications [Di Caro, 1997], la liste n'est pas exhaustive.

L'algorithme que nous présentons à la partie III page 113 s'inspire largement du modèle exprimé ici, bien que son mode d'opération et ses buts soient différents.

Il est intéressant, pour finir, de considérer toutes les approches qui ont découlé de l'observation des mécanismes de la nature avec cette citation de Jean-Louis Deneubourg, dans un article paru en 1995, il y a 10 ans :

> « J'ignore si les idées exposées dans cet article auront un avenir. Le fait qu'elles correspondent à des solutions proposées par la nature ne saurait garantir leur succès. Malgré les tentatives et les rêves des constructeurs des premières machines volantes, il y a aujourd'hui beaucoup plus d'engins à ailes fixes que d'ornithoptères mimant le vol des oiseaux ! Les solutions retenues par un système vivant ne valent en effet que pour un ensemble de contraintes et de paramètres qui ne sont pas forcément universels. Mais c'est bien parce que notre savoir scientifique et technologique reste limité, que nous devons ne laisser aucune piste inexplorée, et ne négliger aucune source de connaissances ou d'inspiration. »

(J-.L. Deneubourg, «Individuellement, les insectes sont bêtes, collectivement, ils sont intelligents...», Le Temps Stratégique, n°65, Genève, Septembre 1995, EdiPresse).

5.1 : Ant System.

t compteur de temps, nc compteur de cycles.

1 $t \leftarrow 0$;
2 $nc \leftarrow 0$;
3 **pour** *tout arc* $e = (u, v)$ **faire**
4 Placer une valeur de phéromone initiale $\tau_e(t) \leftarrow c$;
5 Placer $\Delta^{(t)}(e) \leftarrow 0$;
6 Placer m fourmis sur les n nœuds ;

s l'index dans la liste tabou.

7 $s \leftarrow 1$;
8 **pour** *les fourmis* k *de* 1 *à* m **faire**
9 Placer la ville de départ de la fourmi k dans sa liste tabou;
10 **répéter**
11 **pour** *les fourmis* k *de* 1 *à* m **faire**
12 Choisir la ville u sur laquelle se déplacer avec une probabilité $p_k^{(t)}(e)$;
13 Bouger la fourmi k sur la ville u;
14 Insérer u dans sa liste tabou;
15 **jusqu'à** *ce que toutes les listes tabou soient pleines*;
16 **pour** *les fourmis* k *de* 1 *à* m **faire**
17 Bouger la fourmi k de la dernière ville de la liste tabou à la première;
18 Calculer la longueur L_k de ce tour;
19 Mettre-à-jour le plus court tour trouvé;
20 **pour** *tout arc* $e = (u, v)$ **faire**
21 **pour** *les fourmis* k *de* 1 *à* m **faire**
22 $\Delta_k(e) \leftarrow \begin{cases} \frac{Q}{L_k} & \text{Si la } k\text{-ième fourmi utilise l'arc } e \text{ dans son chemin ;} \\ 0 & \text{Sinon.} \end{cases}$;
23 $\Delta(e) \leftarrow \Delta(e) + \Delta_k(e)$;
24 **pour tous les** *arcs* e **faire** Calculer $\tau^{(t+n)}(e) \leftarrow \rho \cdot \tau^{(t)}(e) + \Delta(e)$;
25 $t \leftarrow t + n$;
26 $nc \leftarrow nc + 1$;
27 **pour tous les** *arcs* e **faire** Placer $\Delta(e) \leftarrow 0$
28 **si** $nc < nc_{max}$ **alors**
29 Vider toutes les listes tabou;
30 Aller à la ligne 7;
31 **sinon**
32 Afficher le tour le plus court;
33 Stop;

Deuxième partie

État de l’art

Chapitre 6

Le problème

Sommaire

Nous nous sommes désormais équipés d'un nombre suffisant de références et de définitions pour entamer une description plus formelle, moins intuitive, du problème de la distribution dynamique adaptative. Cette description est nécessaire avant de passer en revue les techniques et modèles existants afin de mieux cerner les caractéristiques nécessaires à la distribution dynamique.

Nous essayerons tout au long de ces descriptions de conserver une notation homogène, dont les principaux éléments sont répertoriés en fin de manuscrit dans un index des notations (p. 195).

6.1 Objectifs

L'objectif principal, la distribution dynamique adaptative, se décompose en trois points :

1. équilibrer la charge des ressources de calcul ;
2. minimiser la charge réseau ;
3. savoir gérer la dynamique de la simulation, être en mesure de s'y adapter.

Nous verrons dans le chapitre suivant que de nombreuses méthodes ne proposent qu'un ou deux des trois points définis ci-dessus. En règle générale, la dynamique n'est pas gérée, ce qui ne permet pas de comparer facilement le modèle que nous proposons.

6.2 Le problème

Nous représenterons l'application à distribuer par un graphe pondéré dynamique $G(t) = (\mathcal{V}(t), \mathcal{E}(t))$, où $\mathcal{V}$ est un ensemble de sommets représentant les *entités* de l'application à distribuer au temps t et $\mathcal{E}$ un ensemble d'arcs valués représentant des *interactions* entre les entités au temps t. La pondération d'un arc e notée $w^{(t)}(e) \in \mathbb{R}^+$ représentant son importance.

Nous traitons ce graphe comme la représentation d'un réseau d'interactions, mais nous le considérons comme dénué de toute sémantique liée à l'application. C'est-à-dire que si des organisations sont trouvées dans le graphe, elle ne correspondront pas forcément à des organisations faisant sens par rapport à l'application.

Nous avons pris le parti de considérer que les interactions entre les entités sont des communications sous une forme ou sous une autre, ceci en majeure partie parce que nous aurons à distribuer les entités qui composent l'application sur un réseau, et que des communications seront nécessaires, là où il y a interaction. Ainsi la valuation sur les arcs indiquent le poids des communications.

De plus ce graphe est *dynamique*, c'est à dire qu'à tout moment sa topologie et sa valuation peuvent changer. Ainsi à tout moment :

- des sommets peuvent apparaître ou disparaître ;
- des arcs peuvent apparaître ou disparaître ;
- les valuations sur les arcs peuvent changer.

Il est important de noter que nous considérons ici le graphe et sa dynamique comme une structure de donnée informatique dont le nombre de sommets et d'arcs peuvent changer. Ainsi le graphe qui à un moment donné peut avoir 12 sommets, est *le même* que celui qui 10 itérations plus tard possède 31 sommets. Bien entendu, la topologie peut être très différente, mais il s'agit d'une modification de la structure du graphe, avec conservation des données qui peuvent être attachées à un sommet ou à un arc. Une vision mathématique des graphes pourrait faire penser qu'un graphe est une instance de l'ensemble des graphes possibles, tout comme le nombre 4 est une instance d'un réel. Un graphe donné avec sa structure serait alors invariant, et ainsi à chaque étape on aurait un autre graphe, la dynamique étant représentée par une *famille de graphes*. Le second point de vue est aussi valide tant que l'on considère que certaines données sont attachées aux «identifiants» de sommets et des arcs et que d'un graphe à l'autre ces données sont préservées.

Les «évènements» qui font la dynamique du graphe représentant l'application à distribuer ne regroupent pas cependant tous les éléments de dynamique que l'on doit être en mesure de gérer. En effet, parallèlement à ce graphe représentant l'application, un ensemble de ressources de calcul variant avec le temps est maintenu. On considère que toute ressource de calcul peut communiquer avec toute autre. C'est-à-dire que si on les représentait par un graphe, ce dernier serait complet. Cet ensemble est lui aussi changeant, et ajoute ainsi un autre élément de dynamique. Voici la liste complète des «événements» possibles :

- des sommets peuvent apparaître ou disparaître ;
- des arcs peuvent apparaître ou disparaître ;
- les valuations sur les arcs peuvent changer ;
- des ressources de calcul peuvent apparaître ou disparaître.

Afin d'incorporer une indication de distribution, et d'indiquer quel sommet *de-*

vrait se trouver sur quel ressource de calcul, le graphe est coloré. On note $G(t) = (\mathcal{V}(t), \mathcal{E}(t), \mathcal{C}(t))$, avec $\mathcal{C}(t)$ l'ensemble des couleurs au temps t. Chacune des ces couleurs identifie de manière unique une ressource de calcul. De plus, à chaque sommet est associé une couleur qui indique sur quelle ressource de calcul l'entité qu'il représente devrait s'exécuter. Les figures 6.1 et 6.2 page suivante montrent le même graphe mais coloré de deux façons différentes. L'épaisseur des arcs indique l'importance des liens. La première figure identifie une distribution ou la charge machine est correctement égalisée, mais la charge réseau n'est pas minimisée. La seconde donne le même résultat pour la charge machine, mais minimise en outre la charge réseau.

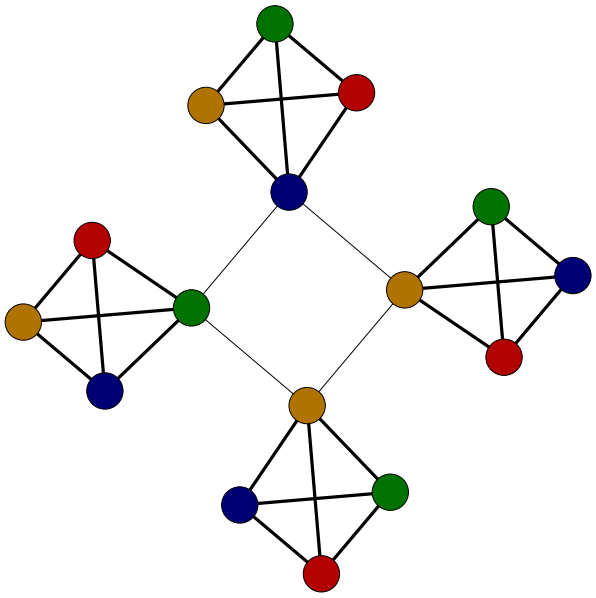


FIG. 6.1: Un graphe pondéré égalisant la charge machine mais ne minimisant pas la charge réseau.

Ce graphe est, comme on l'a dit, distinct de l'application, il fournit par sa coloration l'état de distribution que l'on attend de l'application. Il est bien sûr possible que le graphe indique une couleur pour un sommet qui ne correspond pas à la réalité, l'entité peut ne pas encore avoir migré sur la ressource distante.

Nous utiliserons dans la mesure du possible cette formalisation de l'application et de sa distribution dans la description de notre modèle, ainsi que dans cette partie, qui présente plusieurs méthodes de distributions existantes.

6.3 Propriétés pour un algorithme de distribution dynamique adaptative

Avant de passer à une description des méthodes de distribution déjà existantes nous allons passer en revue certaines caractéristiques essentielles qu'un algorithme doit posséder pour effectuer une distribution dynamique. Ceci nous permettra d'indiquer quels algorithmes répondent à quelles attentes.

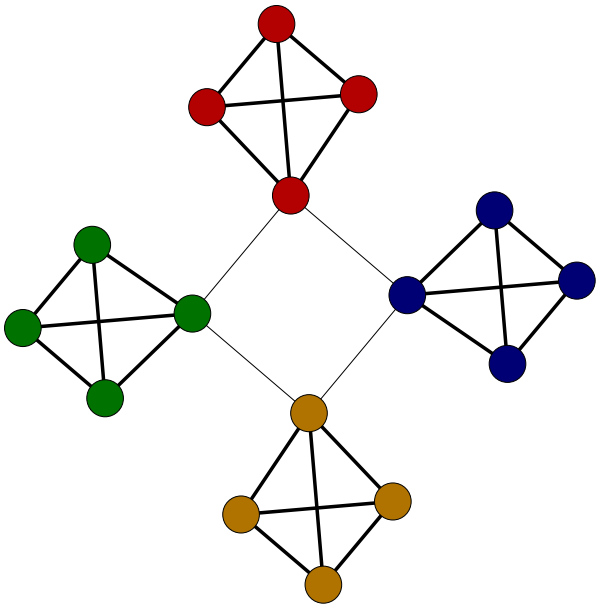


FIG. 6.2: Un graphe pondéré égalisant la charge machine et minimisant la charge réseau.

6.3.1 Dynamique

Est statique un algorithme qui calcule une solution de distribution pour une application donnée une fois pour toutes en amont de l'exécution effective de cette application. A contrario, est dynamique tout algorithme capable de calculer une distribution de l'application à tout moment quelque soit l'état de l'application et en continu. La littérature est abondante pour la première catégorie, et plus faible pour la seconde, à laquelle cette thèse s'intéresse.

C'est une différence importante dans le paradigme d'utilisation, et si elle réside avant tout dans l'algorithme de distribution, elle dépend aussi de l'application à distribuer.

6.3.2 Temps réel au sens large

À la caractéristique de dynamique s'ajoute une autre, qui n'est par définition pas présente dans les méthodes statiques et qui va s'appliquer au travail présenté dans cette thèse.

Nous qualifions de *temps réel* les méthodes de distributions qui peuvent à tout moment donner une solution de distribution pour une application sans recalculer une nouvelle solution complète, mais en améliorant/modifiant les solutions précédentes. Le terme «temps réel» est utilisé dans plusieurs acceptions. Ici il signifie qu'une solution acceptable est toujours consultable par l'application à distribuer, sans qu'un calcul préalable ne soit nécessaire. L'application à distribuer et *AntCO*^{2} s'exécutent de concert. Un tel algorithme n'est pas dirigé vers un optimum car les données, et ainsi le problème à traiter, changent en continu.

Nous utiliserons cette caractéristique dans un classement des approches collectives

au début de la partie III page 113 afin de différentier les approches et les besoins.

6.3.3 Gestion de charges

Nous distinguons une autre propriété importante : la gestion de charge réseau. Certains algorithmes ne distribuent que la charge machine, d'autres tiennent en plus compte des interactions entre les diverses parties distribuées. Ce second point est d'autant plus important que la *granularité* de l'application, c'est-à-dire le nombre d'entités distribuables qui la composent, est importante, comme nous l'avons vu à la section 2.4 page 37.

Si cent hommes sont présents pour creuser un trou, ils n'iront pas cent fois plus vite qu'un seul, car il faudra les organiser, et mettre en œuvre une méthode pour qu'ils se synchronisent ou qu'ils ne se gênent pas.

La figure 6.3 récapitule ces trois propriétés.

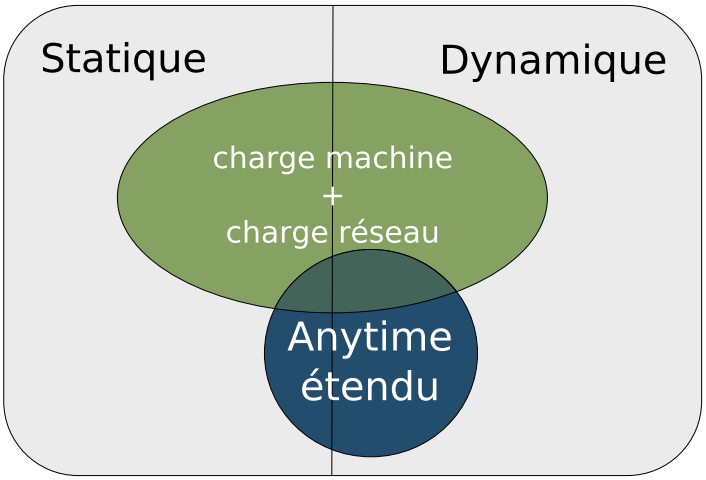


FIG. 6.3: Propriétés des algorithmes de distribution.

6.4 Mesures de qualité

Nous utilisons deux critères principaux de mesure de la qualité des distributions produites. Le premier se rapporte à la charge réseau, le second à la charge machine. Si deux critères sont nécessaires, c'est qu'il est particulièrement délicat d'obtenir une unique mesure de qualité pour des concepts qui s'opposent tel que la charge réseau et la charge machine. De manière évidente en effet, optimiser la charge machine revient à dégrader la charge réseau, et inversement optimiser la charge réseau revient à dégrader complètement la charge machine, en n'en utilisant qu'une seule. Ces critères sont contradictoires et toute approche de distribution dynamique devra trouver un *compromis*.

Le premier critère, r_1, représente la charge de communication. Il introduit la notion de *communication effective*, c'est-à-dire de communication réifiée sur le réseau,

opposé aux communications directes qui se font entre deux entités *sur la même ressource de calcul*. En effet, le temps nécessaire à l'établissement d'une communication entre deux ressources de calcul prend 100 fois et parfois plus de temps qu'une communication directe entre deux entités sur la même ressource. Étant donné le graphe coloré de l'application $G(t)$ au temps t, les communications effectives sont identifiées par l'ensemble $\mathcal{A}(t)$ des arcs entre deux sommets de couleurs différentes au temps t. Les communications directes sont donc $\mathcal{E}(t) \cap \mathcal{A}(t)$. Le critère r_1 est la proportion de communications effectives rapporté au nombre de communications total :

$$r_1 = \frac{\sum_{a \in \mathcal{A}(t)} w^{(t)}(a)}{\sum_{e \in \mathcal{E}(t)} w^{(t)}(e)} \tag{6.1}$$

Plus ce critère est proche de 0, meilleur il est.

Le second critère, r_2, est lié à la charge des ressources de calcul. Pour ce dernier on prend en compte la charge de la ressource la plus occupée, et celle de la ressource la moins occupée. Ainsi, si $\mathcal{V}_c$ représente l'ensemble des sommets ayant la couleur c :

$$r_2 = \frac{min(card(\mathcal{V}_1), \ldots, card(\mathcal{V}_n))}{max(card(\mathcal{V}_1), \ldots, card(\mathcal{V}_n))} \tag{6.2}$$

Plus r_2 est proche de 1, meilleur il est. Le minimum et le maximum sont normalisés en fonction de la puissance de la machine, la machine la plus puissante ayant une puissance de 1. Ceci permet de prendre en compte les ressources de calcul hétérogènes.

Chapitre 7

Méthodes de distribution statiques

Sommaire

Les méthodes de distribution statiques sont étroitement corrélées aux problèmes de *placement* et de *partitionnement* dans les graphes. Il est intéressant d'abord de se pencher sur le vocabulaire usuel du partitionnement afin de cerner les termes et les concepts proches.

7.1 Partitionnement de graphe

Le problème du partitionnement de graphe peut être identifié au notre. Les buts sont d'équilibrer la taille des partitions dans un graphe tout en minimisant le nombre d'arcs inter-partitions. Le partitionnement de graphe est utilisé pour distribuer des tâches sur un ensemble de processeurs, mais aussi par exemple pour trouver des découpes idéales lors de la création et l'intégration de circuits imprimés et de processeurs à très large échelle (VLSI, Very Large Scale Integration), ou encore pour la création et la distribution de réseaux de télécommunication.

La différence majeure, est que la majorité des algorithmes de partitionnement de graphe opèrent sur des graphes statiques. Leur seul but de trouver une solution optimale pour cet unique graphe. En conséquence les méthodes développées ne sont pas adaptées à la dynamique d'une simulation, qui est le but que nous nous sommes fixés.

Dans le vocabulaire du partitionnement de graphe on parle souvent d'*edge-cut* pour indiquer, soit l'ensemble de arcs qui relient deux partitions distinctes, soit leur nombre.

On parle aussi d'*edge-weight* pour désigner la somme des pondérations des arcs inter-partitions :

$$\sum_{e \in \mathcal{A}(t)} w^{(t)}(e) \tag{7.1}$$

Avec $\mathcal{A}(t)$ l'ensemble des arcs inter-partitions. L'edge-cut correspondant à un edge-weight avec tous les arcs pondérés à 1.

En général la charge est représentée par la taille de la plus grande partition, que l'on peut comparer à la taille optimale des partitions qui serait :

$$\frac{n}{p} \tag{7.2}$$

Avec n le nombre de sommets dans le graphe (soit $card(\mathcal{V}(t))$) et p le nombre de processeurs (soit $card(\mathcal{C}(t))$).

Ces mesures sont très proches des notres. Le critère r_1 correspond à :

$$\frac{edgeweight}{card(\mathcal{E}(t))} \tag{7.3}$$

Et le critère r_2 implique à la fois la taille de la partition minimale et maximale, plutôt qu'uniquement la partition maximale. Cela nous permet d'avoir des mesures dans $[0..1]$.

7.2 Bissection récursive

La bissection récursive part du principe qu'il est plus facile de distribuer un graphe sur 2 ressources de calcul que sur $n > 2$, puis de redécouper le problème en sous-problèmes récursivement. C'est la technique «diviser pour régner». Ainsi on découpe d'abord le graphe en 2 partitions que l'on cherche à égaliser, puis on relance le processus récursivement sur chacune des deux partitions.

Cette approche possède l'avantage d'être naturellement parallélisée, et donc facilement distribuable. D'abord sur 2 ressources de calcul, puis sur 4, et toutes les puissances de 2 jusqu'à la profondeur voulue. L'inconvénient majeur étant que seul un nombre puissance de 2 de partitions peut être généré.

La bissection récursive n'est pas une méthode en soit, mais une façon de procéder au découpage. On utilise des méthodes de bissection (simple) diverses que l'on réutilise ensuite récursivement. Nous allons donc nous intéresser aux méthodes de bissection.

Il a été prouvé que ce problème est NP-complet [Garey, 1979]. En effet, pour trouver la bissection optimale, celle qui produit deux sous-graphes de taille équivalente ayant un edge-weight minimal, il faudrait essayer toutes les découpes possibles. Plusieurs heuristiques ont été mise en place. Un grand nombre de ces dernières sont des méthodes dites «géométriques» éloignées de ce que nous cherchons à réaliser. Nous allons nous intéresser aux méthodes à base de graphes.

7.3 Partitionnement «glouton»

L'un des premiers algorithmes dédiés à la bissection simple est la méthode «gloutonne». Le principe de l'algorithme repose sur une amélioration d'une partition existante. Il est possible de commencer avec un graphe partitionné arbitrairement. L'algorithme étant une méthode de bissection, il travaille toujours sur deux partitions à la fois, et les améliore.

Étant donné un graphe pondéré $G = (\mathcal{V}, \mathcal{E})$, coupé arbitrairement en deux ensembles A et B de même taille à un sommet près, $G = A \cup B$. Le principe consiste à échanger deux-à-deux des sommets si un tel échange améliore le cut-weight, jusqu'à ce que le gain maximum soit moins que zéro, ou égal à zéro (ou qu'une autre heuristique détermine qu'il est tant d'arrêter).

Pour déterminer les gains et donc trouver la paire de sommets offrant le gain maximal, on définit :

- Le coût externe E du sommet u est le nombre d'arcs $e = (u, v) \in \mathcal{E}$ avec $v \in B$.
- Le coût interne I du sommet u est le nombre d'arcs $e = (u, v) \in \mathcal{E}$ avec $v \in A$.
- Le coût total D pour u est donc $D(u) = E(u) - I(u)$.
- Le gain d'échange entre u et v devient $gain(u, v) = D(u) + D(v) - 2\omega(u, v)$, ou $\omega(u, v) = 1$ si $(u, v) \in \mathcal{E}$, et $\omega(u, v) = 0$ sinon.

Il est possible d'améliorer la recherche du meilleur gain [Ferencz, 2005] en stockant les coûts internes et externes dans chaque sommet. Cette étape est effectuée au préalable, et les coûts peuvent être mis à jour après chaque échange. Le meilleur choix requiert ainsi de parcourir les $(n/2)^2$ paires de sommets possibles, et la complexité est donc en $O(n^2)$.

7.4 Kernighan et Lin

Une évolution de cet algorithme est donnée par B. Kernighan et S. Lin. En effet l'algorithme glouton ne donne pas de très bons résultats car beaucoup de graphes ont une meilleure bissection qui requiert d'échanger plusieurs sommets à la fois afin de produire un gain. L'algorithme glouton, ne peut chercher de nouvelle bissection que dans l'espace de solutions des bissections qui sont à un échange de la bissection courante. L'algorithme Kernighan-Lin (KL) [Kernighan, 1970] échange plusieurs sommets à la fois afin de chercher des bissections éloignées de plusieurs échanges.

Cet algorithme possède une complexité en $O(n^3)$, mais il existe plusieurs améliorations à cet algorithme de base.

L'algorithme KL essaye à chaque étape d'échanger non pas une, mais k paires de sommets avec $1 \leq k \leq \frac{n}{2}$. L'algorithme est divisé en passes successives. Une passe considère plusieurs échanges de paires de sommets :

1. Considérer l'échange de la paire de sommets ayant le plus haut gain, même si ce gain est négatif, et marquer les sommets de cette paire.
2. Répéter l'étape précédente sur tous les sommets non marqués, jusqu'à ce que tous les sommets soient marqués.
3. Choisir k tel que le coût de la bissection à la répétition k des étapes précédentes soit le plus petit.

4. Échanger les k premières paires déterminées dans le graphe original.

Ainsi cet algorithme plutôt que d'échanger une paire à la fois, considère une séquence d'échange de paires de sommets et choisit la meilleure séquence, si son gain est positif. L'algorithme s'arrête dès que le gain d'une passe est nul ou négatif. L'algorithme 7.1 décrit plus en détail ces étapes.

7.1 : Heuristique de bissection de Kernighan et Lin

Calculer $cutweight \leftarrow$ coût des partitions $\mathcal{V} = A \cup B$;
répéter
  Calculer les coûts $D(v)$ pour tout $v \in \mathcal{V}$;
  Enlever les marques sur tous les sommets du graphe;
  tant que *il reste de sommets non marqués* **faire**
    Trouver une paire de sommets non marqués (u, v) maximisant le $gain(u, v)$;
    Marquer u et v sans les échanger;
    Mettre à jour $D(u)$ pour tous les sommets u non marqués comme si les sommets précédents avaient effectivement été échangés;
  À ce point, nous avons calculé une séquence de paires $(u_1, v_1), \ldots, (u_n, v_n)$, dont les gains sont $gain(1), \ldots, gain(n)$ où $k \leftarrow \frac{card(\mathcal{V})}{2}$ dans l'ordre où les paires ont été marquées.
  Choisir l'indice $k \in [1..n]$ tel que $gaingeneral = \sum_{i=1..k} gain(i)$ est maximal;
  Ce gain général est la réduction de coût pour l'échange de l'ensemble de paires de sommets $(u_1, v_1), \ldots, (u_k, v_k)$.
  si $gaingeneral > 0$ **alors**
    Mettre à jour $A \leftarrow A - \{u_1, \ldots, u_k\} \cup \{v_1, \ldots, v_k\}$;
    Mettre à jour $B \leftarrow B - \{v_1, \ldots, v_k\} \cup \{u_1, \ldots, u_k\}$;
    Mettre à jour $cutweight \leftarrow cutweight - gaingeneral$;
jusqu'à *gain* ≤ 0;

L'algorithme KL donne de bien meilleurs résultats que l'algorithme glouton, mais est significativement plus lent. Fiduccia et Mattheyes ont développés un algorithme [Fiduccia, 1982] améliorant KL, et réduisant sa complexité à $O(card(\mathcal{E}))$ par l'utilisation de structures de données appropriées.

7.5 Partitionnement spectral

La méthode de la bissection spectrale [Pothen, 1990, Teresco, 2005] donne en général de très bons résultats, mais est particulièrement coûteuse. Elle consiste à construire une matrice laplacienne L du graphe dans laquelle :

– chaque élément de la diagonale l_{vv} est le degré du sommet v ;
– chaque élément luv est -1 si un arc $e(u, v)$ existe, 0 sinon ;

Un vecteur propre x associé à la plus petite valeur propre de L différente de zéro est utilisé pour diviser les sommets en deux ensembles. La valeur médiane de x est

déterminée. Pour chaque x_v si x_v est moins que la valeur médiane, le sommet v est associé à l'ensemble A, sinon à l'ensemble B.

Le problème de cette méthode réside dans le calcul des vecteurs propres qui est une opération coûteuse. Cette méthode est utilisée pour les bissections, mais a aussi été développée pour quatre ou huit sections [Hendrickson, 1995].

7.6 Approches multiniveau

Les approches multiniveau sont une technique que l'on peut appliquer à plusieurs méthodes de bissection. Elles reposent sur une reformulation du graphe à plusieurs niveaux ou échelles plus ou moins «grossières» ou simplifiée (les anglo saxons utilisent le terme «coarsening»). Ainsi le graphe est d'abord reformulé sous une forme simple, c'est-à-dire ayant des sommets représentant en fait plusieurs sommets réels. Cette simplification est ensuite partitionnée avec une méthode de bissection telles que celles que nous avons déjà décrit. Enfin le graphe est rétabli dans sa forme initiale, et les partitions calculées sur sa forme simple affinées par une optimisation locale.

Une telle méthode permet de commencer le partitionnement sur une version extrêmement simplifiée du graphe, et d'utiliser des méthodes de partitionnement plus coûteuses uniquement pour affiner ce partitionnement initial. Bien entendu, une telle approche suppose un graphe statique. La figure 7.1 page suivante ([Teresco, 2005]) présente les diverses étapes de simplification du graphe, de partitionnement, et de rétablissement. En (a) les sommets du graphe original sont d'abord groupés par deux pour former une version simplifiée à un premier niveau représentée en (b). A nouveau les sommets de ce graphe sont groupés par deux en (c) pour donner le graphe simplifié au second niveau en (d). Cette version du graphe est bi-partitionnée en (e), puis rétablie au premier niveau de simplification en (f) avec un partitionnement affiné. Enfin la version initiale du graphe est rétablie en (g) et le partitionnement à nouveau affiné.

Il existe de nombreuses façons de simplifier le graphe, en utilisant des algorithmes permettant de grouper les sommets. Il faut tenir compte à cette étape des pondérations des arcs. L'ensemble de arcs incidents aux sommets groupés sont reliés au sommet qui les simplifie. Une méthode très simple et efficace combine les sommets reliés par les arcs de plus forte pondération [Karypis, 1999].

Beaucoup d'approches utilisent le partitionnement spectral sur le graphe simplifié pour la qualité des résultats qu'il produit, puis une approche dérivée du Kernighan-Lin comme Fiduccia-Mattheyes pour améliorer les divers niveaux de simplification, mais de nombreuses combinaisons et améliorations sont possibles telles que [Walshaw, 1997].

7.7 Algorithmes génétiques

Les approches basées sur des heuristiques telles que le recuit simulé ou les algorithmes génétiques ont aussi été appliquées au problème du partitionnement de graphe. Nous avons classé les algorithmes génétiques parmi les approches statiques, bien que certaines méthodes soient en mesure de gérer une faible dynamique [Maini, 1994] car celle-ci reste limitée.

Les algorithmes génétiques (AG, à ne pas confondre avec la programmation génétique, que nous verrons dans la partie III), s'inspirent de l'évolution selon Darwin afin

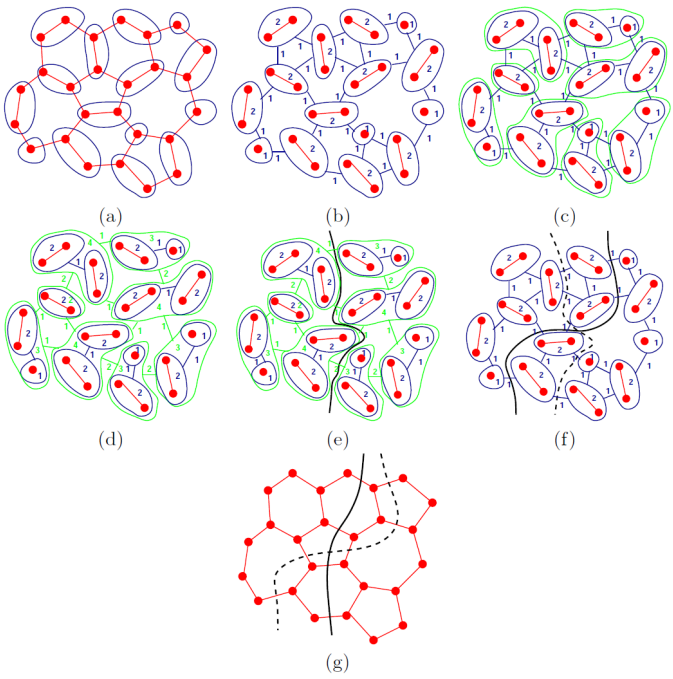


FIG. 7.1: Passage du graphe à sa simplification, partitionnement, puis rétablissement et affinage du partitionnement.

de faire évoluer une population de solutions vers un optimum. Ils ont été introduits par John Henri Holland [Holland, 1975] en 1960.

Dans un algorithme génétique, chaque individu de la population représente une position dans l'espace de solutions . Ces individus sont codés sous forme de chaînes de symboles : les *nucléotides* (les algorithmes génétiques de base utilisaient des chaînes de bits), un gène étant formé de plusieurs nucléotides. Par exemple si on cherche la valeur minimale d'une fonction $f(x)$, la chaîne de symboles sera représentée par les bits formant le paramètre x.

On génère la population initiale aléatoirement. Afin d'améliorer les solutions, on crée ensuite d'autres générations en appliquant des opérateurs génétiques sur les individus de la génération précédente, jusqu'à ce qu'une solution satisfaisante soit trouvée. Les individus sont sélectionnés pour la reproduction en fonction de leur scores.

Un algorithme génétique suit le schéma suivant :

1. codage du problème sous forme d'une chaîne de nucléotides ;
2. génération d'une population initiale d'individus ;
3. calcul du score de chaque individu ;
4. si l'objectif est atteint, sortie ;
5. sélection des reproducteurs en fonction des scores ;
6. construction des descendants par application de différents opérateurs génétiques (étape de reproduction) ;
7. remplacement de la population précédente par les descendants (ou mélange avec les descendants) ; aller à l'étape 3 ;

L'opérateur génétique principal est le *croisement*. Ce dernier génère, à partir de deux individus parents, un troisième dont une partie du génome provient du premier parent et une autre partie du second parent. L'endroit où les deux génomes parents sont coupés, le *locus*, est en général choisi aléatoirement.

L'autre opérateur génétique important est la *mutation*. Il s'agit d'introduire dans le génome d'un individu un nucléotide aléatoire. Cette étape est appliquée après le croisement. Il ne doit en général pas y avoir trop de mutation (moins de 10%) pour éviter la création d'individus à trois têtes et donc non viables. En effet le «bruit» introduit par les mutations risque de détruire des séquences génétiques intéressantes sélectionnées par l'algorithme. Cependant, l'opérateur de mutation est essentiel pour dépasser les limites imposées par le matériel génétique de départ.

Il existe bien sûr d'autres types d'opérateurs génétiques. Le croisement peut être décliné en fonction du nombre de *loci*, d'une probabilité d'échange de nucléotides, *etc*. De même, la mutation peut porter sur plusieurs gènes, *etc*

La sélection des individus pour la reproduction peut se faire de différentes façons. La méthode la plus utilisée, dite *roue de la fortune biaisée*, consiste à ce qu'un individu ait une probabilité d'être choisi directement correspondante à son score. Une autre méthode, dite par tournoi sélectionne au hasard une paire d'individus et ne garde que le meilleur. Une variante, le tournoi stochastique, tire un nombre aléatoire entre 0 et 1 et si ce chiffre est inférieur à une valeur prédéterminée (ex. 0.8), choisit le meilleur individu de la paire, sinon le moins bon.

Bien sûr un individu peut être sélectionné plusieurs fois pour la reproduction, et il est possible de jouer sur le fait que les meilleurs individus soient sélectionnés plus souvent que d'autres. La population précédente peut être complètement remplacée à chaque étape ou remplacée en partie. La partie restante peut alors être par exemple formée de ses meilleurs individus.

On peut aussi éviter la convergence vers un minimum local (une population globalement bonne ayant des caractéristiques très proches, mais ne représentant pas un optimum général au problème) par l'introduction à chaque étape de reproduction d'individus aléatoires.

Historiquement le codage binaire des solutions a souvent été préféré au codage par chaîne de symboles. Néanmoins, ce dernier est plus pratique et laisse une plus grande latitude. John Holland avait préféré le codage binaire car les chaînes générées sont souvent plus longues et fournissent donc plus de loci pour le croisement, opérateur essentiel de l'algorithme.

Dans le cas qui nous occupe, le principe repose sur un codage de la bissection sous forme de gènes représentés de manière binaire. Chaque nucléotide représente un sommet du graphe et peut donc prendre la valeur 1 ou 0. Cependant à ce stade, cette approche rend la recherche de solution particulièrement difficile, en raison du nombre énorme de permutations possibles. Il est donc nécessaire d'aider l'algorithme à mieux sélectionner les individus, et les croiser.

[Maini, 1994], pour accélérer la recherche, utilise un «masque de croisement» constitué de réels dans l'intervalle $[0..1]$ correspondant à la probabilité de choisir un parent ou un autre. Ce masque de croisement est déterminé en fonction de solutions initiale. Il utilise un graphe planaire mis en forme, une partition initiale, ainsi que les coordonnées des nœuds du graphe pour déterminer des informations de voisinage permettant la mise au point de ce masque. La méthode proposée est donc géométrique. La

mise au point du masque utilise le cut-edge d'un sommet par rapport au nombre d'arcs total pour déterminer la probabilité pour ce sommet.

7.8 Colonisation émergente

L'approche proposée par Pascale Kuntz et Dominique Snyers dans [Kuntz, 1994] est proche de la notre et utilise plusieurs concepts que nous reprenons (c.f. III page 113). Ils proposent un système constitué de plusieurs colonies d'animats autonomes colonisant un environnement constitué du graphe à partitionner. Cependant, d'une part l'approche s'applique à des graphes statiques, d'autre part son mode de fonctionnement, basé sur la démographie et la co-évolution d'espèces diffère du notre. Cependant, il est important de noter que l'approche n'utilise pas de fonction objectif explicite, et qu'elle utilise des mécanismes basés sur la stigmergie.

L'algorithme se propose de partitionner un graphe utilisé comme environnement grâce à un nombre donné d'animats initialement distribués uniformément. Il existe un nombre q donné d'espèces d'animats, et un nombre p de ressources de calcul, avec $q \geq p$. La démographie au sein d'une espèce varie, et il est possible que des espèces disparaissent. Chaque animat possède en effet un âge courant et un âge maximal. L'âge de l'animat est incrémenté à chaque itération, et pour tout déplacement d'un sommet du graphe à un autre. L'âge agit comme rétroaction négative dans l'algorithme stabilisant la démographie des espèces.

Les animats utilisent une fonction de satisfaction qui vaut 1 si l'animat est sur un sommet colonisé par son espèce, et 0 sinon. Un sommet est colonisé par une espèce lorsqu'il contient une majorité d'animat de cette espèce. Une heuristique est utilisée pour choisir quelle espèce colonise un sommet dans le cas où deux espèces sont à égalité.

Les animats se reproduisent en suivant leur fonction de satisfaction. C'est la rétroaction positive. Lorsqu'un animat est satisfait, c'est-à-dire sur un sommet colonisé, il renforce cette situation en créant un nombre donné d'animats de son espèce.

La règle de mouvement des animats dépend d'une probabilité de mouvement ainsi que de leur satisfaction :

- Si l'animat est satisfait les mouvement maintenant ce dernier dans sa colonie sont favorisés. Il y favorise alors les arcs de poids important.
- Sinon, l'animat cherche à fuir la colonie dans laquelle il se trouve en favorisant les sommets destination connectés par des arcs de poids faible.

7.9 Clustering par colonie de fourmis pour le partitionnement

Pascale Kuntz et Dominique Snyers associés à Paul Layzell ont aussi proposé dans [Kuntz, 1997] une méthode de partitionnement de graphe capable de détecter des groupes pour partitionner le graphe. Les fourmis et les sommets du graphe sont déposés dans un environnement discrétisé en deux dimensions Γ. Le principe fonctionne comme un tri effectué en parallèle par plusieurs entités ici représentées par des fourmis. On peut citer un algorithme fondé sur le même principe, mais dans le domaine de

la classification non supervisée de données [Azzag, 2003].

Dans algorithme présenté dans [Kuntz, 1997] les fourmis n'utilisent pas de phéromone. Initialement, les fourmis $x \in \mathcal{A}$ et les sommets $v \in \mathcal{V}$ sont distribués de manière aléatoire dans l'environnement Γ. L'algorithme est itératif. À chaque itération t, une fourmi est sélectionnée aléatoirement. Si un sommet se trouve sur sa position, elle peut le saisir et le déplacer. Sinon, elle se déplace aléatoirement.

Lorsqu'une fourmi rencontre ou transporte un sommet, son comportement devient probabiliste :

- plus un sommet est isolé plus il a de chance d'être transporté ;
- plus l'environnement courant contient des sommets similaires au sommet transporté, plus une fourmi à de chance de le déposer ;

Le transport peut, bien entendu, se dérouler sur plusieurs itérations. La probabilité qu'une fourmi déplace un sommet v est :

$$p_d(v) = \left(\frac{k_d}{k_d + f(v)} \right)^2 \tag{7.4}$$

De même la probabilité qu'une fourmi place un sommet est :

$$p_p(v) = \left(\frac{f(v)}{k_p + f(v)} \right)^2 \tag{7.5}$$

Avec k_d et k_p des constantes et f représentant une estimation de la densité de sommets similaires à v dans un voisinage, défini par une aire Σ de $\sigma \times \sigma$ éléments de Γ dont v est le centre :

$$f(v) = \begin{cases} \frac{1}{\sigma^2} \sum\limits_{v;\Pi_t(v) \in \Sigma} (1 - \frac{d(v,u)}{\alpha}) & \text{si } f(v) > 0 \\ 0 & \text{sinon} \end{cases} \tag{7.6}$$

Avec $\Pi_t(v)$ la position du sommet v, et d la différence (non similarité). Un sommet très différent de v sera ainsi moins désirable dans Σ qu'une position non occupé. Ainsi la constante α divise les différences de manière à ce que des sommets différents produisent une valeur de f réduite. La valeur de f n'est maximale que lorsque toutes les positions de Σ sont occupées par des sommets u_i similaires à v, c'est à dire pour lesquels $d(u_i, v) = 0$. Une aire contenant des sommets similaires et différents produira une valeur intermédiaire de f.

La mesure de similarité est avant tout basée sur la topologie du graphe, l'adjacence ainsi que l'importance des arcs, leur pondération, entre en compte.

Les fourmis possèdent une mémoire sous forme de liste tabou des M derniers sommets déplacés. Le déplacement d'un nouveau sommet v est mené à bien par une marche aléatoire biaisée dans la direction du sommet de la liste tabou ayant le plus de similarité avec v. L'algorithme 7.2 page suivante détaille ce comportement. Dans ce dernier $\delta = \delta_1 + \delta_2$ est la distance sur la grille Γ, δ_1 sa projection sur l'axe X, et δ_2 sur l'axe Y. À chaque itération t l'état d'une fourmi $s_t(x)$ est défini par sa position $\Pi(x)$ et le sommet qu'elle déplace, ou 0 si elle n'en porte pas : $s_t(x) = (\Pi_t(x), v)$ ou $s_t(x) = (\Pi_t(x), 0)$. La probabilité p_r introduit une légère perturbation afin d'éviter les blocages où deux fourmis s'empêchent mutuellement d'avancer.

7.2 : Algorithme de Kuntz Layzell et Snyers

Première phase : choix d'une fourmi

Sélectionner une fourmi x aléatoirement dans $\mathcal{A}$;

Seconde phase : déplacement des sommets

si $s_t(x) = (\Pi_t(x), v)$ **alors**
  Placer v en $\Pi_t(x)$ (i.e. $\Pi_{t+1}(v) = \Pi_t(x)$) avec une probabilité de $p_p(v)$;
  Ajouter la liste tabou $list(x)$ de x avec v;

si $s_t(x) = (\Pi_t(x), 0)$ *et* $\exists v \in \mathcal{V}; \Pi_t(v) = \Pi_t(x)$ **alors**
  Déplacer v (i.e. $s_{t+1}(x) = (\Pi_{t+1}(x), v)$) avec une probabilité de $p_d(v)$;
  Choisir $v^\star \in list(x)$ tel que
  $d\left(\Pi_t(x), \Pi_t(v^\star)\right) = \min\left\{d\left(\Pi_t(x), \Pi_t(v_i)\right); v_i \in list(x)\right\}$;
  Placer $\delta = 0$ et $\delta^\star = \delta_1^\star + \delta_2^\star$ (i.e.
  $\delta^\star = \delta_1\left(\Pi_t(x), \Pi_t(v^\star)\right) + \delta_2\left(\Pi_t(x), \Pi_t(v^\star)\right)$;

Troisième phase : déplacement des fourmis

si $s_t(x) = (\Pi_t(x), 0)$ **alors**
  Déplacer x de r éléments de grille dans une direction aléatoire;

si $s_t(x) = (\Pi_t(x), v)$ **alors**
  si $\delta \leq \delta^\star$ **alors**
    Déplacer x de r éléments de grille dans une direction aléatoire avec une probabilité p_r;
  sinon
    Déplacer x de r éléments de grille dans la direction de $v^\star$ avec une probabilité $\delta = \delta + \delta\left(\Pi_t(x), \Pi_{t+1}(x)\right)$;

Il en résulte que :
- si les sommets appartiennent au même groupe ils sont collectés au même endroit ;
- le nombre d'arcs intergroupes est minimisé ;
- des groupes distants sont localisés en des points différents de l'espace.

Cependant, certains problèmes se posent, certains partagés avec l'algorithme précédent des mêmes auteurs :
- Le nombre de groupes peut être différent du nombre de ressources de calcul ;
- La somme des tailles des groupes alloués à chaque ressource de calcul doit être similaire ;
- Leur nombre ainsi que la structure du graphe peut changer.

Pour résoudre le premier problème, les auteurs mentionnent la possibilité de jouer sur les paramètres de l'algorithme de façon à choisir le nombre de groupes à créer. Malheureusement pour une distribution dynamique, il est impossible de connaître le nombre de groupes permettant d'obtenir la meilleure répartition de charge, comme le montre la figure 7.2 page ci-contre. Cette figure montre un graphe composé de quatre groupes «naturels», cependant, les deux plus petits groupes ne devrait en former qu'un pour qu'un équilibre soit trouvé avec trois ressources de calcul.

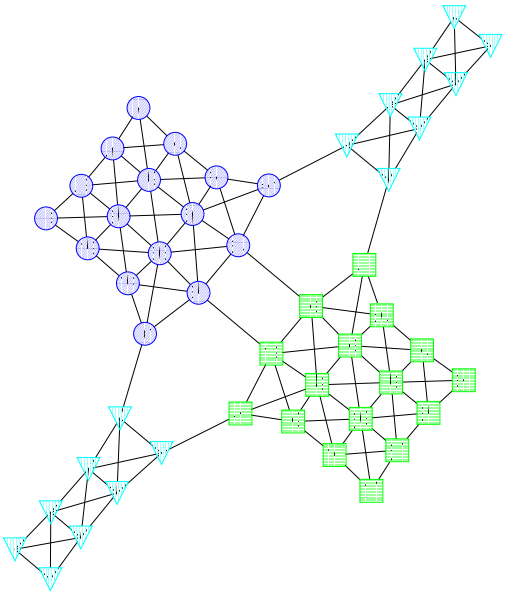


FIG. 7.2: Exemple pour lequel le meilleur nombre de groupes est différent du nombre de ressources de calcul.

Chapitre 8

Méthodes de distribution dynamiques

Sommaire

On peut parler de dynamique sur de nombreux aspects du problème :
- L'état de l'application à distribuer change constamment ;
  - des entités peuvent apparaître et disparaître continuellement ;
  - des interactions peuvent apparaître et disparaître continuellement ;
  - les valeurs de ces interactions changent en continu ;
- L'état des ressources de calcul sur lesquelles on distribue l'application change constamment ;
  - elles peuvent apparaître et disparaître ;
  - leur état peut changer.

De plus ces multiples sources qui conditionnent les sorties de notre méthode de distribution sont imprévisibles.

Les méthodes que nous avons vues, toutes dédiées aux graphes statiques, travaillent sur une instance de graphe fixée, et pour certaines, améliorent itérativement les résultats. Ainsi pour les utiliser sur un graphe dynamique, il serait nécessaire de les appeler à chaque changement dans ce dernier, ou même à chaque changement de l'environnement d'exécution. Cependant, certaines de ces techniques, celles donnant les meilleurs résultats, sont coûteuses. Les méthodes de distribution prenant en compte la dynamique pallient à ce manque, mais sont encore un domaine peu abordé dans la littérature. Elles se concentrent en majorité sur les algorithmes à base de *diffusion* (que nous verrons par la suite), ou modifient une méthode statique. Parmi ces dernières on peut citer deux approches :
- les approches qui effectivement recommencent le calcul de la distribution du début ;
- les approches qui réutilisent les résultats précédents.

Les premières apportent une grande facilité de mise en œuvre au détriment d'un

problème majeur de rapidité, et aussi de stabilité. En effet il est rarement garanti que le résultat d'une itération sera corrélé avec le résultat de l'itération suivante. Or, en ce qui concerne la distribution, il est évident qu'il est important de migrer le moins d'entités possibles.

Les approches réutilisant les résultats précédents sont souvent basées sur une méthode qui déplace les frontières des sous-domaines. Les désavantages majeurs sont que la qualité peut rapidement se dégrader et qu'il faut une dynamique faible. L'algorithme ne s'adaptera pas facilement aux reconfigurations massives.

Les méthodes dynamiques sont développées dès le départ pour prendre en compte la dynamique. Cependant, toutes ne gèrent pas toutes les formes de dynamique. Certaines savent prendre en compte l'arrivée de nouvelles entités ou tâches pour les ordonnancer sur un ensemble de processeurs, mais ne gèrent par les interactions, la charge réseau. D'autres ont un nombre de ressources de calcul fixé, *etc*.

Nous passerons en revue deux algorithmes de distribution dynamique pour lesquels il est possible d'ajouter ou de retirer des entités à tout moment. Seul le second est en mesure de prendre en comptes les interactions entre les entités.

8.1 Algorithmes de diffusion

Le principe des algorithmes de diffusion consiste à faire communiquer les ressources de calcul uniquement avec ses voisins, et d'échanger deux-à-deux des entités. Si nous avons pris le parti que le réseau d'interconnexion physique des ressources de calcul était abstrait et que toute machine peut communiquer avec toute autre, il en va différemment ici : le réseau de machines est connu.

Les modèles de diffusion imitent la façon dont la chaleur se diffuse dans un compartiment. La distribution initiale non homogène de la température produit un déplacement de cette dernière amenant une distribution régulière, uniforme, de la température. Dans le cadre de la distribution dynamique, la charge machine remplace la température. Chaque ressource de calcul mesure sa charge et échange des tâches avec ses voisines afin de l'égaliser. Après plusieurs itérations, la charge générale s'égalise.

Certaines méthodes de diffusion sont *globales* [Lin, 1992], dans le sens où le calcul des déplacement d'entités de ressource de calcul à ressource de calcul est défini en fonction d'informations collectées sur toutes ces dernières. Ces informations sont ensuite traitées par une unique ressource de calcul dédiée. Bien que ces méthodes produisent des résultats de grande qualité, les phases de communication de toutes les ressources de calcul vers une seule pour la collecte des informations de charge agit comme un véritable goulot d'étranglement, monopolisant le réseau d'interconnexion.

L'autre approche consiste à distribuer l'algorithme de diffusion lui-même. Chaque ressource de calcul exécute alors une partie de l'algorithme de diffusion en déterminant les entités à échanger avec ses voisins sur la base d'une estimation locale. En ce sens, l'algorithme de diffusion est facilement distribuable et malgré des décisions de migration uniquement locales, l'algorithme peut aboutir à une répartition uniforme de la charge. Cependant, des problèmes apparaissent, liés en majorité au fait que les processeurs ne sont pas liés entre eux par un réseau formant un graphe complet. De plus la charge, bien que souvent indiquée par des nombres réels, n'est pas découpable régulièrement. En effet, même si toutes les entités représentent la même charge machine, il

n'est pas possible de diviser cette charge unitaire.

Ainsi, dans [Luque, 1995], les auteurs notent que pour certains réseaux de ressources de calcul, il est possible que la distribution locale soit correcte, mais que la distribution globale soit mauvaise comme l'exemple de la figure 8.1. On tombe dans de telles configurations lorsque le réseau de ressources de calcul est peu ou très peu connecté :

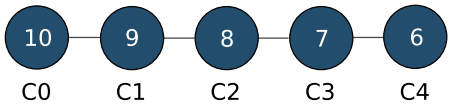


FIG. 8.1: Une situation ou les ressources ont des charges balancées localement, mais pas globalement.

La solution apportée est alors de laisser les domaines des processeurs se recouvrir, ceci afin de permettre une meilleure diffusion.

Qu'elle soit globale ou locale, la diffusion est l'une des technique dynamique les plus employée et a donné lieu à de nombreuses modifications basées sur le même principe. La méthode est itérative. Chaque ressource de calcul essaye de diffuser ses excès de charge vers ses voisins. Bien entendu, une situation d'équilibre global ne peut être atteinte dès le premier échange, ce qui justifie le mode itératif de la méthode. L'algorithme de diffusion de base [Boillat, 1990] est le suivant. À chaque itération, t de l'algorithme, le processeur i envoie à ses voisins une charge égale à la différence entre sa charge et la charge de ses voisins. Ainsi la charge envoyée est :

$$l_i^{(t+1)} = l_i^{(t)} - \sum_{i \leftrightarrow j} \delta_{ij} \tag{8.1}$$

Avec $i \leftrightarrow j$ dénotant l'existence d'un lien entre les processeurs i et j, et :

$$\delta_{ij} = c_{ij} \cdot \left(l_i^{(t)} - l_j^{(t)} \right) \tag{8.2}$$

la différence de charge (on a donc bien $\delta_{ij} = |\delta_{ij}|$) et l_i la charge de la ressource de calcul i. Le coefficient c proposé par Boillat est :

$$c_{ij} = \frac{1}{\max(degree(i), degree(j)) + 1} \tag{8.3}$$

Avec $degree(i)$ le nombre de ressources de calcul voisines de i. Ce coefficient permet d'égaliser la charge distribuée (ou récupérée) sur les processeurs voisins. Cependant, il faut bien noter qu'ici la charge est un nombre réel ne reflétant pas forcément la charge d'une entité.

L'algorithme de diffusion converge vers une distribution uniforme si :

- $c_{ij} \geq 0$ pour $i \leftrightarrow j$;
- $\sum_{j:i \leftrightarrow j} c_{ij} < 1$ pour $d \in \mathcal{C}_i$ avec $\mathcal{C}_i$ l'ensemble des processeurs connectés à i ;
- $c_{ij} = c_{ji}$;
- bien entendu le graphe des ressources de calcul est connexe.

La critique majeure que l'on peut adresser aux algorithmes de diffusion réside dans le fait qu'ils distribuent des entités sans relations entre elles (bien que des améliorations de ceux-ci se proposent de le faire). Ainsi, bien que deux entités communiquent beaucoup, l'algorithme pourra les placer sur des ressources distantes, en ne prenant pas en considération la charge réseau. L'autre inconvénient, en particulier pour une méthode avec décision locale, est que la vitesse de convergence est faible, dans le pire des cas, le nombre d'itération nécessaires est de $O(p^2)$, cette dernière étant corrélée avec la connectivité du graphe de ressources de calcul.

Parmi les variantes possibles[1], on peut citer :

- Des améliorations de la convergence de l'algorithme utilisant le polynôme de Chebyshev dans [Diekmann, 1997, Hu, 1999, Ghosh, 1998] ;
- Avec des algorithmes de pavement, ou d'équilibrage d'arbre [Cougny, 1994, Flaherty, 1997, Flaherty, 1998] ;
- Dans [Horton, 1993], l'auteur propose un algorithme de diffusion multiniveau. On opère une bissection du graphe de ressources de calcul, puis on balance les deux sous graphes. Ce processus est répété récursivement jusqu'à ce qu'il ne soit plus possible de faire de bissections (convergence en $\log(p)$).
- L'algorithme «dimension exchange» [Cybenko, 1989] ;
- On peut aussi citer [Heirich, 1995, Ghosh, 1994] ;

La méthode présentée dans la section suivante s'inspire de la diffusion telle que nous venons de la décrire en prenant en compte les interactions des entités, c'est à dire en ne considérant plus les entités comme de simples tâches sans corrélation.

8.2 Approche particulaire

L'approche par particules [Heiss, 1995] est inspirées des algorithmes de diffusion. Dans cet article, les auteurs donnent la description «intuitive» suivante de l'algorithme : Dans un bassin dont le fond est plat, on verse différents fluides non solubles et de viscosité différente. On observe alors que les fluides les plus fins s'étalent au fond, tandis que les plus visceux restent compacts à un endroit. De plus, après avoir versé chaque fluide, une phase de stabilisation se produit qui même à un équilibre figure 8.2.

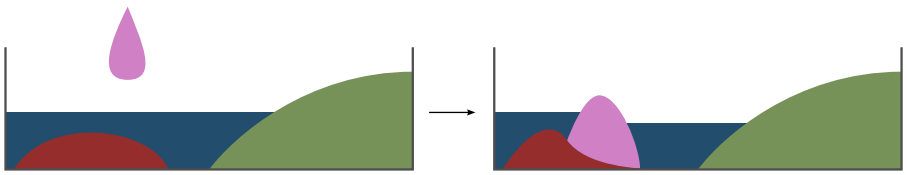


FIG. 8.2: L'ajout d'un fluide (à gauche), puis sa stabilisation (à droite).

Ici la diffusion se fait suivant plusieurs forces :

- la force d'équilibrage de charge, distribue les entités pour égaliser la charge ;
- la force de communication (cohésion), garde les entités très communicantes ensembles ;

[1] En particulier Y. Hu en décrit un grand nombre sur `http://www.dl.ac.uk/TCSC/Staff/Hu_Y_F/PROJECT/pdcp_siam.html`.

– la force de migration (friction), empêche une entité de migrer si cela coûte trop cher.

L'algorithme 8.1 utilise ensuite une combinaison linéaire de ces trois forces pour déterminer les entités devant migrer.

8.1 : Approche particulaire

répéter
  Attendre un changement de charge;
  si *la charge augmente* **alors**
    $M \leftarrow \{v \in \mathcal{V} | x_i = 1\}$;
    $loadbalanced \leftarrow false$;
    tant que $M \neq \emptyset$ *et* $loadbalanced = false$ **faire**
      Obtenir les informations sur les voisins;
      Pour toutes les entités, obtenir les forces dans toutes les directions.
      pour tous les $v \in M$ **faire**
        $\mathcal{V}_v$ ensemble des voisins du sommet v.
        pour tous les $u \in \mathcal{V}_v$ **faire**
          Calculer les forces $f(v, u)$;
      On sélectionne le candidat ayant la plus large force.
      $cand \leftarrow \max_{u \in \mathcal{V}_v} \{f(v, u)\}$;
      On sélectionne la direction de plus large force.
      $direct \leftarrow \max_{u \in \mathcal{V}_v} \{f(cand, u)\}$;
      si $f(cand, direct) > 0$ **alors**
        On fait migrer le candidat dans la direction de plus large force.
        Migrer$(cand, direct)$;
        Envoyer l'information de nouvelle charge aux voisins;
      sinon
        Il n'y a plus de tâches migrables.
        $loadbalanced \leftarrow true$;
  sinon
    La charge diminue.
    Envoyer l'information de charge aux voisins;
jusqu'à *Indéfiniment*;

Voici une description des trois forces à l'œuvre :

Équilibrage Plusieurs mesures de charges sont données par les auteurs. Nous retiendrons celle-ci :

$$V_{load}(c) = card(\mathcal{V}_c) \tag{8.4}$$

avec c le processeur considéré. Les auteurs définissent ensuite les différences de charge de manière absolue, la force d'équilibrage f_{lb} entre deux processeurs i et

j appliquée au sommet v :

$$f_{lb}^{i \leftrightarrow j}(v) = c_{lb} \left(\frac{V_{load}(i) + 1}{V_{load}(j) + 1} \right) \tag{8.5}$$

Les incréments de 1 permettent d'éviter les charge nulles.

Cohésion Les auteurs définissent aussi explicitement la méthode de mesure des communications entre deux entités :

$$V_{com}(c) = \sum_{j \in \mathcal{V}_\rangle} a(i,j)c(i,j) \tag{8.6}$$

Avec $a(i,j)$ le temps nécessaire pour transporter une information élémentaire de la ressource de calcul i à j, et $c(i,j)$ l'intensité de communication entre ces deux ressources de calcul. La force de minimisation des communications est donc le gain en coût de communication si on déplaçait le sommet v du processeur i au processeur j :

$$f_{com}^{i \leftrightarrow j}(v) = c_{com} \left(V_{com}(i) - V_{com}(j) \right) \tag{8.7}$$

Friction La migration, opération qui peut être coûteuse, est considérée comme une force de friction. Plus le déplacement d'une entité est lourde, plus la friction sera élevée. Si $d(v)$ est la somme de données à envoyer pour faire migrer l'entité v la friction est :

$$f_{frict}^{i \leftrightarrow j}(v) = c_{frict} d(v) a(i,j) \tag{8.8}$$

Additionnellement, les entités possèdent un compteur de migrations qui ne doit pas dépasser un seuil constant fixé max_{mig}. Ainsi, la migration du sommet v est autorisée si $x(v) = 1$, avec :

$$x(v) = \begin{cases} 1, & \text{si } r \geq \frac{z(v)}{max_{migs}} \\ 0, & \text{sinon} \end{cases} \tag{8.9}$$

et $z(v)$ le compteur de migrations de l'entité v. Le paramètre r est une valeur aléatoire.

Dans ces forces les constantes c_{lb}, c_{com} et c_{frict} servent à pondérer l'effet combiné des différentes forces. La force générale est donc la combinaison linéaire des trois forces décrites :

$$f^{i \leftrightarrow j}(v) = f_{lb}^{i \leftrightarrow j}(v) + f_{com}^{i \leftrightarrow j}(v) + f_{frict}^{i \leftrightarrow j}(v) \tag{8.10}$$

Troisième partie

Modèle

Cette partie traite essentiellement du modèle que nous proposons, que nous avons nommé *AntCO*^{2}. Cependant, ce modèle s'insère principalement dans le domaine des «Approches Collectives». Ainsi, afin de mieux cerner le modèle que nous allons présenter, et en référence aux concepts présentés dans la partie I page 19 et aux techniques présentées dans la partie II page 87, il est important de replacer notre approche dans son domaine ainsi que par rapport à ses bases inspiratrices. Nous placerons ici l'accent sur les différences significatives entre la méthode que nous proposons et les méthodes qui s'en rapprochent par une caractéristique ou une autre. Ceci nous permettra de lever les ambiguïtés liées aux différents vocabulaires qui naissent forcément avec des approches au confluent de plusieurs domaines (Biologie, Parallélisme, Approches Collectives, *etc*.).

Ces briques de bases étant posées, nous décrirons ensuite en détail l'approche que nous avons suivie, *AntCO*^{2}. Nous présenterons d'abord l'algorithme en lui-même, son fonctionnement général, puis divers cas particuliers auxquels nous avons été confrontés, et enfin l'explication et le détail des divers mécanismes qui ont été utilisés dans l'algorithme pour les traiter. Nous étudierons en particulier l'influence des divers paramètres à notre algorithme.

De nombreuses possibilités d'évolution de la méthode aussi bien dans ses buts que dans son action viennent s'ajouter à cette discussion. N'étant pas directement au cœur du sujet de cette thèse, ils sont reportés dans la partie IV page 143, cependant, une technique de recherche des meilleurs paramètres à notre algorithme s'inspirant de PSO^2 est présentée ici.

Nous proposons sur la figure 8.3 page suivante une petite taxinomie des méthodes ayant trait aux approches collectives. Il s'agit bien sûr d'un classement parmi d'autres. L'intention ici est de montrer les différences entre le modèle que nous proposons, et les approches qui l'ont inspiré.

Nous différencions les approches collectives dédiées à l'optimisation combinatoire ayant une *fonction objectif* explicite ou même implicite globale de celles qui n'en ont pas. Cette fonction est identifiée par une mesure réalisée de manière globale sur la ou les solutions apportées par la méthode (les anglo-saxons utilisent le terme *fitness*), qui rétroagit sur la méthode. Cette rétroaction modifie les itérations futures de la méthode, et permet l'optimisation en vue d'un résultat escompté. Notre méthode n'utilise pas de mesure ou de fitness implicite ou explicite globale ayant un effet rétroactif sur son comportement.

Un tel choix serait possible, mais rendrait la distribution du modèle délicate en introduisant une ou des variables centralisées. La communication de ces informations à tous les éléments du modèle serait un goulot d'étranglement pour une application. De plus le graphe que nous utilisons est dynamique, et il n'est pas possible de conserver des données sur celui-ci très longtemps, la dynamique les rendant caduques.

Nous introduisons aussi, comme effet de l'absence de fonction objectif, une différence au niveau du mode d'exécution, en particulier sur la fin attendue de l'algorithme. Notre approche n'est pas définie pour avoir une fin particulière aboutissant à un résultat optimisé tout au long de son exécution en raison de la dynamique des entrées. Nous avons qualifié cette autre approche *temps réel étendu*.

Ainsi nous séparons dans notre classement les approches basées sur l'évaluation

[2] Section 4.5.2 page 71.

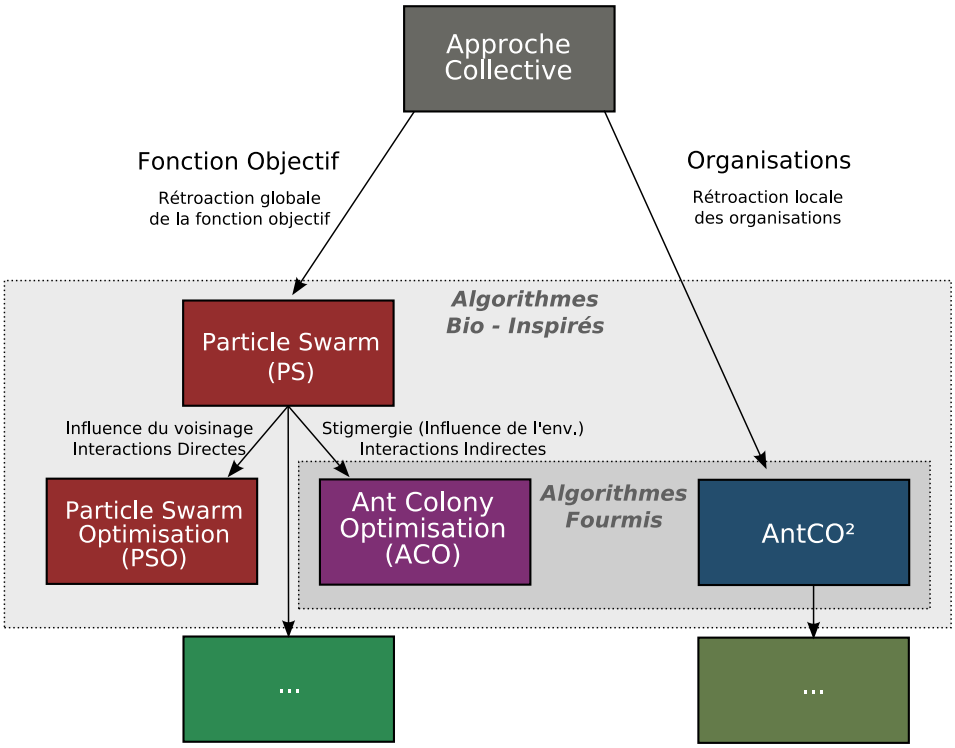


FIG. 8.3: Positionnement de notre approche.

d'une fitness globale, des approches comme la notre qui n'utilisent que des choix locaux. Si sur le diagramme nous avons nommé «essaim de particules» les approches utilisant une fonction objectif, c'est que les très nombreuses méthodes collectives qui en découlent sont quasiment toutes dans le domaine de l'optimisation.

Chapitre 9

AntCO²

Sommaire

Ce chapitre présente d'abord informellement *AntCO²* et ses spécificités, avant de décrire en détail son modèle. Plusieurs mécanismes entrent en jeu dans le fonctionnement de ce dernier que nous décrirons ensuite, pour finir en proposant une architecture dans laquelle *AntCO²* pourrait s'insérer, ainsi qu'une étude des paramètres à l'algorithme, leur relations et leurs effets.

9.1 Présentation et Spécificités

AntCO² est une approche dans laquelle une population d'entités informatiques simples et réactives (par opposition à des agents proactifs) imitant une partie du com-

portement de fourmis naturelles, coopèrent et entrent en compétition pour la colonisation de zone de forte interaction d'un graphe dynamique.

Nous allons informellement faire le tour des spécificités de l'algorithme $AntCO^2$ (pour finir par remettre en cause son statut d'«algorithme») afin de le différencier de plusieurs autres approches basées sur les fourmis qui pourraient sembler similaires.

9.1.1 Fourmis et phéromones colorées : collaboration et compétition

$AntCO^2$ est un modèle dans lequel non seulement des fourmis numériques collaborent pour effectuer une tâche comme pour de nombreuses autres approches fourmi, mais aussi dans lequel ces même individus entrent en compétition. La majorité des algorithmes basés sur les fourmis ne reprennent qu'un de ces aspects : la collaboration.

Dans $AntCO^2$, il existe plusieurs colonies de fourmis, chacune d'une couleur unique, représentant une ressource de calcul donnée, qui se «confrontent» afin de coloniser des zones d'un environnement représentant l'application à distribuer. Ainsi *il y a collaboration au sein d'une colonie, mais compétition entre les colonies*. C'est la collaboration qui va permettre la détection d'organisations au sein des entités de l'application, et donc des groupes à distribuer, et c'est la compétition qui va permettre une distribution de la charge de ces groupes sur différentes ressources de calcul.

Les fourmis colorées d'$AntCO^2$ déposent des phéromones elles-aussi colorées. Ainsi chaque fourmi laisse dans son environnement non seulement une quantité donnée de phéromone, mais aussi un message qualitatif indiquant sa colonie.

9.1.2 Le graphe en tant qu'environnement et solution

Comme pour beaucoup d'algorithmes à base de fourmis, ces dernières se déplacent dans un environnement représenté par un graphe. Ce dernier est lui-même une représentation d'une application à distribuer. Cependant la signification de ce graphe est inconnue aux fourmis, elles ne sont sensibles qu'à la présence ou non d'arcs entre les nœuds, à l'importance de ces arcs par rapport aux autres et aux informations que leur congénères ont déposé dans ce graphe. Comme nous l'avons déjà vu, l'importance d'un arc est définie par l'application elle-même en fonction de ses besoins.

En effet, les fourmis en parcourant le graphe le modifient en déposant des informations sous la forme de phéromones. Ces phéromones sont stockées sur les arcs traversés. C'est le message qualitatif et quantitatif qui permet aux autres fourmis de savoir combien d'autres fourmis de leur colonies sont passées par là, et combien de fourmis d'autres colonies on visité cet endroit. La figure 9.1 page suivante donne un exemple de toutes les informations présentes dans le graphe. Elle montre les proportions de phéromones colorées sur les arcs, indiquant la qualité des phéromones (leur couleur), mais aussi leur quantité en fonction de la taille du diagramme circulaire. Les sommets sont eux-aussi colorés, en fonction des phéromones dominantes sur les arcs adjacents. Chaque sommet peut contenir plusieurs fourmis de couleurs différentes.

Le graphe est une structure de données en mutation constante. Ainsi un arc possède une durée de vie, et à chaque itération de l'algorithme il conserve les messages que les fourmis l'ayant emprunté y ont déposé. Dès sa disparition, ces messages sont perdus, et si un arc réapparaît entre les mêmes nœuds de nouveaux messages doivent continuellement être déposés.

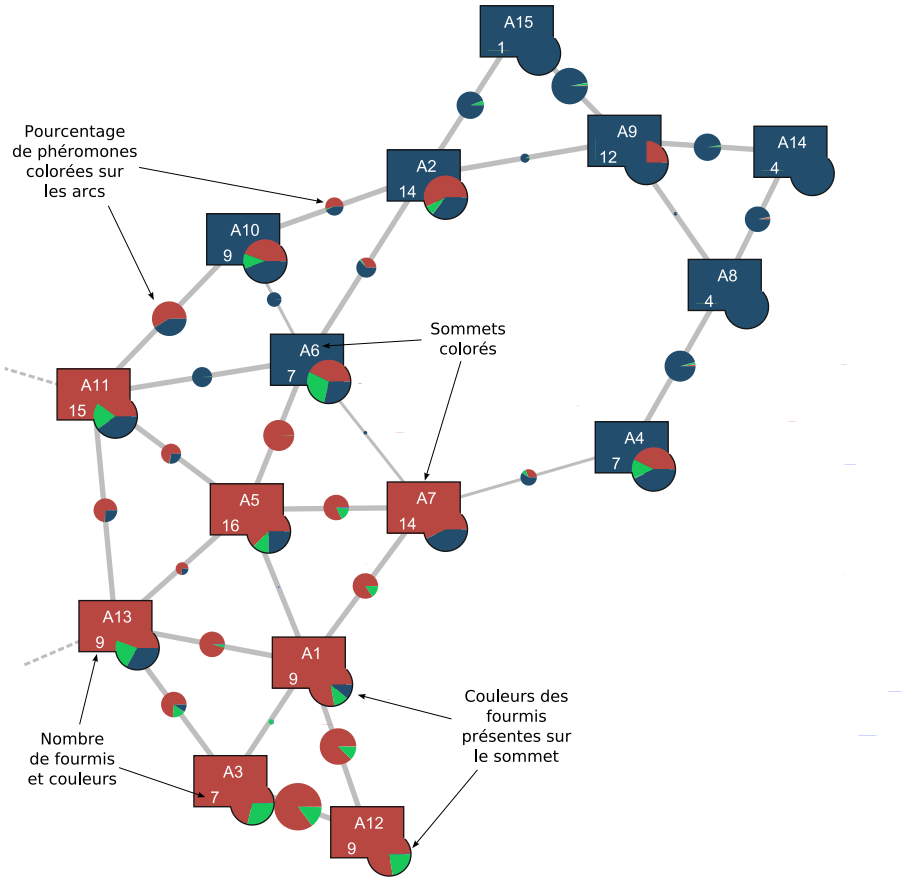


FIG. 9.1: Le graphe dynamique utilisé en tant qu'environnement pour les fourmis, et les diverses informations qu'il contient.

Les fourmis se déplacent dans cet environnement en perpétuelle mutation. L'algorithme complet d'*AntCO*^{2} n'est pas relancé à chaque changement dans le graphe. Il tourne au contraire en continu sur un graphe changeant. À aucun moment les fourmis ne stockent de chemin «solution» qu'elles évaluent par la suite, ni même une autre forme de mémoire «solution». En fait le graphe «environnement» au temps t *est* la solution de placement au temps t par coloration.

9.1.3 Distribution par détection d'organisations

Les fourmis colonisent les nœuds du graphe en les colorant et délimitent ainsi des ensembles de sommets adjacents de même couleur. Ceci forme des sous-graphes connexe qui possèdent une frontière.

Ces derniers ne sont pas délimités aléatoirement. Les fourmis sont attirées par les aires de communication importante. Elles se regroupent sur les grappes d'entités en forte interaction qui forment des organisations au sein du graphe. Chaque couleur correspondant à une ressource de calcul, les sous-graphes délimités indiquent ainsi un placement. On favorise une distribution des organisations constituées de zones très communicantes sur une ressource de calcul qui leur est propre afin de limiter la com-

munication entre machines.

Ainsi, nous définirons une organisation par :

> **Organisation (*AntCO^2*)** :
> Sous-graphe connexe constitué de sommets de même couleur.

Le nombre de fourmis dans une colonie, leur démographie, ainsi que la multiplicité des colonies en compétition permet une répartition de ces zones sur toutes les ressources de calcul. Des forces de répulsion inter-colonies permettent l'égalisation des zones colonisées afin de répartir correctement la charge.

9.1.4 Modèle d'exécution

En fait, nous ne devrions peut-être pas parler d'algorithme en ce qui concerne *AntCO^2* puisque la définition très précise d'algorithme induit une fin attendue, avec un résultat attendu, or *AntCO^2* ne réunit aucune des ces caractéristiques. En effet il ne se termine qu'avec la fin de la simulation qu'il aide à distribuer, et ne produit pas un résultat particulier en fin d'exécution, mais bien des «résultats» à tout moment. Ceci puisque le graphe coloré *est* la solution, et que ce dernier évolue avec l'application. Cependant, par abus de langage, nous parlerons d'algorithme en général pour définir les multiples algorithmes au sein du modèle *AntCO^2*.

On peut voir *AntCO^2* comme un service, ou un filtre. Un service est une entité qui peut être interrogée et qui fournit en retour un travail donné. C'est le cas d'*AntCO^2* à qui on peut demander à tout moment le placement d'un groupe d'entités. De même un filtre est traversé par un flux, ici d'informations, qui est transformé. Dans le cas d'*AntCO^2*, en entrée du filtre on trouve des événements définissant l'état courant des entités de l'application à distribuer et en sortie ces mêmes entités accompagnées d'une information de placement.

Le point commun de ces deux exemples, filtre et service, réside dans le fait qu'ils fonctionnent en continu apportant des réponses lorsqu'on le leur demande, dynamiquement. Ils ne s'exécutent par linéairement d'un début à une fin pour aboutir à un résultat. Ce qui motive ce mode de fonctionnement est bien sûr la manière dont on s'en sert. Les deux reçoivent des entrées non pas uniquement au début de leur fonctionnement pour produire un résultat à la fin, mais bien durant toute leur période de fonctionnement. Ils doivent être en mesure de produire des solutions à chaque invocation, voire dans le cas du filtre, en continu.

9.2 Modèle

Nous allons aborder le modèle d'*AntCO^2* suivant deux axes : la modélisation du graphe environnement et son évolution, puis la modélisation des colonies de fourmis. Nous considérerons ensuite la façon dont la démographie au sein des colonies est gérée, ainsi que divers mécanismes mis en œuvre pour répondre à différentes situations.

9.2.1 Graphe et Environnement

L'algorithme utilise un graphe dynamique pondéré coloré tel que décrit au chapitre 6.2 page 88. Reprenons brièvement cette notation. Le graphe est noté $G(t) = (\mathcal{V}(t), \mathcal{E}(t), \mathcal{C}(t))$ au temps t pour indiquer sa dynamique et celle des ensembles de sommets $\mathcal{V}(t)$, d'arcs $\mathcal{E}(t)$ et de couleurs $\mathcal{C}(t)$. Les arcs sont pondérés par un poids noté $w^{(t)}$ indiquant leur importance au niveau de l'application.

Nous ajoutons à cette description une information sur la phéromone déposée par les fourmis. Ainsi, les arcs contiennent un ensemble de messages représentant les phéromones des fourmis les ayant traversés. Tout comme les fourmis, les phéromones sont colorées. Chaque arc contient donc $card(\mathcal{C}(t))$ messages de phéromones au temps t, ces messages représentant tous la quantité de phéromone de leur couleur présente 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 un arc e entre les itérations t et $t+1$ dépose une quantité donnée de phéromone de couleur c notée :

$$\Delta_k^{(t)}(e, c) \tag{9.1}$$

On note la quantité totale de phéromone de couleur c déposée sur l'arc e par toutes les fourmis de couleur c ayant traversé cet arc entre t et $t+1$:

$$\Delta^{(t)}(e, c) = \sum_{k \in \mathcal{F}_c(t)} \Delta_k^{(t)}(e, c) \tag{9.2}$$

De même, la quantité totale de phéromones toutes couleurs confondues, déposée sur l'arc e au temps t est :

$$\Delta^{(t)}(e) = \sum_{c \in \mathcal{C}(t)} \Delta^{(t)}(e, c) \tag{9.3}$$

Lorsque $\Delta^{(t)}(e) \neq 0$, la proportion de phéromones de couleur c sur e par rapport aux autres couleurs entre t et $t+1$ est :

$$K_c^{(t)}(e) = \frac{\Delta^{(t)}(e, c)}{\Delta^{(t)}(e)} \tag{9.4}$$

avec $K_c^{(t)}(e) \in [0, 1]$.

La quantité totale de phéromone de couleur c présente sur l'arc e entre t et $t+1$ est notée :

$$\tau^{(t)}(e, c) \tag{9.5}$$

et la quantité totale toutes couleurs confondues :

$$\tau^{(t)}(e) \tag{9.6}$$

Au départ, cette valeur est fixée à :

$$\tau^{(0)}(e) = \epsilon \tag{9.7}$$

nous verrons par la suite que cette quantité ne peut être à zéro. Ensuite cette phéromone s'évapore à chaque itération. Ainsi la quantité totale de phéromone de couleur c sur l'arc e au temps t est définie par la récurrence :

$$\tau^{(t)}(e, c) = \rho \cdot \tau^{(t-1)}(e, c) + \Delta^{(t-1)}(e, c) \tag{9.8}$$

Où $\rho \in]0, 1]$ est un facteur de persistance des phéromones, c'est-à-dire la quantité de phéromones restant après évaporation. En d'autres termes l'évaporation est $1 - \rho$.

La couleur $\xi^{(t)}(u)$ d'un sommet u du graphe est ensuite définie par la couleur principale de tous ses arcs adjacents, ainsi :

$$\xi^{(t)}(u) = \arg\max_{c \in \mathcal{C}(t)} \sum_{e \in \mathcal{E}_u(t)} \tau^{(t)}(e, c) \tag{9.9}$$

avec $\mathcal{E}_u(t)$ l'ensemble des arcs adjacents au sommet u au temps t.

L'algorithme 9.1 reprend, de manière très simple, le fonctionnement de l'environnement.

9.1 : Fonctionnement de l'environnement dans *AntCO²*.

répéter
- Récupérer les informations de l'environnement d'exécution (ressources de calcul détruites ou crées) et mettre à jour le graphe;
- Récupérer les informations sur l'application (entités et communications) et mettre à jour le graphe;
- **pour** *toute fourmi* x **faire**
  - c.f. algorithme 9.2;
- **pour** *tout arc* e **faire**
  - Appliquer le facteur d'évaporation ρ sur la phéromone présente en utilisant l'équation 9.8 page précédente;
- **pour** *tout sommet* v **faire**
  - Définir la couleur du sommet en fonction de l'équation 9.9;
- Communiquer ces informations si nécessaire;

jusqu'à *indéfiniment*;

9.2.2 Fourmis et Comportement

Les fourmis utilisent le graphe $G(t)$ comme environnement et se déplacent de nœud en nœud. Ainsi au temps t chaque fourmi est associée à un nœud particulier, et un nœud peut contenir plusieurs fourmis. *AntCO²* est une méthode itérative. À chaque itération, les fourmis traversent un arc.

La phéromone perçue par les fourmis est fonction de $\tau^{(t)}(e, c)$, mais modulée par la présence d'autres phéromones sur l'arc. C'est cette partie qui permet le comportement de répulsion face aux phéromones de colonies «adverses». Les fourmis d'une couleur c sont bien attirées par les phéromones de couleur c, et plus la quantité de cette phéromone est élevée plus elles sont attirées, mais en outre cette attraction est modulée par le fait qu'il peut y avoir bien plus ou bien moins de phéromones d'autres couleurs. Dans le premier cas on minimise l'attraction dans le second cas non. On utilise le facteur $K_c^{(t)}(e)$ pour définir la phéromone perçue sur un arc e par une fourmi de couleur c :

$$\Omega^{(t)}(e, c) = K_c^{(t)}(e)\tau^{(t)}(e, c) \tag{9.10}$$

Ainsi à chaque itération, une fourmi k de couleur c placée sur un nœud u choisit d'emprunter un arc plutôt qu'un autre en fonction des phéromones — de sa couleur et

des autres couleurs — présentes sur ces arcs. On note $p^{(t)}(e,c)$ la probabilité pour que cette fourmi traverse l'arc $e=(u,v)$ entre t et $t+1$:

$$\begin{cases} p^{(t)}(e,c) &= \dfrac{w^{(0)}(e)}{\sum\limits_{e_i \in \mathcal{E}_u(t)} w^{(0)}(e_i)} & \text{si } t=0 \\ \\ p^{(t)}(e,c) &= \dfrac{\left(\Omega^{(t)}(e,c)\right)^{\alpha} \cdot \left(w^{(t)}(e)\right)^{\beta}}{\sum\limits_{e_i \in \mathcal{E}_u(t)} \left(\Omega^{(t)}(e_i,c)\right)^{\alpha} \cdot \left(w^{(t)}(e_i)\right)^{\beta}} & \text{si } t \neq 0 \end{cases} \tag{9.11}$$

où les paramètres α et β (tous deux > 0) tout comme dans l'algorithme original de Marco Dorigo (section 5.2.2 page 81, voir l'équation 5.6) permettent de modifier l'importance relative des phéromones et des pondérations d'arcs, et $\mathcal{E}_u(t)$ est l'ensemble des sommets adjacents à u.

Afin d'éviter des mouvements oscillatoires, dans lesquels une fourmi passe d'un sommet u à un sommet v puis à nouveau au sommet u en boucle, on introduit dans chaque fourmi une mémoire des sommets visités. Cette technique est très similaire à une liste tabou. On introduit donc dans l'équation 9.11 un coefficient $\eta \in]0,1]$ dont le but est d'empêcher les fourmis de faire demi tour sans pour autant les bloquer s'il n'y a pas d'autre chemin. Ainsi, chaque fourmi peut se rappeler les M derniers arcs traversés, stockés dans une liste $\mathcal{W}_k$ avec $card(\mathcal{W}_k) \leq M$, M étant un seuil constant fixé par avance. La valeur de η pour une fourmi k sur le sommet u, considérant l'arc $e=(u,v)$ est :

$$\eta_k(v) = \begin{cases} 1 & \text{si } v \notin \mathcal{W}_k \\ \eta & \text{si } v \in \mathcal{W}_k \end{cases} \tag{9.12}$$

Pour une fourmi k de couleur c sur le sommet u, la probabilité de traverser l'arc $e=(u,v)$ entre t et $t+1$ est :

$$p^{(t)}(e,c) = \frac{\left(\Omega^{(t)}(e,c)\right)^{\alpha} \cdot \left(w^{(t)}(e)\right)^{\beta} \cdot \eta_k(u)}{\sum\limits_{e_i \in \mathcal{E}_u(t)} \left(\Omega^{(t)}(e_i,c)\right)^{\alpha} \cdot \left(w^{(t)}(e_i)\right)^{\beta} \cdot \eta_k(v_i)} \tag{9.13}$$

L'algorithme 9.2 page suivante reprend le fonctionnement des fourmis.

9.2 : Comportement d'une fourmi de couleur c dans *AntCO²*.

répéter
  $\tau_c \leftarrow 0$;
  $\tau \leftarrow 0$;
  pour *chaque arc* e *adjacent au sommet courant* v **faire**
    Calculer la probabilité $p(e, c)$ de choisir cet arc en fonction de la couleur c de la fourmi (éq. 9.15 page 125);
    $\tau_c \leftarrow \tau_c + \tau_c(e)$;
    $\tau \leftarrow \tau + \tau(e)$;
  si $\frac{\tau_c}{\tau} < \phi$ **alors**
    Mécanisme de répulsion.
    Fuir;
  sinon
    Déplacement normal.
    Sélectionner le prochain arc à traverser e_{next} parmi tous les arcs connectés au sommet v en fonction de sa probabilité;
    Déposer sur e_{next} une quantité $\Delta(e, c)$;
    Se déplacer sur le sommet u connecté à v par e_{next};
jusqu'à *indéfiniment*;

9.2.3 Gestion de Population et démographie

Dans un tel algorithme, il faut un nombre critique de fourmis pour maintenir les solutions, les organisations, malgré les reconfigurations du graphe. Cependant il faut veiller à ne pas surcharger le graphe en fourmis afin de ne pas trop influencer les résultats, d'abord la solution elle-même, mais aussi le coût d'exécution d'*AntCO²*.

Cependant, le nombre de sommets dans le graphe ainsi que le nombre de couleurs, c'est-à-dire de ressources de calcul sont des variables. La population de fourmi ne peut donc pas être constante, et une politique de gestion démographique doit être mise en place.

Nous verrons par la suite d'autres mécanismes tels que la «pression démographique» qui nécessitent une gestion de la population, mais le mécanisme majeur est celui de la répartition de charge. C'est le nombre de fourmis au sein d'une colonie donnée qui détermine sa capacité à coloniser de nouvelles organisations au sein du graphe. Si le nombre de fourmi est trop limité, elles ne pourront déposer suffisamment de phéromones sur une partie donnée du graphe pour la colorer, et une autre colonie se l'appropriera. Ainsi il est aussi nécessaire de faire varier la population de fourmis d'une couleur donnée en fonction de la puissance d'une ressource de calcul.

Nous utilisons une politique de population *proportionnelle* à la taille du graphe. Par taille du graphe nous entendons son nombre de sommets, mais ce paramètre peut varier (on peut penser au nombre d'arcs, ou au diamètre par exemple). À chaque sommet apparaissant, on ajoute un nombre δ donné de fourmis. De la même manière, à chaque sommet disparaissant, on retire ce même nombre de fourmis. Ces nombres sont bien entendu modulés pour que l'on ajoute et retire un nombre de fourmis comparable dans

chaque colonie.

Il pourrait sembler idéal de créer $\delta = card(\mathcal{C}(t))$ fourmis par nœud (créer un nombre de fourmis égal au nombre de couleurs par nœud), mais le nombre de colonies varie et rend cette politique parfois inefficace. Cependant il faut garder à l'esprit que la taille du graphe est à considérer *en même temps* que le nombre de ressources de calcul à disposition. En fonction de la charge induite par les entités de l'application il serait probablement inutile d'utiliser six ressources de calcul pour un graphe de seulement une centaine de sommets.

Si l'on sait que le nombre de ressources de calcul restera fixe, une politique donnant de bons résultats est effectivement d'allouer $\delta = card(\mathcal{C}(t))$ fourmis par nœud apparaissant (et identiquement de détruire $\delta = card(\mathcal{C}(t))$ fourmis par nœud disparaissant). Si l'on sait au contraire que le nombre de sommets du graphe ne variera presque pas, mais que de nouvelles ressources de calcul peuvent devenir disponible à tout moment, il peut être préférable de fixer δ et de créer moins de fourmis par nœud apparaissant qu'il y a de couleurs. On essaye alors de répartir les fourmis de manière approximativement égale.

Si une telle politique est préférée, c'est qu'elle est *locale*. En effet on cherche, pour des raison de facilité de distribution et d'encombrement réseau minimal, à éviter tout mécanisme de communication global nécessitant un gestionnaire centralisé. Il serait possible de mettre en œuvre des mécanismes plus précis de gestion de la population avec de tels gestionnaires, mais au prix d'une perte de facilité de distribution.

9.2.4 Conditions initiales

Il est possible de faire démarrer *AntCO^2* avec un graphe déjà connu représentant une application, par exemple pour faire une expérimentation, ou parce qu'*AntCO^2* démarre alors que l'application est déjà en train de s'exécuter. Mais il est plus probable que l'application et *AntCO^2* vont démarrer de concert. *AntCO^2* sera alors utilisé pour non seulement distribuer l'application, mais aussi la déployer.

Dans les deux cas, la question de l'initialisation d'*AntCO^2* se pose. Comment va-t'on placer les fourmis sur les nouveaux nœuds ? Au départ les arcs n'auront qu'une quantité minime et quasiment égale de phéromones de toutes couleurs[1], mais le nombre de fourmis d'une couleur donnée sur une zone peut très vite amener à la colonisation d'une partie plutôt que d'une autre. Il est impossible cependant de choisir un «meilleur» emplacement pour les fourmis (puisque ce sont elles qui détectent les organisations), et on a donc choisi de les répartir le plus uniformément possible.

Avec une telle répartition, pour un graphe déjà créé dès le départ, on a une valeur de r_2 qui est quasi optimale, et une valeur de r_1 généralement très mauvaise.

9.2.5 Ajouts de couleurs

Lorsqu'une ressource de calcul apparaît il faut ajouter une nouvelle colonie d'une autre couleur. Puisque nous avons choisi une politique de population proportionnelle à la taille du graphe, on peut procéder de deux façons :

[1] Il est en effet préférable d'éviter de placer exactement la même valeur initiale de phéromone sur tout le graphe pour éviter dans le premières étapes que trop de fourmis suivent le même comportement, bien que celui-ci est aussi probabiliste.

– ajouter un nombre de fourmis proportionnelle à la taille du graphe, c'est-à-dire d'augmenter δ (par exemple en fixant $\delta = card\mathcal{C}(t)$).
– garder δ constant et répartir une proportion de fourmis déjà existantes dans la nouvelle couleur.

Ceci étant à décider en fonction du mode d'évolution du graphe et des ressources de calcul.

9.2.6 Gestion de conditions particulières et Mécanismes Supplémentaires

L'algorithme de base tel qu'il a été décrit jusqu'à présent, rencontre plusieurs types de situations que l'on peut toutes ramener au fait que le graphe entier doit être parcouru à tout moment pour qu'une coloration correcte soit maintenue. On peut isoler trois situations importantes :

Les encerclements sont des zones où les fourmis d'une couleur sont «capturées» par les fourmis d'une autre couleur. La zone est supérieure à la taille de la mémoire des fourmis et le mécanisme de répulsion les empêche d'aller explorer des nœuds autres que ceux de la partie encerclée.

Les embouteillages sont des zones dans lesquelles beaucoup trop de fourmis se pressent. Les rétroaction positives (de la phéromone attire la fourmi qui en dépose plus) ne cessent d'amplifier l'attirance pour les fourmis qui recrutent de plus en plus de la population d'une colonie à ce point.

Les déserts à l'inverse des embouteillages sont des zones que les fourmis ne visitent pas ou plus. La phéromone dans ces aires est devenue voisine de zéro par rétroaction négative de l'évaporation et les chances pour que des fourmis d'une couleur ou d'une autre les choisissent, bien que non nulles, deviennent trop faibles pour entamer un processus de colonisation.

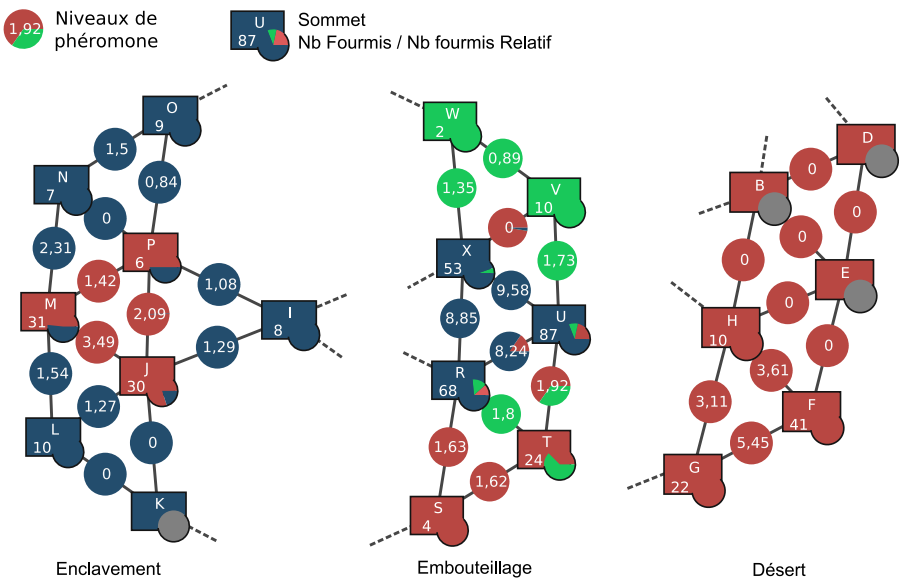


FIG. 9.2: Problèmes auxquels est confronté l'algorithme de base.

La figure 9.2 page précédente donne un exemple de chacune de ces difficultés. Deux mécanismes locaux sont à l'œuvre dans *AntCO*^{2} pour les contrecarrer :

– la pression démographique ;
– les mécanismes de fuite.

Pression démographique

La pression démographique s'inspire directement des phénomène d'embouteillages que l'on a décrit à la section 5.1.2 page 75 sur le pont à double voies de tailles identiques. Lorsque les fourmis constatent que l'un des chemins, bien que plus attirant par sa phéromone, est bouché par un nombre trop important d'individus, elles sélectionnent l'autre chemin. En d'autres termes, le nombre d'individu sur un chemin peut jouer un rôle répulsif même si chemin est très chargé en phéromones.

Ce mécanisme est quasi-directement implanté dans le comportement des fourmis, de la même manière que η, par un facteur de répulsion minimisant l'importance d'un arc. On note $\gamma(u)$ ce facteur qui est défini en fonction du nombre de fourmis sur le sommet u, par rapport à un seuil N^* fixé :

$$\gamma(u) = \begin{cases} 1 & \text{si } N(u) \leq N^* \\ \gamma \in]0,1] & \text{sinon} \end{cases} \tag{9.14}$$

Ainsi l'équation 9.13 est modifiée ainsi :

$$p^{(t)}(e,c) = \frac{\left(\Omega^{(t)}(e,c)\right)^{\alpha} \cdot \left(w^{(t)}(e)\right)^{\beta} \cdot \eta_k(u) \cdot \gamma(u)}{\sum\limits_{e_i \in \mathcal{E}_u(t)} \left(\Omega^{(t)}(e_i,c)\right)^{\alpha} \cdot \left(w^{(t)}(e_i)\right)^{\beta} \cdot \eta_k(v_i) \cdot \gamma(v_i)} \tag{9.15}$$

Le mécanisme de la pression démographique aide les colonies de fourmis à mieux se disperser sans créer des zones de surpopulation et résout ainsi les situations d'embouteillage, et en partie les encerclements.

Mécanismes de fuite

Les mécanismes de fuite permettent aux fourmis de détecter les environnements particulièrement hostiles et de s'en échapper plus rapidement que ne leur permettrait leur comportement de base. La détection d'environnement «hostile» est effectuée en comparant la proportion des valeurs de phéromones de sa propre couleur par rapport aux valeurs de toutes les couleurs sur l'ensemble des arcs adjacents au sommet sur lequel elle se trouve. Si cette proportion est inférieure à un seuil $\phi \in]0,1[$, la fourmi choisit la fuite. Ainsi la fourmi de couleur c sur le sommet u fuit si :

$$\frac{\sum\limits_{e \in \mathcal{E}_u(t)} \tau^{(t)}(e,c)}{\sum\limits_{c_i \in \mathcal{C}(t)} \left(\sum\limits_{e \in \mathcal{E}_u(t)} \tau^{(t)}(e,c_i) \right)} \leq \phi \tag{9.16}$$

Il existe plusieurs méthodes pour implanter la fuite. La plus simple consiste à faire «sauter» la fourmi à un endroit aléatoire sur le graphe. On peut comparer cette

technique à une mort suivie d'une éclosion (ce qui préserve donc une population constante). Cette méthode possède le désavantage de n'être pas locale, il faut pouvoir considérer tout le graphe. Dans la suite nous décrivons une architecture possible pour l'implantation d'*AntCO*^{2} dans laquelle le graphe est distribué. Dans ce cas, la fourmi peut sauter aléatoirement sur la zone distribuée du graphe uniquement, ce qui n'implique aucune communication distante.

Une autre technique consiste à laisser la fourmi parcourir aléatoirement (tout en conservant le mécanisme de mémoire) n nœuds en une seule itération. Cette technique est purement locale, et permet à la fourmi de s'échapper très rapidement d'une zone encerclée.

L'avantage direct de la fuite, quelle que soit son implantation, est de permettre de sortir de mauvaises solutions. Elle permet :

- de recoloniser des zones désertes ;
- de casser les encerclements ;
- et de bouger rapidement des fourmis qui perturberait une zone en colonisation par une autre couleur.

En effet, l'un des autres avantages, mais présent uniquement dans la première implantation du mécanisme (par «saut» ou par «mort / éclosion»), est qu'il permet de sortir des composantes du graphe non-connexe. En effet, la dynamique des entrées ne nous permet pas d'assurer une connexité au graphe, et le comportement de base des fourmis ne leur permet pas de s'échapper s'il n'y a pas d'arcs. Ainsi, il est possible que des fourmis de deux couleurs soient emprisonnées en nombre quasi égal sur une petite composante, empêchant sa coloration uniforme en faisant osciller les couleurs sur ses nœuds. Or le mécanisme de fuite par saut permet à l'une des colonies de migrer d'une telle zone.

Autres mécanismes

Les phéromones jouent un rôle prépondérant dans le fonctionnement de l'algorithme. Cependant, il est impossible de borner exactement la quantité de phéromone qui peut être présente sur un arc (bien qu'au niveau des fourmis, on observe uniquement les proportions de phéromones). Outre les phéromones, la population de fourmis à un instant donné à un endroit particulier joue aussi un rôle important. Ceci nous a mené à tester la remise à zéro des phéromones sur tous les arcs du graphe afin d'observer la capacité des fourmis à retrouver une solution, en partant de la simple répartition démographique.

On pourrait nommer ce mécanisme «colonisation démographique». Bien que la phéromone soit ramenée à un seuil identique sur tous les arcs, la coloration du graphe est influencée par la répartition démographique. Le nombre de fourmis présentes sur les sommets influence donc encore leur comportement de manière indirecte.

L'algorithme est très réactif, et dans le cas de graphes statiques, comme dans le cas du pont à double voies de tailles différentes, il est possible qu'une mauvaise solution soit d'abord choisie aléatoirement, puis par rétroaction positive malheureusement renforcée. Lorsque le graphe est dynamique, les modifications sont généralement suffisantes pour permettre une variation significative des solutions. Dans le cas statique, il est possible que ces mauvaises solutions perdurent. La remise à zéro des phéromones permet alors de briser ce problème.

Ces cas sont en fait rares, mais on a aussi constaté un «lissage» des solutions grâce à cette technique. En effet lorsque la solution sur un graphe statique approche de la meilleure solution, la remise à zéro permet d'effacer les légers défauts liés aux trois conditions défavorables décrites plus haut qui n'auraient pas été corrigés par les mécanismes de fuite ou de pression démographique. Certaines des expérimentations sur les graphes statiques qui sont données dans la partie IV page 143 utilisent donc cette remise à zéro.

Ce mécanisme est cependant délicat car il est global. Nous l'avons cependant utilisé sans lien avec une quelconque fitness, c'est-à-dire qu'il a été déclenché sur le graphe à des intervalles réguliers. Il serait donc possible de l'utiliser de manière distribuée.

9.3 Architecture

Nous avons décrit plus haut *AntCO*^{2} comme un filtre ou un service. C'est en effet ainsi qu'il s'insèrerait dans une application. Dans un tel cadre, *AntCO*^{2} est à la fois client et partie d'un *middleware* chargé des divers échanges entre les composants de l'application :

- *AntCO*^{2} est un service car le middleware est utilisé pour faire communiquer *AntCO*^{2} et les composants de l'application à distribuer ;
- mais c'est aussi un client du middleware, car ce dernier fait le lien entre *AntCO*^{2} et l'environnement global d'exécution.

Il peut en particulier fournir des d'évènements en relations avec l'apparition et la disparition des ressources de calcul ainsi que leur charge. La figure 9.3 montre comment *AntCO*^{2} s'insère dans l'architecture d'une application distribuée.

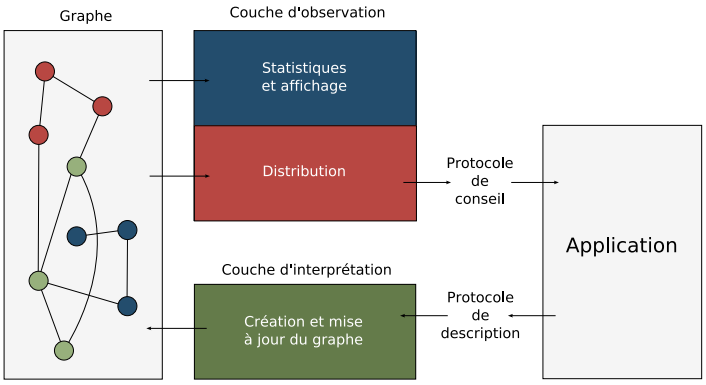


FIG. 9.3: Échanges de données entre l'application et *AntCO*^{2}.

En tant que service, *AntCO*^{2} fournit des suggestions de placement pour les entités composant l'application. On utilise le terme «suggestion» car l'application n'est à aucun moment obligée de suivre ces indications pour diverses raisons. Par exemple, l'entité peut ne pas être en mesure de migrer pour une période donnée par ce qu'elle est liée à une ressource particulière non migrable qu'elle utilise pour un temps.

9.3.1 Modèle en Couches

AntCO² tel que présenté est un algorithme *réactif*. Les fourmis réagissent sans planification, ceci étant très largement du à la nature locale des choix qu'elles effectuent. Bien que des organisations stables apparaissent malgré les reconfigurations constantes de leur constituants de base, leurs frontières oscillent parfois. Si un tel comportement était conservé, et si l'application respectait instantanément ces suggestions, certaines entités ne cesserait de migrer d'une ressource de calcul à une autre.

AntCO² est prévu pour s'intégrer dans une architecture dans laquelle une couche supérieure viendrait s'intercaler entre elle et l'application. Cette couche aurait pour but de «lisser» les résultats retournés par *AntCO²* afin de limiter les oscillations. On pourrait ajouter des techniques très simples, telles que n'accepter un changement de couleur qu'après une période donnée de stabilité. Cette période serait calculée en fonction de la dynamique de l'application. Une telle technique permettrait déjà l'élimination des oscillations.

De plus cette couche serait utilisée pour prendre en compte des heuristiques spécifiques à l'application à distribuer. Il serait ainsi possible de prendre en compte les entités non migrables, parce que liées à une ressource particulière se trouvant sur une machine donnée. Cette couche spécifique est montrée sur la figure 9.3 page précédente.

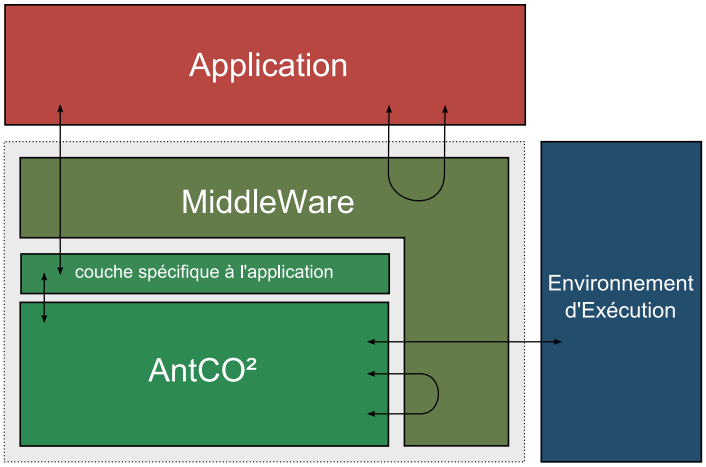


FIG. 9.4: L'architecture dans laquelle s'insère *AntCO²*.

9.3.2 Comment Distribuer *AntCO²* Lui-même

L'un des intérêts majeurs d'*AntCO²* est que le modèle n'utilise que des informations *locales*. Cette propriété le rend facilement distribuable, et bien que sur la figure 9.4 *AntCO²* soit présenté comme une entité unique, il est possible que le service soit physiquement présent sur chaque ressource de calcul mise en œuvre pour exécuter une application distribuée tierce.

Ainsi il est possible :

– qu'*AntCO²* fonctionne sur une ressource de calcul dédiée, à part ;
– qu'*AntCO²* fonctionne sur des ressources de calcul dédiées ;

– qu'$AntCO^2$ fonctionne sur toutes ou partie des ressources de calcul utilisées par l'application à distribuer.

Les figures 9.5, 9.6 et 9.7 page suivante reprennent ces modes de fonctionnement. Nous privilégions la dernière possibilité, car elle implique beaucoup moins de communications réseau entre $AntCO^2$ et l'application à distribuer, cependant elle à un impact sur la charge machine qu'il est nécessaire de considérer.

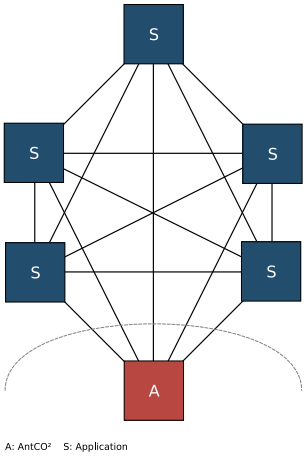


FIG. 9.5: $AntCO^2$ à part sur une ressource de calcul dédiée.

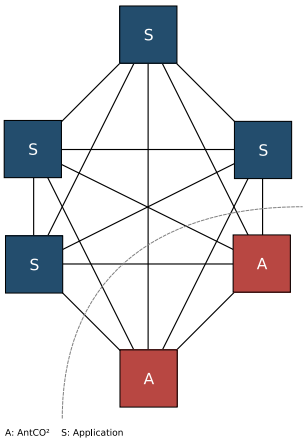


FIG. 9.6: $AntCO^2$ à part sur plusieurs ressources de calcul dédiées.

Cependant, être en mesure de distribuer $AntCO^2$ lui-même soulève plusieurs problèmes : comment va-t-on distribuer le graphe de l'application ? Comment va-t-on gérer le passage d'informations entre les différentes instances d'$AntCO^2$? Comment va-t-on distribuer l'algorithme de distribution lui-même ?

En ce qui concerne les deux dernières questions, les réponses sont triviales. Pour la première, le passage d'informations, on sait que seules les informations locales sont utilisées. Il n'y a donc que des migrations de fourmis entre chaque instance d'$AntCO^2$.

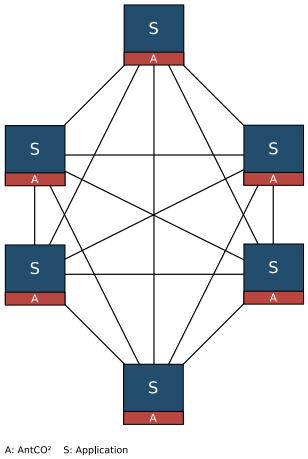


FIG. 9.7: *AntCO²* sur les mêmes ressources de calcul que l'application à distribuer.

Pour la seconde question, la qualité de la distribution d'*AntCO²* dépendra de la qualité de la distribution déterminée par *AntCO²*. En effet, l'une des méthode les plus simple pour distribuer l'algorithme est d'associer l'instance d'*AntCO²* avec les entités qu'elle gère. Dès qu'une entité migre d'une ressource de calcul A sur une ressource de calcul B, sa gestion est confiée à l'instance d'*AntCO²* qui s'exécute sur B.

Ceci nous donne d'ailleurs un indice sur une réponse possible à la première question : comment distribuer le graphe. En effet si chaque instance d'*AntCO²* gère uniquement les entités qui s'exécutent physiquement sur la ressource de calcul où elle se trouve, le graphe est automatiquement découpé et partagé. Il est cependant probablement nécessaire à chaque instance d'*AntCO²* de connaître plus que la partie du graphe qu'elle gère. En effet elle doit savoir, pour chaque nœud combien d'arcs permettent de passer vers d'autre nœuds. Cette information est passée par l'application elle-même. La figure 9.8 représente un tel découpage du graphe.

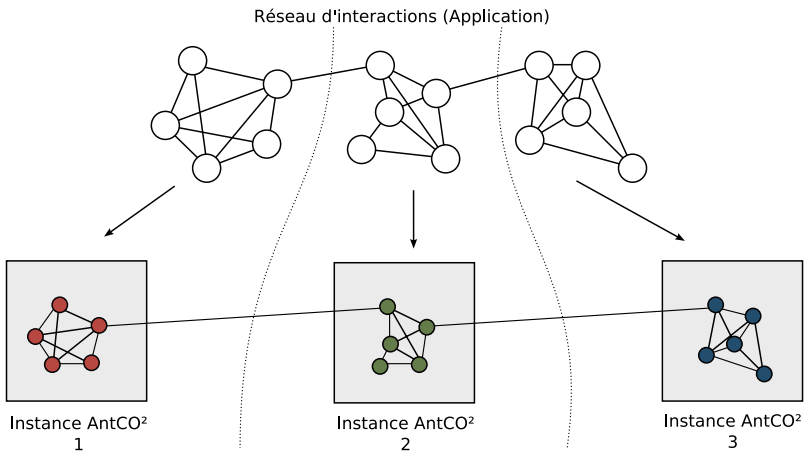


FIG. 9.8: Découpage du graphe sur les différentes instances d'*AntCO²*.

Une telle méthode est fonctionnelle, mais pas forcément totalement efficace. En effet pour les sommets à la frontière des zones gérées par l'instance courante d'*AntCO*^{2}, il y a possiblement échange d'informations avec les autres instances d'*AntCO*^{2}. On peut noter que cette frontière devrait correspondre rapidement à la frontière des organisations détectées, qui sont déjà découpées pour éviter le plus de communications. Cependant, il serait intéressant d'éviter encore ces communications. Nous proposons une méthode de gestion du découpage du graphe par recouvrement et découverte. Le principe reposerait sur des sommets «fantômes» représentant des entités gérées par d'autres instances d'*AntCO*^{2} afin que les diverses parties de graphes se recoupent en se chevauchant. Cette information serait elle aussi tenue à jour, mais par «découverte». Ce serait les fourmis qui en migrant apporterait en même temps l'information des sommets d'où elle proviennent qui permettrait la mise à jour de ces recouvrements. La figure 9.9 présente ce mode de fonctionnement.

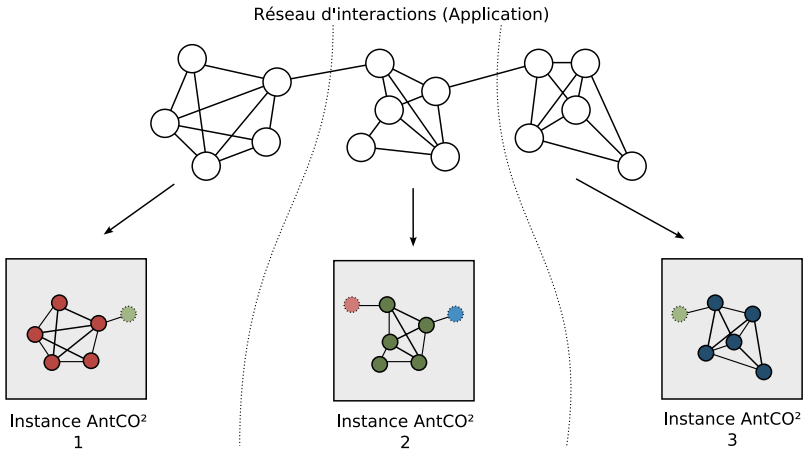


FIG. 9.9: Découpage et recouvrement du graphe sur les différentes instances d'*AntCO*^{2}.

Toutes les informations gérées par *AntCO*^{2} ne sont pas complètement locales. Certaines, comme l'ajout ou le retrait de ressources de calcul, donc de colonies, sont globales. Cependant de telles informations sont gérées par le middleware, et le coût d'un tel «broadcast» n'est pas très important. Il suffit ensuite d'utiliser la politique choisie de gestion de la démographie pour créer ou retirer les fourmis de la couleur concernée.

Pour conserver un comportement local lorsqu'une colonie apparaît, il est possible de ne faire naître les fourmis correspondantes que sur la ressource de calcul associée, et laisser ces fourmis coloniser peu à peu les ressources voisines. Ce fonctionnement serait comparable à celui de découverte de recouvrement de graphes décrit précédément : dès qu'une fourmi d'une couleur inconnue apparaît sur une ressource, sa colonie est «découverte» et enregistrée. Cependant ce fonctionnement ne gère pas le retrait de colonies, mais il est toujours possible de faire passer des messages de proche en proche entre instances d'*AntCO*^{2} par diffusion.

9.3.3 Synchronisme et Asynchronisme

Deux problèmes de synchronismes se posent :

– entre l'application à distribuer et $AntCO^2$;
– entre les différentes instances d'$AntCO^2$ lorsqu'elle est elle-même distribuée.

En ce qui concerne le premier point, la première question que l'on peut se poser est : «à quelle vitesse va $AntCO^2$ par rapport à l'application qu'elle distribue ?», en d'autres termes quelle gestion du temps utiliser ? Il semble que ce problème de vitesse respective soit mesurable en particulier en fonction des itérations (pour les applications fonctionnant ainsi) mais surtout aussi vis-à-vis des flots d'événements.

En effet la rapidité avec laquelle arrivent les informations sur l'application permet de quantifier «la dynamique» de cette dernière. Plus l'application change rapidement, plus $AntCO^2$, qui est une méthode basée sur le renforcement de zones par dépôts de phéromones, aura du mal à fixer ces zones, tout au moins au niveau de la phéromone. Cependant il y a plusieurs facteurs à considérer :

– le flot d'événements en provenance de l'application même rapide peut avoir un débit faible.
– les fourmis n'utilisent pas que la phéromone, la densité de population en un point elle aussi influe, et c'est un mécanisme très rapide.

Si le débit est faible, les changements au sein des organisations ne seront pas importants, et surtout, la durée de vie de ces organisations permettra une colonisation et un maintien facile. Si les organisations ont une très faible durée de vie, $AntCO^2$ aura plus de problèmes pour les détecter. La densité de population aide cependant à contrecarrer cette situation. Lorsque les fourmis d'une colonie sont massées en un point, elles ont une tendance à repousser les autres colonies et à rester globalement dans cette zone. Si l'application est extrêmement dynamique, cette inertie de population permet de conserver une coloration convenable. Ainsi, même pour des applications très dynamiques, dans lesquelles des entités apparaissent et disparaissent constamment, l'algorithme permet de conserver une distribution correcte grâce aux forces de répulsion entre colonies et aux mécanismes de pression démographique. Dans un tel environnement les entités à très faible durée de vie vont très vite être placées correctement dès le début grâce à la densité de population.

L'autre forme de synchronisme est liée aux diverses instances d'$AntCO^2$. Faut-il toutes les synchroniser ? Cette question n'a pas encore été complètement investie. Synchroniser $AntCO^2$ reviendrait à se limiter à la ressource de calcul la plus lente. Les points en défaveur de l'asynchronisme sont :

– quelque soit les machines utilisées, la plus lente règle la vitesse ;
– il faut une horloge globale ;
– l'envoi de messages de synchronisation est coûteux.

L'un des points que l'on peut avancer en faveur d'une désynchronisation serait que chaque ressource de calcul sur laquelle s'exécute une instance d'$AntCO^2$ possède une charge proportionnelle à ce qu'elle peut gérer. Dans ce cas la marge de temps machine disponible pour les instances d'$AntCO^2$ est approximativement la même sur toutes les machines. Cependant, une telle assertion ne peut jamais être réellement vérifiée, et il peut de tout façon se trouver des situations de charge très faible où la plus rapide des ressources de calcul possède plus de temps machine que la plus lente.

9.4 Paramètres

Voici une vue générale des principaux paramètres de l'algorithme :

α	Paramètre affectant l'importance des phéromones dans le comportement des fourmis.
β	Paramètre affectant l'importance des poids sur les arcs du graphe dans le comportement des fourmis.
ρ	Paramètre de «conservation» des phéromones ($1 - \rho =$ évaporation).
N^*	Seuil de pression démographique.
M	Taille de la mémoire des fourmis, en nombre de nœuds visités.
φ	Seuil de «fuite».
$n\mathcal{F}$	Nombre de fourmis à créer par sommet.

Le dernier paramètre, est fonction de la politique démographique choisie. L'un des objectif serait de pouvoir trouver un jeu de paramètres donnant de bons résultats dans quasiment tous les cas. Des mécanismes ajoutant de la robustesse à *AntCO^2* tels que la pression démographique et la «fuite» des fourmis permettent ce genre d'égalisation. Malheureusement en général les paramètres dépendent des propriétés des graphes auxquels on applique *AntCO^2*.

Le paramètre de mémoire, M, comme nous le verrons par la suite dans la partie consacrée aux expérimentations, est quasiment toujours correct avec une valeur de 3. Il permet d'éviter les aller-retours sur les cycles en triangles, ce qui dans la majeure partie des cas est suffisant. L'un des rare cas où il est préférable de le changer, apparaît avec les graphes ayant globalement une forme d'arbre. Dans ce cas, il est préférable d'agrandir la mémoire afin de laisser les fourmis visiter les branches en entier.

Le paramètre ρ à durant nos expérimentations sur des graphes de test quasiment toujours donné de bons résultats à 0.8. Mais ce paramètre dépend de la dynamique du graphe. Plus le graphe est dynamique au sens des organisations, plus il faut baisser ρ afin de permettre un renouvellement plus rapide des organisations détectées.

Le paramètre φ doit être en général être inférieur à $\frac{1}{card(\mathcal{C}(t))}$ pour donner de bons résultats. Il peut être réglé quasi-automatiquement à cette valeur.

Le paramètre N^* doit être supérieur au nombre moyen d'individus par sommet, $\frac{card(\mathcal{F}(t))}{card(\mathcal{V}(t))} + 2$ donne de bons résultats. Plus ce paramètre est proche de $\frac{card(\mathcal{F}(t))}{card(\mathcal{V}(t))}$, plus les populations s'étalent.

Les paramètres principaux sont α et β qui sont tous les deux liés, sont plus difficiles à déterminer. Ils proviennent directement du modèle de Jean-Louis Deneubourg. Un réglage de paramètres donnant de bons résultats est $\alpha = 1$ et $\beta = 3$.

Le nombre de fourmis est aussi un paramètre délicat. Il faut s'assurer qu'il y a un nombre de fourmis suffisant pour que les organisations détectées soient entretenues malgré leur dynamique. La borne basse est aux alentours de $card(\mathcal{V}(t))$, cependant on considère que $card(\mathcal{V}(t)) \cdot card(\mathcal{C}(t))$ est meilleur, afin d'éviter les zones de désert.

Si l'on compare le nombre de fourmis utilisé à celui des algorithmes «traditionnels», on pourra constater que le nombre de fourmis est beaucoup plus grand. Généralement, ceci est du au fait qu'à tout instant, le graphe dans sa totalité doit être colorié par les fourmis.

Chapitre 10

Améliorations de la méthode

Sommaire

Le modèle proposé par *AntCO*^{2} à l'inconvénient, comme beaucoup d'heuristiques de ce type, de nécessiter le réglage de plusieurs paramètres. Ce réglage est délicat. Même s'il est possible de trouver un ensemble de paramètres qui donnent des résultats satisfaisants avec quasiment tous les réseaux d'interactions testés, et c'est le rôle des divers mécanismes d'autogestion (population, colonies, saut...) ajoutés, ces paramètres ne permettent pas une convergence aussi rapide que souhaitée des solutions.

Régler à la main les paramètres pour un type d'application particulier devient très rapidement une tâche laborieuse : certains paramètres l'influencent et il devient vite difficile d'en changer plus d'un à la fois. De plus les tests doivent être menés sur une série suffisamment large d'exécutions afin d'en tirer une signification éventuelle.

Ces problèmes suggèrent une approche automatique permettant de déterminer, pour une application ou classe d'application donnée, les meilleurs paramètres. Lorsque l'on cherche à optimiser plusieurs paramètres, un grand nombre de metaheurisitques se présentent à nous. Une approche particulièrement séduisante aurait peut-être été d'utiliser un système de fourmis ! Mais elle ne s'adapte pas directement à notre problème. L'approche qui nous a semblé la plus immédiate a été l'approche basée sur les essaims de particules, PSO.

Tout d'abord, une critique générale sur le but consistant à déterminer les paramètres d'*AntCO*^{2} automatiquement par une quelconque méthode. À quoi ressemble l'espace des solutions ? Où sont les bonnes valeurs ? Il est possible que les bons paramètres sur un graphe donné soient complètement différents des bons paramètres pour un autre graphe. On cherche à trouver des paramètres qui fonctionnent pour tous les graphes (même si on en a déjà une approche, il serait intéressant de pouvoir les obtenir

à nouveau par un moyen automatisé d'optimisation multicritère comme les algorithmes génétiques) et peut-être des paramètres qui fonctionnent «mieux» pour des catégories de graphes par exemple les graphes statiques (évolution dans le temps), ou encore les grilles (topologie).

Cherchons nous une valeur précise des paramètres, telle que : «avec α à 1, β à 3 et ρ à 0.7 on obtient de bons résultats», ou plutôt un rapport entre les paramètres ? Ce rapport existe-t-il ? Ne peut on plutôt dire : «avec α deux fois moins grand que β et une évaporation proportionnelle au nombre de fourmis, on obtient de bons résultats» ?

10.1 Réglage automatique de paramètres avec PSO

De nombreuses metaheuristiques seraient utilisables pour effectuer le réglage des paramètres. On peut citer en particulier les *essaims de particules* (PSO, Particle Swarm Optimization), qui sont dans leur inspiration très proches des *Boids* développés par C. Reynolds [Reynolds, 1987] (voir les sections 12.3 page 166). Ici cependant, le but n'est pas de simplement reproduire le comportement de troupeaux d'animaux, de bancs de poissons ou de vols d'oiseaux, mais bien d'optimiser une fonction, les Boids n'étant qu'une inspiration.

10.1.1 Principe

PSO est comparable à l'approche par algorithme génétique par le fait qu'elle utilise une population d'individus représentant chacun une solution dans un hyperespace de solutions. Cependant, contrairement aux AG elle ne définit pas de croisement ou de mutation, et les mêmes individus sont créés et utilisés pendant toute la durée de la recherche.

Si on la compare aux algorithmes génétiques, l'optimisation par essaim de particule utilise une approche *épigénétique*, plutôt qu'une approche *phylogénétique*[1]. En un mot on s'attache ici au développement du comportement de l'individu (codé très frustrement par une position de l'espace et un vecteur vitesse), plutôt qu'à la façon dont il est constitué.

10.1.2 Application au réglage de paramètres

L'avantage de cette métaheuristique est sa grande adaptation au problème de réglage de paramètres qui nous occupe. L'application est directe, chaque dimension de l'hyperespace de recherche représente un paramètre de l'algorithme fourmi : α, β, ρ, *etc*.

Une telle approche nous permettrait de tracer les résultats de la méthode en relation avec les changements dans les paramètres, dans la mesure où on peut suivre le comportement des particules, afin de voir s'il y a un effet direct, ou au contraire, si un petit changement dans les paramètres produit un changement massif dans les résultats.

Cette méthode souffre cependant, dans notre cas, du même problème que celle utilisant les AG : la fonction de fitness est l'algorithme *AntCO*^{2} lui-même.

[1] En sautant au passage une possible approche *ontogénétique* qui pourrait ressortir dans ce qui est décrit à la section suivante 10.2 page suivante.

10.1.3 Quelques résultats préliminaires

Les figures 10.1 et 10.2 page suivante montrent deux tests de cette approche avec 5 particules. Chaque particule est représentée par sa courbe de fitness $pbest$. La fitness est ici simplement :

$$\frac{r_1 + r_2}{2} \tag{10.1}$$

Chaque particule est représentée par un vecteur à trois composants $\vec{x} = (\alpha, \beta, \rho)$, on ne fait donc varier dans cette implantation que trois paramètres. Le voisinage utilisé ici est un graphe complet, ce qui explique la forme des courbes, dans lesquelles, une bonne solution a été trouvée très tôt, avec les autres particules se dirigeant vers cette solution.

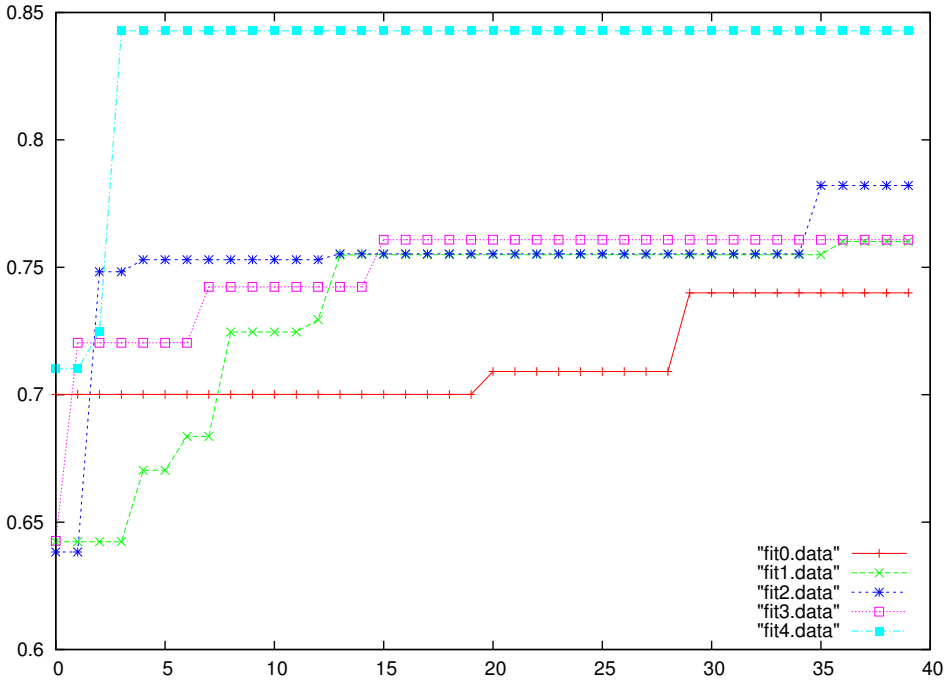


FIG. 10.1: Évolution de la valeur $pbest$ pour cinq particules sur un graphe en grille 5×8.

10.2 Programmation génétique

La programmation génétique (PG), à l'instar des algorithmes génétiques, utilisent les mécanismes de reproduction, les croisements, les mutations, la sélection naturelle, *etc*. Cependant, elle ne fait pas cela sur un codage par gènes d'une solution, mais applique plutôt ses opérateurs directement sur le programme qui sera chargé de donner la solution.

Ici une étape supplémentaire est ajoutée, et la fonction de fitness n'est plus simplement l'évaluation d'une solution, mais l'exécution du programme généré génétique-

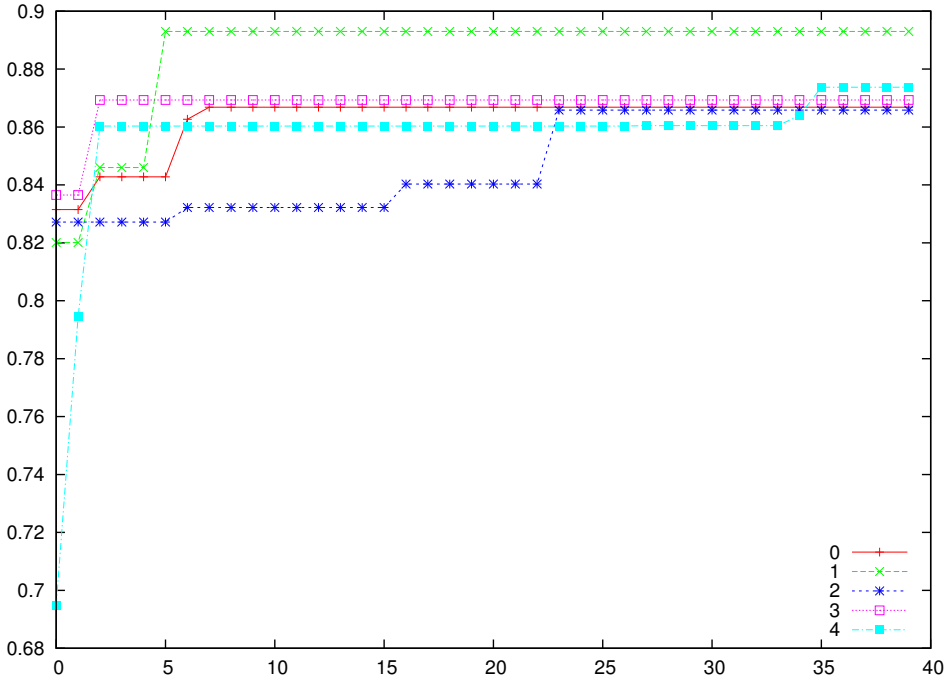


FIG. 10.2: Évolution de la valeur $pbest$ pour cinq particules sur un graphe invariant d'échelle de 400 sommets.

ment, afin de voir si la solution qu'il produit est bonne[2].

10.2.1 Principe

La programmation génétique a été initiée par Nichael Lynn Cramer [Cramer, 1985] puis largement développée par John R. Koza [Koza, 1992].

Dans cette approche, les gènes sont ici directement les instructions d'un programme. De plus, au lieu de coder les gènes séquentiellement les uns après les autres, le chromosome est exprimé sous la forme d'un arbre d'expressions. En effet, tout programme peut être ramené à un arbre comparable à celui présenté sur la figure 10.3 page ci-contre. Dans un tel arbre, les nœuds figurent les instructions. Les instructions prenant des arguments ont un nombre correspondant de branches vers d'autres instructions. Lorsqu'un nœud représente une instruction qui ne prend aucun argument, sur cette figure, il peut quand même avoir une branche qui mène à l'instruction suivante à exécuter.

Les opérateurs génétiques sont ensuite appliqués à l'arbre. La mutation change un nœud d'un sous arbre, et le croisement intervertit des sous-arbres. L'espace des solutions possible peut paraître immense. Cependant, les programmes que l'on peut

[2]Une troisième approche consisterait à boucler la boucle en faisant en sorte qu'un algorithme génétique génère des séquences de gènes qui sont ensuite interprétées en programmes, ces programmes étant exécutés et leur résultats utilisés dans la fonction de fitness. On se rapprocherait encore plus de la façon dont les choses se produisent dans la nature, et c'est l'approche retenue par Samuel Landau dans [Landau, 2003].

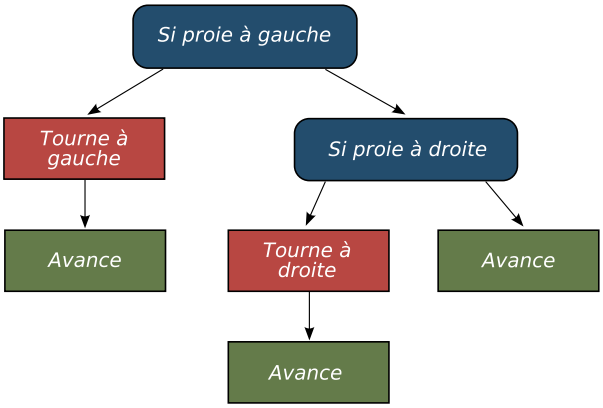


FIG. 10.3: La représentation d'un programme sous la forme d'un arbre d'expressions.

produire sont restreints aux arbres valides :

- certains nœuds ne peuvent avoir qu'une certaine catégorie de nœuds enfants ;
- l'arbre peut-être typé, ce qui restreint encore les possibilités ;
- enfin, la taille de l'arbre peut être contrainte.

Ensuite, à l'instar des AG, la programmation génétique utilise une population de d'arbres et la sélection naturelle pour choisir les meilleurs arbres et les faire évoluer.

10.2.2 Adaptation à notre modèle

À la section 9.2.2 page 120 nous avons passé en revue plusieurs espèces de fourmis crées durant cette thèse, pour répondre à différents besoins, et au fur et à mesure que notre compréhension du problème évoluait. Il pourrait être intéressant de rendre cette procédure automatique. Deux approches peuvent être suivies :

1. Créer de nouvelles espèces par programmation génétique en utilisant *AntCO*^{2} comme fonction de fitness. Ceci implique que la population de fourmis est homogène durant l'exécution.
2. Utiliser de nouveaux groupes de fourmis directement pendant l'exécution en continu d'*AntCO*^{2}, et laisser des mécanisme de gestion de population (mort, naissance), effectuer le rôle de la fitness. Dans ce cas, la sélection serait naturelle dans l'environnement du graphe dynamique.

Quatrième partie

Expérimentations et travaux connexes

Cette partie présente les diverses expérimentations menées sur *AntCO*^{2}. Plusieurs tests sur des graphes statiques ont d'abord été mené à bien, permettant de mettre en évidence certaines caractéristiques du modèle.

Cependant ce dernier à été développé dans le but de distribuer des applications dynamiquement. Le second chapitre se concentre donc sur la dynamique, d'abord au en provenance de l'environnement d'exécution, puis et surtout en provenance de l'application à distribuer.

Le chapitre suivant présente succinctement les deux implantations qui ont été faites d'*AntCO*^{2}, et qui ont permis ces expérimentations. Enfin un dernier chapitre décrit les diverses modifications du modèle appliquées à d'autre domaines qui ont été développées dans le cadre de cette thèse.

Chapitre 11

Graphes statiques

Sommaire

AntCO^2 est prévu à la base pour gérer des applications faites d'un grand nombre d'entités en interaction, et donc particulièrement dynamiques. Cependant, afin de tester certaines caractéristiques de l'algorithme nous l'avons appliqué à divers types de graphes ayant différentes propriétés.

11.1 Grilles

La grille est probablement le modèle de réseau d'interaction le plus simple, et le plus courant. Un grand nombre d'applications sont particulièrement régulières et peuvent se définir suivant une grille. On appelle aussi les grilles des *treillis* (lattice).

Les grilles présentent l'avantage d'avoir un degré quasi-uniforme (à l'exception des bords). Dérivé des grilles, les *tores* peuvent présenter un degré totalement uniforme (ce n'est pas le cas de tous les tores).

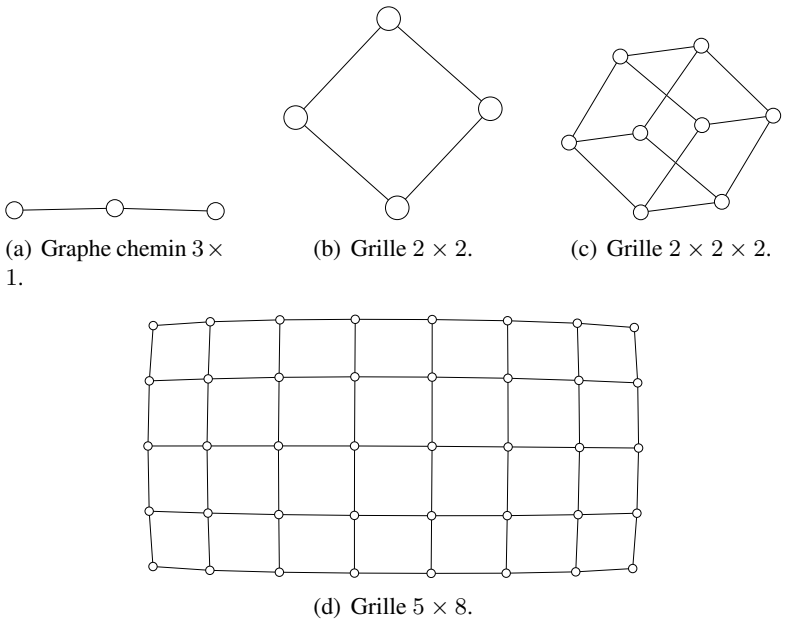
> **Grille** :
> Un graphe en grille $G_{m,n}$ de $m \times n$ sommets, est le produit de deux graphes chemin, de m et n sommets respectivement. Un graphe chemin est un arbre ayant deux nœuds de degré 1 et $n-2$ nœuds de degré 2. On peut généraliser les graphes grille à $G_{m,n,r,...}$.

Les grilles, par leur uniformité, sont particulièrement éloignées des réseaux d'interaction que l'on se propose de distribuer, car elles ne présentent pas d'organisations «topologiques» (ou forment une organisation unique) justement en raison de leur grande

uniformité. Ces organisations peuvent cependant apparaître dans les pondérations des arcs, avec les communications. De plus, le nombre important d'applications se traduisant par un réseau d'interaction en grille rend leur exploration intéressante, et il est intéressant d'observer le comportement de l'algorithme de distribution en l'absence d'organisations distinctes, afin de s'assurer que la charge machine est correctement répartie.

Les grilles sont en quelque sorte à l'opposé des graphes aléatoires que nous verrons par la suite, qui eux aussi ne présentent aucune organisation mais ne sont pas réguliers.

La figure 11.1 montre quelques exemples de grilles.



(a) Graphe chemin 3×1.

(b) Grille 2×2.

(c) Grille $2 \times 2 \times 2$.

(d) Grille 5×8.

FIG. 11.1: Exemples de grilles.

La construction d'un réseau en grille de pondération uniforme est particulièrement triviale comme l'expose l'algorithme 11.1 page suivante :

11.1 : Génération d'une grille uniforme.

Entrées : n, m la largeur et la hauteur de la grille en nœuds.
Sorties : Un graphe G aléatoire.

Création de $m \times n$ sommets dans G. Les sommets sont numérotés de 0 à $(m \times n) - 1$

Créer $m \cdot n$ sommets;

Liaison des sommets entre eux :

pour y *de* 0 *à* $m - 1$ **faire**
  pour x *de* 0 *à* $n - 1$ **faire**
    $s \leftarrow y * n + x$;
    $r \leftarrow s + 1$;
    $b \leftarrow (y + 1) * n + x$;
    si $r < n$ **alors**
      Créer un arc entre v_s et v_r;
    si $b < m$ **alors**
      Créer un arc entre v_s et v_b;

Deux types d'expériences peuvent être intéressantes avec les grilles statiques :
– S'assurer que pour une grille ayant des pondérations uniformes, la charge est correctement répartie et les grappes d'entités le moins éparpillées possible.
– S'assurer que l'algorithme trouve les organisations sous forme de pondération uniquement dans un graphe qui autrement est uniforme.

11.1.1 Grille avec pondération uniforme

La figure 11.2 page 155 présente un test sur une grille 20×20 dont tous les arcs ont une pondération uniforme colorées par 4 colonies. Voici les paramètres utilisés :

Paramètre	*Valeur*
α	1
β	3
ρ	0.8
N^*	10
M	3
ϕ	0.3
Individu/sommet	8
RAZ-P	30
Espèce	*AgoraphoMyrmex*

On constate que l'algorithme est en mesure de trouver un quasi optimal sur la grille[1] : le fait que les couleurs soient parfaitement alignées sur la grille n'est pas un hasard, mais correspond à une surface de contact entre couleurs minimale. Cette configuration est un attracteur pour les fourmis. Quand au critère r_2, ici, il est de 1, c'est-à-dire un équilibrage parfait.

[1] Le problème du partitionnement de graphe est NP-complet.

11.1.2 Grille pondérée

La figure 11.3 page 156 présente un test sur une grille 8×8 dont les arcs sont pondérés pour délimiter quatre zones contenant 16 sommets. Cette figure met en évidence les pondérations.

Voici les paramètres utilisés pour ce test avec 4 colonies :

Paramètre	*Valeur*
α	1
β	3
ρ	0.8
N^*	10
M	9
ϕ	0.3
Individu/sommet	4
RAZ-P	0
Espèce	*AgoraphoMyrmex*

La figure 11.4 page 157 présente les résultats obtenus. L'algorithme trouve exactement les zones de fortes pondération[2] ce qui correspond à une solution ayant de valeurs de r_1 et r_2 optimales.

11.2 Graphes aléatoires

Les graphes aléatoires ont été défini initialement par Paul Erdös et Alfréd Rényi en 1959 dans [Erdös, 1959], et ont été extensivement étudiés par la suite. Nous nous y intéressons dans la mesure où à l'instar des grilles ils ne présentent aucune organisation particulière, mais ne sont pas réguliers.

Les graphes que nous avons utilisés ont une distribution de degrés suivant une loi de Poisson, et ont été construit par l'algorithme 11.2 page suivante. Cet algorithme, simple à implémenter, peut assigner à certains sommets légèrement plus d'arcs que nécessaire, mais au final le résultat correspond toujours à une loi de Poisson[3].

[2]Une vidéo est disponible à l'adresse `http://www-lih.univ-lehavre.fr/~dutot/videos/Grid8x8.mpeg`.

[3]L'algorithme décale le degré moyen demandé, mais la distribution reste une distribution de Poisson

11.2 : Génération d'un graphe aléatoire.

Entrées : d degré moyen d'un sommet.
Sorties : Un graphe G aléatoire.

Création de n sommets dans G :

Créer n sommets;

Liaison des sommets entre eux :

pour *tout sommet* v **faire**

Définition du nombre de voisins d_v de v en fonction d'une loi de Poisson centrée sur d :

$d_v \leftarrow poisson(d)$;

On ne prend pas en compte les arcs créés précédemment par d'autres sommets :

$d_v \leftarrow (d_v - degre(v))$;

Choix aléatoire de d_v sommets différents de v :

$V \leftarrow ensembleDeSommetsAleatoire(G, d_v, v)$;

Création d'un arc entre v et chaque sommet choisi :

pour *tout sommet* v_i *de* V **faire**
  Créer un arc entre v et v_i;

La figure 11.5 page 157 montre la distribution de degrés des sommets d'un graphe de 3000 sommets et un degré moyen de 12, généré avec le programme précédent. En abscisse est indiqué le degré, en ordonnée le nombre de sommets ayant ce degré.

Puisqu'il n'y a aucune organisation, l'algorithme ne sera donc pas en mesure d'améliorer la charge réseau et on ne cherche pas à obtenir une valeur particulière pour r_1. Par contre, l'algorithme doit toujours être en mesure de distribuer au mieux la charge machine, c'est l'objectif de l'expérience présentée ici. La figure 11.6 page 158 montre un graphe aléatoire statique de 300 sommets et 602 arcs, pondéré de manière uniforme, coloré par 4 colonies de fourmis. Voici les paramètres utilisés :

Paramètre	*Valeur*
α	1
β	3
ρ	0.86
N^*	10
M	3
ϕ	0.3
Individu/sommet	8
RAZ-P	0
Espèce	*AgoraphoMyrmex*

Tout comme pour les grilles, il est possible de définir des organisations grâce aux pondération des arcs. Nous avons ici créé des organisations en définissant un arbre de recouvrement du graphe, lui-même coupé aléatoirement pour créer une forêt. Ensuite

nous avons pondéré fortement les arcs appartenant à la forêt, laissant les autres arcs faiblement pondérés.

La figure 11.7 page 159, montre cette forêt une fois colorée par *AntCO²*, comparée au graphe lui-même, ainsi que les tracés r_1 et r_2 sur 4000 étapes. Voici les paramètres utilisés :

Paramètre	*Valeur*
α	1
β	3
ρ	0.86
N^*	10
M	3
ϕ	0.3
Individu/sommet	8
RAZ-P	0
Espèce	*AgoraphoMyrmex*

On constate que le critère r_1 est amélioré par la présence d'organisations. L'algorithme est en mesure de grouper les organisations de forte pondération et d'utiliser comme frontière les faibles pondérations. Le critère r_2 souffre légèrement de ce fait, car les organisations définies sont de taille aléatoire et par conséquent ne permettent pas une distribution uniforme. On constate quelques faibles problèmes de coloration dans certaines organisations dus à la perturbation impliquée par le très grand nombre d'arcs faiblement valués.

11.3 Graphes Complets

Les graphes complets font partie des graphes uniformes sans organisation excepté la totalité du graphe. La figure 11.8 page 160 présente l'évolution des critères r_1 et r_2 sur un graphe complet de 100 nœuds. On constate que r_2 donne de bons résultats malgré la perturbation importante introduite par le degré élevé de chaque nœud. Naturellement, r_1 ne peut être rendu petit, et n'est pas dans le cas des graphes complets informatif. En effet ici, chaque nœud joue le même rôle étant donné que les degrés sont uniformes, et il n'y a pas de meilleure coupe. La courbe bleue pointillée indique la meilleur valeur théorique pour r_1 si les partitions ont la même taille. Ce n'est qu'une valeur informative, la courbe r_1 produite par *AntCO²* coupe cette valeur théorique, car la taille des partitions n'est pas toujours identique. Cette valeur est définie par :

$$\frac{card(\mathcal{V})(card(\mathcal{C})-1)}{card(\mathcal{C})(card(\mathcal{V})-1)} \tag{11.1}$$

Si tous les clusters n'ont pas la même taille, r_1 est égal à :

$$\frac{\sum_{j=1}^{p}\left(card(\mathcal{V}_j(t))\left(\sum_{i=1,i\neq j}^{p} card(\mathcal{V}_i(t))\right)\right)}{n(n-1)} \tag{11.2}$$

Tout comme pour les graphes aléatoires, la figure 11.9 montre l'évolution des critères r_1 et r_2 sur le graphe complet utilisé précédemment, mais pondéré. La même technique à été utilisée. Un arbre de recouvrement est défini sur le graphe complet, il est coupé aléatoirement formant une forêt, et des pondérations élevées sont appliquées aux arcs de cette forêt, tandis que des poids faibles sont placés sur les autres arcs.

Le critère r_1 devient significatif. On constate sur cette figure qu'il est très proche de l'optimal théorique donné par la courbe bleue pointillée. Cette fois encore cette valeur théorique indique une valeur de r_1 pour des tailles de partition identiques. Ce nombre est défini ainsi. Soit $\bar{h}$ et $\bar{l}$ les moyennes de poids hautes et basses respectivement. Soit $\mathcal{E}_h(t)$ et $\mathcal{E}_l(t)$ l'ensemble des arcs étant pondérés fortement et faiblement respectivement au temps t. Soit $\mathcal{A}_h(t)$ et $\mathcal{A}_l(t)$ l'ensemble des arcs de communication effective au temps t étant pondérés fortement et faiblement respectivement. L'optimum présume que $\mathcal{A}_l(t)$ possède le nombre le moins important d'éléments (pour le graphe que nous utilisons, il n'y en a pas) :

$$\frac{card(\mathcal{A}_h(t))\bar{h} + card(\mathcal{A}_l(t))\bar{l}}{card(\mathcal{E}_h(t))\bar{h} + card(\mathcal{E}_l(t))\bar{l}} \tag{11.3}$$

Comme pour le graphe aléatoire, la taille très variable des sous-arbres créés perturbe la distribution.

11.4 Graphes invariants d'échelle

Les graphes invariants d'échelle (mais on utilise plus courrament le terme anglo-saxon *scale-free*, qui se traduirait plus littéralement par «sans échelle»), sont nommés ainsi parce que qu'il ne présentent pas de degré particulier, contrairement aux grilles par exemple qui présentent une structure d'échelle régulière. Ils forment le pont entre les graphes aléatoires, dont le degré des nœuds sont effectivement de tout ordre mais aléatoires, et les graphes réguliers dont le degré des nœuds est quasiment identique en tout point.

La distribution de degrés d'un tel réseau n'est donc pas confinée à une échelle particulière. Et en effet, la «signature» d'un graphe invariant d'échelle est sa distribution de degrés en loi de puissance [Albert, 2002]. Cela signifie qu'il y a beaucoup de nœuds très peu connectés, et très peu de nœuds de degré très élevé. C'est cette caractéristique qui rend les graphes invariants d'échelle «résistants». En effet on a plus de chances, si la destruction est aléatoire, de casser un nœud faiblement connecté qu'un nœud de degré élevé, et il est donc plus probable que des chemins alternatifs existent toujours pour aller d'un point A vers un point B, malgré la reconfiguration.

Ces graphes présentent aussi parfois la caractéristique d'être «petit-monde». Un graphe petit-monde [Watts, 1998] est informellement un graphe dans lequel on peut joindre deux nœuds quelconques du réseau par un nombre limité de sauts de nœud en nœud. Pour tout couple de nœuds il existe donc un chemin court les reliants. Les graphes ou réseaux «petit-monde» sont une généralisation de la constatation courante dans la vie de tous les jours qu'il est probable de rencontrer un inconnu auquel on est lié par une série courte de connaissances communes, et en effet on peut construire de tels graphes en observant les réseaux d'accointances et les réseaux sociaux de tous types (expérience de Stanley Milgram [Milgram, 1967]).

Ces réseaux ou graphes sont souvent constitués au fil du temps, par un processus de construction entretenu. On croyait par exemple au départ que la structure de l'internet était un graphe aléatoire. Mais il a été découvert qu'en réalité l'internet possède la propriété d'invariance d'échelle [Barabasi, 2002]. Il existe de très nombreux autres exemples de tels graphes dans la nature : les réseaux sociaux, les réseaux d'interaction proétiques, *etc* [Newman, 2003]. De tels réseaux sont créés au fil du temps suivant un processus «d'attachement préférentiel». Les liens se font de préférence sur les nœuds qui sont déjà très fortement connectés (et donc souvent «désirables»). Les nœuds les plus connectés ayant tendance, par construction à devenir des «concentrateurs», rares.

La structure d'un tel graphe, et la propriété d'invariance d'échelle qui l'accompagne, émerge donc par construction, sans volonté générale de la part des entités qui contribuent à l'entretien du réseau, à l'exemple d'internet, dont on ne connaissait pas la propriété d'invariance d'échelle. Ces réseaux sont donc particulièrement intéressants, car il est probable de les retrouver dans un nombre important de simulations.

La figure 11.10 page 161 montre un réseau invariant d'échelle coloré par *AntCO*^{2} avec 4 couleurs. Voici les paramètres utilisés :

Paramètre	*Valeur*
α	1
β	3
ρ	0.86
N^*	9
M	9
ϕ	0.3
Individu/sommet	8
RAZ-P	0
Espèce	*AgoraphoMyrmex*

Cette figure montre la capacité d'*AntCO*^{2} à trouver les organisations d'entités communiquantes, r_1 est bon, et r_2 bien qu'en retrait est correct. Il est normal, dans un algorithme qui recherche un compromis entre deux critères contradictoires, de voir r_2 baisser un peu. Ici les organisations présentes dans le graphe sont toutes de tailles très variables, certaines sont très importantes, d'autres moins[4]. *AntCO*^{2} ayant tendance à privilégier explicitement les communications, et donc r_1, certaines organisations de grande taille influencent les résultats d'équilibrage de charge. De plus ces organisations sont très peu connectées, et les fourmis ont bien moins de chance de traverser les arcs les joignant.

Comme on peut le voir, la mémoire des fourmis utilisées pour colorer ce graphe est plus élevée. En effet ce dernier a une structure ressemblant beaucoup à un arbre, ce qui oblige certaines fourmis à faire de très large demi-tours.

[4]Le gestionnaire de mise en forme de graphes utilisé pour réaliser ces figures affiche un graphe qui n'est pas planaire au mieux, il y a donc différentes organisations dans ce graphe qui se recouvrent malheureusement.

11.5 Archive de partitionnements de graphes

La «graph partitionning archive» [Soper, 2004, Soper, 2005] (archive de partitionnements de graphes) propose un classement de nombreux algorithmes de partitionnement sur des graphes de tailles et types divers. Cependant, tous les graphes proposés sont statiques et non pondérés, bien que certains des algorithmes présentés soient en mesure de gérer les pondérations. Il faut aussi noter que les graphes de l'archive ne représentent pas d'application particulière. Ainsi, beaucoup sont très réguliers, de type grille, ne présentant pas d'organisations, ou comme souvent avec les graphes réguliers une unique organisation.

Cependant, cette archive, outre le nombre important de graphes qu'elle met à disposition, permet d'observer le comportement d'*AntCO*^{2} sur des graphes de toute taille et de tout type, et de comparer les résultats. Nous avons déjà abordé et décrit le domaine du partitionnement de graphe à la section 7.1 page 93. La plupart des méthodes proposées dans l'archive utilisent la technique de la bissection récursive, et en conséquence ne sont capable de produire qu'un nombre de partitions puissance de 2. Bien entendu toutes les méthodes présentées ne gèrent pas les graphes dynamiques, et de par la méthode employée (bissection récursive), ne serait pas facilement applicables à de tels graphes.

Nous avons testé *AntCO*^{2} sur trois graphes de large taille de l'archive. Les résultats ne sont pas optimaux lorsque l'on considère les deux critères en même temps, mais sont proches des solutions fournies par les meilleurs algorithmes dédiés ($r_1 < 0.1$ et $r_2 > 0.9$). Ces résultats sont détaillés dans les trois tables 11.1, 11.2 et 11.3. Ces tables sont organisées ainsi : chaque colonne donne les résultats pour un nombre donné de partitions. Sur la première ligne se trouve les résultats du meilleur algorithme référencé par l'archive de partitionnements de graphes. Comme ces algorithmes ne proposent de résultats que pour un nombre de partitions puissance de 2, certaines cases sont vides. Le premier nombre dans chaque case indique le *cut-weight*, le second, entre parenthèses, indique la taille de la partition la plus large. Ce nombre peut être comparé à l'optimal $\frac{n}{p}$. Les lignes suivantes montrent les résultats d'*AntCO*^{2}. D'abord les résultats où le *cut-weight* est minimal, puis les résultats où la «partition maximale est minimale», enfin des compromis entre ces critères.

Partitions	2	3	4	5	6
Meilleur Algo.	612 (1198)		1203 (599)		
Cut-Weight Min.	382 (1552)	913 (993)	1208 (660)	1365 (587)	1712 (541)
Meilleure Part. Max.	946 (1198)	1258 (803)	2397 (609)	4176 (484)	4185 (412)
Compromis	690 (1272)	971 (886)	1208 (660)	1365 (587)	1713 (536)

Partitions	7	8	12	16
Meilleur Algo.		1758 (300)		2116 (150)
Cut-Weight Min.	1162 (592)	1950 (411)	2453 (269)	2806 (290)
Meilleure Part. Max.	4364 (346)	5209 (309)	6650 (209)	5879 (158)
Compromis	1162 (346)	1950 (411)	2453 (269)	2806 (290)

TAB. 11.1: Comparaison d'*AntCO*^{2} avec les algorithmes de la «graph partitioning archive» sur le graphe «add20» (2395 nœuds et 7462 arcs).

Partitions	2	3	4	5	6
Meilleur Algo.	191 (1426)		429 (713)		
Cut-Weight Min.	178 (2059)	258 (1142)	352 (1077)	339 (909)	467 (757)
Meilleure Part. Max.	262 (1426)	8545 (954)	9474 (718)	11851 (574)	12487 (479)
Compromis	252 (1427)	293 (1027)	409 (899)	504 (696)	575 (621)

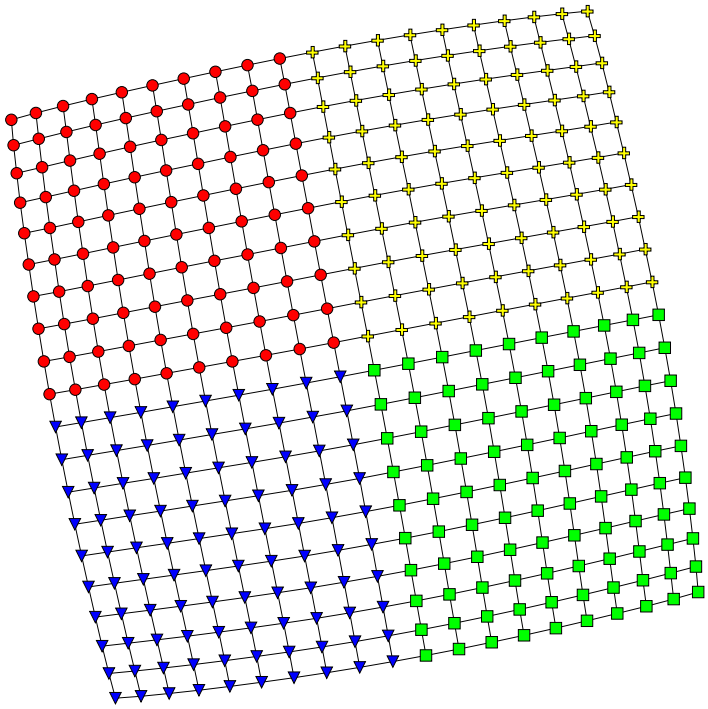
Partitions	7	8	12	16
Meilleur Algo.		728 (357)		1245 (179)
Cut-Weight Min.	563 (691)	540 (587)	861 (404)	1071 (329)
Meilleure Part. Max.	12786 (441)	13098 (360)	13619 (241)	13989 (182)
Compromis	648 (510)	552 (568)	861 (404)	1075 (311)

TAB. 11.2: Comparaison d'*AntCO*^{2} avec les algorithmes de la «graph partitioning archive» sur le graphe «data» (2851 nœuds et 15093 arcs).

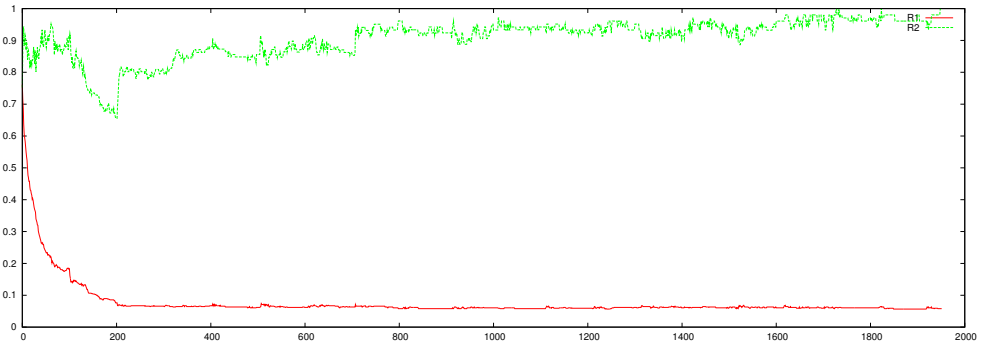
Partitions	2	3	4	5
Meilleur Algo.	10343 (6164)		19245 (3082)	
Cut Weight Min.	11671 (6554)	17461 (4332)	21510 (3308)	27289 (3013)
Meilleure Part. Max.	11945 (6164)	20419 (4110)	123309 (3088)	131206 (2477)
Compromis	11671 (6554)	17461 (4332)	21510 (3308)	27289 (3013)

Partitions	6	7	8
Meilleur Algo.			25468 (1541)
Cut Weight Min.	29809 (2425)	31755 (2072)	33258 (1869)
Meilleure Part. Max.	137189 (2064)	141173 (1767)	144189 (1549)
Compromis	29809 (2425)	31755 (2072)	33258 (1869)

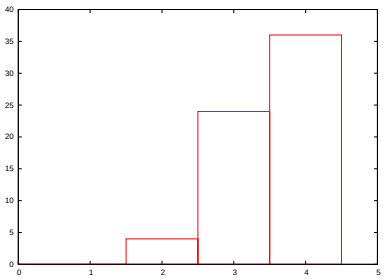
TAB. 11.3: Comparaison d'*AntCO*^{2} avec les algorithmes de la «graph partitioning archive» sur le graphe «vibrobox»(12328 nœuds et 165250 arcs).



(a) Grille 20×20.



(b) Évolution de r_1 and r_2.



(c) Distribution de degrés.

FIG. 11.2: Grille 20×20 de pondération uniforme.

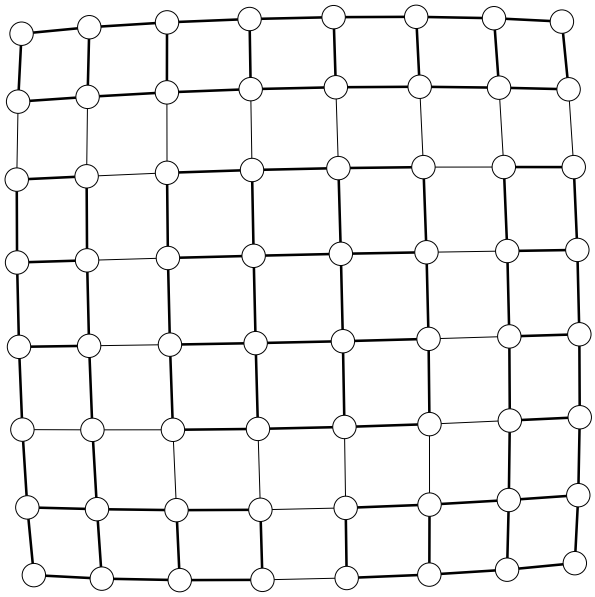
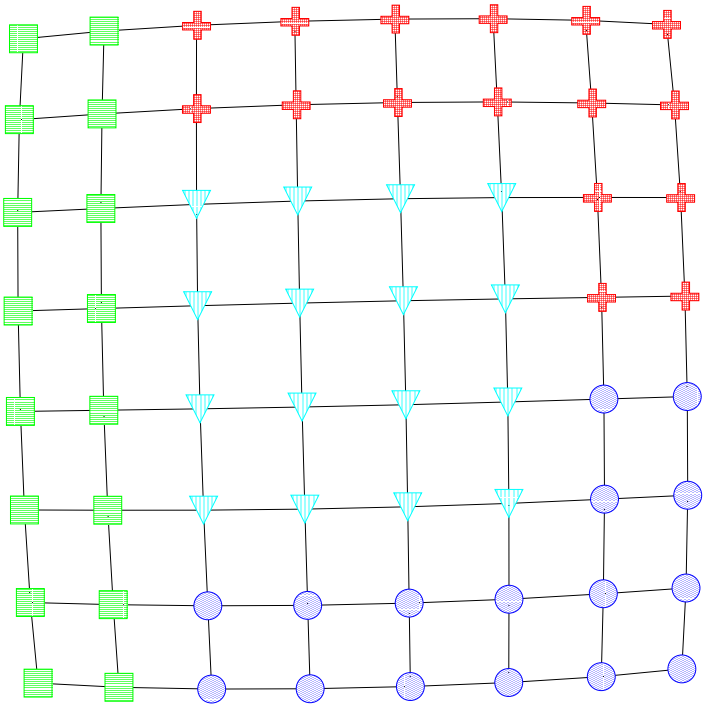
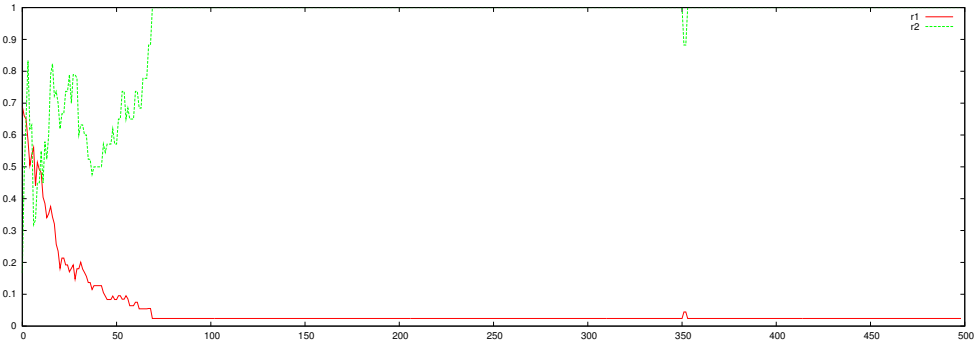


FIG. 11.3: Pondérations sur une grille 8×8



(a) Grille 20×20.



(b) Évolution de r_1 and r_2.

FIG. 11.4: Grille 8×8 pondérée en 4 groupes de 16 nœuds.

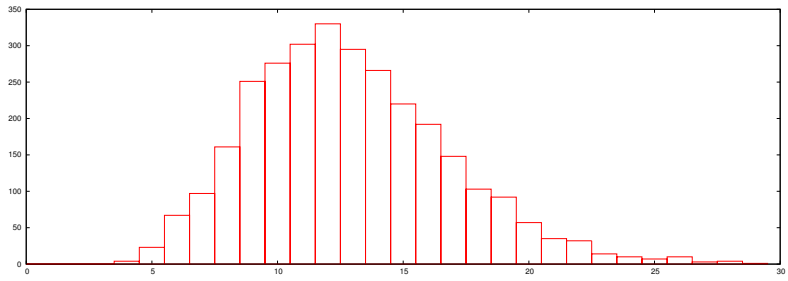
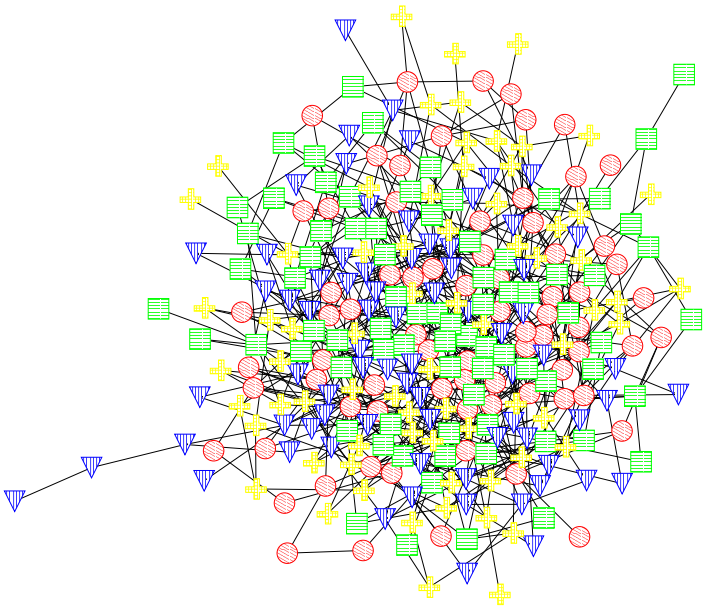
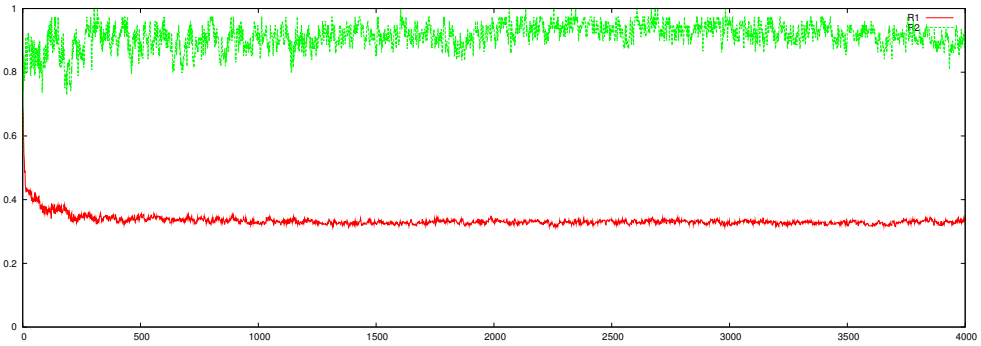


FIG. 11.5: Distribution de degrés des sommets d'un graphe aléatoire de 3000 sommets.

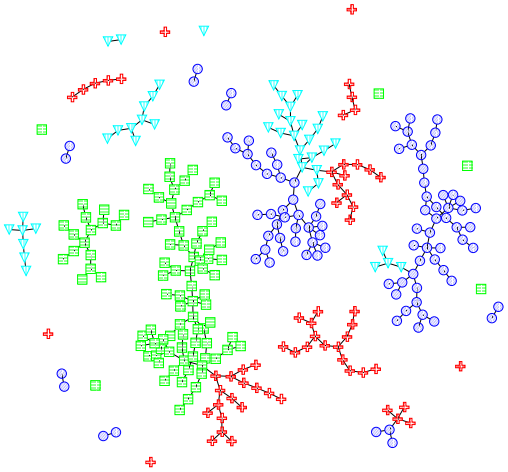


(a) Graphe aléatoire coloré (étape 4000).

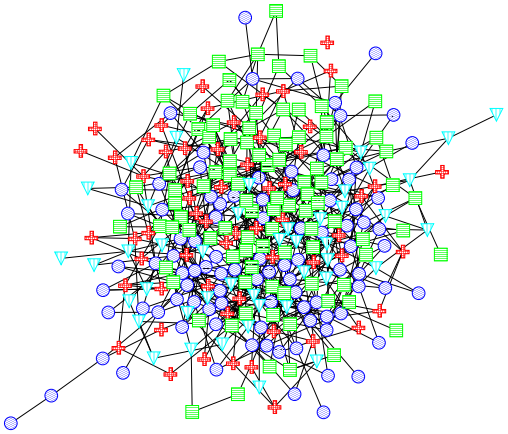


(b) Évolution de r_1 and r_2 sur 4000 étapes.

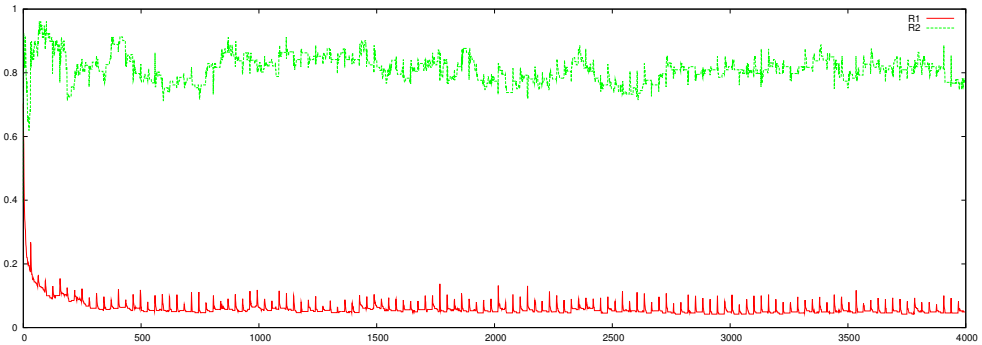
FIG. 11.6: Graphe aléatoire non pondéré (300 sommets, 602 arcs).



(a) Graphe aléatoire pondéré coloré (étape 4000).



(b) Graphe aléatoire pondéré coloré (étape 4000).



(c) Évolution de r_1 and r_2 sur 4000 étapes.

FIG. 11.7: Graphe aléatoire pondéré (300 sommets, 602 arcs).

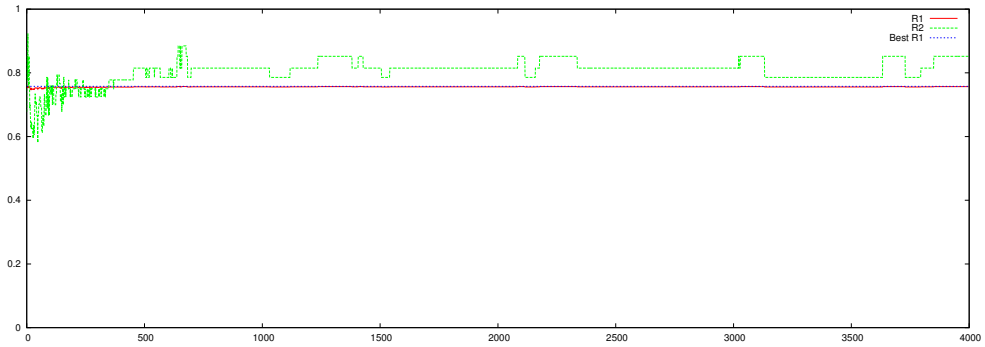


FIG. 11.8: Évolution de r_1 et r_2 sur un graphe complet de 100 sommets non pondéré.

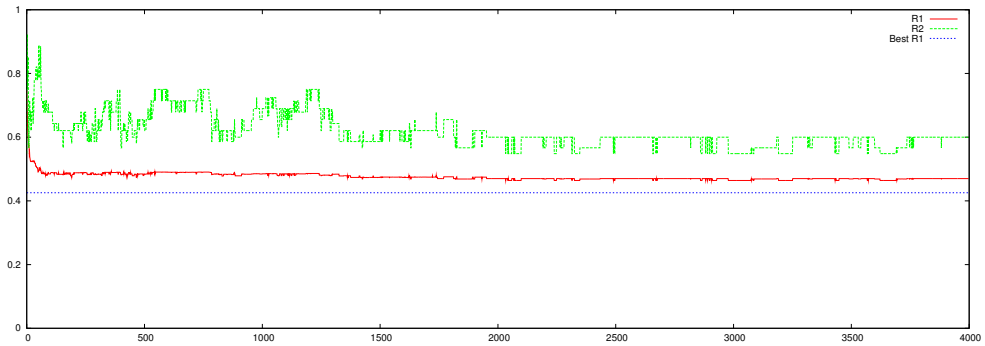
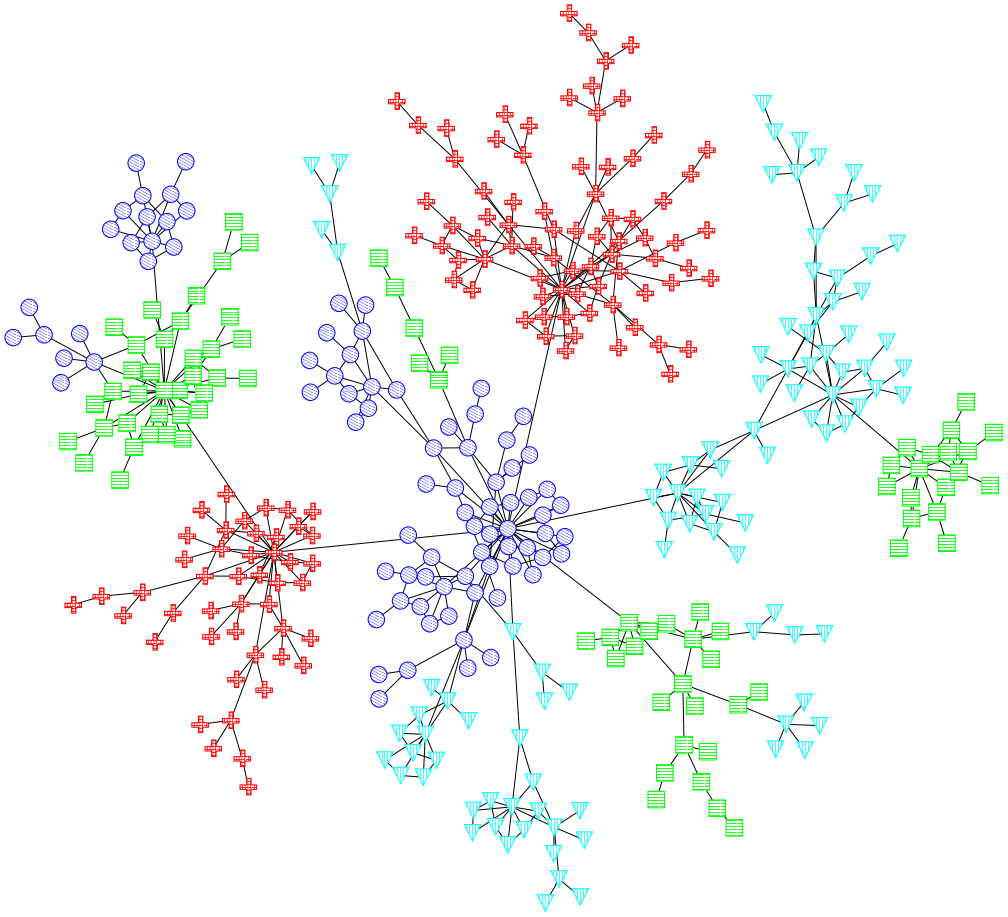
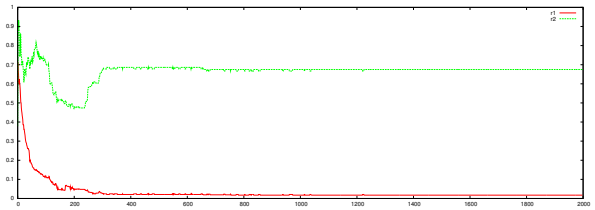


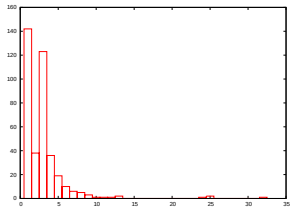
FIG. 11.9: Évolution de r_1 et r_2 sur un graphe complet de 100 sommets pondérés.



(a) Graphe invariant d'échelle coloré.

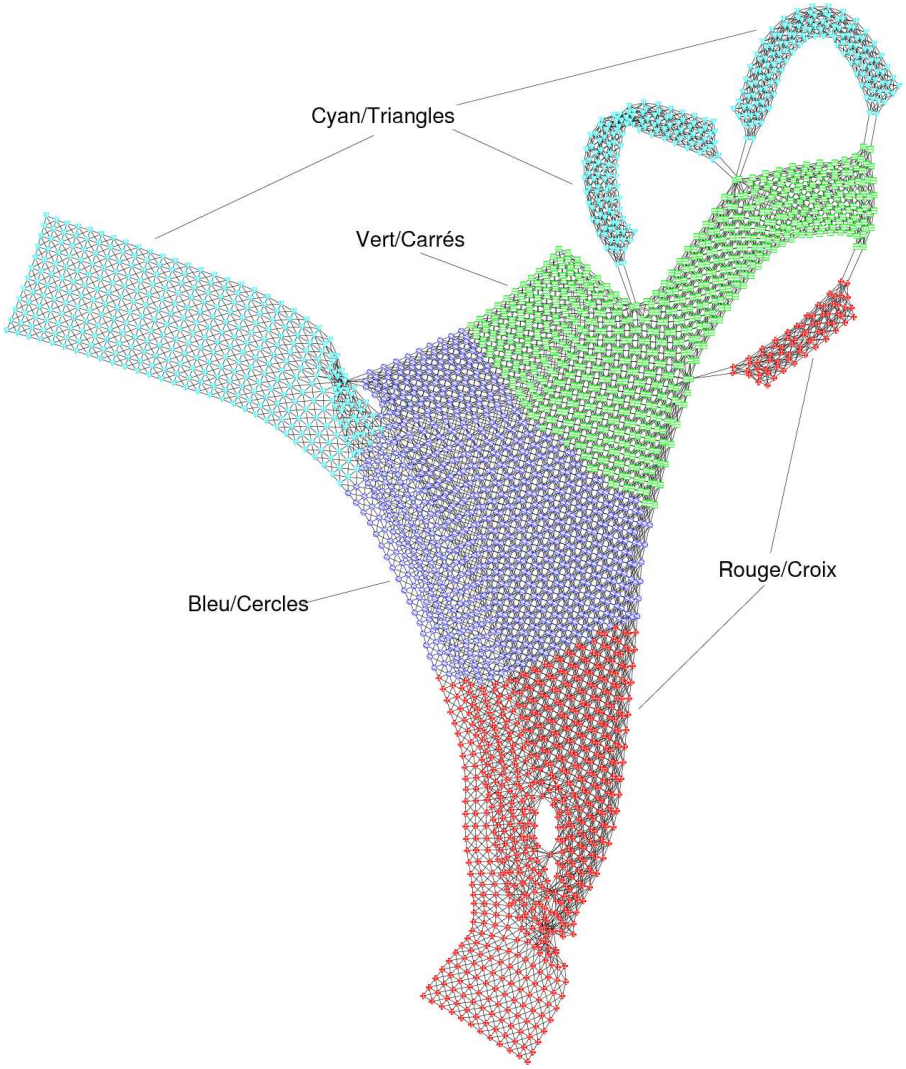


(b) Évolution de r_1 and r_2 sur 2000 étapes.

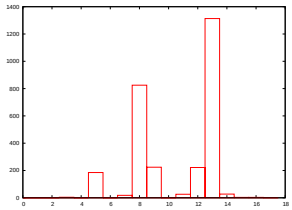


(c) Distribution de degrés.

FIG. 11.10: Graphe invariant d'échelle de 391 sommets et 591 arcs.



(a)



(b) Distribution de degrés.

FIG. 11.11: Le graphe «data» coloré par 4 colonies (le graphe n'est pas planaire et par endroits deux couches se superposent).

Chapitre 12

Graphes dynamiques

Sommaire

Nous avons déterminé plusieurs sources de dynamique. Le graphe représentant le réseau d'interactions de l'application à distribuer en est une. Le système sur lequel s'exécute l'application, et *AntCO*^{2}, en est une autre. Nous allons dans un premier temps décrire quelques expériences sur des graphes statiques dans lesquels la dynamique ne provient que des ressources de calcul. Nous verrons ensuite quelques graphes dynamiques simples, utilisés surtout dans les phases préliminaires pour tester *AntCO*^{2}. Nous étudierons ensuite de manière plus complète une application de simulation particulière définissant des essaims de particules au comportement très similaires aux boids que nous avons rencontrés déjà plusieurs fois, ainsi que le comportement d'*AntCO*^{2} avec cette application.

12.1 Dynamique dans les ressources de calcul

Nous avons repris dans la figure 12.1 page suivante l'exemple d'une grille 30×30 statique et avons introduit de nouvelles colonies à intervalle régulier toutes les 1000 étapes. On peut comparer cette approche à la bissection récursive que nous avons vue à la section 7.2 page 94, mais avec un nombre de partitions qui peut être différent d'une puissance de 2. Ce test simule un environnement de calcul dans lequel de plus en plus de ressources de calcul deviennent disponibles. La figure montre aussi l'évolution des critères de qualité r_1 et r_2 durant ce test. On constate que la taille restreinte de la grille vis-à-vis du nombre de plus en plus élevé de ressources de calcul amène une convergence légèrement moins rapide vers une solution de bonne qualité, mais que cette dernière se maintient.

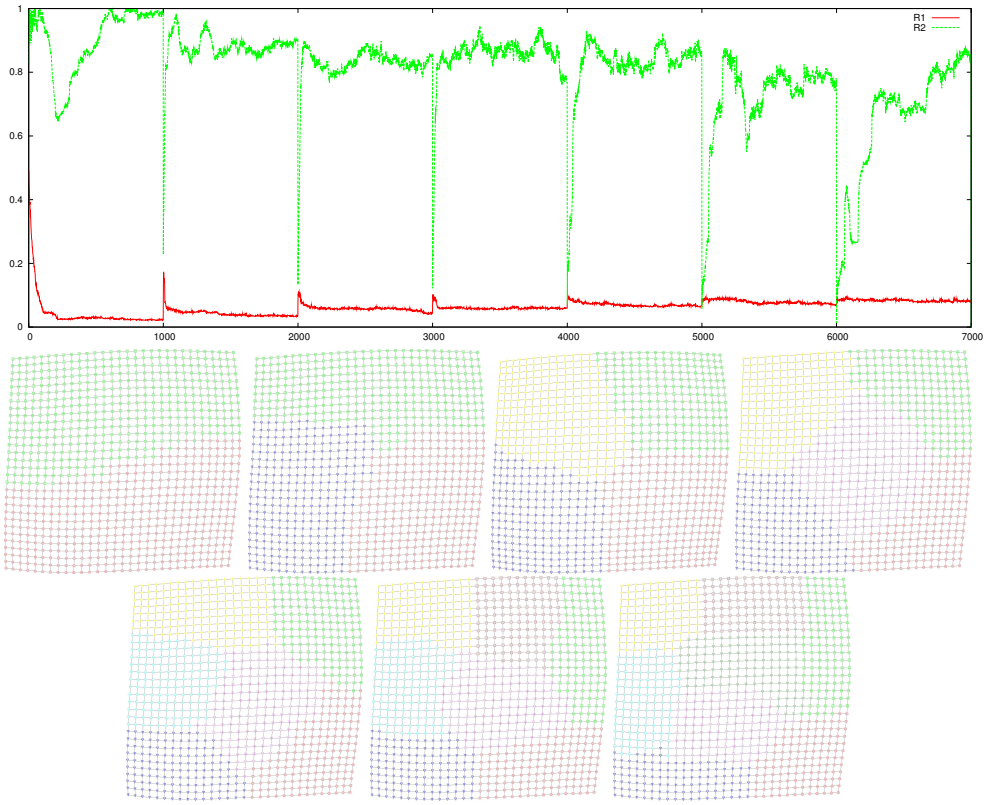


FIG. 12.1: Ajout d'une colonie toutes les 1000 étapes dans une grille 30×30.

Il est intéressant de voir qu'aucune réorganisation massive du graphe n'est opérée lorsque les ressources de calcul sont ajoutées. Certaines zones sont coupées tandis qu'une partie reste en position, minimisant ainsi le nombre de migrations.

Le même processus à été testé sur un graphe bien plus important extrait de l'archive de partitionnements de graphes de 2851 sommets et 15093 arcs (montré figure 11.11). La figure 12.2 page ci-contre montre l'évolution des critères r_1 et r_2 avec des résultats similaires à l'expérience précédente.

12.2 Divers graphes dynamiques

L'algorithme que nous proposons est dédié aux graphes dynamiques. L'intérêt de la méthode utilisée réside dans le calcul incrémental de la solution. Le calcul peut être interrompu à tout moment par un changement dans les entrées (le graphe, les ressources de calcul) et recommencer de là où il en est à détecter des organisations.

En effet, au niveau de ses composants de bases, le graphe dynamique est constamment en reconfiguration. Par contre, pris dans son ensemble, des organisations durables apparaissent. Elles sont le reflet des organisations se créant dans l'application distribuée dont le graphe est tiré. Ces organisations ont une durée de vie souvent bien plus grande que la durée de vie moyenne d'un arc ou d'un sommet du graphe. Au sein de ces organisations la communication est plus élevée, non seulement en volume mais

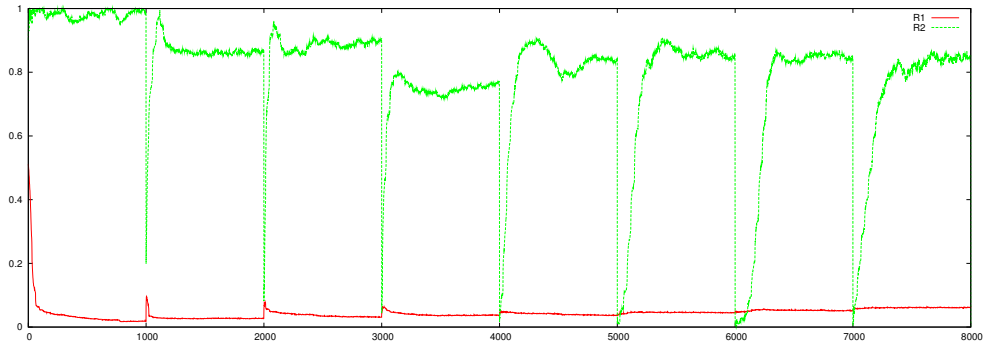


FIG. 12.2: Ajout d'une colonie toutes les 1000 étapes sur le graphe «data» de 2851 sommets et 15093 arcs.

aussi souvent en connectivité. Ce sont les critères utilisés par les fourmis pour créer des grappes d'entités, détectées comme des organisations.

Cette section présente quelques graphes dynamiques simples utilisés en majorité pour tester le comportement d'*AntCO*^{2}. Ces graphes ne sont pas obtenus à partir d'une application particulière, mais artificiellement créés en générant un graphe de topologie donnée, puis en le modifiant par l'intermédiaire d'événements tels que :

– l'ajout ou le retrait d'un sommet ;
– l'ajout ou le retrait d'un arc ;
– la modification d'une valeur.

La figure 12.3 page 169 montre un graphe simple de 18 sommets dont les arcs apparaissent et disparaissent au cours du temps. Sur cette figure les sommets sont représentés par les rectangles, sur les arcs sont affichés des diagrammes circulaires indiquant les taux relatifs de phéromones, le niveau maximum étant indiqué textuellement. Sur les sommets sont indiqués, à gauche le nombre total de fourmis, et à droite un diagramme circulaire indiquant le nombre relatif de fourmis de chaque couleur sur le sommet.

Le graphe est naturellement découpé en trois groupes de sommets liés dans le centre par une série d'arcs apparaissant et disparaissant. Dans chaque groupe, des arcs apparaissent et disparaissent aussi. Ce graphe pourrait représenter une petite application dans laquelle chaque ensemble d'entités travaille à part, se communiquant des informations par intermittence. Au sein de ces groupes d'entités, tous les éléments ne sont pas en communication constante.

Voici les paramètres utilisés :

Paramètre	*Valeur*
α	1
β	4
ρ	0.8
N^*	5
M	4
ϕ	0.3
Individu/sommet	10
RAZ-P	0
Espèce	*AgoraphoMyrmex*

La figure 12.4 page 170 représente un graphe de 32 sommets qui passe régulièrement par trois configurations. Cette figure montre les mêmes vues du graphe sur deux colonnes, la première reprenant les mêmes informations que la figure précédente, l'autre n'affichant que les groupes colorés, pour plus de lisibilité. Ce test montre un graphe constitué principalement de 3 groupes naturels, coloré par 4 colonies. La difficulté réside dans la répartition de charge, puisque le nombre de groupes d'entités observé est inférieur au nombre de ressources de calcul disponibles. De plus ce graphe change continuellement de structure les entités au sein de organisations communicant plus ou moins.

On constate que la répartition de charge est correcte, et que les organisations détectées restent en place malgré leurs reconfigurations importantes. L'un des trois groupes est coupé en deux pour répartir la charge également sur les quatre ressources de calcul.

La figure 12.5 page 171 montre un graphe de type grille sur lequel se déplace une plus petite grille, s'y connectant. La second grille se connecte par intermittence à tous les sommets de la plus large grille[1]. Ce graphe simule une organisation d'entités parcourant un maillage d'entités, que l'on peut comparer par exemple dans une simulation aquatique à un banc de poissons (la petite structure) passant dans un environnement (la grille). Les communications au sein du banc de poissons sont basées sur la vision de ces derniers et est donc importante et durable au sein du banc, alors que la perception de l'environnement est momentanée.

La plus petite structure conserve sa coloration tout au long de l'expérience, bien qu'elle croise plusieurs domaines de couleur différente. Ceci est du en partie au fait que les communications au sein de la petite grille sont élevées.

12.3 Écosystème marin

Une application de simulation d'écosystème marin [Tranouez, 2005] a été développée afin de tester *AntCO*^{2}. Cette application repose sur le principe des Boids de Craig Reynolds que nous avons détaillé à la section 4.5.1 page 68, avec plusieurs modifications. Cette application non seulement permet de tester *AntCO*^{2} sur un problème réel, mais a aussi servi à comparer ses performances avec deux autres méthodes de distribution.

[1] Deux vidéos de cette simulation sont disponibles aux adresses `http://www-lih.univ-lehavre.fr/~dutot/videos/MovingStruct1.avi` et `http://www-lih.univ-lehavre.fr/~dutot/videos/MovingStruct2.avi`.

12.3.1 Simulation

Le comportement général des boids définis dans cette simulation est identique à celui proposé par Reynolds, cependant, l'environnement définit additionnellement un flux porteur qui agit sur leur déplacement. De plus, il existe plusieurs espèces de boids, chacune pouvant être paramétrée différemment. Deux individus d'espèces différentes se repoussent.

Ainsi les rétroactions positives :
- attraction vers le barycentre du groupe perçu ;
- orientation dans le direction générale du groupe ;
- adaptation à la vitesse du groupe,

sont contrecarrées pas plusieurs rétroactions négatives :
- distance minimale vis-à-vis des membres du groupe ;
- angle de vue inférieur à 360 degrés ;
- réaction de fuite vis-à-vis d'autres espèces.

Ces mécanismes permettent la création d'organisations de boids qui, au lieu de ne former ultimement un unique groupe, se scindent en plusieurs organisations, évoluant au fil du temps, pouvant être brisées par d'autres espèces.

L'implantation de cette simulation utilise une grille évitant ainsi de parcourir tout l'espace pour déterminer le voisinage d'individus à prendre en compte lors du déplacement. De plus cette grille est utilisée pour simuler un mode de distribution par maillage de l'environnement.

12.3.2 Mode de distribution

Trois modes de distributions ont été testés avec cette simulation :

1. aléatoire,
2. par maillage,
3. $AntCO^2$.

Dans le mode aléatoire, les boids se voient assigner un processeur dès leur apparition et n'en changent plus par la suite. Dans la simulation que nous avons utilisé, le nombre de boids est fixé par avance, et ne varie pas par la suite. Ce mode de distribution donne un équilibrage de la charge optimal. Chaque machine est chargée de manière équivalente. Par contre la charge de communication n'est pas prise en compte. Au sein d'une organisation il y a donc de fortes chances que deux boids communicants soient sur des processeurs différents. Ce mode de distribution est donc le pire en ce qui concerne la charge réseau.

Dans le mode par maillage, l'environnement est découpé en mailles, et ces dernières sont assignées chacune à une ressource de calcul. Les boids s'exécutent sur la ressource de calcul assignée à la maille dans laquelle ils se trouvent. Ce mode améliore la minimisation des communications au détriment de l'égalisation de la charge machine. En effet dans ce mode, les boids se déplaçant sans contrainte, il est possible que tous les individus migrent sur une machine donnée. De plus, vis-à-vis des communications, il est possible qu'une organisation stagne à la frontière de deux mailles gérées par des processeurs différents, plaçant donc des communications de forte intensité sur le réseau.

12.3.3 Résultats

Ces deux modes ont été comparés aux résultats produits par *AntCO*^{2}. On constate que ce dernier opère un bon compromis entre charge réseau et charge machine. En ce qui concerne le critère de qualité r_1, les communications, les résultats sont meilleurs que les deux autres approches. Pour r_2, il donne des résultats similaires voir meilleurs que le maillage. Il est par contre bien entendu, impossible de dépasser le mode aléatoire sur le critère r_2, le nombre de Boids étant invariant, et l'allocation parfaite dès le début.

Les figures 12.6 page 172 et 12.7 page 172 montrent l'évolution des critères r_1 et r_2 respectivement sur un test avec 200 boids répartis dans 4 espèces durant 5000 étapes de temps. Les figures comparent les trois stratégies évoquées : aléatoire, par maillage et *AntCO*^{2}. Pour r_2, l'allocation aléatoire toujours idéale et n'est pas montrée.

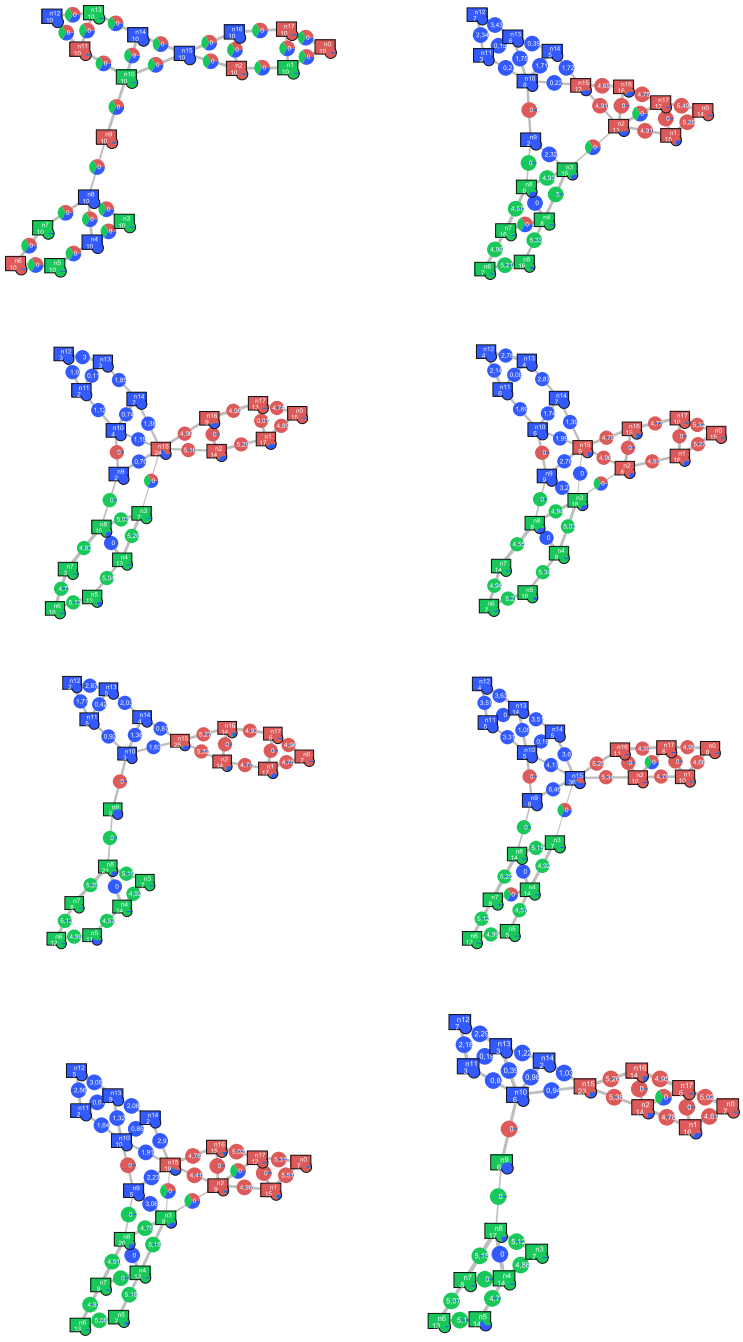


FIG. 12.3: Graphe dynamique de 18 sommets (de gauche à droite, puis de bas en haut).

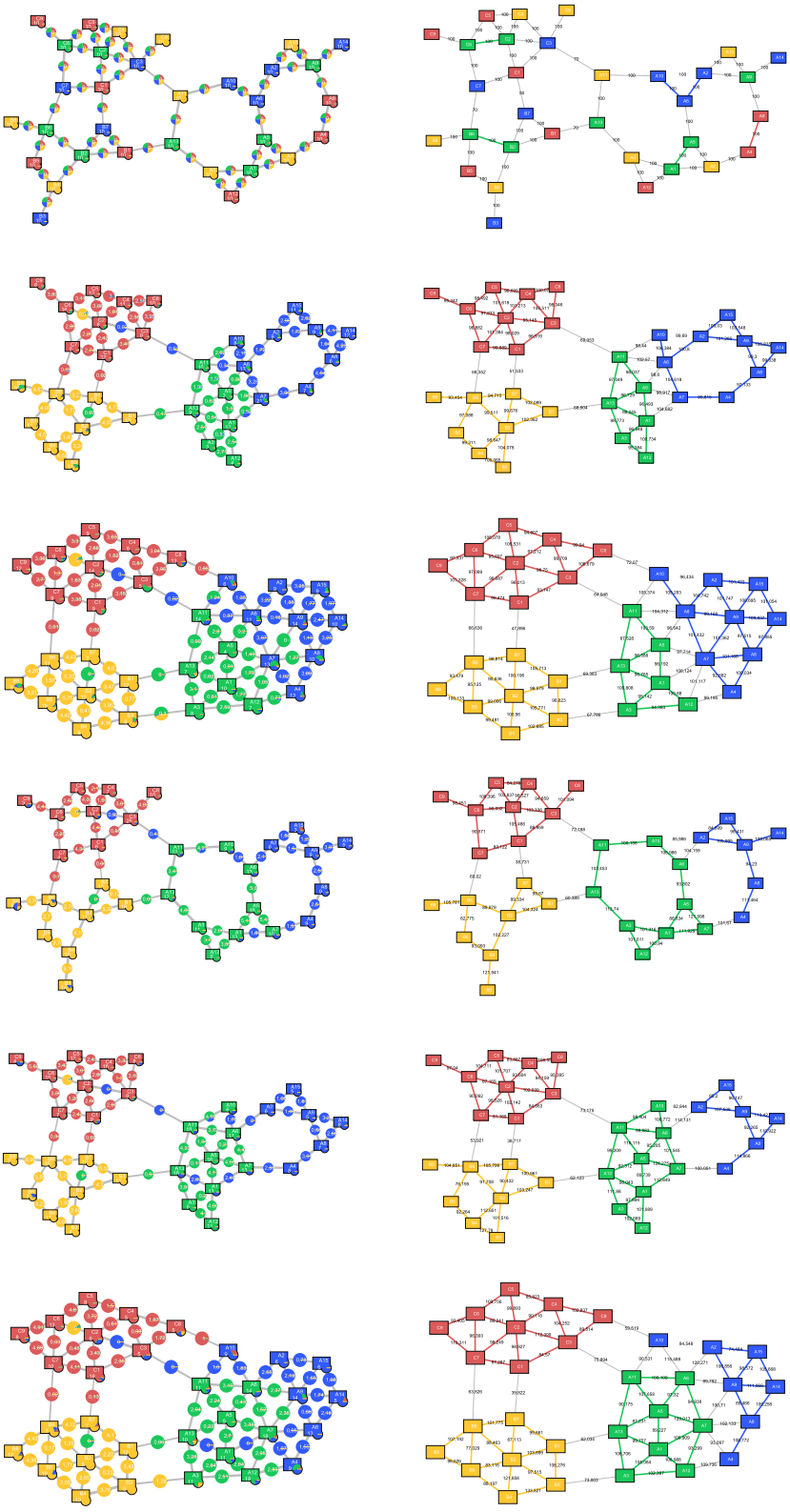


FIG. 12.4: Graphe dynamique de 32 sommets.

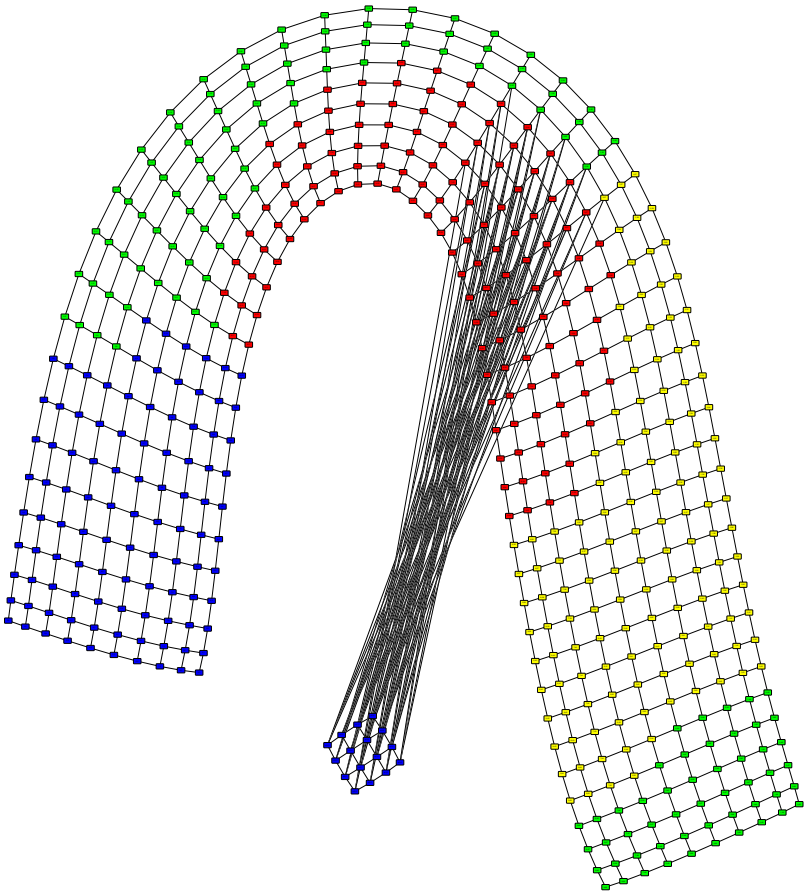


FIG. 12.5: Une structure (4×4) se déplace sur une plus large (10×50).

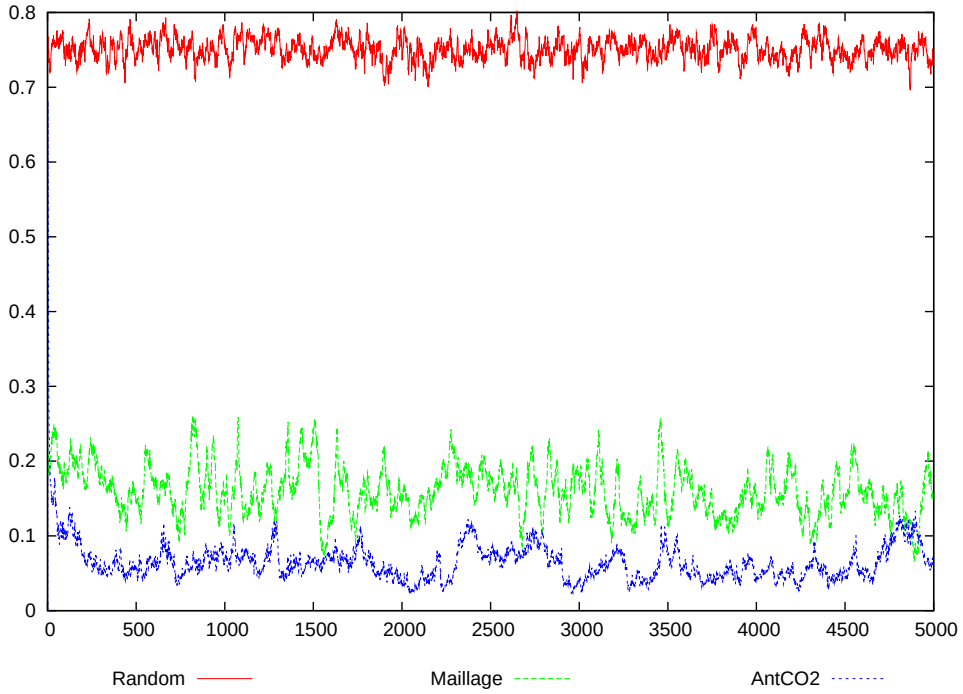


FIG. 12.6: Comparaison sur le critère r_1 entre les stratégies «aléatoire», «maillage» et «*AntCO*^{2} » avec 200 boids.

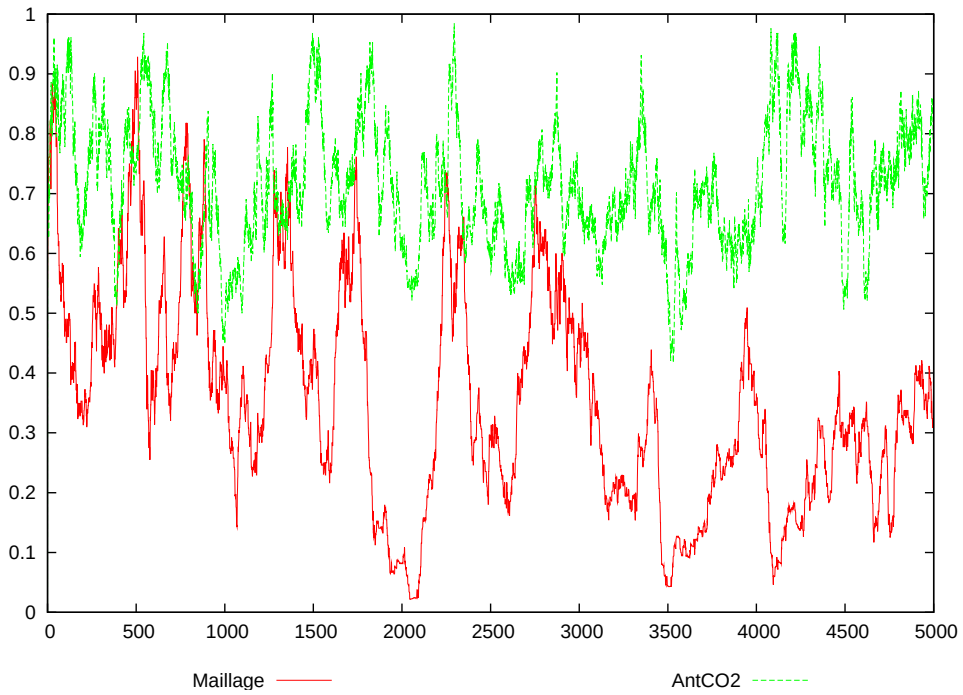


FIG. 12.7: Comparaison sur le critère r_2 entre les stratégies «maillage» et «*AntCO*^{2} » avec 200 boids.

Chapitre 13

Implantation

Sommaire

Ce chapitre décrit rapidement les deux implantations qui ont été faites d'*AntCO*^{2}. La première est une version simple non distribuée permettant de tester le paramétrage avec divers graphes représentant des applications «possibles». La seconde version est un prototype distribuable intégrable avec diverses applications.

La méthode d'implantation à son importance pour des modèles tels que celui que nous avons utilisé pour lesquels les détails de mise en œuvre peuvent influer sur les résultats. En effet, le choix de la gestion du temps, du mode de synchronisme (dont nous avons déjà parlé à la section 9.3.3 page 131), ainsi que des modes de couplage avec une application ou des «simulations» d'applications (de simulation...), permet de mieux appréhender la méthode.

13.1 Première Implantation : Distribution Simulée

Dans cette première implantation, l'application à distribuer était représentée par un graphe pondéré dynamique la simulant. Dans cette implantation le simulateur d'application exécute un événement dans le graphe à chaque itération. Ce simulateur peut aller n fois plus vite que *AntCO*^{2} (n itérations du graphe pour une itération d'*AntCO*^{2}) ou n fois moins vite (une itération du graphe pour n itérations d'*AntCO*^{2}).

L'implantation a été utilisée uniquement pour les tests et n'est pas distribuée. Le graphe et *AntCO*^{2} s'exécutent sur la même machine. Le nombre de fourmis dépassant largement le nombre de threads qu'il serait possible de faire tourner sur les ordinateurs qui ont été utilisés, ces derniers n'ont pas été utilisés. Les fourmis se déplacent chacune à leur tour sur le graphe par cycles. Un cycle est terminé lorsque toutes les fourmis se sont déplacées.

Cependant une telle technique pose un problème particulier : si toutes les fourmis s'exécutent en séquentiel, toujours dans le même ordre. En effet, les modifications effectuées par la première fourmi sur le graphe seront détectées par les suivantes. De

plus, cette première fourmi trouvera un environnement différent de celui que découvriront les suivantes, ce qui ne correspondrait pas à la description que nous avons donné de notre modèle.

Pour résoudre ce problème, les fourmis travaillent sur un environnement préservé, et stockent leurs modifications dans un environnement temporaire. À la fin de chaque cycle, les changements sont appliqués au graphe de manière à ce que les fourmis le retrouvent telles qu'elle l'ont modifié au prochain cycle. On peut comparer ce mécanisme à une protection par transactions qui ne sont validées qu'à la fin de chaque cycle.

Les figures 13.1 et 13.2 page suivante montrent des captures d'écran de cette première implantation.

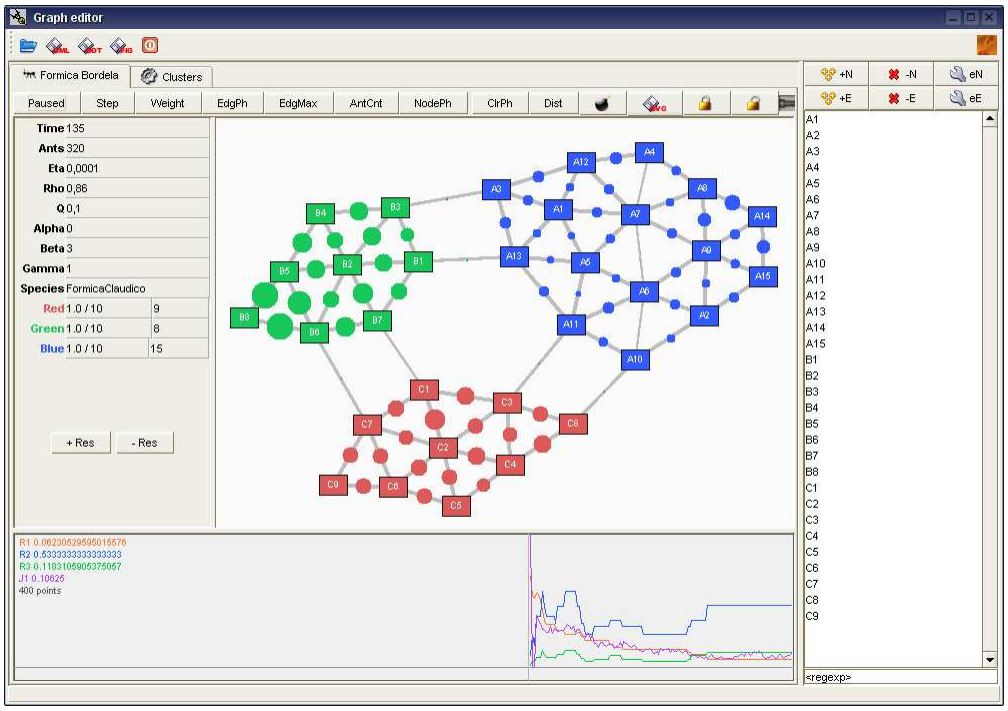


FIG. 13.1: Capture d'écran de la première implantation du modèle.

13.2 Seconde Implantation : Prototype Distribué

Cette implantation reprend les même capacités que la précédente en ce qui concerne la simulation d'applications avec des graphes, à la différence qu'il est possible de définir des itérations de l'application directement dans le graphe. Ceci permet d'avoir des itérations produisant 1 événement en même temps que des itérations produisant n événements. Le graphe peut aussi simuler une application envoyant des informations en continu.

Cette version sépare les composants d'*AntCO*^{2} et l'application, et ces composants peuvent se situer sur des machines différentes. Dans cette application, les diverses

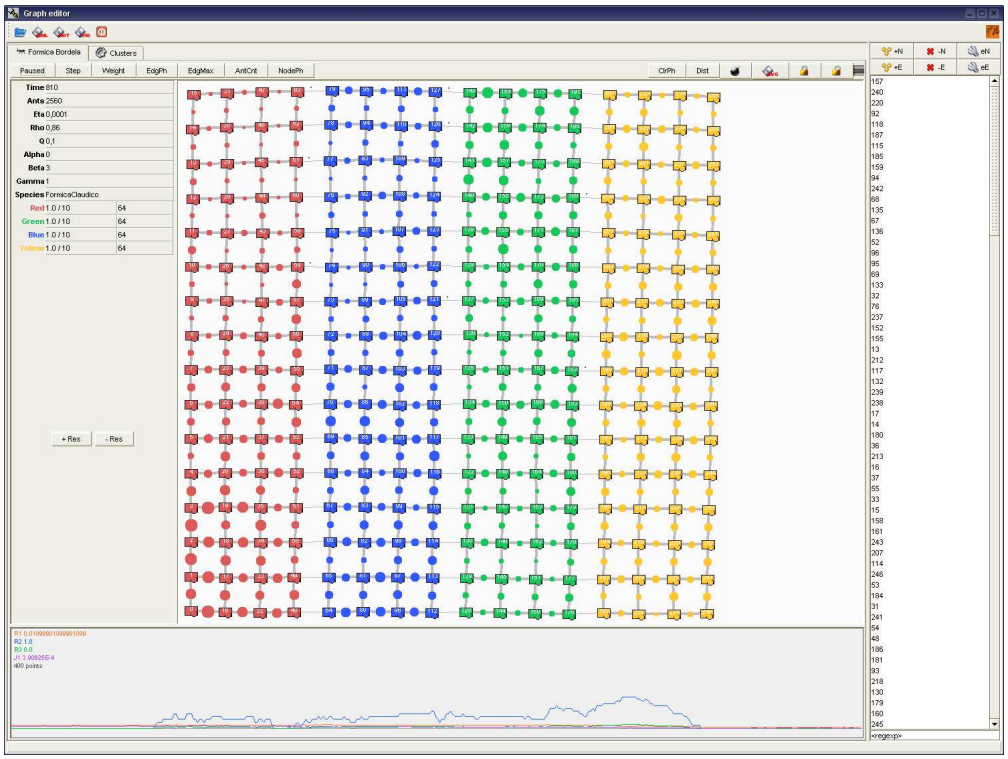


FIG. 13.2: Capture d'écran de la première implantation du modèle.

instances d'*AntCO*^{2} *peuvent* être synchronisées. Cependant, il est aussi possible de laisser chaque instance travailler à la vitesse de la ressource de calcul sur laquelle elle se trouve. Plus *AntCO*^{2} peut travailler vite par rapport à l'application, meilleurs sont les résultats. *AntCO*^{2} est prévu pour être une application extrêmement légère.

L'architecture de la partie fourmi de ce logiciel est donnée dans le diagramme UML des figures 13.3 et 13.4. *AntCO*^{2} est un service, mais un composant d'interface graphique a été mis en place qui permet de se connecter aux instances d'*AntCO*^{2} et d'afficher l'état de la distribution. Les figures 13.5 et 13.6 montrent une vue de cette interface.

De nombreux développements ont été effectués sur *AntCO*^{2}. Outre le modèle qui peut être téléchargé sur la page

http ://www-lih.univ-lehavre.fr/ dutot/programmes/ants.html

des bibliothèques de gestion des graphes sont disponibles sur la page

http ://www-lih.univ-lehavre.fr/ dutot/programmes/graphs.html .

Cette implantation n'est qu'une première étape vers un couplage plus complet avec une véritable application distribuée. Pour l'instant seule la simulation d'écosystème aquatique a été couplé logiciellement avec *AntCO*^{2}.

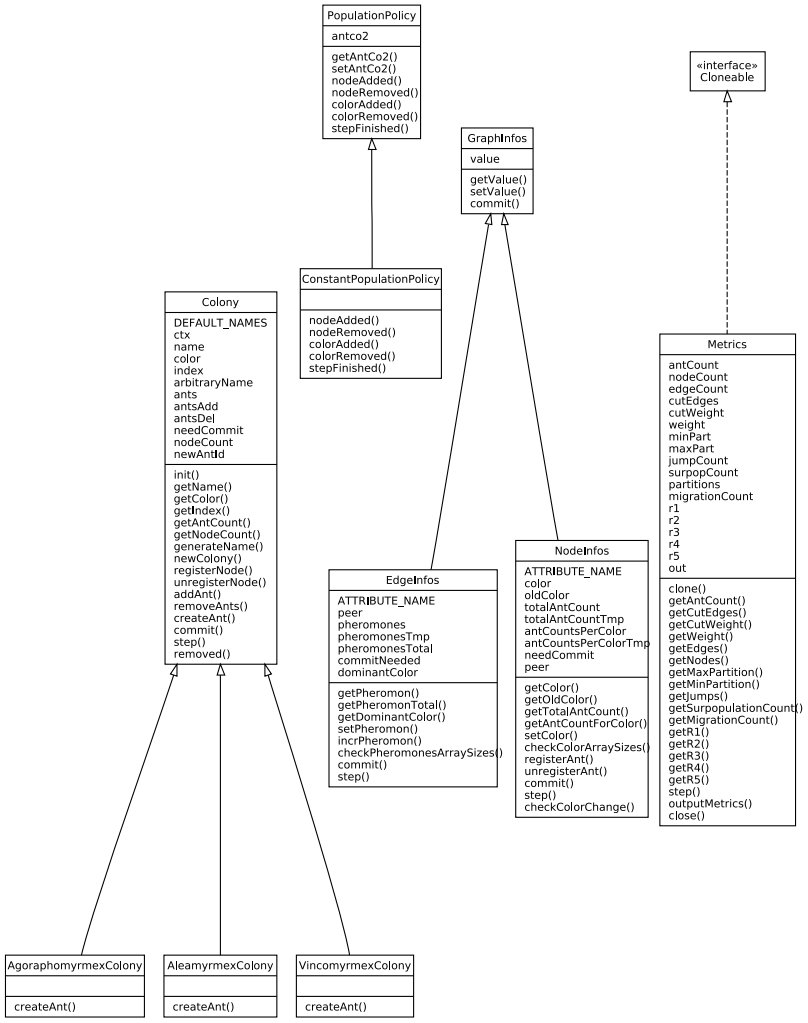


FIG. 13.3: Diagramme UML de la partie logicielle implémentant le modèle d'*AntCO*^{2} (A).

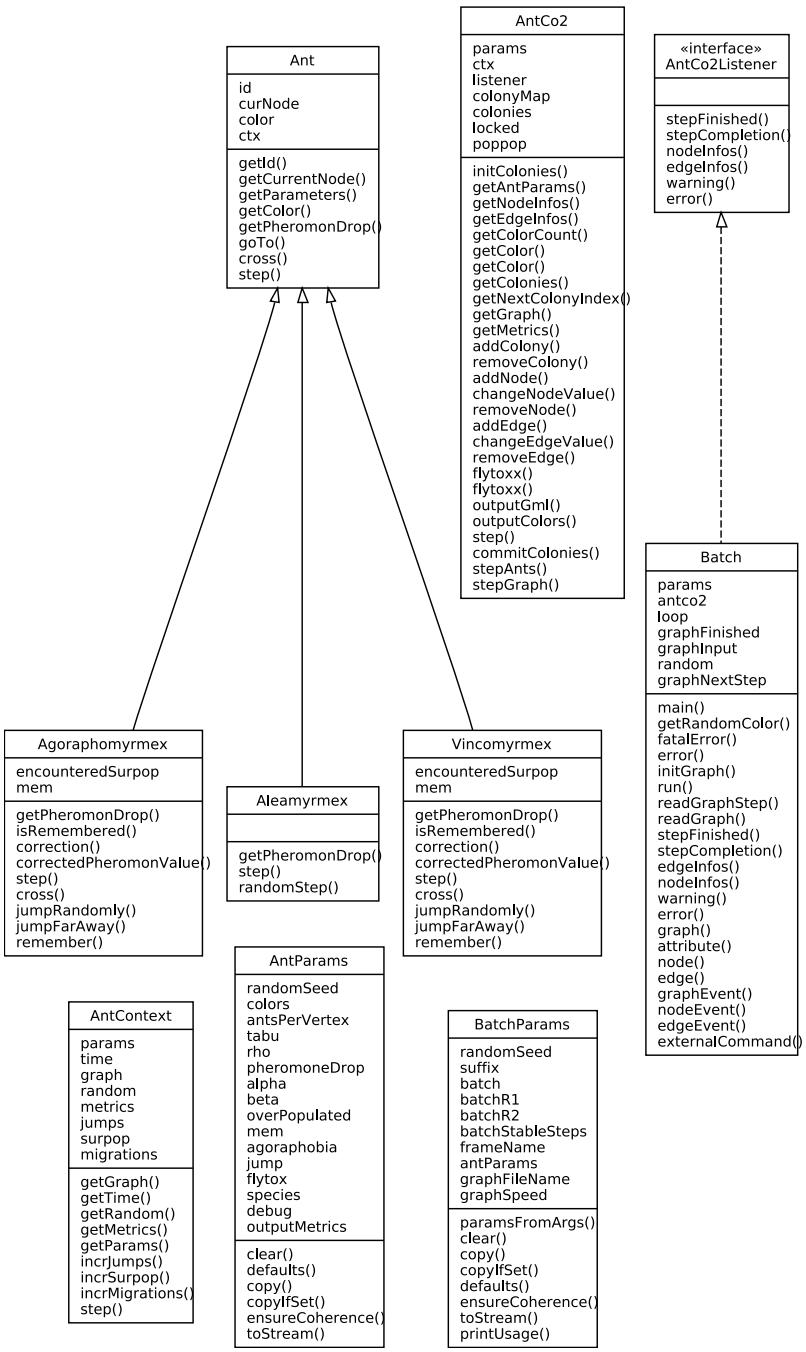


FIG. 13.4: Diagramme UML de la partie logicielle implémentant le modèle d'*AntCO*^{2} (B).

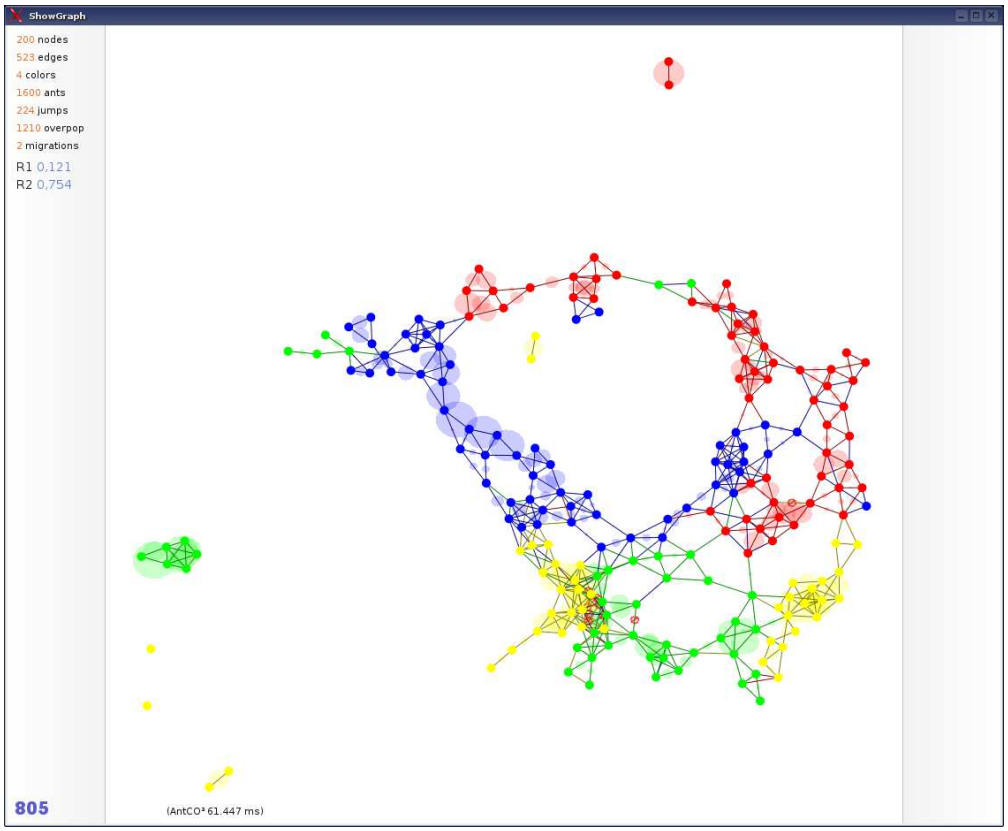


FIG. 13.5: Capture d'écran de la seconde implantation du modèle sur une application de boids.

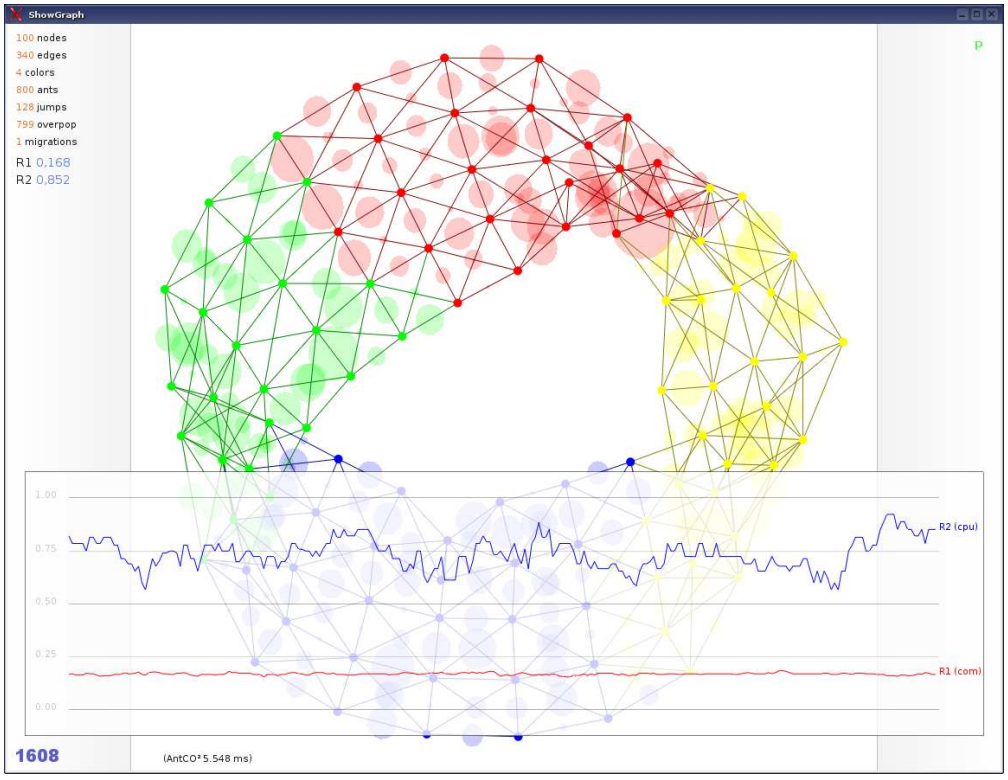


FIG. 13.6: Capture d'écran de la seconde implantation du modèle sur le graphe «mœbius».

Chapitre 14

Modifications du modèle

Sommaire

Ce chapitre expose diverses modifications du modèle d'*AntCO*^{2} afin de l'adapter à des problèmes voisins. Certains de ces problèmes ont été en partie abordés durant le déroulement de cette thèse, d'autres sont des pistes pour la poursuite de ce travail ou simplement des voies qu'il serait intéressant d'explorer.

14.1 Système d'aide à la décision pour le trafic routier

L'intelligence collective et les algorithmes fourmis ont déjà été exploités à propos du routage dans les réseaux de télécommunications [Di Caro, 1997, White, 1997]. Un modèle très proche consiste à concevoir un système d'aide à la décision dans un autre domaine du routage : le trafic sur un réseau routier. Durant cette thèse, des travaux ont été entrepris dans cette direction, qui on donné lieu à deux publications [Lerebourg, 2003, Bertelle, 2003b].

On distingue deux objectifs pour la gestion du trafic routier :

– permettre à des utilisateurs du réseau routier de choisir le chemin le plus court vers une destination arbitraire à partir de n'importe quelle position ;
– réguler le trafic afin d'éviter les embouteillages.

Ces deux objectifs sont en relation l'un avec l'autre, l'amélioration du second pouvant améliorer le premier, et l'utilisation du premier pouvant améliorer le second...

Le modèle présenté dans [Bertelle, 2003b] utilise plusieurs éléments afin de réguler le trafic :

– un système de collecte d'informations sur le trafic utilisées à l'étape de recherche des meilleurs chemins ;

– un graphe dynamique pondéré et orienté représentant le réseau routier et les informations sur le trafic ;
– un système de régulation s'appuyant sur ce graphe et gérant le système de contrôle ;
– un système d'aide à la décision, utilisant le graphe dynamique et le système de contrôle. Une interface multimodale informe et aide les utilisateurs en fonction de leur profil.

La figure 14.1 synthétise l'architecture globale du modèle présenté. On y voit les deux parties principales interagissant entre-elles : le *système de contrôle* et le *système d'aide à la décision*. Les informations circulent entre les deux, allant du premier au second, puis du second au premier en une boucle de rétroaction typique des systèmes complexes.

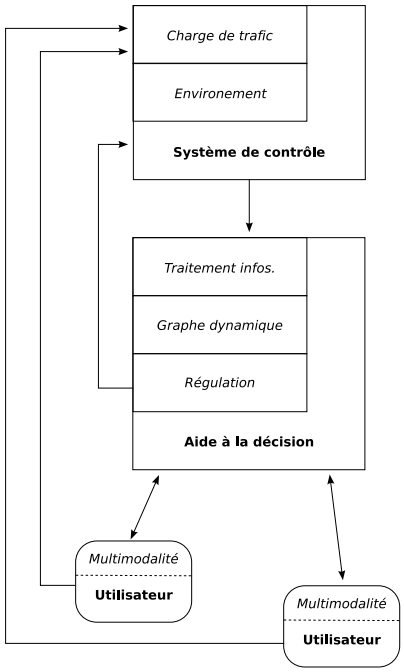


FIG. 14.1: Architecture globale pour la gestion de trafic routier.

Cependant, nous ne présenterons ici que l'un des points de ce modèle, le système d'aide à la décision, car il est basé sur un algorithme fourmi. Le système de régulation utilise algorithme de perceptron multicouche avec rétro-propagation qui est décrit en détail dans [Bertelle, 2003b].

14.1.1 Principe

Le modèle présenté figure la cartographie routière par un graphe orienté et pondéré $G = (\mathcal{V}, \mathcal{E})$, où $\mathcal{V}$ est l'ensemble des sommets et $\mathcal{E}$ l'ensemble des arcs. Chaque sommet $v \in \mathcal{V}$ représente une intersection ou un lieu important et chaque arc $e = (u, v) \in \mathcal{E}$ représente une route entre ces intersections. Les arcs sont valués en fonction du temps nécessaire pour traverser l'arc.

Le graphe est dynamique. Les valuations sont mise-à-jour en fonction du trafic observé par un système de collecte d'information. Ainsi un arc qui pouvait être traversé rapidement peut soudainement devenir quasiment impraticable en raison d'un embouteillage. Cette dynamique implique l'utilisation d'un algorithme robuste et adaptatif face aux reconfigurations constantes de l'environnement. Tout comme pour la distribution dynamique adaptative, la métaphore des fourmis s'adapte particulièrement bien à ce problème.

14.1.2 L'algorithme

Le but du système d'aide à la décision est de fournir à un utilisateur le meilleur chemin d'un point de départ vers un point d'arrivée. La méthode de résolution utilise des mécanismes d'auto-organisation et est distribuée.

Des fourmis numériques, qui constamment déposées dans le graphe dynamique G, ont pour tâche de trouver les meilleurs chemins entre deux sommets v_0 et v_n. Les fourmis ne déposent de la phéromone que lors de leur retour, c'est-à-dire après avoir trouvé un chemin entre v_0 et v_n. Au retour, elles suivent bien entendu le chemin exact qu'elles ont suivi à l'aller. La quantité de phéromone déposée par chaque fourmi dépend de la qualité du chemin trouvé (congestion, longueur). Les fourmis sont attirées par les chemins les plus rapides en fonction de leur pondération, ainsi que par les phéromones.

L'algorithme est découpé en deux parties :

L'environnement Représenté par le graphe dynamique G, a pour tâche de gérer la population de fourmis, l'évaporation, ainsi que la simulation ou la répercution des changements dans le trafic ou sur le réseau routier sur les poids et les arcs de G.

De plus, le sommet v_n contient le meilleur temps qu'une fourmi à mis pour passer de v_1 à v_n, noté W_{0n} ainsi que le meilleur chemin. En raison de la dynamique du graphe, il est possible que le meilleur chemin deviennent moins bon qu'un autre, et on note t_{0n} l'instant où le chemin correspondant à W_{0n} a été trouvé ou mis à jour. L'algorithme 14.1 page suivante représente ces actions, avec l'évaporation $(1 - \rho)$ mise à jour à chaque itération de l'environnement :

$$q_{ij}^{new} = \rho q_{ij}^{old} \tag{14.1}$$

Où $\rho \in]0, 1]$, et q_{ij}^{old} et q_{ij}^{new} sont respectivement le taux de phéromone sur l'arc entre les sommets v_i et v_j avant et après la mise-à-jour.

Les fourmis Elles se déplacent dans le graphe dynamique G, essayant d'aller de v_0 à v_n. Trois états sont possibles pour une fourmi :

1. *recherche de nourriture*,
2. *sur le sommet* v_n, et
3. *retour à la source*.

L'algorithme 14.2 page 185 décrit le comportement local d'une fourmi située sur un sommet v_i, avec :

– q_{ij} la quantité de phéromone déposée sur l'arc connectant les sommets v_i et v_j ;

– w_{ij} le pondération de l'arc entre les sommets v_i et v_j, qui est une valeur dynamique dépendant du trafic sur l'axe routier correspondant ;
– p_{ij} la probabilité qu'une fourmi située sur le sommet v_i se déplace jusqu'au sommet v_j, exprimée par :

$$p_{ij} = \frac{(q_{ij})^{\alpha} \left(\frac{1}{w_{ij}}\right)^{\beta}}{\sum_{k \in A_i} (q_{ik})^{\alpha} \left(\frac{1}{W_{ik}}\right)^{\beta}} \tag{14.2}$$

– A_i est l'ensemble des sommets adjacents à v_i qui n'ont pas encore été traversés par la fourmi ;

Une fourmi qui a trouvé un chemin entre v_0 et v_n, revient en suivant exactement le même chemin sur v_0 en déposant une quantité Δq de phéromone sur chaque arc qu'elle traverse avec :

$$\Delta q = \frac{K}{W_{ij}} \tag{14.3}$$

où K est une constante et W_{ij} le coût global du chemin entre v_i et v_j. La quantité de phéromone sur l'arc entre v_i et v_j est donc :

$$q_{ij}^{new} = q_{ij}^{old} + \Delta q \tag{14.4}$$

14.1 : Environnement des fourmis routières.

$t_{env} =$ temps discret pour l'environnement.;
pour *toujours* **faire**
  Naissance de fourmis sur le sommet v_0.;
  Évaporation des phéromones (eq. 14.1 page précédente).;
  Mise à jour des poids en fonction du trafic.;
  si $t_{0n} << t_{env}$ **alors**
    $W_{0n} \leftarrow +\infty$;
  $t_{env} \leftarrow t_{env} + 1$;

14.2 : Comportement d'une fourmi routière.

$ant\_state \in \{search, arrived, go\_back\}$;
t_{ant} = temps discret pour les fourmis.;
$vertex \leftarrow v_0$;
$ant\_state \leftarrow search$;
$vie \leftarrow vrai$;
tant que vie **faire**
  if $ant\_state = search$ **then**
    La fourmi doit choisir un sommet adjacent à i.
    A_i ensemble des sommet adjacents à i non déjà traversés par la fourmi.;
    pour $j \in A_i$ **faire**
      $p_{ij} \leftarrow$ probabilité que la fourmi choisisse d'aller de i vers j (eq. 14.2 page ci-contre).;
    Sélectionner le prochain sommet v_k en fonction de la probabilité p_{ij};
    Attendre $w_{ik} - 1$ unités de temps.;
    $vertex \leftarrow v_{ik}$;
    si $vertex = v_n$ **alors**
      $ant\_state \leftarrow arrived$;
  if $ant\_state = arrived$ **then**
    Mettre à jour si besoin est le plus court chemin et le temps t_{0n}.;
    $ant\_state \leftarrow go\_back$;
  if $ant\_state = go\_back$ **then**
    Déposer des phéromones sur le chemin trouvé (eq. 14.4 page précédente);
    $vie \leftarrow faux$;

14.1.3 Premières expérimentations

Les figures 14.2 page 187, 14.3 page 188 et 14.4 page 189 montrent quelques essais préliminaires sur différents styles de graphes routiers. La figure 14.2 a servi de test sur la capacité des fourmis à détecter un changement dans le meilleur chemin entre le sommet A et le sommet F. Comme le montrent les figures 14.2(b) et 14.2(c), la pondération de l'arc (K,J) est changée durant le calcul forçant les fourmis à trouver un autre chemin. Les paramètres utilisés pour cette expérience sont $\alpha = 3$, $\beta = 1$, $\rho = 0.6$ et $K = 1$.

Les figures 14.3 et 14.4 représentent des villes entourées d'un périphérique. En temps normal, utiliser le périphérique est moins coûteux (figure 14.3(b) et 14.4(b)), cependant, en modifiant les poids du périphériques pour simuler un embouteillage (figure 14.3(c) et 14.4(c)), on constate que les fourmis sont capables de trouver un chemin moins coûteux en passant par le centre ville. Pour ces expériences, les paramètres utilisés sont $\alpha = 3$, $\beta = 1$, $\rho = 0.9$ et $K = 2$.

14.2 Recherche de communautés dans les réseaux

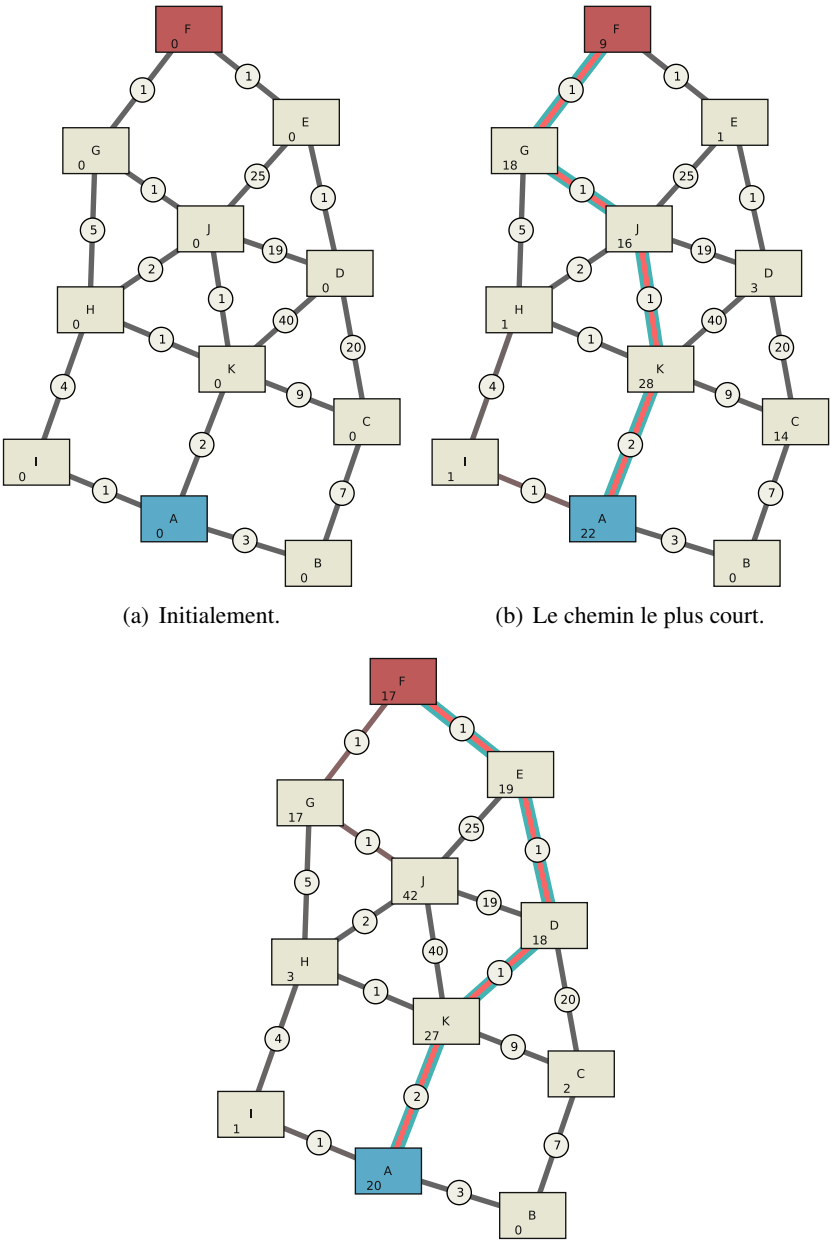
Le modèle que nous avons proposé, *AntCO*^{2}, détecte les organisations dans un réseau d'interaction pour mieux distribuer la charge machine et réseau. Intuitivement, passer à la recherche de communautés dans les réseaux d'interaction consisterait à désactiver la partie distribution (par exemple les contraintes de répartition de charge qui coupent les organisations) d'*AntCO*^{2} pour ne garder que la recherche d'organisations.

En effet, on peut facilement identifier notre algorithme «fourmi» à l'un des algorithmes proposé par M.E.J. Newman dans [Newman, 2004]. Dans cet article, l'auteur présente plusieurs méthodes de recherche de communautés dont une est basée sur une marche aléatoire dans le graphe. Cette marche considère un nombre important de parcours du graphe sans cycle, c'est-à-dire qu'un parcours ne passe jamais deux fois par le même arc. Chaque arc contient un compteur qui permet de savoir combien de parcours le traversent. Lorsqu'un nombre suffisant de parcours ont été effectués, les arcs ayant un compteur de passage les plus élevés sont éliminés. Le but consiste donc à trouver des points d'articulation du graphe et les couper pour isoler les communautés.

Nous avons appliqué ce principe à nos fourmis. Ces dernières réalisent une marche dans le graphe et déposant des phéromones sur les arcs qui sont en quantité d'autant plus importante qu'un nombre élevé de fourmis à traversé un arc.

Plusieurs expériences ont été faites avec le modèle de comportement des fourmis décrit dans la partie III page 113, à la différence près que ces dernières ne sont plus influencées par la phéromone et la pression démographique. Cependant, elles continuent à déposer de la phéromone. À la différence du modèle proposé par Newman, les fourmis ont une mémoire limitée, et se déplacent en continu. Elle utilisent des phéromones colorées pour marquer les arcs, ce qui permet d'accentuer le mécanisme de découverte de communautés. Ici les arcs à éliminer sont les arcs les moins chargés en phéromone.

La figure 14.5 page 190 montre le graphe invariant d'échelle que nous avons déjà utilisé précédemment, colorié par des fourmis utilisant le mécanisme décrit ci-dessus. Ici les clusters sont de toutes tailles puisqu'il n'y a plus de contraintes d'égalisation de la charge. Les communications sont par contre toujours privilégiées, ce qui permet la détection des clusters de forte interaction.

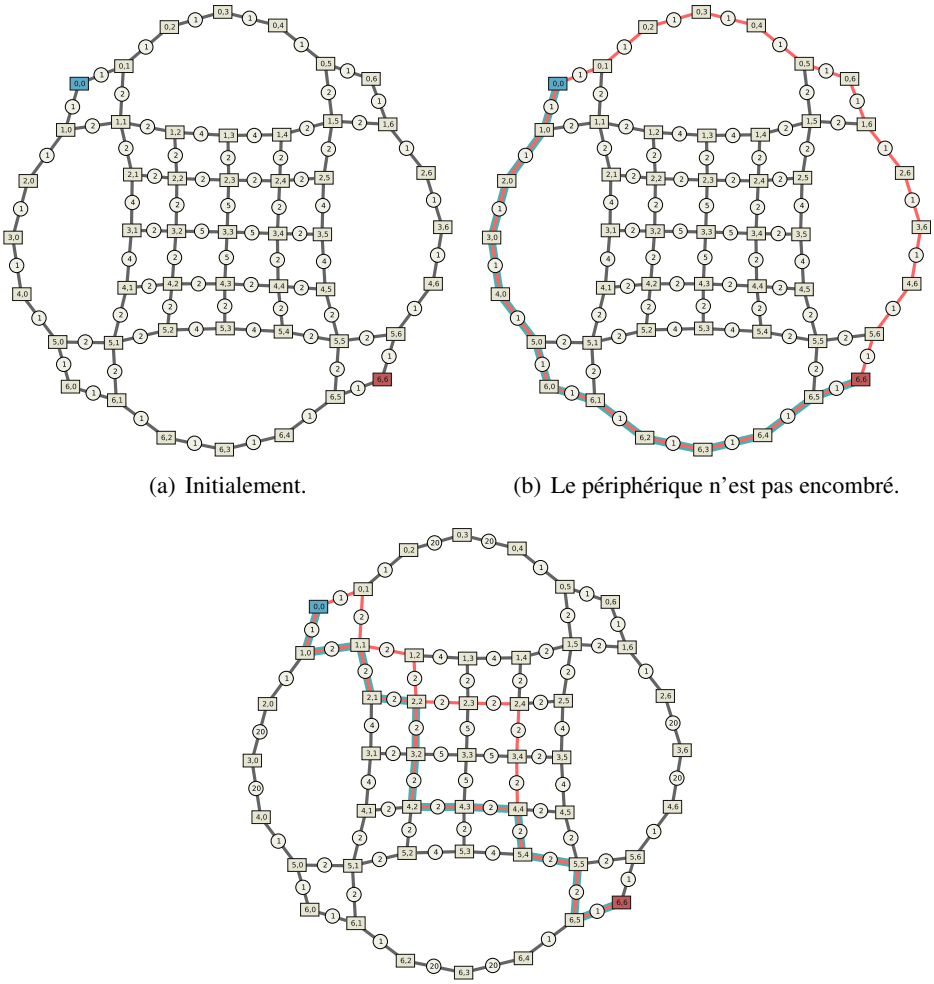


(a) Initialement.

(b) Le chemin le plus court.

(c) Le chemin le plus court disparaît, un autre est déterminé.

FIG. 14.2: Une ville et son périphérique.

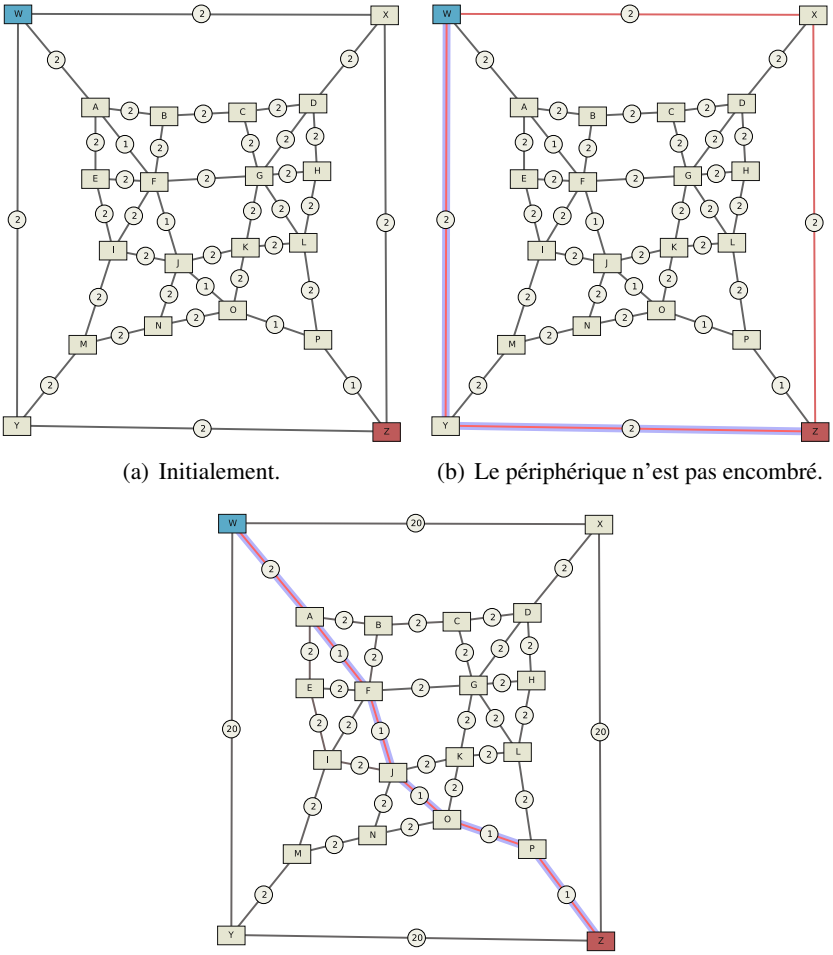


(a) Initialement.

(b) Le périphérique n'est pas encombré.

(c) Le périphérique devient encombré, passer par le centre ville est plus rapide.

FIG. 14.3: Une ville et son périphérique.



(a) Initialement.

(b) Le périphérique n'est pas encombré.

(c) Le périphérique devient encombré, passer par le centre ville est plus rapide.

FIG. 14.4: Une ville et son périphérique.

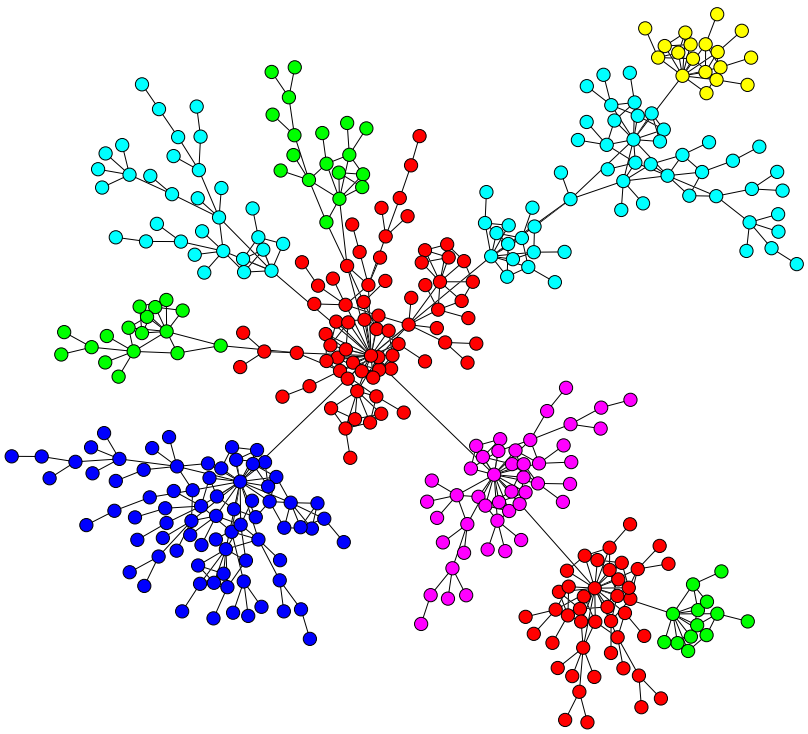


FIG. 14.5: Un graphe invariant d'échelle de 391 sommets et 591 arcs coloré pour trouver les communautés.

Bilan et perspectives

Nous avons présenté une approche utilisant des mécanismes d'intelligence collective pour résoudre un problème de distribution d'applications réparties constituées de multiples entités en interaction de manière dynamique. La méthode repose sur la détection d'organisations au sein du réseau d'interaction de l'application, modélisé par un graphe. Dans son principe elle repose sur l'interaction de fourmis numériques colorées utilisant les mécanismes de collaboration et de compétition.

Cette approche vient compléter l'ensemble des techniques et modèles déjà proposés dans le domaine de la distribution dynamique. Elle est destinée à gérer :

- la dynamique de l'environnement d'exécution ;
- la dynamique de l'application ;
- l'égalisation de la charge machine ;
- la minimisation des communications ;
- les organisations au sein du réseau d'interactions de l'application.

Le modèle développé est implicitement distribué car il ne repose que sur des décisions locales. Plusieurs expérimentations ont été menés sur ce dernier montrant sa capacité à distribuer la charge et minimiser les communications dans des graphes d'applications aussi bien statiques que dynamiques. Sur ce dernier point, le modèle à été mis en œuvre dans deux implantation dont une distribuable, couplée à une simulation d'écosystème marin qui a permis de le comparer à deux autres politiques de distribution de manière favorable.

La méthode est adaptative, non pas au niveau des individus, ces derniers étant réactifs, mais dans son ensemble. C'est cette propriété qui lui permet de calculer en continu la distribution sur un graphe en perpetuelle reconfiguration. Les rétroactions positives maintiennent les chemins entre les sommets fortement corrélés, tandis que les rétroactions négatives isolent ces communautés. Les premières sont controlées par les fourmis (dépots de phéromones), les secondes par l'environnement (évaporation, disparition d'arcs, pondérations). Ce sont les rétroactions négatives qui rendent l'algorithme adaptatif en permettant l'oubli des communautés invalides introduites par la dynamique.

Ce travail ouvre de nombreuses pistes. Dans un premier temps il serait intéressant d'explorer plus avant certaines modifications du modèle pour d'autres tâches, en particulier lorsqu'il est nécessaire de détecter des organisations. La recherche de communautés dans les réseaux en est un exemple. D'autres possibilités sont ouvertes aussi sur l'amélioration du modèle, en particulier pour le réglage automatique des paramètres, ainsi que sur la création d'une couche d'observation qui se situerait entre l'application et $AntCO^2$.

La vue que nous avons présentée de ce travail est celle d'un service au sein d'un

environnement d'exécution réparti, utilisant un middleware. Cependant, pour terminer ce travail, il est intéressant d'en présenter une autre approche, telle qu'on pourrait l'observer avec un macroscope [de Rosnay, 1975]. Commençons par une définition :

> **Écosystème** :
> Ensemble structuré comprenant un substrat et la biocénose qui y vit. Cet ensemble est caractérisé par des échanges de matière et d'énergie dus aux interactions entre les organismes vivants et le substrat.

Et considérons maintenant la définition que nous pourrions avoir d'une simulation telle que nous les avons considérées jusqu'à présent :

> **Simulation** :
> Ensemble structuré comprenant un environnement d'exécution en éventuelle évolution, et les composants logiciels qui s'y exécutent. Cet ensemble est caractérisé par des échanges d'information dus aux interactions entre les composants du système.

Les applications que nous avons décrit présentent de grandes similitudes avec la vue que l'on a d'un écosystème. Ainsi, quelles sont les propriétés d'un écosystème ?

- Il y a auto-organisation de ses constituants et rétroaction du système sur lui-même ;
- L'écosystème et ses sous-parties passent par des phases juvéniles, adultes et sénescentes ;
- L'écosystème est un système complexe, dans le sens où le tout est plus que la somme de ses parties. En d'autres termes, décrire les composants à part n'est pas suffisant.
- Les interactions en grand nombre au sein du système sont primordiales.
- Le système est adaptatif dans son ensemble. Cependant sa capacité d'adaptation est limitée, si on perturbe trop son état, ce dernier s'effondre (on peut citer par exemple les phénomènes d'eutrophisation comme la nitrification).
- Le système possède une frontière perméable, et des flots de matière, d'énergie et d'information le traversent et le structurent.

Les parallèles avec un système informatique sont nombreux :

- Dans une simulation faite d'un nombre important d'entités en interaction, il y a des mécanismes d'auto-organisation.
- Des organisations émergent, à divers niveaux d'échelle, et ces dernières passent par les phases juvéniles, adultes, et sénescentes.
- Les interactions en grand nombre au sein du système sont primordiales.
- La simulation utilisant ces mécanismes est plus que la somme de ses parties. Les composants ne sont pas explicitement programmés pour produire le résultat global. Ceci est vrai aussi pour *AntCO*^{2}.
- Le système s'adapte à ses entrées dans son ensemble, même s'il est constitué de parties réactives. Cependant, sa capacité d'adaptation est limitée, trop de changements le font s'effondrer.
- Le système possède une frontière perméable, et des flots de matière, d'énergie et d'information le traversent et le structurent.

Pour désigner un tel système on peut utiliser le terme «écosystème computationel» proposé par A. B. Huberman et T. Hoggs dans [Huberman, 1988], cependant avec une acception différente.

Un tel écosystème computationel, tout comme sa contrepartie naturelle possède plusieurs niveaux d'échelle. Considérons une simulation couplée à $AntCO^2$. L'ensemble des deux dans l'environnement d'exécution forme l'écosystème computationel complet. Chacun est un sous-système avec ses entrées et sorties bouclées l'une sur l'autre. Ces deux systèmes rétroagissent mutuellement en s'observant. Au sein de chacun, de nombreuses entités en interactions forment des organisations (entités de l'application, fourmis), elles mêmes délimitées par une frontière, *etc*. juqu'au niveau des entités elles-mêmes.

De même, on peut prendre l'exemple d'un écosystème estuarien, formant un tout, composé de sous systèmes (populations diverses, faune, flore, ou a divers niveaux dans la chaîne alimentaire, poissons, plancton). Au sein de ces sous-systèmes on trouve des organisations, en fonction des espèces, et des modes d'interaction, ceci jusqu'au individus eux-mêmes.

Index des notations

Voici un récapitulatif des diverses notations utilisées dans ce document, en particuler à la partie III page 113.

Notation	*Signification*
$G(t) = (\mathcal{V}(t), \mathcal{E}(t))$	Graphe d'interactions dynamique au temps t.
$G(t) = (\mathcal{V}(t), \mathcal{E}(t), \mathcal{C}(t))$	Graphe d'interactions dynamique coloré au temps t.
$u, v \in \mathcal{V}(t)$	Sommets.
$e = (u, v) \in \mathcal{E}(t)$	Un arc.
$c \in \mathcal{C}(t)$	Une couleur.
$n \in \mathbb{N}^+$	Nombre de sommets.
$p \in \mathbb{N}^+$	Nombre de couleurs / ressources de calcul.
$t \in \mathbb{N}^+$	Temps.
x	Une fourmi.
$N(v) \in \mathbb{N}$	Nombre de fourmis sur le sommet v.
$w^{(t)}(e) \in \mathbb{R}$	Poids d'un arc e au temps t.
$\mathcal{V}(t)$	Ensemble de sommets au temps t.
$\mathcal{V}_c(t)$	Ensemble de sommets de couleur c au temps t.
$\mathcal{E}(t)$	Ensemble d'arcs au temps t.
$\mathcal{E}_u(t)$	Ensemble d'arcs adjacents au sommet u au temps t.
$\mathcal{A}(t)$	Ensemble d'arcs de communication effectives au temps t.
$\mathcal{C}(t)$	Ensemble de couleurs au temps t.
$\mathcal{B}(t)$	Edge-cut au temps t.
$\mathcal{D}(t)$	Partition de $G(t)$ au temps t.
$\mathcal{D}_i(t)$	Domaines formant la partition $\mathcal{D}(t)$.
$\mathcal{F}(t)$	Population de fourmis au temps t.
$\mathcal{F}_c(t)$	Population de fourmis de couleur c au temps t.
$\Delta_x^{(t)}(e, c)$	Quantité de phéromone de couleur c déposée par la fourmi x sur l'arc e durant l'intervalle de temps $]t-1, t]$.
$\Delta^{(t)}(e, c)$	Quantité de phéromone de couleur c déposée par toutes les fourmis de couleur c sur l'arc e durant l'intervalle de temps $]t-1, t]$.
$\Delta^{(t)}(e)$	Quantité de phéromone déposée par toutes les fourmis sur l'arc e durant l'intervalle de temps $]t-1, t]$.

Notation	*Signification*
$K_c^{(t)}(e)$	Taux de phéromone de couleur c sur l'arc e au temps t.
$\tau^{(t)}(e,c)$	Quantité totale de phéromone de couleur c présente sur l'arc e au temps t.
$\tau^{(t)}(e)$	Quantité totale de phéromone présente sur l'arc e au temps t. t.
$\Omega^{(t)}(e,c)$	Facteur de renforcement corrigé.
$\xi^{(t)}(u)$	Couleur du sommet u au temps t.
$p^{(t)}(e,c)$	Probabilité qu'une fourmi de couleur c traverse l'arc e au temps t.
α	Importance des phéromones.
β	Importance des poids.
$\rho \in]0,1]$	Facteur de persistance des phéromones.
$\gamma(v)$	Pression démographique sur le sommet v.
N^*	Seuil de pression démographique.
W_x	Liste tabou pour la fourmi x.
M	Taille de la liste tabou.
$\eta_x(u)$	Facteur de pénalisation si la fourmi x visite à nouveau le sommet u.
ϕ	Seuil de fuite.
r_1	Proportion de communications effectives.
r_2	Taille de la partition minimale sur la taille de la partition maximale.

Bibliographie

[Albert, 2002] Albert, R. and Barabasí, A. "Statistical mechanics of complex networks". *Reviews of modern physics*, Vol. 74, p. 47–97 (2002).

[Amdahl, 1967] Amdahl, G. "Validity of the single processor approach to achieving large-scale computing capabilities". Proc. of the *AFIPS Conference*, pages 483–485 (1967).

[Azzag, 2003] Azzag, H., Monmarché, N., Slimane, M., Guinot, C., and Venturini, G. "Anttree : a new model for clustering with artificial ants". Proc. of the *7th European Conference on Artificial Life (ECAL 2003)*, pages 14–17, Dortmund, Germany (2003).

[Barabasi, 2002] Barabasi, A.-L. "Linked : How everything is connected to everything else and what it means". *Plume ed.* (2002).

[Beni, 1989] Beni, G. and Wang, J. "Swarm intelligence in cellular robotic systems". Proc. of the *NATO Advanced Workshop on Robots and Biological Systems*, Il Ciocco, Tuscany, Italy (1989).

[Bertalanffy, 1968] Bertalanffy, L. "Théorie générale des systèmes". *Dunod ed.* (1968).

[Boillat, 1990] Boillat, J. E. "Load balancing and poisson equation in a graph". *Concurrency : Practice and Experience*, Vol. 2, n°. 4, p. 289–313 (1990).

[Bonabeau, 1999] Bonabeau, E., Dorigo, M., and Théraulaz, G. "Swarm intelligence — from natural to artificial systems". *Oxford University Press ed.* (1999).

[Bullnheimer, 1999] Bullnheimer, B. "Ant colony optimization in vehicle routing". *PhD thesis*, University of Vienna (1999).

[Cardon, 2005] Cardon, A. "La complexité organisée — systèmes adaptatifs et champ organisationnel". *Hermes Lavoisier ed.* (2005).

[Costa, 1997] Costa, D. and Hertz, A. "Ants can colour graphs". *Journal of Operation Research Society*, Vol. 3, n°. 48, p. 295–305 (1997).

[Cougny, 1994] Cougny, H. L. D., Devine, K. D., Flaherty, J. E., Loy, R. M., Özturan, C., and Shepard, M. S. "Load balancing for the parallel adaptive solution of partial differential equations". *Applied Numerical Mathematics*, Vol. 16, p. 157–182 (1994).

[Cramer, 1985] Cramer, N. "A representation for the adaptive generation of simple sequential programs". Proc. of the Morgan-Kaufmann, editor, ICGA, pages 183–187 (1985).

[Cybenko, 1989] Cybenko, G. "Dynamic load balancing for distributed memory multi-processors". *J. Parallel Distrib*, Vol. 7, p. 279–301 (1989).

[de Rosnay, 1975] de Rosnay, J. "Le macroscope, vers une vision globale". *Seuil ed.* (1975).

[Deneubourg, 1989] Deneubourg, J.-L. and Goss, S. "Collective patterns and decision making". *Ethology Ecology and Evolution*, Vol. 1, n°. 4, p. 295–311 (1989).

[Di Caro, 1997] Di Caro, G. and Dorigo, M. "Antnet : A mobile agents approach to adaptive routing". *Technical report*, IRIDIA, Université libre de Bruxelles, Belgium (1997).

[Diekmann, 1997] Diekmann, R., Muthukrishnan, S., and Nayakkankuppan, V. "Enginneering diffusive load balancing algorithms using experiments". Proc. of the *Lecture Notes in Computer Science, 1253*, pages 111–122 (1997).

[Dorigo, 1992] Dorigo, M. "Optimization, learning and natural algorithms". *PhD thesis*, Department of Electronics Politecnico di Milano, Italy. In italian (1992).

[Dorigo, 1997] Dorigo, M. and Gambardella, L. "Has-sop : An hybrid ant system for the sequential ordering problem". *Technical report*, IDSIA, Lugano, Switzerland (1997).

[Dorigo, 1996] Dorigo, M., Maniezzo, V., and Colorni, A. "The ant system : optimization by a colony of cooperating agents". *IEEE Trans. Systems Man Cybernet.*, Vol. 26, p. 29–41 (1996).

[Erdös, 1959] Erdös, P. and Rényi, A. "On random graphs". *Pubiones Mathematicaelicat*, Vol. 6, p. 290–297 (1959).

[Ferencz, 2005] Ferencz, A., Szewczyk, R., Weinstein, J., and Wilkening, J. "Bisection". *http ://http ://www.cs.berkeley.edu/ szewczyk/cs270/.* consulté le 9/05 (2005).

[Fiduccia, 1982] Fiduccia, C. M. and Mattheyses, R. M. "A linear time heuristic for improving network partitions". Proc. of the *ACM IEEE Design Automation Conference*, pages 175–181 (1982).

[Flaherty, 1998] Flaherty, J. E., Loy, R. M., Özturan, C., Szymanski, B. K., Teresco, J. D., and Ziantz, L. H. "Parallel structures and dynamic load balancing for adaptive finite element computation". *Applied Numerical Mathematics*, Vol. 26, p. 241–263 (1998).

[Flaherty, 1997] Flaherty, J. E., Loy, R. M., Shepard, M. S., Szymanski, B. K., Teresco, J. D., and Ziantz, L. H. "Adaptive local refinement with octree load balancing for the parallel solution of three-dimentional conservation laws". *Journal of Parallel and Distributed Computing*, Vol. 47, p. 139–152 (1997).

[Forsyth, 1997] Forsyth, P. and Wren, A. "An ant system for bus driver scheduling". Proc. of the *7th International Workshop on Computer-Aided Scheduling of Public Transport*, Boston (1997).

[Gambardella, 1999] Gambardella, L. M., Taillard, E., and Dorigo, M. "Ant colonies for the quadratic assignment problem". *Journal of the Operational Research Society*, Vol. 50, p. 167–176 (1999).

[Garey, 1979] Garey, M. R. and Johnson, D. S. "Computers and intractability". *Freeman ed.*, San Francisco, CA (1979).

[Ghosh, 1994] Ghosh, B. and Muthukrishnan, S. "Dynamic load balancing on parallel and distributed networks by random matchings". Proc. of the *Sixth Annual ACM Symposium on parallel Algorithms and Architectures*, pages 226–235 (1994).

[Ghosh, 1998] Ghosh, B., Muthukrishnan, S., and Schultz, H. "First- and second-order diffusive methods for rapid, coarse, distributed load balancing". *Theory of Computing Systems*, Vol. 31, p. 331–354 (1998).

[Goss, 1989] Goss, S., Aron, S., Deneubourg, J.-L., and Pasteels, J. M. "Self-organized shortcuts in the argentine ant". *Naturwissenchaften*, Vol. 76, p. 579–581 (1989).

[Goss, 1990] Goss, S., Aron, S., Deneubourg, J.-L., and Pasteels, J. M. "The self-organizing exploratory pattern of the argentine ant". *Insect Behavior*, Vol. 3, p. 159–168 (1990).

[Heirich, 1995] Heirich, A. and Tayler, S. "A parabolic load blancing method". Proc. of the *International Conference on Parallel Processing* (1995).

[Heiss, 1995] Heiss, H.-U. and Schmitz, M. "Decentralized dynamic load balancing : The particles approach". *Information Sciences*, Vol. 84, p. 115–128 (1995).

[Hendrickson, 1995] Hendrickson, B. and Leland, R. "An improved spectral graph partitioning algorithm for mapping parallel computations". *SIAM J. Scien. Comput.*, Vol. 16, n°. 2, p. 452–469 (1995).

[Heppner, 1990] Heppner, F. and Grenader, U. "The ubiquity of chaos", chapter A Stochastic Nonlinear Model for Coordinated Bird Flocks. *Krasner*, aaas publications edition (1990).

[Holland, 1975] Holland, J. H. "Adaptation in natural and artificial systems : An introductory analysis with applications to biology, control, and artificial intelligence". *University of Michigan Press (second edition : MIT Press 1992) ed.*, Ann Arbor, MI (1975).

[Horton, 1993] Horton, G. "A multi-level diffusion method for dynamic load balancing". *Parallel Computing*, Vol. 9, p. 209–218 (1993).

[Hu, 1999] Hu, Y. "An improved diffusion algorithm for dynamic load balancing". *Parallel Computing*, Vol. 25, p. 417–444 (1999).

[Huberman, 1988] Huberman, B. and Hogg, T. "The behavior of computational ecologies". Proc. of the Huberman, B., editor, The Ecology of Computation, pages 77–115. Elsevier Science, North-Holland, Amsterdam (1988).

[Karypis, 1999] Karypis, G. and Kumar, V. "A fast and high quality multilevel scheme for partitioning irregular graphs". *SIAM J. Scien. Comput.*, Vol. 20, n°. 1 (1999).

[Kennedy, 1995] Kennedy, J. and Eberhart, R. "Particle swarm optimization". Proc. of the *IEEE International Conference on Neural Networks*, volume IV, pages 1942–1948, Piscataway, NJ (1995).

[Kennedy, 2001] Kennedy, J. and Eberhart, R. "Swarm intelligence". *Academic Press ed.* (2001).

[Kernighan, 1970] Kernighan, B. and Lin, S. "An efficient heuristic procedure for partitioning graph". *The Bell System Technical Journal*, Vol. 49, n°. 2, p. 192–307 (1970).

[Koza, 1992] Koza, J. "Genetic programming". *MIT Press ed.* (1992).

[Kuntz, 1997] Kuntz, P., Layzell, P., and Snyers, D. "A colony of ant-like agents for partitioning in VLSI technology". Proc. of the *Fourth European Conference on Artificial Life*, pages 417–424, Cambridge, MA :MIT Press (1997).

[Kuntz, 1994] Kuntz, P. and Snyers, D. "Emergent colonization and graph partitionning". Proc. of the Cliff, D., Husbands, P., Meyer, J., and Wilson, S., editors, From animals to animats 3 (SAB 94 Int. Conf.), pages 494–500. MIT Press (1994).

[Landau, 2003] Landau, S. "Des AG vers la GA – Sélection Darwinienne et Systèmes Multi-Agents". *PhD thesis*, Université Paris VI, Paris (2003).

[Le Moigne, 1990] Le Moigne, J. "La modélisation des systèmes complexes". *Dunod ed.* (1990).

[Lin, 1992] Lin, H. and Raghavendra, C. "A dynamic load-balancing policy with a central job dispatcher". *IEEE Trans. Software Engrg.*, Vol. 18, n°. 2, p. 148–158 (1992).

[Luque, 1995] Luque, E., Ripoll, A., Cortés, A., and Margalef, T. "A distributed diffusion method for dynamic load balancing on parallel computers". Proc. of the *PDP 1995*, pages 43–50 (1995).

[Maini, 1994] Maini, H., Mehrotra, K., Mohan, C., and Ranka, S. "Genetic algorithms for graph partitioning and incremental graph partitioning". Proc. of the *1994 conference on Supercomputing*, pages 449–457, Washington, D.C. IEEE Computer Society Press (1994).

[Marin, 2003] Marin, O., Bertier, M., and Sens, P. "DARX - a framework for the fault-tolerant support of agent software". Proc. of the *14th. IEEE International Symposium on Software Reliability Engineering, (ISSRE 03)*, Denver, USA (2003).

[Milgram, 1967] Milgram, S. "The small-world problem". *Psychology Today*, Vol. 1, p. 60–67 (1967).

[Mullender, 1993] Mullender, S. "Distributed systems". *Addison Wesley ed.* (1993).

[Newman, 2003] Newman, M. E. J. "The structure and function of complex networks". *SIAM Review*, Vol. 45, p. 167–256 (2003).

[Newman, 2004] Newman, M. E. J. and Girvan, M. "Finding and evaluating community structure in networks". *Phys. Rev*, Vol. 69 (2004).

[Pothen, 1990] Pothen, A., Simon, H., and Liou, K.-P. "Partitioning sparces matrices with eigenvectors of graphs". *SIAM J. Mat. Anal. Appl.*, Vol. 11, n°. 3, p. 430–452 (1990).

[Rennard, 2002] Rennard, J.-P. "Vie artificielle". *Vuibert ed.* (2002).

[Resnick, 2000] Resnick, M. "Turtles, termites and traffic jams". *MIT Press ed.* (2000).

[Reynolds, 1987] Reynolds, C. W. "Flocks, Herds, and Schools : A Distributed Behavioral Model". *Computer Graphics*, Vol. 21, n°. 4, p. 25–34. SIGGRAPH '87 Conference (1987).

[Soper, 2004] Soper, A. J., Walshaw, C., and Cross, M. "A combined evolutionary search and multilevel optimisation approach to graph partitioning". *Global Optimization, 29(2)*, pages 225–241. originally published as Univ. Greenwich Tech. Rep. 00/IM/58) (2004).

[Soper, 2005] Soper, A. J., Walshaw, C., and Cross, M. "Graph partitioning archive". *http ://staffweb.cms.gre.ac.uk/ c.walshaw/partition/http ://staff-web.cms.gre.ac.uk/ c.walshaw/partition/http ://staff-web.cms.gre.ac.uk/ c.walshaw/partition/.* consulté le 9/05 (2005).

[Teresco, 2005] Teresco, J. D., Devine, K. D., and Flaherty, J. E. "Numerical solution of partial differential equations on parallel computers", chapter Partitioning and Dynamic Load Balancing for the Numerical Solution of Partial Differential Equations. *Springer-Verlag*, are magnus bruaset and petter bjørstad and aslak tveito edition (2005).

[Théraulaz, 1997] Théraulaz, G. and Bonabeau, E. "Auto-organisation et comportement", chapter Auto-Organisation et comportement collectifs : la modélisation des sociétés d'insectes, pages 91–140. *Hermes*, guy théraulaz and françois spitz edition (1997).

[Tranouez, 2005] Tranouez, P. "Contribution à la modélisation et à la prise en compte informatique de niveaux de description multiples". *PhD thesis*, Laboratoire d'Informatique du Havre (LIH), Univesité du Havre (2005).

[Walshaw, 1997] Walshaw, C., Cross, M., and Everett, M. "Parallel dynamic graph-partitioning for unstructured meshes". *J. Parallel Distrib. Comput.*, Vol. 47, n°. 2, p. 102–108 (1997).

[Watts, 1998] Watts, D. J. and Strogatz, S. H. "Collective dynamics of «small-world» networks". *Nature*, Vol. 393, p. 440–442 (1998).

[White, 1997] White, T. "Routing with swarm intelligence". *Technical Report SCE-97-15*, SCE (1997).

Publications

[Bertelle, 2002a] Bertelle, C., Dutot, A., Guinand, F., and Olivier, D. "DIMANTS : a distributed multi-castes ant system for dna sequencing by hybridization". Proc. of the *NETTAB 2002*, pages 1–7, AAMAS 2002 Conf, Bologna (Italy) (2002).

[Bertelle, 2002b] Bertelle, C., Dutot, A., Guinand, F., and Olivier, D. "Distribution of agent based simulation with colored ant algorithm". Proc. of the *ESS2002*, pages 39–43, Dresden (Germany) (2002).

[Bertelle, 2002c] Bertelle, C., Dutot, A., Guinand, F., and Olivier, D. "DNA sequencing hybridation based on multi-castes ant system". Proc. of the *NETTAB Workshop on Agents in Bioinformatics* (2002).

[Bertelle, 2002d] Bertelle, C., Dutot, A., Guinand, F., and Olivier, D. "DNA Sequencing Hydridization based on multi-castes ant system". Proc. of the *BIXMAS 2002 Conference Workshop of AAMAS 2002*, Bologna, Italy (2002).

[Bertelle, 2003a] Bertelle, C., Dutot, A., Guinand, F., and Olivier, D. "Dynamic placement using ants for object based silulations". Proc. of the *DOA, International Symposium on Distributed Objects and Applications*, Catania (Sicily) (2003).

[Bertelle, 2004] Bertelle, C., Dutot, A., Guinand, F., and Olivier, D. "Colored ants for distributed simulations". Proc. of the LNCS, editor, ANTS04, Bruxel. Dorigo et al. (2004).

[Bertelle, 2003b] Bertelle, C., Dutot, A., Lerebourg, S., and Olivier, D. "Road traffic management base on ant system and regulation model". Proc. of the *MAS, Modeling and Applied Simulation, Italy*, Bergigi (Italy) (2003).

[Bertelle, 2002e] Bertelle, C., Dutot, A., Olivier, D., and Prévost, G. "Active objects to develop computer games for blind children". Proc. of the *GAME'ON 2002 Conf on Intelligent Games and Simulation*, pages 178–180, London (UK). SCS (2002).

[Dutot, 2005] Dutot, A., Cardon, A., Olivier, D., and Guinand, F. "Competing ants for organization detection, application to dynamic distribution". Proc. of the *ECCS 05, European conference on complex systems*. à paraître, 17 nov. 2005 (2005).

[Dutot, 2004] Dutot, A., Fisch, R., Olivier, D., and Pigné, Y. "Dynamic distribution of an entity-based simulation". Proc. of the *JICCSE 04 - Jordan International Conference on Computer Science and Engineering*, Alt-Salt, (Jordan) (2004).

[Dutot, 2002] Dutot, A., Olivier, D., and Archambault, D. "Tl a language to create games for visually impaired children". Proc. of the Miesenberger, K., Klaus, J., and Zagler, W., editors, Computer Helping People with Special Needs, pages 193–195. Springer (2002).

[Lerebourg, 2003] Lerebourg, S., Dutot, A., Bertelle, C., and Olivier, D. "Decision support system and regulation system for road traffic management". Proc. of the *ESS, 15th European Simulation Symposium*, pages 529–534, Delft (Netherlands) (2003).

Résumé

Ce travail présente une méthode de distribution dynamique et adaptative, pour des applications distribuées constituées de multiples entités en interaction, dans un environnement de calcul versatile.

L'équilibrage de charge ainsi que la minimisation des coûts de communication sont pris en compte.

La méthode proposée repose sur la détection d'organisations au sein de l'application afin de mieux la distribuer. Les organisations sont identifiées comme des groupes d'entités en très forte communication.

Les organisations évoluent, apparaissent, se renforcent, s'affaiblissent et disparaissent. Les ressources disponibles de calcul sur lesquelles l'application s'exécutent varient également. Ces contraintes imposent à la distribution de s'adapter dynamiquement.

La méthode est basée sur des colonies de fourmis numériques qui tentent de recruter les entités de l'application. Les fourmis coopèrent au sein d'une même colonie et sont en compétition lorsqu'elles n'appartiennent pas à une même colonie. Elles tentent de s'approprier les organisations au sein de l'application, chaque colonie travaillant pour une ressource de calcul distincte. La compétition inter-colonies permet la répartition de la charge. La collaboration au sein de chaque colonie permet la détection des organisations, en plaçant les très fortes communications ensembles sur la même ressource de calcul. Enfin la gestion de la population permet de prendre en compte l'hétérogénéité des ressources de calcul.

Abstract

This work presents a dynamic adaptive distribution method for distributed applications made of large number of interacting entities in a versatile computation environment.

Load balancing as well as communication minimization are taken into account.

The proposed method is based on the detection of organizations inside the application to better distribute it. Organizations are identified as groups of highly communicating entities.

Organizations evolve, appear, strengthen, weaken and disappear. Available computing resources where the application runs also change. Such constraints dictate that the distribution be dynamic and adaptive.

The method is based on colonies of numerical ants trying to gather entities of the application. Ants cooperate inside a unique colony and compete when they are not in the same colony. They try to capture organizations inside the application, each colony working for a distinct computing resource. Competition between colonies allow the load balancing. Collaboration inside colonies allow to detect organizations, putting highly communicating sets on the same computing resource. Finally, population management allow to take into account computing resources heterogeneity.