Simulations distribuées par un algorithme fourmi

Cyrille Bertelle, Antoine Dutot, Frédéric Guinand, Damien Olivier

RENPAR'15 / CFSE'3 / SympAAA'2003
La Colle sur Loup, France, 15 au 17 octobre 2003

Simulations distribuées par un algorithme fourmi

Cyrille Bertelle, Antoine Dutot, Frédéric Guinand et Damien Olivier

LIH - Université du Havre
25 Rue Philippe Lebon - BP 540
76058 Le Havre cedex, France
Email : {Cyrille.Bertelle, Antoine.Dutot, Frederic.Guinand, Damien.Olivier}@univ-lehavre.fr

---

Résumé
On considère des simulations distribuées sur un ensemble de ressources de calcul. Elles sont composées d'un grand nombre d'entités informatiques communicantes en perpétuelle évolution (objets, acteurs, agents ...). À l'aide d'un algorithme fourmi nous détectons les organisations présentes afin de rapprocher par migration les entités qui les constituent, en respectant l'équilibrage de charge. Nous utilisons un graphe dynamique de communication pour modéliser les applications. Plusieurs colonies de fourmis, chacune d'une couleur distincte représentant une ressource (ex : un processeur), entrent en compétition pour marquer les sommets du graphe en utilisant des phéromones colorées. Lorsque la couleur d'un sommet change, cela signifie que l'entité correspondante peut éventuellement migrer, ceci en fonction des contraintes de l'application.

Mots-clés : Algorithme fourmi, distribution dynamique, clustering, auto-organisation.

---

1. Introduction

Les simulations sont souvent utilisées pour étudier des systèmes complexes comme les écosystèmes, le trafic routier, la gestion de crises .... Ils sont constitués d'un grand nombre d'entités dont le comportement ainsi que leurs interactions décrivent la trajectoire du système. Bien que nous soyons en présence de systèmes naturellement distribués du fait de leur nature hétérogène organisée, ce type de problème ne permet de placement statique à cause des aspects dynamiques. En effet des organisations se font et se défont en cours de simulation. Une façon donc de distribuer de telles applications est de détecter les organisations en cours de simulation afin de minimiser en particulier les coûts de communication dans le respect d'un bon équilibrage de charge. Cela revient donc à du clustering dynamique (organisations) sous contraintes (équilibrage). Une fois les clusters détectés les entités sont rapprochées par migration en fonction des contraintes de l'application.

2. Graphe dynamique de communication

La simulation est représentée par un graphe $G = (\mathcal{V}, \mathcal{E})$. $\mathcal{V}$ est l'ensemble des sommets du graphe, ils représentent les entités qui constituent l'application. $\mathcal{E} = \mathcal{V} \times \mathcal{V}$ est l'ensemble des arcs. Chaque arc $e = (v_i, v_j)$ représente les communications entre les entités associées aux sommets v_i et v_j, les communications sont considérées de façon non directionnelle et le graphe n'est donc pas orienté. À l'instant t chaque arc (v_i, v_j) est valué par le poids des communications. Ce poids est une fonction dépendante de l'application, il prend en compte en particulier le volume de communication, mais également par exemple la dynamicité des entités associées aux sommets. Au départ, chaque sommet est affecté à une ressource de calcul. Cette distribution initiale est fixée par la simulation et nous la considérons aléatoire pour le modèle. Les communications ont alors lieu de façon locale ou de façon externe à une ressource de calcul.

Définition 1 (Communication effective)
On appelle communication effective toute communication entre deux entités qui ne sont pas situées sur la même ressource de calcul.

On cherche donc à limiter ces communications effectives en identifiant les clusters constitués d'entités fortement communicantes entre elles tout en tenant compte de l'équilibrage de charge. Le modèle de graphe retenu est dynamique, le poids des arcs peut varier durant le calcul et des sommets apparaissent et disparaissent modifiant en profondeur les propriétés du graphe (la connexité par exemple). Une approche monotone permet de rechercher les clusters dans le graphe, mais il faut alors effectuer le traitement sur une copie gelée du graphe alors que le celui-ci évolue. Le graphe risque d'être modifié de façon significative et les résultats obtenus ne sont plus utilisables à cause de divergence entre l'état réel de l'application et les propositions de migration. De plus, à chaque gel l'algorithme est appliqué.
Il nous faut donc à la fois un algorithme non monotone, incrémental et anytime, la simulation continuant à se dérouler durant le calcul afin qu'à tout instant une solution puisse être proposée. Le graphe dynamique constitue l'environnement des entités qui composent l'application. Il prend en compte les changements dès qu'ils apparaissent. Ces changements permanents sont l'une des principales motivations dans l'utilisation d'un algorithme fourmi. En effet, cette classe d'algorithme est bien adaptée pour traiter ce type de problème et respecter les contraintes énoncées précédemment.

3. Algorithmes fourmis

Ces algorithmes appartiennent à la classe des algorithmes approchés itératifs. Ils sont basés sur le comportement naturel des sociétés d'insectes collectifs où une solution peut émerger par les interactions souvent indirectes (à travers l'environnement) entre individus[6]. Les fourmis déposent des *phéromones*, qui vont plus ou moins attirer les autres en fonction de leur quantité. Les phéromones s'évaporant, seuls les chemins régulièrement empruntés subsistent, les plus longs chemins tendant à disparaître. C'est une technique robuste, dynamique, distribuée et n'utilisant que des informations locales. Elle a déjà été appliquée avec succès à différents problèmes d'optimisation combinatoire [4], routage dynamiquen séquencage de l'ADN, partitionnment de graphe.

4. Distribution dynamique par un algorithme fourmi coloré

L'algorithme doit à la fois minimiser les communications effectives et respecter l'équilibrage de charge. Ces deux contraintes sont antagonistes, en effet pour minimiser les communications effectives, il suffit de placer la totalité des entités sur une unique ressource de calcul mais alors la charge est concentrée sur cette unique ressource. L'algorithme fourmi, dans notre cadre, est utilisé pour détecter les clusters regroupant les entités qui communiquent beaucoup. Les fourmis numériques sont alors fortement attirées par les forts poids des arcs et les phéromones comme on le verra par la suite. Par ailleurs on introduit un mécanisme de compétition entre les fourmis en attribuant à chaque ressource de calcul une couleur et des << fourmis recruteuses >> de sa couleur. On parlera dans la suite de *fourmis colorés* et de leur *phéromone de couleur*. Pour présenter notre algorithme nous étendons la définition proposée au paragraphe 2.

Définition 2 (Graphe dynamique coloré de communication)
Un graphe dynamique coloré de communication est un graphe valué non orienté $G = (\mathcal{V}, \mathcal{E}, \mathcal{C})$ *tel que :*
- $\mathcal{C}$ *est un ensemble de* p *couleurs,* p *est le nombre de ressources de calcul.*
- $\mathcal{V}$ *est l'ensemble des sommets. Chaque sommet possède une couleur appartenant à* $\mathcal{C}$*.*
- $\mathcal{E}$ *est l'ensemble des arcs. Chaque arc est valué par un poids. Un poids* $w(u, v) \in \mathbb{N}^+$ *associé à un arc* $(u, v) \in \mathcal{V} \times \mathcal{V}$ *est fonction des communications entre les entités associées aux sommets* u *et* v*.*

La figure 1 (graphe de gauche) montre un exemple de graphe dynamique coloré de communication. La couleur d'un sommet est modifiée si cette modification minimise les communications effectives et/ou la charge de la ressource de calcul. L'algorithme doit colorer les sommets constituant des clusters possédant des forts poids de communication, comme c'est montré sur le graphe de droite (fig. 1) qui est une solution de la figure 1 avec par exemple les sommets verts a, b, c, d, e, f, g, h. Ces sommets peuvent changer de couleur ensuite en fonction de l'évolution de la simulation.

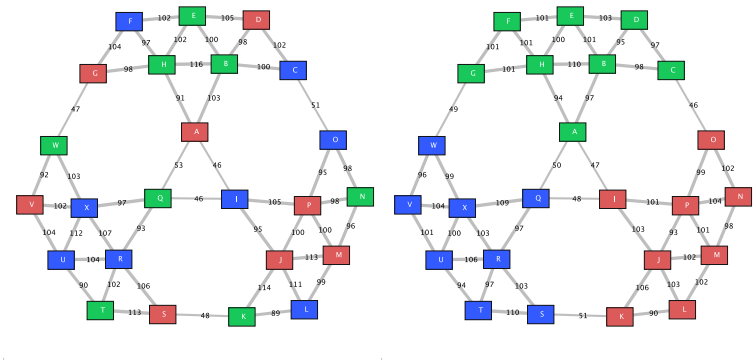


FIG. 1 – Exemple de graphe dynamique coloré de communication à l'itération 0 et l'itération 45

4.1. Algorithme fourmi coloré

L'algorithme que nous proposons [2] est basé sur Ant System de Dorigo [4]. Il sert de base à nos expérimentations et nous présenterons dans la suite des améliorations (cf. 4.3) au niveau de la gestion des populations de fourmis. L'algorithme se décompose de la façon suivante :

1. Soit un graphe dynamique coloré de communication $G = (\mathcal{V}, \mathcal{E}, \mathcal{C})$.
2. À chaque ressource de calcul on affecte une couleur, cette couleur est donnée initialement aux sommets qui représentent les entités situées sur cette ressource. Pour chaque ressource de calcul de couleur $c \in \mathcal{C}$ un nombre de fourmis de couleur c, proportionnel à la puissance de calcul est uniformément réparti sur les sommets de couleur c.
3. L'algorithme est basé sur un processus itératif. Les fourmis parcourent à chaque pas de temps un arc entre deux sommets et déposent à cette occasion des phéromones de leur couleur. De plus chaque fourmi mémorise le sommet d'où elle vient. On définit :
   - La quantité de phéromones de couleur c déposée par une fourmi x sur l'arc (u, v) entre les pas de temps $t-1$ et t : $\Delta_x^{(t)}(u, v, c)$.
   - La quantité de phéromones de couleur c déposée par les fourmis sur l'arc (u, v) entre les pas de temps $t-1$ et t est égale à :
   $$\Delta^{(t)}(u, v, c) = \sum_{x \in \mathcal{F}} \Delta_x^{(t)}(u, v, c) \tag{1}$$
   Où $\mathcal{F}$ représente l'ensemble de toutes les fourmis.
   - La quantité totale de phéromones déposée sur l'arc (u, v) entre les pas de temps $t-1$ and t est égale à :
   $$\Delta^{(t)}(u, v) = \sum_{c \in \mathcal{C}} \Delta^{(t)}(u, v, c) \tag{2}$$
   - Si $\Delta^{(t)}(u, v) \neq 0$, le taux de phéromones de couleur c sur l'arc (u, v) entre les pas de temps $t-1$ et t est égal à :
   $$K_c^{(t)}(u, v) = \frac{\Delta^{(t)}(u, v, c)}{\Delta^{(t)}(u, v)}, \text{ et on a } K_c^{(t)}(u, v) \in [0, 1] \tag{3}$$
4. La quantité courante de phéromones de couleur c présente sur l'arc (u, v) au pas t est notée $\tau^{(t)}(u, v, c)$. Sa valeur initiale ($t = 0$) est 0, sinon

   $\tau^{(t)}(u, v, c) = \rho\tau^{(t-1)}(u, v, c) + \Delta^{(t)}(u, v, c)$, où $\rho \in [0, 1]$ modélise le mécanisme d'évaporation

5. À cette étape de l'algorithme nous avons calculé la quantité courante de phéromones $\tau^{(t)}(u, v, c)$. Ces phéromones de couleur sont utilisées comme un facteur de renforcement pour détecter les clusters en construisant des chemins colorés. Il nous faut maintenant prendre en compte l'équilibrage de charge. Pour cela il nous faut contrebalancer le facteur de renforcement initial $\tau^{(t)}(u, v, c)$

avec $K_c^{(t)}(u,v)$, l'importance relative instantanée d'une couleur par rapport aux autres. Ce facteur de renforcement utilisant $K_c^{(t)}(u,v)$ peut rendre instable le processus de calcul. Nous regardons donc l'importance relative d'une couleur par rapport aux autre couleurs dans une fenêtre de temps $q \in \mathbb{N}^+$. Cela donne :

$$K_c^{(t,q)}(u,v) = \sum_{s=t-q}^{t} K_c^{(s)}(u,v). \tag{4}$$

Le facteur de renforcement est donc :

$$\Omega^{(t)}(u,v,c) = K_c^{(t,q)}(u,v)\tau^{(t)}(u,v,c) \tag{5}$$

6. Nous avons défini, l'évolution des phéromones sur les arcs du graphe dans ce qui précède, il nous faut maintenant préciser les déplacements des fourmis. Les déplacements se font en fonction d'une probabilité calculée à partir des phéromones présentes et des poids des arcs. Soit $p(u,v_k,c)$ la probabilité qu'une fourmi de couleur c emprunte l'arc (u,v_k) incident au sommet u et dont le poids est $w(u,v_k)$.
   - A $t = 0$,

$$p(u,v_k,c) = \frac{w(u,v_k)}{\sum_{v \in \mathcal{V}_u} w(u,v)} \tag{6}$$

   - Pour $t \neq 0$,

$$p(u,v_k,c) = \frac{(\Omega^{(t)}(u,v_k,c))^{\alpha}(w(u,v_k))^{\beta}}{\sum_{v_q \in \mathcal{V}_u} (\Omega^{(t)}(u,v_q,c))^{\alpha}(w(u,v_q))^{\beta}} \tag{7}$$

   Où $\mathcal{V}_u$ est l'ensemble de tous les sommets adjacents à u. Les paramètres α et β permettent d'équilibrer l'influence des phéromones et le poids des communications. Le choix de l'arc emprunté par une fourmi dépend donc de la probabilité $p(u,v_k,c)$. Toutefois, pour éviter les oscillations entre deux sommets nous avons introduit un facteur de pénalisation $\eta \in [0,1]$. Soit $\overline{v}_x$ le dernier sommet visité par la fourmi x, le calcul de probabilité devient :

$$p_x(u,v_k,c) = \frac{(\Omega^{(t)}(u,v_k,c))^{\alpha}(w(u,v_k))^{\beta}\eta_{x,k}}{\sum_{v_q \in \mathcal{V}_u} (\Omega(t)(u,v_q,c))^{\alpha}(w(u,v_q))^{\beta}\eta_{x,q}} \tag{8}$$

   Où

$$\eta_{x,q} = \begin{cases} 1 & \text{si } v_q \neq \overline{v}_x \\ \eta & \text{si } v_q = \overline{v}_x \end{cases} \tag{9}$$

7. La couleur d'un sommet u, notée $\xi(u)$ est définie par la principale couleur de ses arcs incidents :

$$\xi(u) = \arg\max_{c \in \mathcal{C}} \sum_{v \in \mathcal{V}_u} \tau^{(t)}(u,v,c) \tag{10}$$

4.2. Qualité de la solution

Il est nécessaire d'évaluer la qualité de la solution pour l'améliorer et suivre l'aspect dynamique de l'application. Nous devons prendre en compte à la fois le coût global des communications et l'équilibrage de charge. Ces deux critères sont opposés, aussi pour évaluer les solutions obtenues nous avons défini deux critères de qualité r_1 et r_2. Le premier critère r_1 permet de savoir entre deux solutions laquelle présente proportionnellement le moins de communications effectives. Nous avons :

$$r_1 = \frac{e}{s} \text{ où } \begin{cases} e & \text{volume global des communications effectives} \\ s & \text{volume global des communications totales} \end{cases}$$

Plus r_1 est proche de 0, plus le volume de communications effectives est faible, ce qui est l'un des objectifs recherchés. Le deuxième critère r_2 prend en compte l'équilibrage de charge. Soit ν_c le nombre de sommets de couleur c et p_c la puissance de la ressource de calcul affectée à c. On définit :

$$r_2 = \frac{min\mathcal{E}}{max\mathcal{E}} \quad \text{avec} \quad \mathcal{E} = \left\{ \frac{\nu_c}{p_c}; c \in \mathcal{C} \right\}$$

Plus r_2 est proche de 1, plus l'équilibrage est bon. Sur l'exemple (figure 1) on obtient $r_1 = 0.7$, $r_2 = 1.0$ pour $t = 0$ et $r_1 = 0.1$, $r_2 = 1.0$ pour $t = 45$.
Ces critères sont utilisés durant le calcul pour vérifier l'amélioration de la solution, mais également pour conserver la meilleure solution.

4.3. Gestion des populations des fourmis
Il est difficile de calibrer les paramètres de l'algorithme en particulier α et β. Lorsque ceux-ci ne sont pas fixés correctement l'algorithme bloque alors sur des minima locaux. En effet, on constate que pour les clusters composés de sommets qui communiquent beaucoup, les fourmis ont tendance à suivre des chemins privilégiés qui forment des *boucles*. Cela est dû en général, au fait que les poids des arcs sont voisins et dans ce cas les phéromones prennent une part prépondérante dans le choix des chemins. Des sommets sont ignorés alors qu'ils devraient être traversés pour fixer leur couleur. Ceci conduit à trois catégories de problèmes : *l'encerclement*, *la famine* et *la surpopulation*. Nous détaillons cela sur trois figures (cf. figure 2) extraites du graphe présenté sur la figure 1. Chaque sommet est étiqueté par une lettre et le nombre de fourmis présentes alors que sur les arcs figure la quantité de phéromones.

L'encerclement : Un petit groupe de fourmis d'une couleur donnée est encerclé par un groupe de fourmis en plus grand nombre d'une autre couleur. Dans ce cas, elles ne peuvent pas s'échapper à cause des facteurs de répulsion et elles ne peuvent pas aller coloniser ou renforcer d'autres clusters de leur couleur. La figure 2 montre des fourmis rouges qui ne peuvent se déplacer vers le cluster rouge (qui ne figure pas ici), elles sont encerclées par les bleus. Ce phénomène se renforce, les fourmis rouges sont attirées par leur cycle et les bleus les repoussent les forçant à rester dans la boucle.

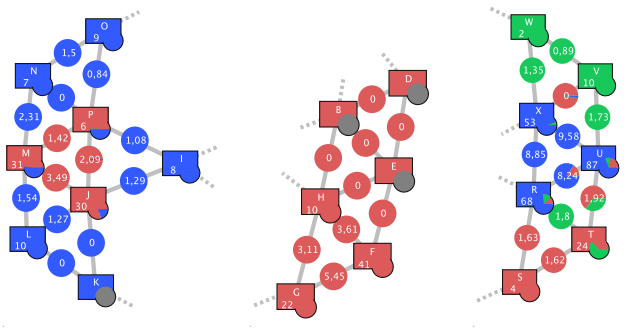
La famine : Des parties du graphe ont de moins en moins de phéromones et sont de moins en moins souvent parcourues par des fourmis, ce phénomène s'amplifie encore avec le mécanisme d'évaporation. La figure 2 montre ce phénomène où les fourmis rouges restent dans une petite boucle en bas laissant toute une partie inoccupée, alors que le poids des arcs est important.

La surpopulation : Le nombre de fourmis en cours de calcul n'est pas borné, on peut donc voir apparaître des phénomènes de surpopulation dans une partie du graphe alors qu'il y a sous population dans une autre partie. Dans le pire des cas, le déséquilibre de population empêche le facteur de répulsion d'opérer correctement. La figure 2 montre ce phénomène où la distribution initiale avait alloué 20 fourmis pour chaque sommet. Il y a maintenant plus de 50 fourmis sur chaque sommet bleu, ces fourmis tournent dans une boucle. Plusieurs autres parties du graphe sont vides. En jouant sur les paramètres α et β il est possible de corriger ce déséquilibre.

Pour sortir des extrema locaux nous avons introduit des mécanismes qui permettent la naissance et la mort de fourmis. Cela permet de perturber la répartition des fourmis, bloquées dans un minimum local qui conduit à la détection de petits clusters stables. Ces mécanismes sont également utilisés pour gérer la création et la suppression de sommets et d'arcs en cours de simulation. Ceci s'intègre donc dans notre approche qui tente de proposer une solution à tout moment dans un environnement en perpétuelle évolution. Pour mettre en œuvre le mécanisme de gestion de population des fourmis nous apportons les modifications suivantes à l'algorithme :

1. On définit les paramètres :
   - La quantité de phéromones de couleur c déposée sur tous les arcs incidents au sommet u :

$$\tau^{(t)}(u, c) = \sum_{\nu_q \in \mathcal{V}_u} \tau^{(t)}(u, \nu_q, c) \tag{11}$$



Encerclement Famine Surpopulation

FIG. 2 – Différents problèmes liés aux boucles

- La quantité de phéromones déposée sur tous les arcs incidents au sommet u :

$$\tau^{(t)}(u) = \sum_{c \in \mathcal{C}} \tau^{(t)}(u, c) \tag{12}$$

- L'importance relative de la phéromone de couleur c par rapport aux autres phéromones déposées sur les arcs incidents à u :

$$\varphi_c(u) = \frac{\tau^{(t)}(u, c)}{\tau^{(t)}(u)}, \text{ on a } \varphi_c(u) \in [0, 1] \tag{13}$$

2. Nous introduisons ici, le mécanisme de mort et de naissance. La population est constante pour un nombre de sommets donné. À chaque étape il est nécessaire de déterminer si une fourmi meurt ou pas. On détermine cela à l'aide d'un paramètre $\phi \in [0, 1]$. Pour une fourmi de couleur c sur un sommet u :
   - Si $\varphi_c(u) < \phi$, la fourmi meurt et une nouvelle naît sur un sommet x. Pour cela, nous sélectionnons aléatoirement un ensemble $\mathcal{V}_n$ de n sommets. Le sommet choisi $x \in \mathcal{V}_n$ pour faire naître la fourmi, satisfait à :

$$x = \arg\min_{v \in \mathcal{V}_n} (\text{card}(\mathcal{F}(v))) \tag{14}$$

     où $\text{card}(\mathcal{F}(v))$ est le nombre de fourmis sur le sommet v.
   - Sinon un arc est choisi pour être traversé par la fourmi (équation 6 et suivantes).

Ceci élimine le problème d'encerclement et de famine, les fourmis meurent lorsqu'elles sont encerclées et naissent dans des zones de famine. Cela résout également les problèmes de surpopulation puisque ces derniers conduisent à un encerclement. Cependant, ceci n'élimine pas totalement le problème des boucles qui tendent à réapparaître. Pour résoudre cela les fourmis peuvent mémoriser plusieurs sommets et ainsi modifier l'équation 8.

5. Comparaison

La comparaison de l'algorithme CAS avec d'autres techniques de répartition est rendu malaisée par des approches différente en fonction du domaine : parallélisme, systèmes d'exploitation répartis, bases de données réparties, objets répartis... Pour comparer l'algorithme CAS avec les autres techniques de répartition automatique, nous définissons plusieurs critères classés en quatre catégories :

local ou global L'algorithme est local si aucun contrôle central n'est utilisé.

adaptabilité structurelle L'adaptabilité est la capacité de l'algorithme à prendre en compte dynamiquement, les variations du nombre d'entités à distribuer, du nombre de ressources de calcul, les changements de topologie du réseau connectant ces ressources et des liens entre les entités.

dynamisme d'exécution Ceci regroupe les capacités :

- **anytime** C'est la capacité à fournir un résultat *exploitable* à tout moment.
- **non-monotone** L'algorithme peut modifier un résultat établi en cours de résolution lorsque des informations changent.
- **incrémental** Les calculs à l'itération i sont effectués à partir des calculs de l'itération $i-1$.

detection d'organisations L'algorithme ne prend pas uniquement en compte que la charge des ressources de calcul, mais aussi les relations entre entités afin de mieux les répartir. Ces relations peuvent être :

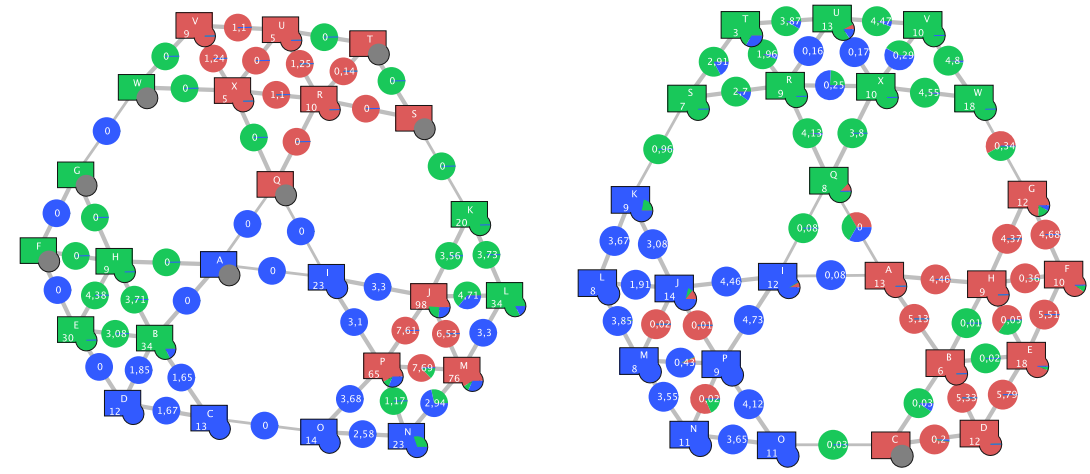
- des contraites (e.g. attachement à des ressources),
- des communications (interdépendances, limitation du coût des communications).

Le regroupement suivant ces liens forme des organisations.

	local	adaptabilité			exécution			detect. org.	
		entités	ressources	topologie	anytime	non-monotone	incrémental	comms.	liens
CAS	×	×	×	×	×	×	×	×	×
Aléatoire	×	×	×	0	s.objet	s.objet	0	s.objet	s.objet
Paire[3]	×	×	×	/	×	s.objet	×	s.objet	s.objet
Vecteur[1]	×	×	×	/	×	s.objet	×	s.objet	s.objet
Drafting[8]	×	×	×	0	×	×	0	s.objet	s.objet
Gradient[7]	0	×	×	0	0	0	0	s.objet	s.objet
Enchères[5, 9]	×	×	×	/	×	0	0	s.objet	s.objet
Round-robin (PVM)	×	×	×	0	s.objet	s.objet	0	s.objet	s.objet

6. Résultats

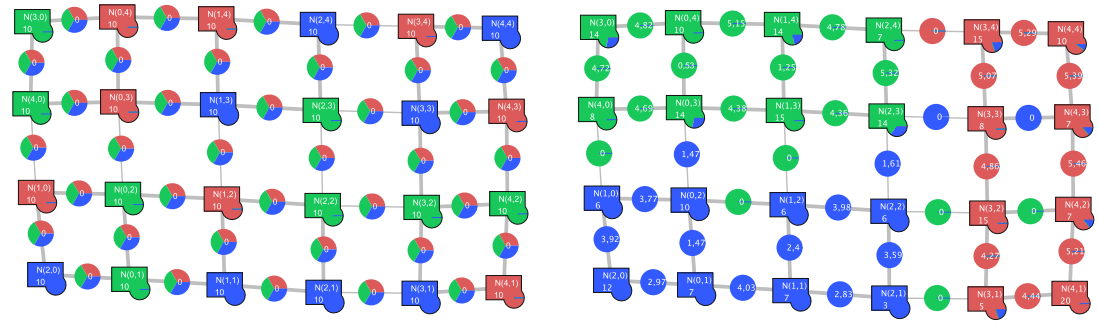
L'algorithme de base (cf. 4.1) a été testé sur de nombreux graphes (aléatoire, dense, ...) et donne souvent de bons résultats. Nous présentons ici un graphe (figure 1) qui a mis en évidence les problèmes développés en 4.3. Ainsi l'algorithme détermine une solution éloignée de la bonne (figure 3 à gauche avec des paramètres $\alpha = 1$, $\beta = 4$, $\eta = 0.0001$, $\rho = 0.8$. Ce résultat est obtenu après 30 pas et reste identique après 570 pas supplémentaires.
Lorsque nous utilisons l'algorithme modifié (cf. 4.3) sur le même problème avec les mêmes paramètres et le paramètre supplémentaire $\phi = 0.3$ et une mémorisation de 4 sommets une solution correcte est obtenue après 23 itérations (figure 3 droite) et reste stable après 200 itérations.



Algorithme de base (600 pas) Algorithme modifié (23 pas)

FIG. 3 – Graphe de test avec l'algorithme de base et l'algorithme modifié

Nous avons également testé l'algorithme sur une grille et nous obtenons les résultats présentés sur la figure 4.



Configuration initiale Configuration après 100 pas

FIG. 4 – Calcul sur une grille

7. Conclusion et perspectives

Nous avons présenté un algorithme qui permet une distribution dynamique en proposant des conseils de migration. Nous n'avons pas développé l'aspect dynamique dans ce papier, mais les tests que nous avons déjà effectués montrent que l'algorithme fonctionne correctement dans un environnment dynamique à la fois au niveau du graphe de communication mais également au niveau des ressources de calcul. En effet l'introduction ou la suppression d'une nouvelle couleur est correctement prise en compte. Des tests ont également été réalisés sur des graphes de plus grande taille. Enfin nous développons une couche heuristique permettant de prendre en compte les contraintes liées à l'application comme par exemple des entitées non migrables (ex : utilisant une base de données), mais aussi des informations propres à l'application.

Bibliographie

1. A. Barak and A. Shiloh. A distributed load-balancing policy for a multicomputer. *Software Practice and Experience*, 15(9) :901–913, July 1985.
2. C. Bertelle, A. Dutot, F. Guinand, and D. Olivier. Distribution of agent based simulation with colored ant algorithm. In *ESS2002*, pages 39–43, Dresden (Germany), 2002.
3. R. Bryant and R. Finkel. A stable distributed scheduling algorithm. In *2nd Int. Conf. on Distributed Computer Systems*, pages 314–323, April 1981.
4. M. Dorigo, V. Maniezzo, and A. Colorni. The ant system : optimization by a colony of cooperating agents. *IEEE Trans. Systems Man Cybernet.*, 26 :29–41, 1996.
5. D.J. Farber. The distributed computing system. In *Compcon Spring 73*, pages 31–34, 1973.
6. C.G. Langton, editor. *Artificial Life*. Addison Wesley, 1987.
7. F. Lin and R. Keller. Gradient model : A demand-driven load balancing scheme. *IEEE*, pages 329–336, 1986.
8. L.M. Ni, C.W. Xu, and T.B. Gendreau. A distributed drafting algorithm for load balancing. *IEEE Trans. on soft. engineering*, SE-11(10) :1153–61, October 1985.
9. Laurent Philippe. *Contribution à l'étude et à la réalisation d'un système à image unique pour multi-calculateur*. PhD thesis, Université de Franche-Comté, Besançon, Janvier 1993.