Chapitre 4

Détection d'organisations par des techniques de collaboration et de compétition

Sommaire

4.1 Introduction

L'intelligence en essaim est d'inspiration éthologique et privilégie les modèles simples en prenant exemple, en particulier, sur les comportements collectifs des insectes sociaux, ceci se traduisant méthodologiquement dans des systèmes auto-organisés. Cela consiste donc à construire des agents réactifs organisés par l'intermédiaire de leurs interactions capables de déterminer des solutions complexes par effet de bord. Les agents ont une vision locale et décident de manière non centralisée et autonome de leurs actions. Le mécanisme naturel est sans holoptisme[38], aucun individu n'a une idée quelconque de ce qui émerge. Ce qui stabilise et dirige est obtenu par l'intermédiaire de l'environnement. Dans le respect de la métaphore naturelle, le fonctionnement du système est décentralisé. Il est constitué d'agents ne faisant appel à aucune représentation ni mécanisme de raisonnement sophistiqué. La résolution n'est que le fait des interactions et de la dynamique du système : l'intelligence naît de façon collective. De tels systèmes se caractérisent par leur adaptabilité et leur robustesse, du fait d'un contrôle décentralisé, chaque agent réagit en fonction de ses propres perceptions aux modifications de son contexte et il est capable de s'adapter continuellement aux variations et à la dynamique de celui-ci. L'absence de centralisation rend le système robuste, la défaillance de l'un des agents ne remet pas en cause l'obtention de la solution. Un tel système peut changer de comportement en cours de fonctionnement afin de s'adapter aux évolutions de son environnement et donc prendre en compte la dynamicité de celui-ci. Un autre aspect important, de notre point de vue est l'économie cognitive, en effet il n'y pas de représentation symbolique ainsi que peu de mémoire individuelle et les modèles de comportement sont extrêmement simples basés sur des mécanismes de perception-action ce qui permet d'introduire éventuellement des comportements adaptatifs simples. La communication n'y est pas intentionnelle et utilise le plus souvent l'environnement comme médium, cet environnement rétroagissant sur les agents.

Bien que l'on puisse considérer que les systèmes complexes peuvent être très différents en ce qui concerne leur formes et leur fonctions, ils partagent une caractéristique commune qui est qu'ils sont constitués d'un ensemble d'entités en interactions. La connectivité est donc un attribut universel des systèmes complexes. [Green, 1993] montre qu'ils présentent des schémas de connectivités souvent semblables et qui plus est, les dépendances exprimées dans des modèles matriciels, les systèmes dynamiques, les automates cellulaires, les semigroupes et des ensembles partiellement ordonnés sont isomorphes aux graphes orientés.

Nous nous proposons d'étudier de tels graphes d'interaction, qui évoluent donc en fonction du temps, en recherchant en particulier les organisations. Pour cela on va utiliser des mécanismes d'intelligence collective qui comme on l'a dit plus haut sont robustes, adaptatifs et de plus distribués. Les agents réactifs vont être attirés par les organisations puis «capturés» par ces organisations contenues dans le graphe dynamique. Ces agents utilisent leur environnement c'est-à-dire le graphe pour interagir par l'intermédiaire de

[38] *Holos* le tout et *optisme* voir

signaux déposées sur celui-ci. Les interactions créeront à la fois des mécanismes de collaboration mais également des mécanismes de compétition qui permettront en particulier de détecter les organisations. La compétition introduit un mécanisme de régulation et permet par exemple la possibilité de découvrir des organisations à intersection non vide.

4.2 Détection d'organisations par un algorithme fourmi

Notre approche consiste à utiliser une population d'agents simples et réactifs dont le comportement s'inspire de celui exhibé par des fourmis naturelles.

Le but est donc de trouver des organisations à l'intérieur d'un graphe $G = (V, E)$ qui représente donc les interactions E entre des entités V. Les arcs peuvent être valués en fonction d'une mesure sur l'interaction. Le poids de l'arc e est noté $|e|$. D'autre part, nous définissons la notion d'organisation dans le graphe G comme étant un ensemble d'entités/sommets plus étroitement reliées ensembles qu'aux autres parties du graphe ; plus étroitement reliées voulant dire que les sommets peuvent avoir un plus grand nombre d'arcs en commun, mais aussi que les poids peuvent être plus importants. Cette formulation est très voisine de celles que l'on rencontre dans la littérature [Danon et al., 2005, Girvan and Newman, 2002, Newman, 2004b].

L'algorithme que nous utilisons s'inspire des algorithmes fourmis ACO (Ant Colony optimization) décrits dans [Dorigo and Stützle, 2004] néanmoins deux différences sont notables : la compétition que nous introduisons par l'intermédiaire de plusieurs colonies et l'absence de fonction d'objectif. Dans notre algorithme de l'auto-organisation émerge, elle modifie l'état de l'environnement et donc le graphe et influence de ce fait le comportement des fourmis numériques mais elle n'est pas réintroduite explicitement dans l'algorithme.

Plusieurs colonies de fourmis se déplacent dans le graphe pour détecter les organisations. Chaque colonie a une couleur distincte et lorsque les fourmis numériques parcourent le graphe elles déposent sur les arcs des phéromones de leur couleur. Les fourmis détectent les phéromones et leur quantité, elles sont attirées par les phéromones de leur couleur en fonction de la quantité et tendent à être repoussées par les phéromones des autres couleurs. Lorsque le graphe est pondéré, les fourmis favorisent les arcs dont la valuation est la plus grande.

Le principe de l'algorithme est de colorer les organisations contenues dans le graphe en utilisant les phéromones. Les fourmis d'une colonie donnée vont donc collaborer pour coloniser des zones et entrer en compétition avec les autres colonies pour maintenir leurs propres zones, la figure 4.1 est un exemple d'un graphe pour lequel on a obtenu quatre organisations avec trois couleurs. Des solutions vont émerger et être entretenues par le comportement des fourmis. Les solutions vont être les couleurs de chaque sommet dans le graphe. Cette couleur est déterminée en fonction de la couleur dominante sur les arcs incidents.

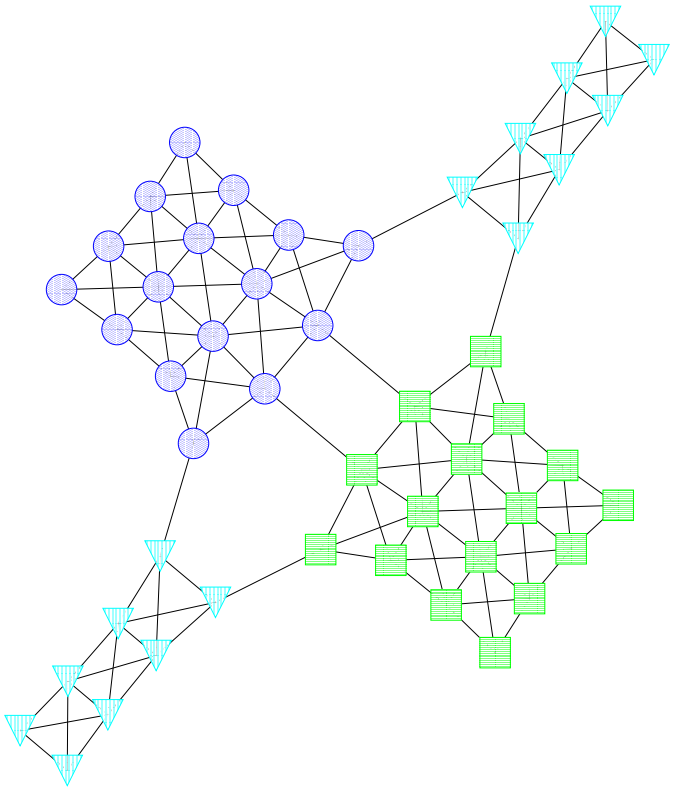


FIG. 4.1 – Exemple d'un graphe coloré avec détection des organisations

4.2.1 Le graphe comme environnement et solution

Les fourmis numériques se déplacent dans un environnement représenté par un graphe. Ce dernier est une représentation d'un système composé d'entité en interaction. Cela peut être par exemple, un graphe de communication, un réseau d'interaction protéique, le graphe de communication d'une application à distribuer ... 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 déposées dans ce graphe. L'importance d'un arc est définie par le système.

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 ont visité cet endroit. La figure 4.2 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

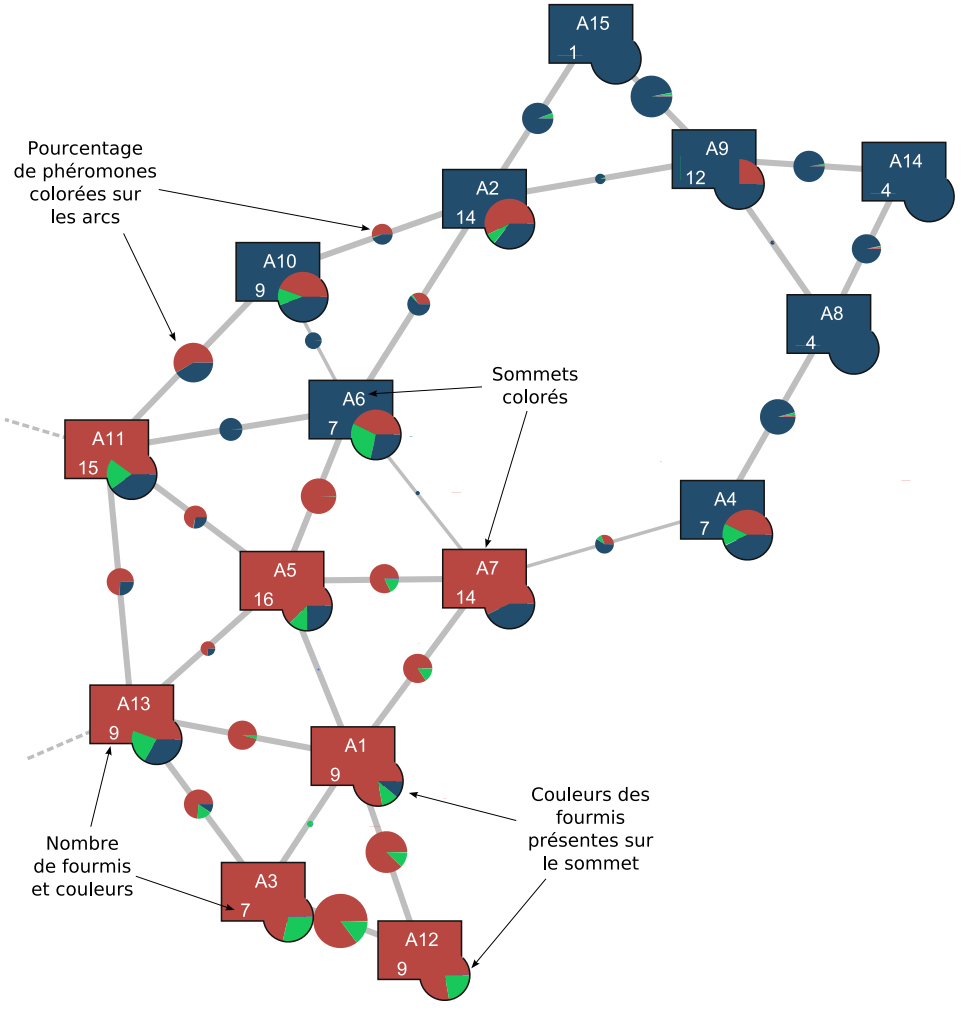


FIG. 4.2 – Graphe dynamique utilisé en tant qu'environnement pour les fourmis, et les diverses informations qu'il contient.

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.

Les fourmis se déplacent dans cet environnement en perpétuelle mutation. L'algorithme 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 au temps t par coloration.

4.2.2 Graphe dynamique coloré

Nous considérons donc le graphe comme étant *dynamique*, c'est-à-dire qu'à tout moment sa topologie et sa valuation sont susceptibles de changer. Ainsi les événements possibles sont les suivants :

- Des sommets peuvent apparaître ou disparaître ;
- Des arcs peuvent apparaître ou disparaître ;
- Les valuations des arcs peuvent changer.

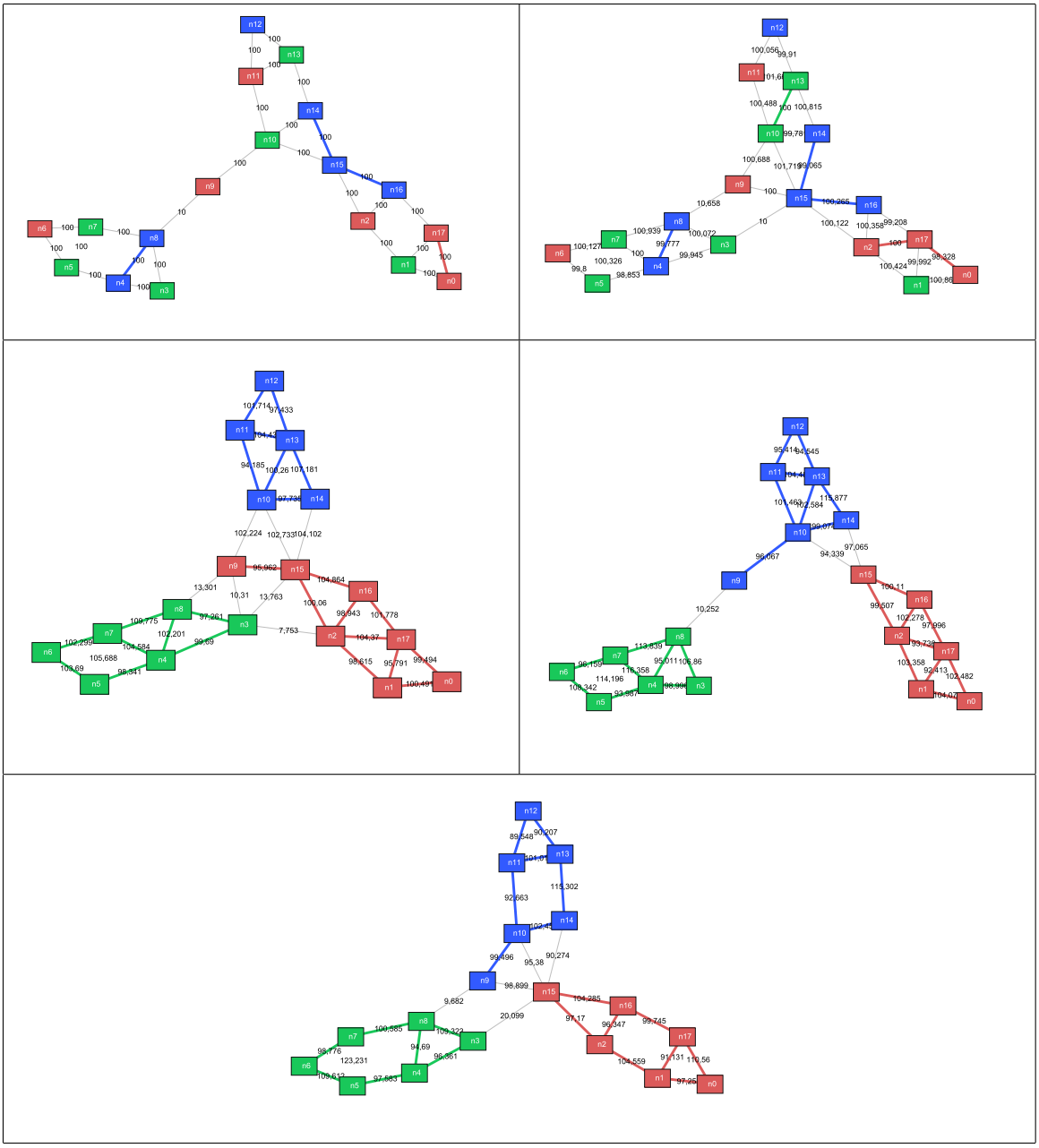
De plus nous allons associer des couleurs aux sommets qui seront l'expression des solutions.

On comprend bien que dans ce contexte les solutions elles aussi peuvent changer et on peut alors voir apparaître/disparaître ou changer des organisations. 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.

Un graphe dynamique coloré $G(t) = (E(t), V(t), C(t))$ (cf. figure 4.3) est défini de la façon suivante :

- $V(t)$ l'ensemble des sommets au temps t. Chaque sommet v est caractérisé par :
  - une couleur $c \in C(t)$,
- $E(t)$ l'ensemble des arcs au temps t. Chaque arc e est caractérisé par :
  - un poids $|e| \in \mathbb{N}^+$ qui correspond à l'importance de l'interaction entre les sommets qui constituent e.
  - une quantité de phéromones de chaque couleur.
- $C(t)$ un ensemble de couleurs qui représente les colonies de fourmis au temps t.

On a donc défini un environnement d'exécution et solution de nos algorithmes, nous allons maintenant présenter ces derniers.



De gauche à droite et de haut en bas.

FIG. 4.3 – Exemple d'un graphe dynamique coloré à cinq stades de son évolution et détection d'organisation

4.2.3 Recherche d'organisations ou de communautés dans les réseaux

Des structures sont souvent présentes dans les grands réseaux d'interactions et elles traduisent l'organisation du réseau sous forme de communauté, ainsi on rencontre des communautés dans des réseaux biologiques, sociaux, d'informations... Si ces organisations sont souvent présentes elles sont néanmoins à la fois difficiles à détecter et même à définir. On se contente, le plus souvent d'une définition intuitive comme celle que nous avons donnée précédemment qui considère donc une organisation comme un ensemble de sommets dont la densité de connexions internes est plus forte que la densité externe, la borne restant à fixer. Le but est alors de trouver une partition des sommets en organisations qui vérifient le critère que nous venons d'énoncer sans en connaître a priori le nombre.

L'algorithme

Le comportement des fourmis colorés est défini par l'algorithme 4.1 qui est exécuté itérativement pour chaque fourmi. À chaque itération l'algorithme fixe le nœud que la fourmi va visiter. Deux paramètres sont utilisés, il s'agit de A et de T. A contrôle la compétition entre les fourmis, ainsi quand la proportion de phéromone de la même couleur que la fourmi comparée aux autres couleurs est inférieure à A, la fourmi est alors dans un environnement hostile et elle fuit dans une autre partie du graphe choisie aléatoirement. Le paramètre T est un paramètre de régulation qui sert à stabiliser éventuellement le comportement des fourmis. En effet, il évite qu'une fourmi fuie constamment de nœud en nœud lorsqu'elle est toujours dans un environnement hostile. Cela permet de mieux coloniser les zones. Habituellement, $T = 2$ et $A = \frac{1}{\text{Le nombre de colonies}}$.

Chaque colonie lutte pour coloniser des sous-graphes, comme on peut le constater sur la figure 4.1 la même colonie peut occuper une ou plusieurs zones. Si cela permet d'envisager d'utiliser un petit nombre de couleurs pour découvrir les organisations dans le graphe, cependant il est possible que deux organisations voisines soient colonisées par la même colonie. Pour éviter cela, l'algorithme commence uniquement avec n colonies en fonction du nombre minimal d'organisations ($n = 2$ si ce nombre n'est pas connu) et on introduit ensuite de nouvelles colonies, tout en gardant la population constante, jusqu'à ce qu'il ne soit plus possible d'en ajouter plus. Pour chaque sommet du graphe on calcule le ratio de la couleur dominante, cette couleur est considérée comme stable si elle représente une proportion plus grande qu'une borne fixée ω. On détermine alors la stabilité globale moyenne du graphe, ce qui permet de savoir s'il faut ajouter de nouvelles colonies. On ajoute des colonies tant que la stabilité globale est supérieure ou égale à ω, $\omega = 80\%$ en général. Cet algorithme à notre connaissance, contrairement à ceux de la littérature, permet par de déterminer les organisations recouvrantes en prenant en compte plusieurs couleurs par arc.

Algorithme 4.1 : Comportement d'une fourmi.

n : nœud courant;
t : temps;
A : borne contrôlant la fuite;
T : temps restant;
Δt : compteur mesurant le temps;
si *degré(n)=0* **alors**
    Sauter aléatoirement sur un autre nœud;
sinon
    $w \leftarrow$ Somme de tous les poids des arcs incidents à n;
    $\tau \leftarrow$ Somme de toutes les phéromones de chaque couleurs sur chaque arc incident à n;
    $\tau_c \leftarrow$ Somme des phéromones de la couleur de la fourmi sur chaque arc incident à n;
    $a \leftarrow \frac{\tau_c}{\tau}$;
    si $\Delta t < T$ **alors**
        Choisir un arc à traverser de façon aléatoire biaisée par le poids des arcs s'ils existent;
        Déposer des phéromones de la couleur de la fourmi sur l'arc traversé;
        $n \leftarrow$ sommet sur lequel la fourmi est arrivée;
        $\Delta t \leftarrow \Delta t + 1$;
    sinon
        si $a < A$ **alors**
            Sauter aléatoirement sur un autre nœud;
            $\Delta t \leftarrow 0$;
        sinon
            Choisir un arc à traverser de façon aléatoire biaisée par le poids des arcs s'ils existent ;
            Déposer des phéromones de la couleur de la fourmi sur l'arc traversé ;
            $n \leftarrow$ sommet sur lequel la fourmi est arrivée;

Quelques résultats

L'exemple de référence est un club de karaté étudié par [Zachary, 1977], dans lequel une dispute interne à conduit à un schisme séparant le club en deux plus petits clubs. C'est un cas réel pour lequel on connaît les organisations résultantes. Cet exemple a été étudié en particulier par Newman dans [Newman, 2004b, Newman, 2004c, Newman, 2004a] ou encore [Bagrow and Bollt, 2005]. Dans ces articles les auteurs proposent différents algorithmes qui déterminent les deux communautés (cf. figure 4.4(a)), néanmoins des problèmes sont souvent rencontrés sur le sommet 10 qui est de degré 2, les deux arcs incidents connectant le nœud au deux clubs. Notre algorithme trouve la solution et met en évidence ce nœud particulier qui n'est jamais réellement stable (cf. figure 4.4(b)).

En naviguant sur Amazon^{TM}site de vente de livres en ligne, il est possible, connaissant le numéro international standard d'un livre (ISBN) de déterminer les autres livres achetés par les consommateurs en même temps. Nous avons construit un graphe à partir de ces données et cherché les organisations. Nous en donnons un exemple sur les figures 4.5 et 4.6.

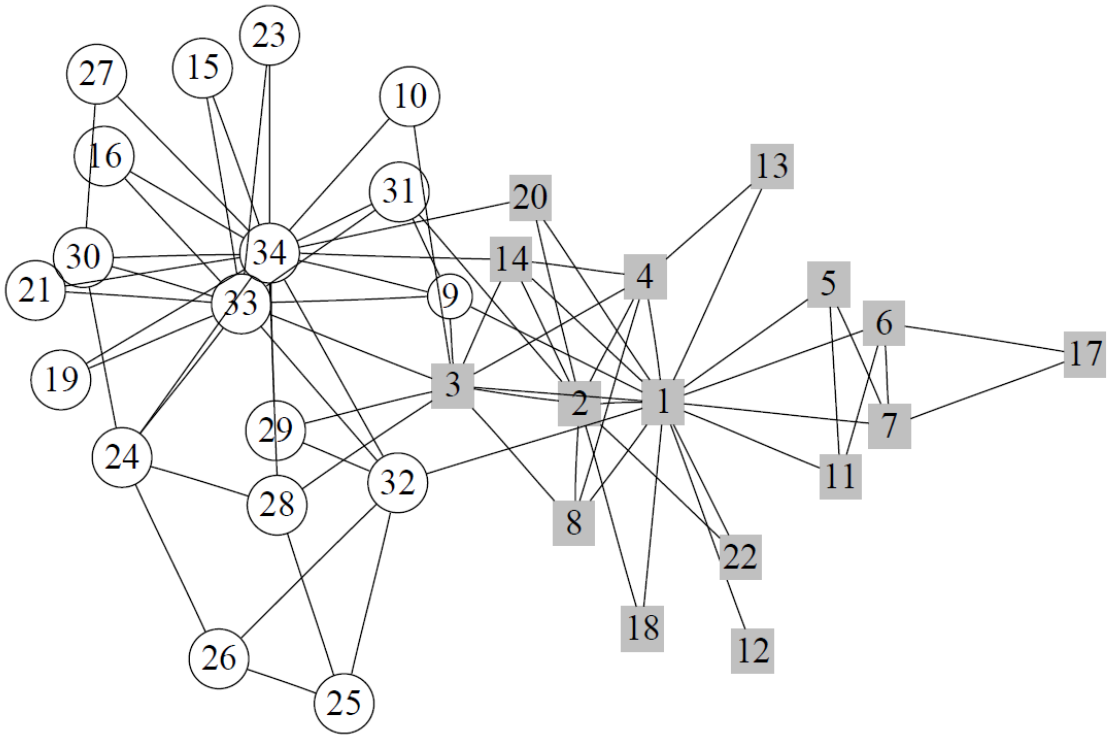
Chaque nœud représente un livre et chaque arc un lien du type «customers having bought this book also bought». Il est intéressant de constater que les recommandations sont presque toujours sur le même sujet. Ainsi sur la figure 4.5 on voit apparaître[39] des zones mises en évidence par les couleurs dont les sujets sont « le web», «programming», «architecture» ou encore «systems».

La figure 4.5 est obtenue à partir de l'une des éditions de «The art of computer programming, volume 1 [Knuth, 1997]. L'algorithme trouve 9 partitions distinctes de différentes tailles.

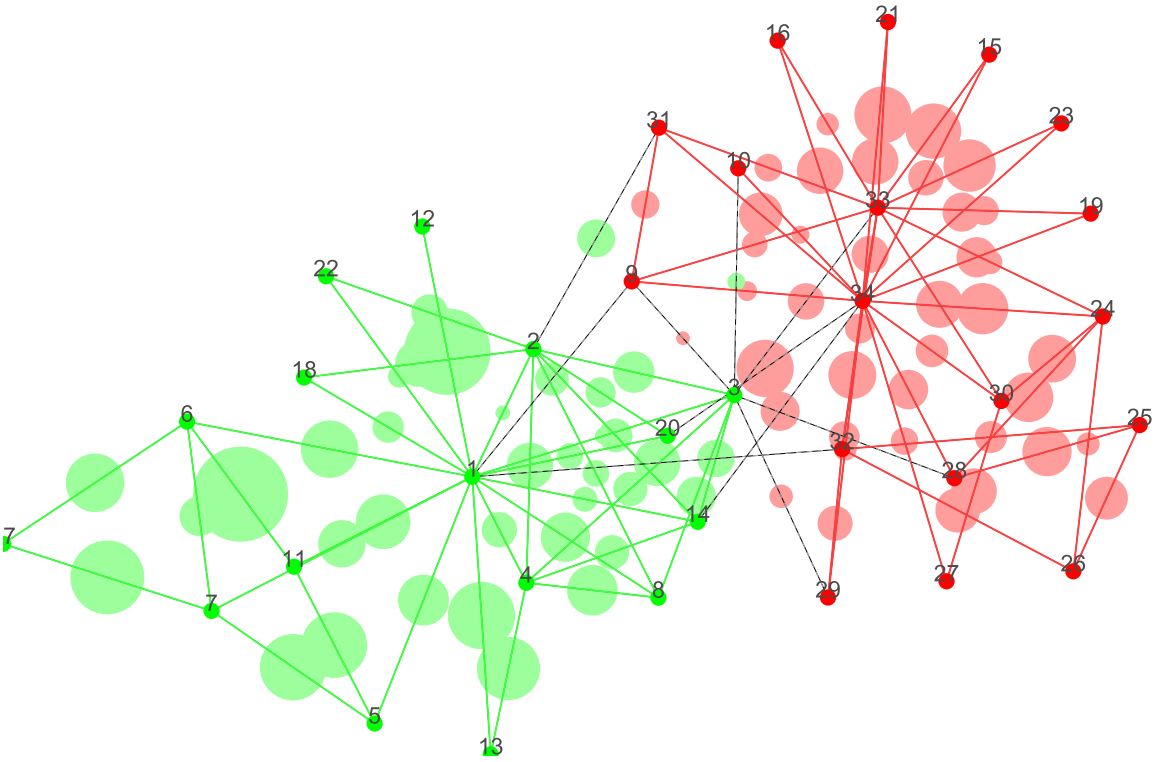
Deux vidéos [Dutot and Olivier, 2006a, Dutot and Olivier, 2006b] de plus grands graphes sont téléchargeables .

Nous avons également appliqué notre méthode à un graphe homologique de protéines, dans un tel graphe chaque nœud est une protéine et un arc existe entre deux protéines si elles sont proches et sont fonctionnellement apparentées. La source du graphe est [Adai et al., 2004] et on peut voir sur [Dutot and Olivier, 2006c] une application de notre algorithme mettant en évidence des groupes de protéines fonctionnellement voisines.

[39]Il faut reconnaître que nous demandons au lecteur des efforts certains, mais nous avons essayé de faire pour le mieux ....



(a) Solution obtenue par Newman



(b) Solution obtenue par notre algorithme. Les cercles sur les arcs représentent la phéromone dominante, le diamètre la quantité et l'épaisseur de l'arc l'importance de celui-ci

FIG. 4.4 – Organisations au sein du club de karaté étudié par Zachary

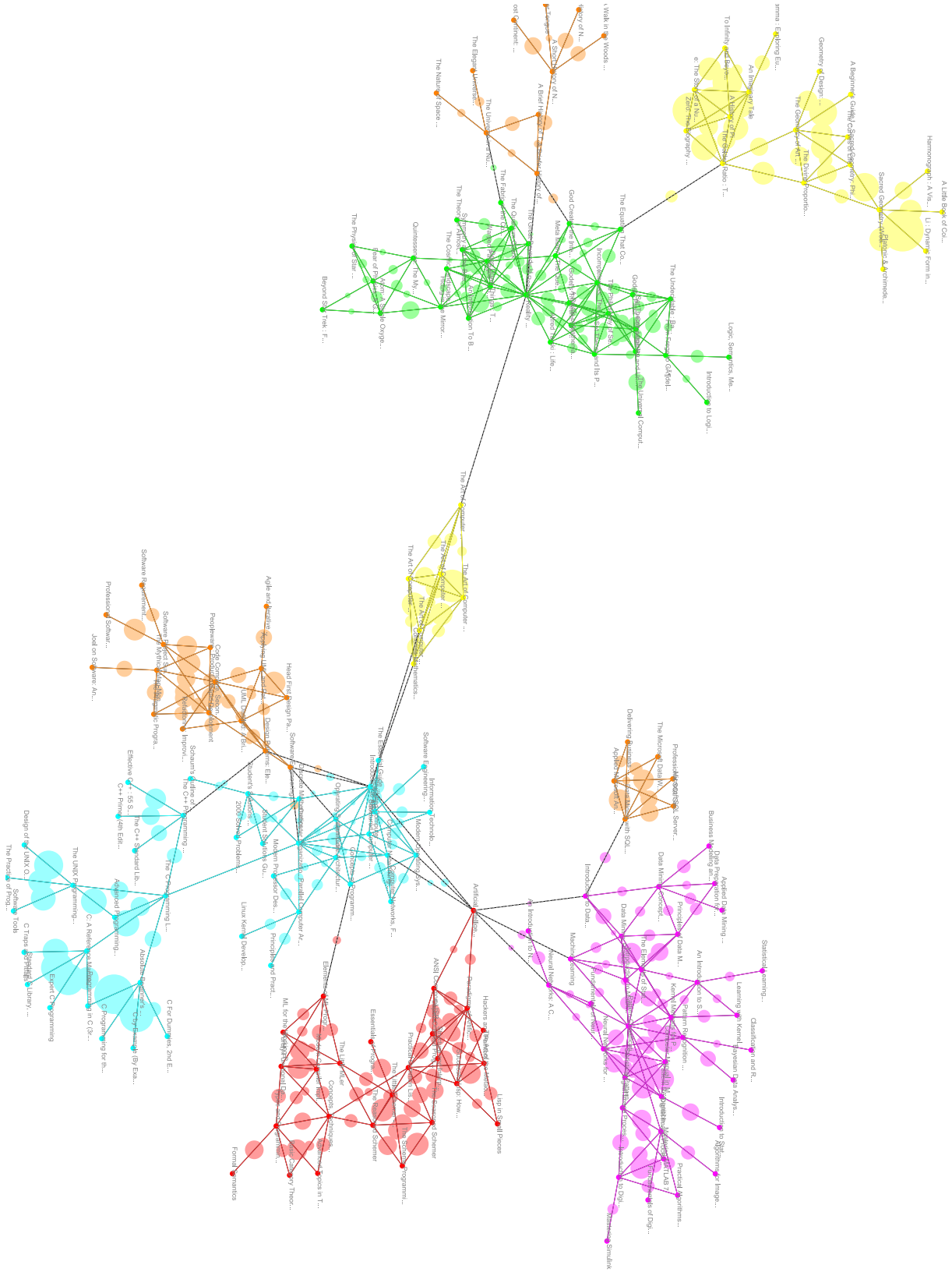


FIG. 4.5 – Exploration récursive de profondeur maximale 12 des bases de données d'Amazon.com à partir de "The Art of Computer Programming".

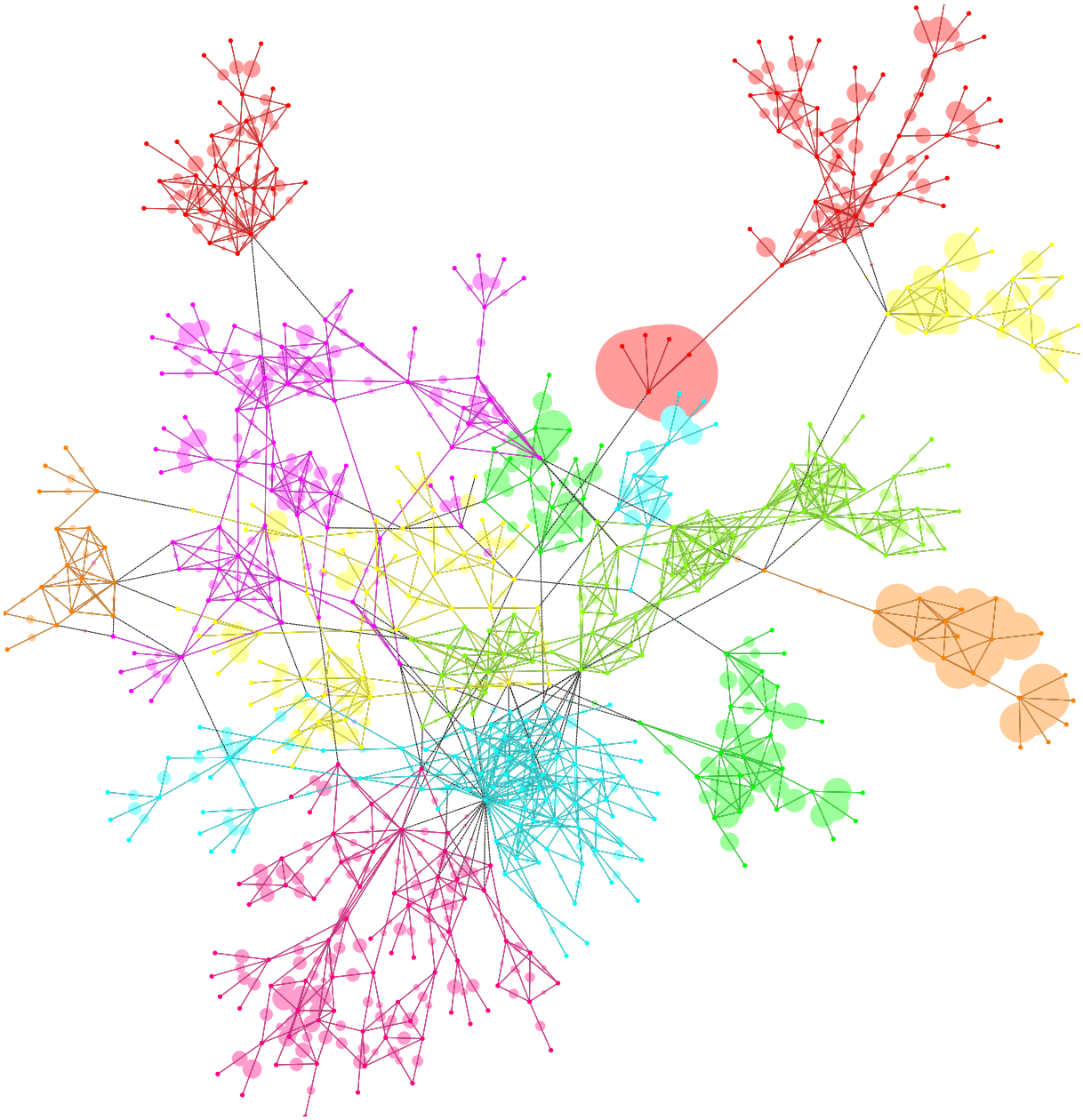


FIG. 4.6 – Exploration récursive de profondeur maximale 12 des bases de données d'Amazon.com à partir de "The Ruby Way".

4.3 Distribution dynamique adaptative

Le problème auquel nous nous intéressons 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 en prenant 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 appelons *entités*, l'exemple auquel nous pensons en premier est bien évidemment la simulation d'un écosystème tel que nous l'avons présenté dans le chapitre 3. Sur la figure 4.7 ces entités sont représentées par des cercles dans la partie «représentation logicielle» et pourront être par exemple des holons, des vortex ou encore des grandeurs physiques. 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.

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, ... 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 même 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

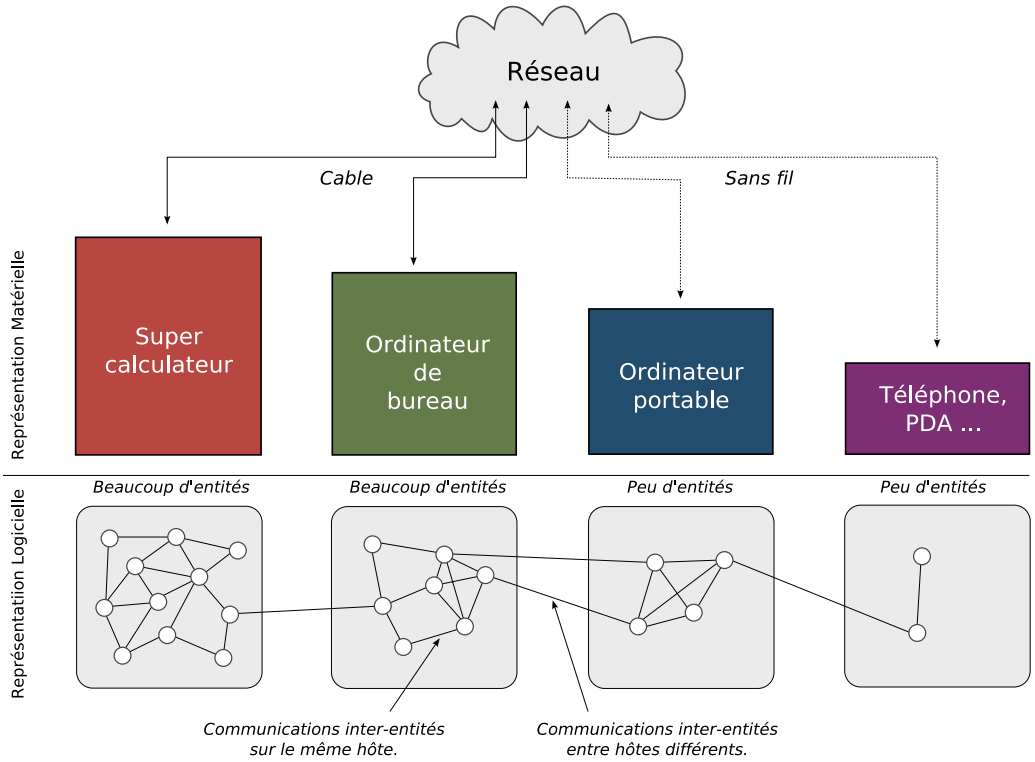


FIG. 4.7 – Exemple d'application répartie nécessitant une distribution dynamique adaptative.

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 4.7 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.

Ce problème a déjà été abordé de manière importante pour des entités non com-

muniquantes dans la littérature 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.

La méthode proposée 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 de [Dorigo et al., 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.

Nous proposons sur la figure 4.8 une petite taxinomie des méthodes ayant trait aux approches collectives. L'intention ici est de montrer les différences entre le modèle que nous proposons, et les approches qui l'ont inspirées.

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, 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

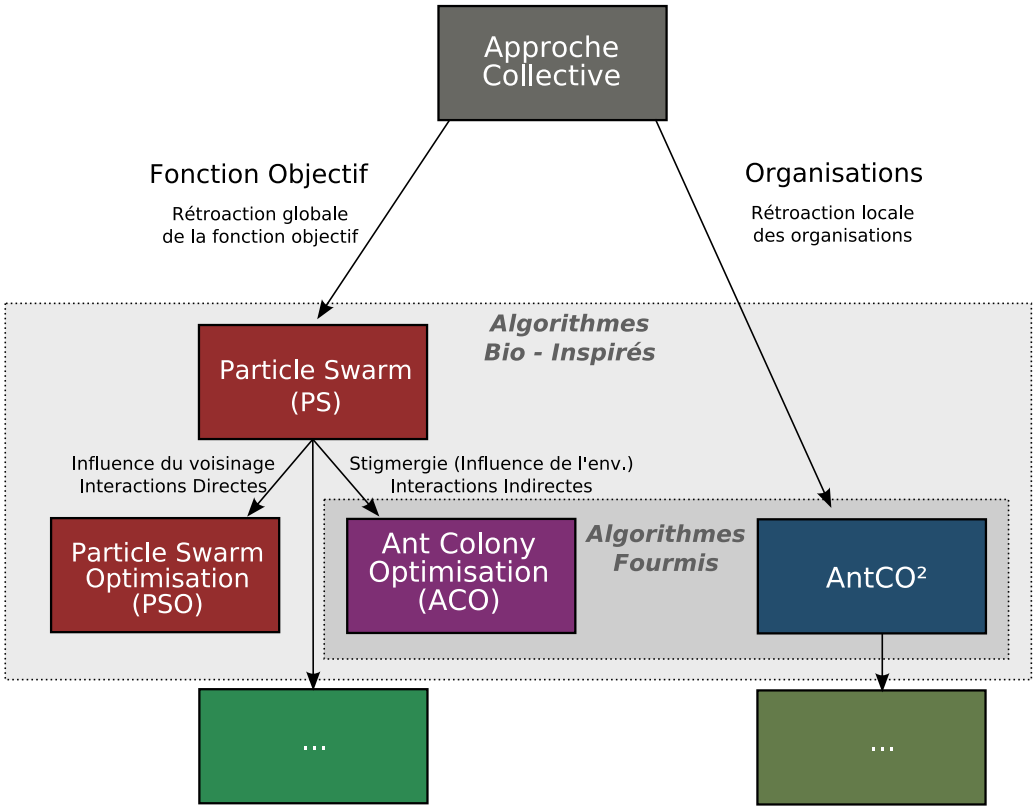


FIG. 4.8 – Positionnement de l'approche.

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.

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

Nous verrons que si certains traits saillants de ces approches caractérisent la méthode proposée, elle se dirige aussi vers une approche basée sur les systèmes complexes, que nous comparons aux écosystèmes, et que l'on qualifie d'écosystème computationel.

4.3.1 Le modèle

Nous allons préciser dans ce qui suit le modèle et l'algorithme nous permettant de réaliser la distribution dynamique.

AntCO^2 n'est pas autre chose qu'une extension du principe de détection des organisations tel que nous l'avons présenté au paragraphe 4.2. L'objectif est de détecter les organisations à l'intérieur desquelles les communications sont fortes et d'équilibrer la charge. Le graphe constitue donc à la fois l'environnement et la solution, c'est une structure de données en constante évolution. Les différentes colonies de fourmis représentent une ressource de calcul donnée, la taille de la colonie la capacité de la ressource et les différentes fourmis

vont alors s'«affronter» afin de coloniser des zones du graphe qui est la projection de l'application à distribuer. Il y a collaboration au sein d'une même colonie et compétition entre les colonies. La collaboration permet la détection d'organisation au sein des entités de l'application et la compétition une répartition de la charge.

L'algorithme utilise un graphe dynamique pondéré coloré. 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 :

$$\Delta_k^{(t)}(e, c) \tag{4.1}$$

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 l'intervalle de temps t et $t + 1$:

$$\Delta^{(t)}(e, c) = \sum_{k \in \mathcal{F}_c(t)} \Delta_k^{(t)}(e, c) \tag{4.2}$$

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

$$\Delta^{(t)}(e) = \sum_{c \in \mathcal{C}(t)} \Delta^{(t)}(e, c) \tag{4.3}$$

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 :

$$K_c^{(t)}(e) = \frac{\Delta^{(t)}(e, c)}{\Delta^{(t)}(e)} \tag{4.4}$$

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 :

$$\tau^{(t)}(e, c) \tag{4.5}$$

et la quantité totale toutes couleurs confondues :

$$\tau^{(t)}(e) \tag{4.6}$$

Au départ, cette valeur est fixée à :

$$\tau^{(0)}(e) = \epsilon \tag{4.7}$$

nous verrons par la suite que cette quantité ne peut être nulle. 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 :

$$\tau^{(t)}(e,c) = \rho \cdot \tau^{(t-1)}(e,c) + \Delta^{(t-1)}(e,c) \tag{4.8}$$

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 définie par la couleur principale de tous ses arcs adjacents, ainsi :

$$\xi^{(t)}(u) = \arg\max_{c \in \mathcal{C}(t)} \sum_{e \in \mathcal{E}_u(t)} \tau^{(t)}(e,c) \tag{4.9}$$

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

L'algorithme 4.2 reprend, de manière très simple, le fonctionnement de l'environnement.

Algorithme 4.2 : Fonctionnement de l'environnement dans $AntCO^2$.

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**
  - cf.. algorithme 4.3;
- **pour** *tout arc e* **faire**
  - Appliquer le facteur d'évaporation ρ sur la phéromone présente en utilisant l'équation 4.8;
- **pour** *tout sommet v* **faire**
  - Définir la couleur du sommet en fonction de l'équation 4.9;
- Communiquer ces informations si nécessaire;

jusqu'à *indéfiniment* ;

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. $AntCO^2$ 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 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 :

$$\Omega^{(t)}(e,c) = K_c^{(t)}(e)\tau^{(t)}(e,c) \tag{4.10}$$

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 $t+1$:

$$\begin{cases} p^{(t)}(e,c) = \dfrac{w^{(0)}(e)}{\sum\limits_{e_i \in \mathcal{E}_u(t)} w^{(0)}(e_i)} & \text{si } t = 0 \\ p^{(t)}(e,c) = \dfrac{\left(\Omega^{(t)}(e,c)\right)^{\alpha} \cdot \left(w^{(t)}(e)\right)^{\beta}}{\sum\limits_{e_i \in \mathcal{E}_u(t)} \left(\Omega^{(t)}(e_i,c)\right)^{\alpha} \cdot \left(w^{(t)}(e_i)\right)^{\beta}} & \text{si } t \neq 0 \end{cases} \tag{4.11}$$

où les paramètres α et β (tous deux > 0) 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 4.11 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 :

$$\eta_k(v) = \begin{cases} 1 & \text{si } v \notin \mathcal{W}_k \\ \eta & \text{si } v \in \mathcal{W}_k \end{cases} \tag{4.12}$$

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 :

$$p^{(t)}(e,c) = \frac{\left(\Omega^{(t)}(e,c)\right)^{\alpha} \cdot \left(w^{(t)}(e)\right)^{\beta} \cdot \eta_k(u)}{\sum\limits_{e_i \in \mathcal{E}_u(t)} \left(\Omega^{(t)}(e_i,c)\right)^{\alpha} \cdot \left(w^{(t)}(e_i)\right)^{\beta} \cdot \eta_k(v_i)} \tag{4.13}$$

L'algorithme 4.3 reprend le fonctionnement des fourmis.

4.3.2 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

Algorithme 4.3 : Comportement d'une fourmi de couleur c dans $AntCO^2$.

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. 4.15);
  - $\tau_c \leftarrow \tau_c + \tau_c(e)$;
  - $\tau \leftarrow \tau + \tau(e)$;
- **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* ;

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'$AntCO^2$.

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 fourmis 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 fourmis 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.

4.3.3 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 non seulement pour distribuer l'application, mais aussi pour la déployer.

Dans les deux cas, la question de l'initialisation d'$AntCO^2$ se pose. Comment allons nous 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[40], 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 (cf. eq. 4.18, mesure la qualité de l'équilibrage de charge) qui est quasi optimale, et une valeur de r_1 (cf. eq. 4.17, mesure de la quantité de communication entre ressources), généralement très mauvaise.

4.3.4 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 :

[40]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 proportionnel à la taille du graphe, c'est-à-dire 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 doit être fixé en fonction du mode d'évolution du graphe et des ressources de calcul.

4.3.5 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.

La figure 4.9 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 phénomènes naturels d'embouteillages (cf. photo 4.10(c)). 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'individus sur un chemin peut jouer un rôle répulsif même si ce chemin est très chargé en phéromones.

Ce mécanisme est quasi-directement implanté dans le comportement des fourmis numériques, 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 :

$$\gamma(u) = \begin{cases} 1 & \text{si } N(u) \leq N^* \\ \gamma \in]0,1] & \text{sinon} \end{cases} \tag{4.14}$$

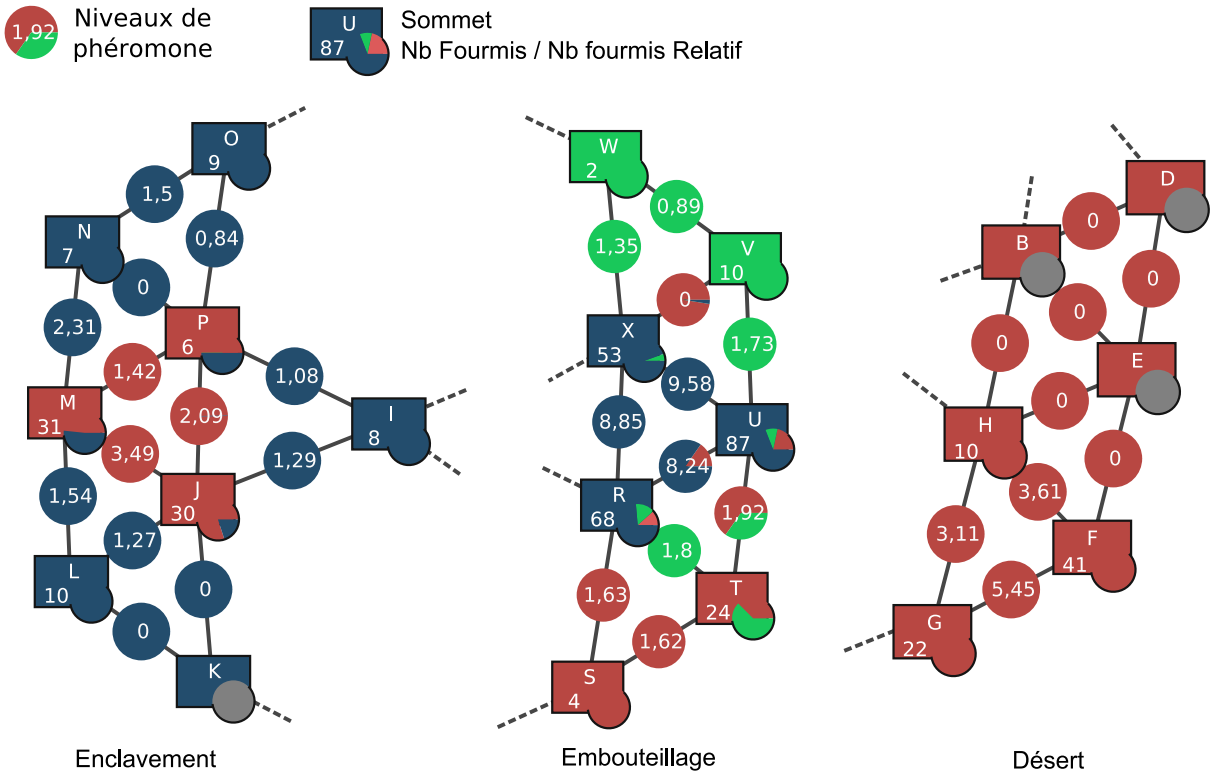


FIG. 4.9 – Problèmes auxquels est confronté l'algorithme de base.

Ainsi l'équation 4.13 est modifiée ainsi :

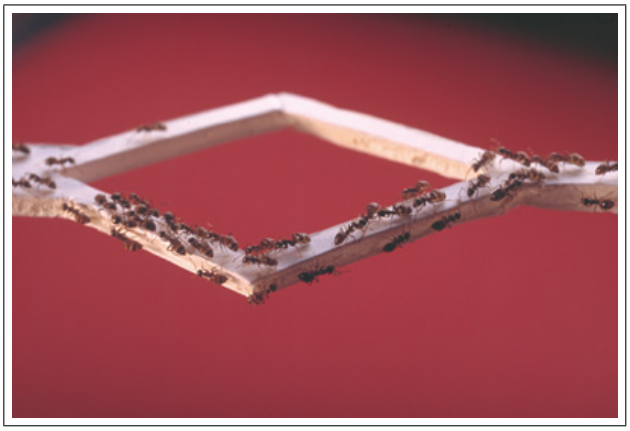
$$p^{(t)}(e,c) = \frac{\left(\Omega^{(t)}(e,c)\right)^{\alpha} \cdot \left(w^{(t)}(e)\right)^{\beta} \cdot \eta_k(u) \cdot \gamma(u)}{\sum_{e_i \in \mathcal{E}_u(t)} \left(\Omega^{(t)}(e_i,c)\right)^{\alpha} \cdot \left(w^{(t)}(e_i)\right)^{\beta} \cdot \eta_k(v_i) \cdot \gamma(v_i)} \tag{4.15}$$

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.

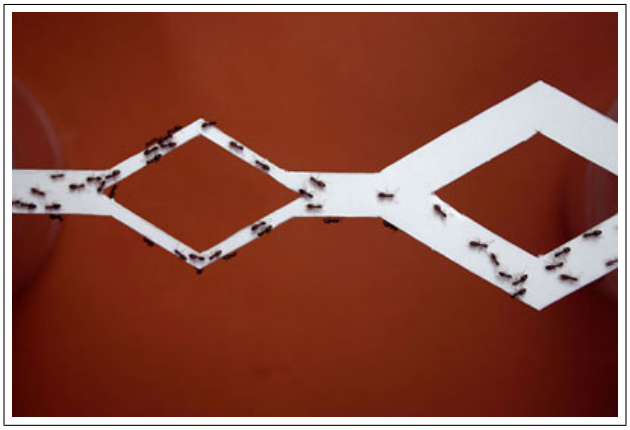
Mécanismes de fuite

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

(a) Au départ les fourmis passent indifféremment par les deux voies du pont



(b) Par amplification des fluctuations, un chemin est sélectionné



(c) Les embouteillages rétablissent l'usage de la seconde voie, ici démontré par l'utilisation de voies larges et fines

FIG. 4.10 – Photos d'une expérience réelle.

fourmi de couleur c sur le sommet u fuit si :

$$\frac{\sum_{e \in \mathcal{E}_u(t)} \tau^{(t)}(e, c)}{\sum_{c_i \in \mathcal{C}(t)} \left(\sum_{e \in \mathcal{E}_u(t)} \tau^{(t)}(e, c_i) \right)} \leq \phi \tag{4.16}$$

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 perturberaient une zone en cours de 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 de graphe 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.

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 est donc possible de l'utiliser de manière distribuée.

4.3.6 Architecture et implémentation

AntCO^2 est à la fois un service d'un middleware et un utilisateur de ce dernier. Le middleware offre un service de communication utilisé par l'application et *AntCO*^{2} et des capacités de migration et de mesure pour ce qui concerne en particulier les communications. Il sert également à fournir les événements en relation avec l'apparition ou la disparition de ressources de calcul ainsi que leur charge. La figure 4.11 montre l'architecture.

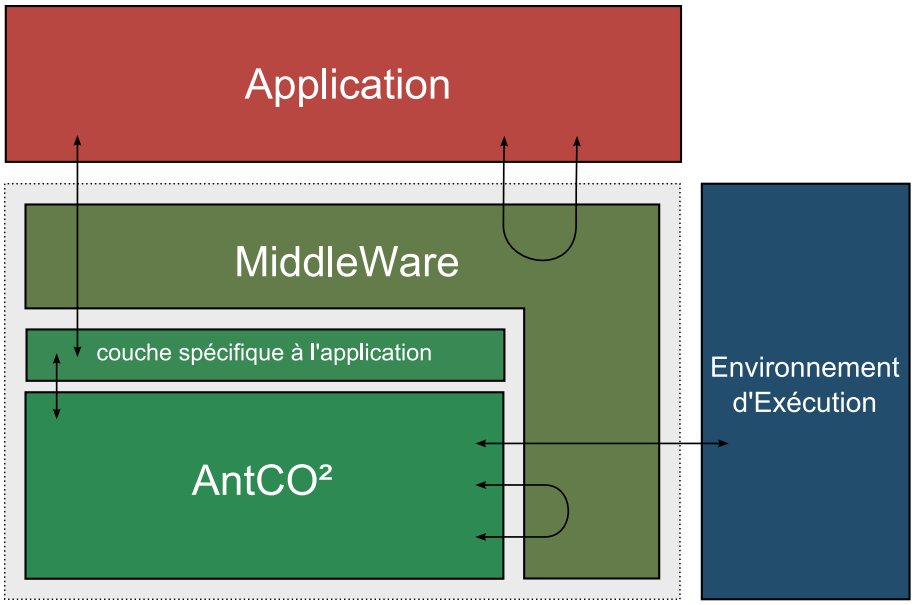
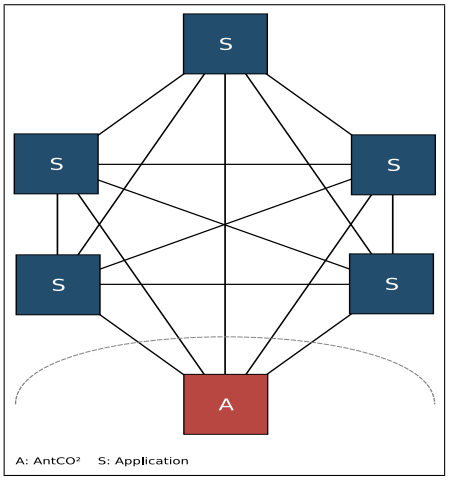


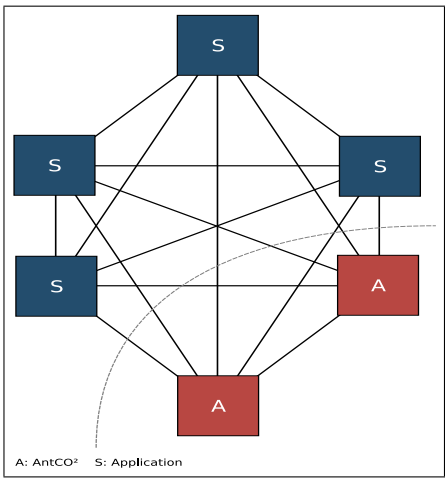
FIG. 4.11 – Architecture dans laquelle s'insère *AntCO*^{2}.

En tant que service, $AntCO^2$ fournit des suggestions de placement pour les entités composant l'application. 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.

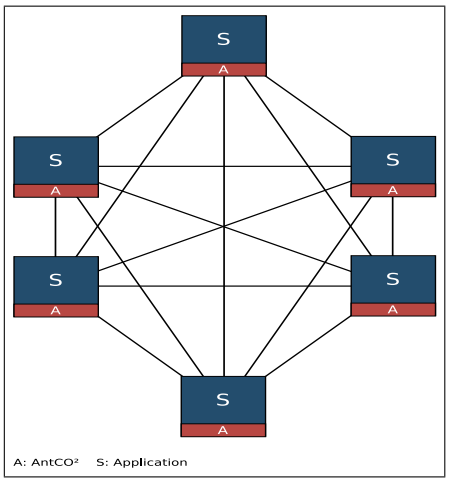
Plusieurs méthodes peuvent être utilisée pour distribuer $AntCO^2$ (cf. figure 4.12). La figure 4.12(a) décrit un environnement où l'application utilise la totalité du cluster pour son utilisation propre excepté la ressource dédiée à $AntCO^2$, la seconde figure 4.12(b) montre la même architecture mais avec plusieurs machines dédiées à $AntC0^2$. Sur la figure 4.12(c), tout comme l'application $AntCO^2$ est également distribué avec éventuellement une instance sur chaque ressource de calcul utilisée par l'application.



(a) $AntCO^2$ à part sur une ressource de calcul dédiée.



(b) $AntCO^2$ à part sur plusieurs ressources de calcul dédiées.



(c) $AntCO^2$ sur les mêmes ressources de calcul que l'application à distribuer.

FIG. 4.12 – Différentes méthodes pour distribuer $AntCO^2$.

Chaque instance d'$AntCO^2$ (cf. algorithme 4.4) a un ensemble de fourmis et un sous-

graphe qui représente la partie de l'application qu'il sert. Au fil des itérations les fourmis sont échangées entre les instances d'*AntCO*^{2} et les événements sont reçus de l'application comme par exemple : création/suppression de nœuds, nouvelle valuation d'un arc .... Des événements sont également reçus de l'environnement d'exécution (ressources de calcul ajoutées ou supprimées par exemple). À intervalle régulier ou sur requêtes de l'application les instances d'*AntCO*^{2} envoient des conseils à l'application. Une migration est proposée quand la couleur d'un sommet change, indiquant que l'entité correspondante de l'application devrait être localisée sur une autre ressource de calcul. L'application est libre de suivre ou pas cette recommandation en fonction des informations spécifiques qu'elle possède ainsi que des contraintes qu'elle doit respecter (cf. algorithme 4.5).

Algorithme 4.4 : Instance d'*AntCO*^{2}.

```
pour toujours
    pour toutes les fourmis faire
        si la fourmi ne fuit pas (eq. 4.16) alors
            Choisir le nouveau nœud à visiter en fonction du taux de phéromone et
            du poids des arcs (eq. 4.15);
    pour chaque arc faire
        Appliquer le facteur de persistance ρ sur les phéromones τ^(t−1) (eq. 4.8);
    pour cahque nœud faire
        Déterminer la couleur de la phéromone dominante en fonction de tous les
        arcs incidents (eq. 4.9) Colorer le nœud avec;
    Prendre en compte les événements issus de l'application et de l'environnement
    et modifier le graphe en conséquence;
    Envoyer les conseils de migration à l'application;
fin
```

Algorithme 4.5 : Instance de l'application

```
pour toujours
    ...;
    Regarder les suggestions provenant d'AntCO² et faire migrer les entités en
    fonction des contraintes et des heuristiques;
    Envoyer les mesures, évènements et migration éventuelle à AntCO²;
fin
```

Considérons l'exemple suivant : soit l'architecture décrite figure 4.12(c) où chaque ressource de calcul héberge à la fois une partie de l'application et une partie de *AntCO*^{2}. Pour simplifier nous fixons le nombre de ressources à deux (R_1 et R_2) mais cela peut être étendu sans problème. L'application démarre ses deux instances App_1 et App_2 (figure 4.13), deux instances d'*AntCO*^{2} sont également créées, respectivement Ant_1 and Ant_2. Des entités de l'application apparaissent sur App_1, respectivement sur App_2 et sont notifiées à

Ant_1 respectivement à Ant_2. Les couleurs initiales sont fixées par la couleur de la ressource de calcul où sont apparues les entités. À chaque fois qu'une entité de l'application App_1 (resp. App_2) interagit avec une entité de App_1 (resp. App_2), Ant_1 (resp. Ant_2) est averti et un arc est créé entre ces entités. Des entités peuvent interagir fortement avec certaines alors qu'elles sont faiblement en relation avec d'autres, créant ainsi des organisations. Si une entité de l'application App_1 (resp. App_2) interagit avec une entité de App_2(resp. App_1), Ant_1 et Ant_2 sont averties. Imaginons maintenant que cette interaction devienne de plus en plus importante, l'arc partagé par Ant_1 et Ant_2 est mis à jour et voit son poids augmenter, une organisation se crée ou une jonction d'organisation est en train de se faire. De plus en plus de fourmis vont emprunter cet arc et il est également possible que de nouveaux arcs apparaissent renforçant encore l'organisation (la réorganisation) naissante. Ces mécanismes vont conduire les fourmis de l'une des couleurs à coloniser l'organisation, des changements de couleur vont donc s'effectuer sur des nœuds du graphe indiquant que les entités devraient migrer si cela est possible en fonction de la stratégie de l'application et des contraintes. Ainsi si il y a des interactions fortes entre des entités, elles vont probablement s'«exécuter» sur la même ressource.

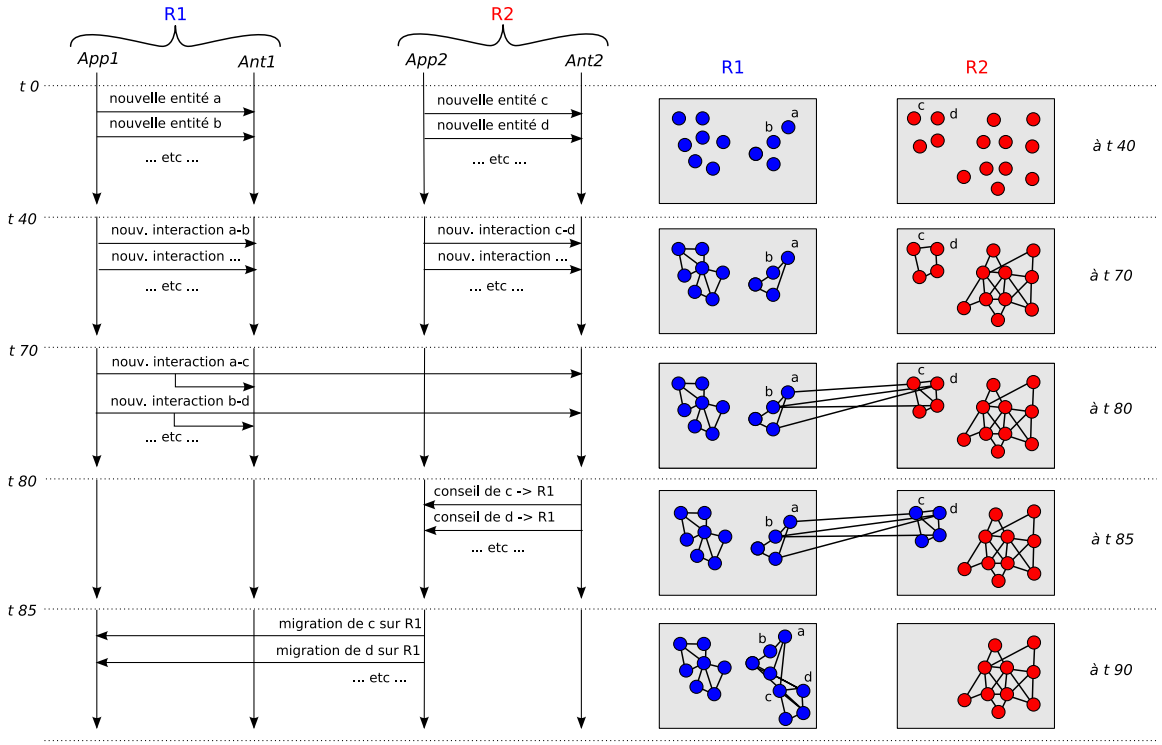


FIG. 4.13 – Diagramme de séquence d'*AntCO*^{2} et une application fonctionnant en parallèle.

De la même façon une organisation, s'exécutant par exemple sur R_1, peut se diviser en deux, alors l'application informe *AntCO*^{2} que l'importance d'un ensemble d'interaction décroît ou disparaît, ceci se traduisant numériquement sur le poids des arcs correspondants. Si l'une des «sous»-organisations résultantes avait des interactions faibles au sens de l'organisation précédente avec une organisation de R_2, elle va pouvoir être colonisée par des fourmis de la couleur de R_2 et *AntCO*^{2} détectera un changement de couleur et

informera l'application App_1 qu'elle devrait faire migrer les entités correspondantes sur R_2.

4.3.7 Expérimentations et résultats

L'équilibrage dynamique de charge entre dans la catégorie des problèmes dépendant très fortement du temps. Il semble donc difficile voire impossible d'effectuer une comparaison des résultats d'$AntCO^2$ avec les solutions optimales sur des graphes dynamiques, à cause de notre incapacité à calculer ces dernières. Supposons maintenant que nous soyons capable de diviser le problème dynamique en une série d'états statiques, trouver une solution optimale est en soit un problème difficile, pour s'en convaincre on pourra consulter [Garey and Johnson, 1979] de plus les solutions optimales obtenues pour t et $t+1$ ne le sont pas du fait des migrations générées par les choix statiques. D'autre part les organisations ne sont pas nécessairement conservées d'un pas à l'autre.

Mesures

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 plus de temps et pafois même d'avantage 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 :

$$r_1 = \frac{\sum_{a \in \mathcal{A}(t)} w^{(t)}(a)}{\sum_{e \in \mathcal{E}(t)} w^{(t)}(e)} \tag{4.17}$$

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 :

$$r_2 = \frac{min(card(\mathcal{V}_1), \ldots, card(\mathcal{V}_n))}{max(card(\mathcal{V}_1), \ldots, card(\mathcal{V}_n))} \tag{4.18}$$

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.

Résultats

Bien qu'*AntCO*^{2} soit prévu à la base pour gérer des applications faites d'un grand nombre d'entités en interaction, et donc particulièrement dynamiques, nous ne l'avons pas testé uniquement sur des graphes dynamiques mais aussi sur des graphes statiques.

Nous ne détaillerons pas la totalité des résultats obtenus, en particulier pour les cas statiques, le lecteur intéressé pourra consulter [68] en particulier sur les points suivants :

- Grilles :
  - Grille avec pondération ;
  - Grille pondérée ;
- Graphes aléatoires ;
- Graphes complets ;
- Graphes invariants d'échelle (figure 4.14), graphes petit-monde ;
- Archive de partitionnements de graphe.

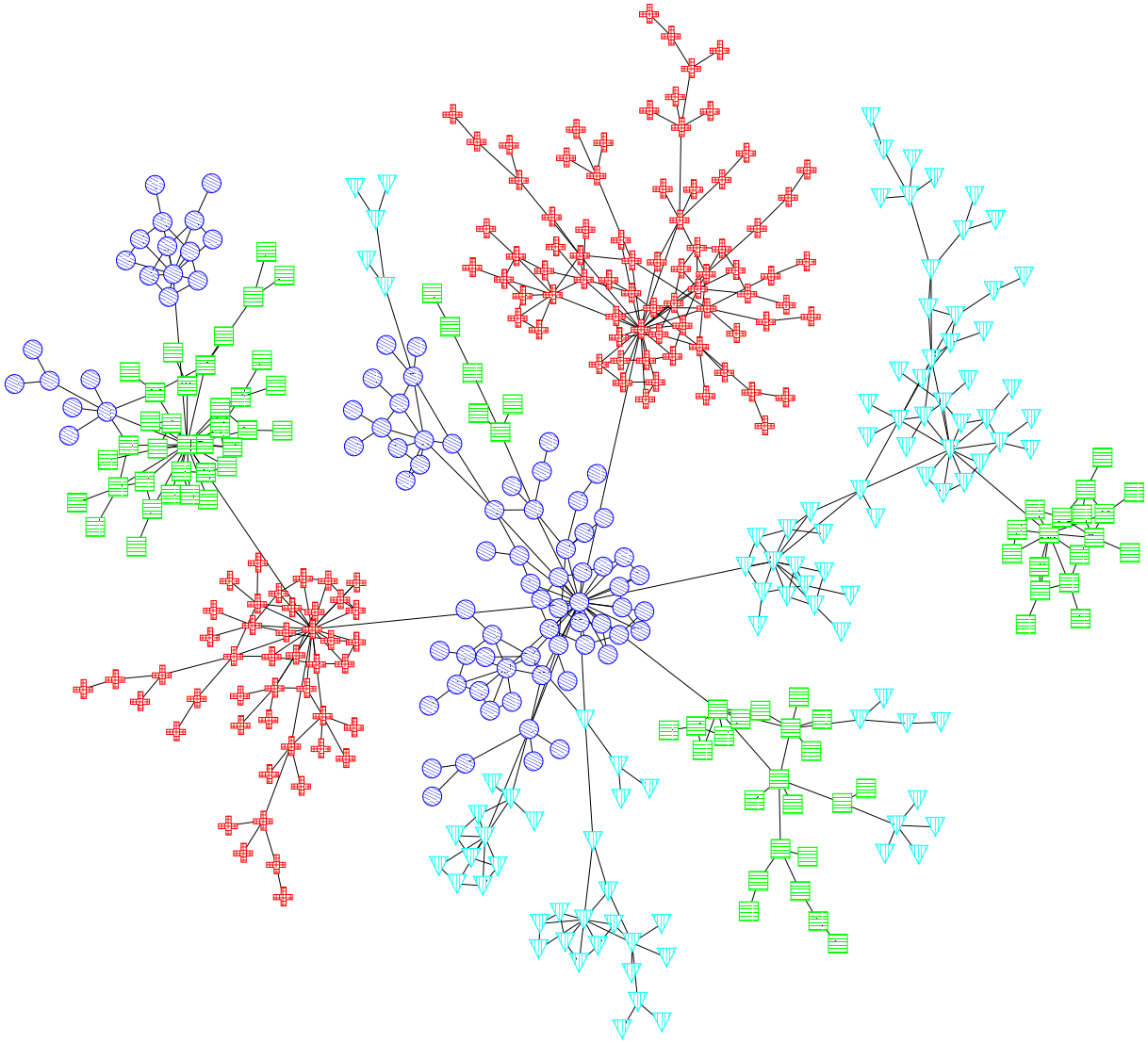
Avant de présenter les résultats obtenus dans le cadre d'une simulation d'un écosystème nous allons détailler deux expériences effectuées avec de la dynamicité. Pour chacun de ces tests une application est simulée en créant un graphe et en appliquant des événements dessus. Les événements sont l'apparition ou la disparition d'un arcs, d'un nœud ou d'une ressource de calcul, mais aussi des modifications au niveau des poids sur les arcs.

Le premier graphe dynamique représente une grille statique que parcourt une autre plus petite. Des connections ont lieu continuellement, lors des déplacements, entre les sommets de deux grilles en fonction de la proximité. Cela représente, par exemple, une organisation qui explore un plus grand ensemble d'entités en interaction (cf. figure 4.15[41].).

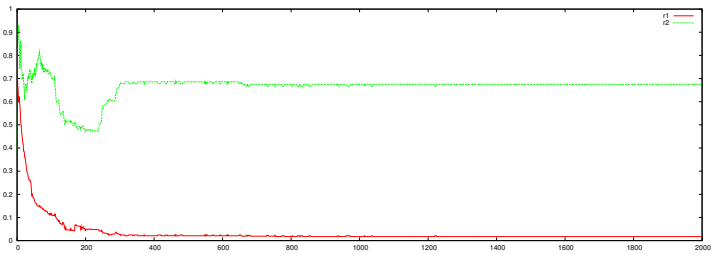
Du fait que les communications dans le petit graphe sont plus fortes que celles avec le grand graphe et que ces dernières durent moins longtemps, le petit graphe garde la même couleur durant l'expérience bien qu'il croise différentes zones de couleurs différentes. L'organisation formée par le petit graphe n'est donc pas perturbé par ses interactions avec la grille. On constate que la dynamique du graphe et la structure sont utilisées pour déterminer les clusters de façon correcte. En effet c'est le comportement dans le temps de la petite grille, avec des apparitions et des disparitions d'interactions, qui permet d'en faire une organisation à part entière Ce graphe est une réduction d'un phénomène rencontré dans les écosystèmes aquatiques dans lesquel un banc de poissons évolue dans un environnement. Les interactions à l'intérieur du banc sont basées sur le cône de vision, et sont plus importantes et durables qu'avec l'environnement.

Pour notre deuxième exemple que nous présentons rapidement, le graphe n'est pas dynamique dans sa topologie, mais par contre l'environnement est changeant, c'est à dire que des ressources de calcul peuvent apparaître ou disparaître à tout moment. On considère que la projection des entités génère une grille comme par exemple dans des

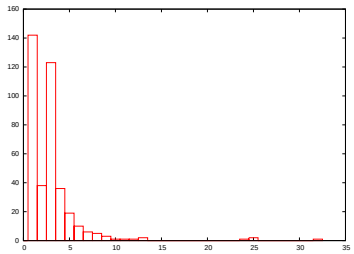
[41] la dynamique peut être observée sur une vidéo [Dutot et al., 2004]



(a) Graphe invariant d'échelle coloré.



(b) Évolution de r_1 and r_2 sur 2000 étapes.



(c) Distribution de degrés.

FIG. 4.14 – Graphe invariant d'échelle de 391 sommets et 591 arcs.

modèles de flux, ou tout autre modèle physique qui utilisent des maillages. Une grille de 30×30 est utilisée (cf. figure 4.16) et on introduit une nouvelle colonie représentant donc

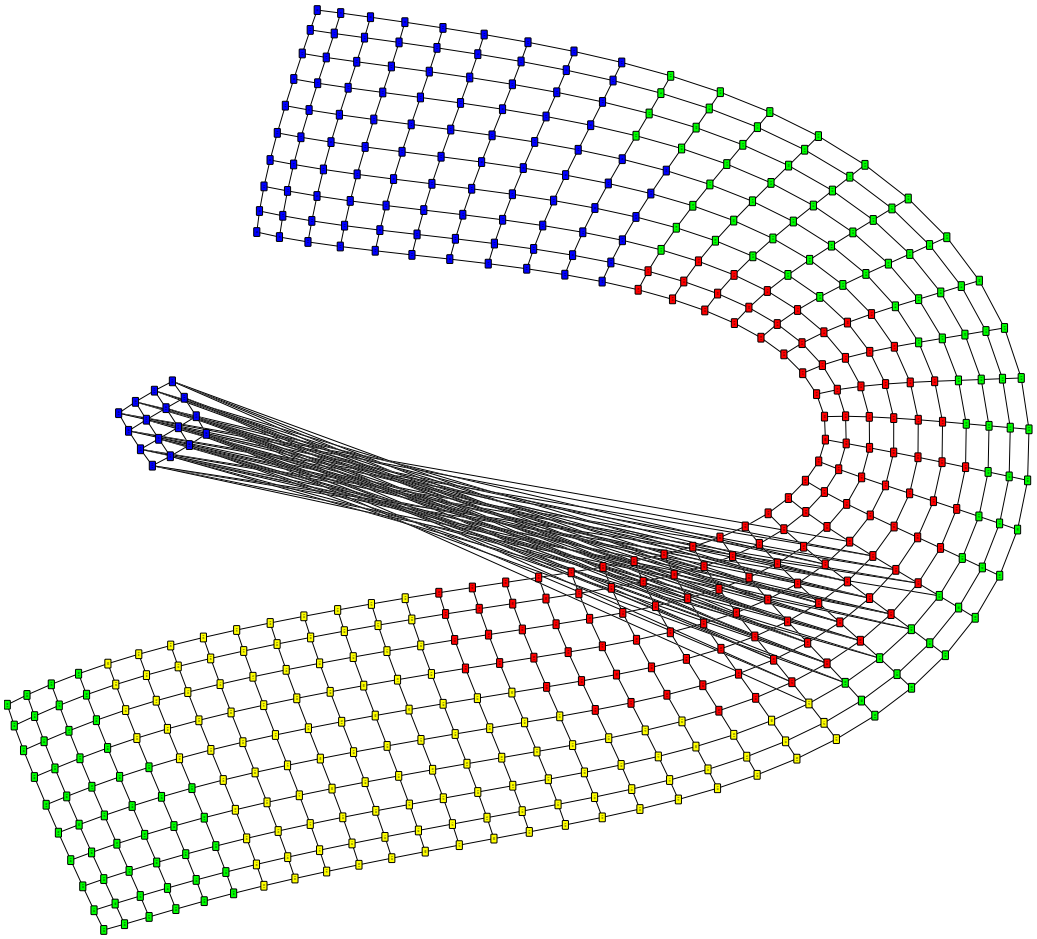


FIG. 4.15 – Organisation interagissant avec une plus grande structure

une nouvelle ressource de calcul toutes les 1000 étapes. Nous présentons le résultat obtenu lorsque le nombre de ressources croît, avec la proposition de répartition et l'évolution de nos deux critères de qualité r_1 et r_2. 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.

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.

Une expérience analogue a également été effectuée sur un graphe de plus grande taille (cf. figure 4.17(a)) issu d'une archive de partitionnement de graphe (2851 sommets et 15093 arcs) [Soper et al., 2004, Soper et al., 2005]. La figure 4.17(b) montre l'évolution des critères r_1 et r_2 avec des résultats similaires à l'expérience précédente.