Colored Ants for Distributed Simulations

Cyrille Bertelle Antoine Dutot Frédéric Guinand Damien Olivier

Laboratoire d'Informatique du Havre
Université du Havre
25 rue Philippe Lebon
76600 Le Havre
email: Antoine.Dutot@univ-lehavre.fr

Abstract. A simulation may be modeled as a set of interacting entities within an environment. Such applications can be represented by a graph which evolves with a one-to-one mapping between vertices and entities and between edges and communications. As for classical applications, performances depend directly on a good load balancing of the entities between available computing devices and on the minimization of the impact of the communications between them. However, both objectives are contradictory and good performances may be achieved if and only if a good trade-off is found. Our method for finding such a trade-off leans on an extension of the ant-based approach. We use competing colonies of ants, each depositing distinctly colored pheromones, to find clusters of highly communicating entities. Ants are attracted by communications and their own colored pheromones, while repulsion interactions between colonies allow to preserve a good distribution. Colors are mapped to processing resources and colored entities are placed on them, allowing to put highly communicating clusters on the same processing resource without overloading it. Several graphs categories (grids, scale-free, random, static or not) are studied.
Keywords: Colored Ant algorithms, dynamic interaction graph, self-organization, clustering, distributed simulation, load balancing .

1 Introduction

Complex system simulations need a very large number of entities and we deal with their implementation on distributed computing environment. The induced interaction graph and its dynamic discourages a static distribution before application execution. As the system evolves communications between entities change. Communications and entities may appear or disappear, creating stable or unstable organizations. As a consequence, an entity location that was correct at the beginning, can severely impact performance later. Therefore we need an anytime distribution method that advices the application on better locations for each entity preserving load-balancing between computing resources, but ensuring that entities that communicate heavily are close together (ideally on the same processing resource). In this paper, a method based on the Ant System[4] is described that recommends on a possible better location of some entities according to the trade-off between load balancing and minimization of communications overhead.
The Paper is organized as follows. Section 2 details the dynamic graph model, the colored ant algorithm. It describes also the dynamic aspects and previous works. In section 3, some local corrections on the previous algorithm are studied. Finally, our model is implemented and illustrated by some experiments described in section 4. In section 5, we conclude with further expected improvements and perspectives for this system.

2 Graph Model and General Algorithm

The model uses a simulation context based on an environment where reactive agent populations act. The environment corresponds to a graph and the population is constituted with numerical ants which move and interact on this graph, following an original colour based algorithm.

2.1 Dynamic Communication Colored Graph

We model the application by a graph $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is a set of vertices representing entities of the application and $\mathcal{E} = \mathcal{V} \times \mathcal{V}$ is a set of edges $e = (v_i, v_j)$ representing communications between entities associated to vertices v_i and v_j. Communications direction being without effect on the ant algorithm, edges are undirected. Edges are labeled by weights which correspond to communication volumes (and possibly more attributes). Each vertex is assigned to an initial processing resource at start. No assumption is made about this initial mapping.

We distinguish two different kinds of communications. On the one hand communications occurring between entities located on the same computing resource are supposed negligible. On the other hand, communications between entities located on distinct computing devices, called *actual communications*, constitute the source of the communication overhead. Our goal is to reduce the impact of actual communications by identifying sets of highly communicating entities in order to map all entities belonging to one set on the same computing resource. Of course, one trivial solution is obtained by mapping all entities on only one computing resource but this solution doesn't respect the load-balancing. In order to avoid this, we use several colored ant colonies that produce colored sets, each color corresponding to one computing resource. An extension of ant algorithm is used to detect clusters of highly commu-

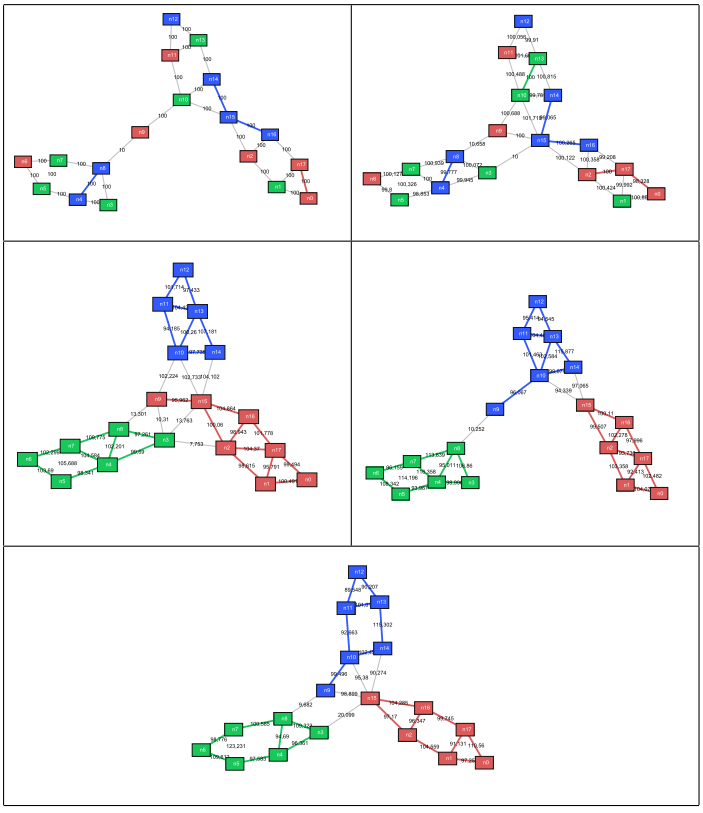


Fig. 1. *Example of a dynamic communication graph at five stages of its evolution*

nicating entities. To solve load balancing problems we introduce *colored ants* and *colored pheromones* that correspond to available processing resources. To suit our algorithm we extend our graph definition:

Definition 1 (Dynamic Communication Colored Graph). *A dynamic communication colored graph is a weighted undirected graph $G = (\mathcal{V}(t), \mathcal{E}(t), \mathcal{C}(t))$ such that:*

- *$\mathcal{C}(t)$ is a set of p colors where p is the number of processing resources of the distributed system at time t.*
- *$\mathcal{V}$(t) is the set of vertices at time t. Each vertex has a color belonging to $\mathcal{C}(t)$.*
- *$\mathcal{E}$(t) is the set of edges at time t. Each edge is labelled with a weight. A weight $w(t, u, v) \in \mathbb{N}^+$ associated with an edge $(u, v) \in \mathcal{V}(t) \times \mathcal{V}(t)$ corresponds to the importance of communications between the couple of entities at time t, corresponding to vertices u and v.*

The figure 1 shows an example of a dynamic communication colored graph at several steps of its evolution. The proposed method changes the color of vertices if this change can improve communications or processing resource load. The algorithm tries to color vertices of highly communicating clusters with the same colors. Therefore a vertex may change color several times, depending on the variations of data exchange between entities.

2.2 Dynamic aspects and load balancing

The graph represents the application as it runs, it is therefore dynamic at several levels:

- weights change continuously;
- edges, representing communications, can appear and disappear at any time;
- vertices, representing entities, can appear and disappear at any time.
- processing resources can appear or disappear and change power at any time.

These changes in both topology and valuation are one of the major motivation for using ant algorithms.

So, the problem may be formulated as the on-line allocation, reallocation and migration of tasks to computing resources composing a distributed environment that may be subject to failure.

There exist many variants of this problem which has been extensively studied, and two main classical approaches may be distinguished. The static one consists in computing at compile-time an allocation of the tasks using the knowledge about the application and about the computing environment. This problem is known as the mapping problem [2]. A global optimization function, representing both load-balancing and communication overhead minimization, should be decreased. Strategies for achieving this objective often leans on graph partitioning techniques using optimization heuristics like simulated annealing, tabu search or evolutionary computing methods. However this approach is not suitable as-is in our context since we don't have information at compile-time.

The second approach is dynamic load-balancing. Many works have been dedicated to the special case of independent tasks [5,10,12]. When communications are considered, the problem is often constrained by some information about the precedence relation between tasks. In [8], for instance, the proposed heuristic, named K1, for scheduling a set of tasks with some dynamical characteristics takes into consideration precedence tasks graphs with inter-tasks communication delays, but, the execution of the task set is supposed to be repeated in successive cycles and the structure of the precedence task graph remains fixed, and only numerical values (execution times and communication delays) may change during the execution. When no information is available about the precedence relation between tasks, Heiss and Schmitz have proposed a decentralized approach based on a physical analogy by considering tasks as particles subject to forces [7]. Their application model is similar to our, but they do not consider changes that may happen to the execution environment. Moreover, their approach is based on local negotiations between neighbors resources and may spread out, depending of the load.

Ant-based algorithms have been proposed by Kuntz et al. [9] for graph partitioning, able to detect "natural" clusters within a graph and it also has the ability of gathering vertices such that:

1. if they belong to the same cluster they are gathered in the same place,
2. the number of inter-cluster edges is minimized and,
3. distinct clusters are located in different places in the space.

Points 1. and 2. are relevant for our application, however, additional issues have to be considered:

1. the number of natural clusters may be different of the number of processing resources,
2. the sum of the sizes of the clusters allocated to each processing resource has to be proportional to its power,
3. and their number as well as the graph structure may change.

For the first issue, in [3], the authors mentioned the possibility of parameterize the KLS algorithm in order to choose the number of clusters to build. Unfortunately, for our application, we do not know in advance what should be the best number of clusters for achieving the best load balancing, as shown in Figure 3, where elements of the two opposite and not directly linked clusters should be allocated to the same processing resource.

2.3 Color Ant Algorithm

Our algorithm is inspired by the Ant System[4]. We consider a dynamic communication colored graph G.

- Each processing resource is assigned to a color. Each vertex gets its initial color from the processing resource where it appears. For each processing resource, ants are allocated as proportionally to its power.
- The algorithm is based on an iterative process. Between steps $t-1$ and t, each ant crosses one edge and reaches a new vertex. During its move, it drops pheromone of its color on the crossed edge.
  We define the following elements:
  - The quantity of pheromone of color c dropped by one ant x on the edge (u, v), between the steps $t-1$ and t is noted $\Delta_x^{(t)}(u, v, c)$.
  - The quantity of pheromone of color c dropped by the ants when they cross edge (u, v) between steps $t-1$ and t is noted:

$$\Delta^{(t)}(u, v, c) = \sum_{x \in \mathcal{F}(t)} \Delta_x^{(t)}(u, v, c) \tag{1}$$

    where $\mathcal{F}(t)$ is the ant population at time t
  - The total quantity of pheromone of all colors dropped by ants on edge (u, v) between steps $t-1$ and t is noted:

$$\Delta^{(t)}(u, v) = \sum_{c \in \mathcal{C}(t)} \Delta^{(t)}(u, v, c) \tag{2}$$

  - If $\Delta^{(t)}(u, v) \neq 0$, the rate of pheromone of color c on the edge (u, v) between the steps $t-1$ and t is noted

$$K_c^{(t)}(u, v) = \frac{\Delta^{(t)}(u, v, c)}{\Delta^{(t)}(u, v)} \tag{3}$$

    This rate verifies $K_c^{(t)}(u, v) \in [0, 1]$.
- The current quantity of pheromone of color c present on the edge (u, v) at step t is denoted by $\tau^{(t)}(u, v, c)$. Its initial value (when $t = 0$) is 0 and then is computed following the recurrent equation:

$$\tau^{(t)}(u, v, c) = \rho\tau^{(t-1)}(u, v, c) + \Delta^{(t)}(u, v, c)$$

  Due to evaporation, we define the persistence of the pheromones on an edge: $\rho \in [0, 1]$.

- At this stage of the algorithm, we have computed the current quantity of pheromone, $\tau^{(t)}(u,v,c)$ classically, as a reinforcement factor for clustering formation based on colored paths. We need now to take into account the load balancing in this self-organization process. For this purpose, we need to balance this reinforcement factor with $K_c^{(t)}(u,v)$, the relative importance of considered color with regard to all other colors. This corrected reinforcement factor is computed as following:

  $$K_c^{(t)}(u,v)\tau^{(t)}(u,v,c)$$

  Unfortunately, this corrected reinforcement factor can generate an unstable process. So we prefer to use a delay-based relative importance of considered color with regard to all other colors. For a time range $q \in \mathbb{N}^+$, we define:

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

  According to this definition, we compute the new corrected reinforcement factor :

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

- Let us define $p(u,v_k,c)$ the probability for one arbitrary ant of color c, on the vertex u, to walk over the edge (u,v_k) whose weight is noted $w(t,u,v_k)$ at time t.
  - At the initial step $(t=0)$,

    $$p(u,v_k,c) = \frac{w(0,u,v_k)}{\sum_{v\in\mathcal{V}_u} w(0,u,v)} \tag{6}$$

  - After the initial step $(t \neq 0)$,

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

  Where $\mathcal{V}_u$ is the set of vertices adjacent to u.
  The relative values of α and β give the weighting between pheromone factor and weights. We will see later that this weighting is a major factor in the way the algorithm achieves its goals.
- The color of a vertex u, noted $\xi(u)$ is obtained from the main color of its incident arcs:

  $$\xi(u) = \arg\max_{c\in\mathcal{C}} \sum_{v\in\mathcal{V}_u} \tau^{(t)}(u,v,c) \tag{8}$$

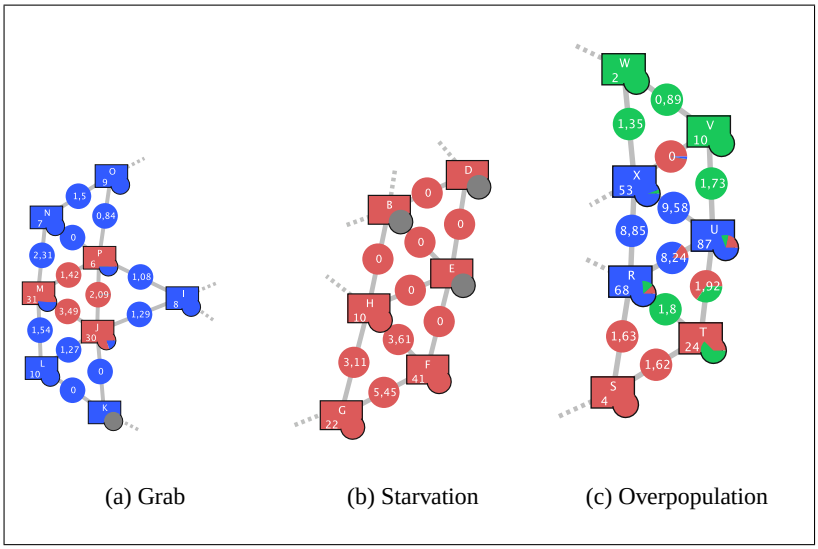
3 Local corrections on General Algorithm

The original algorithm reveals several difficulties due to lack of control over the relations between the numerous parameters. In clusters of high communication, ants tend to follow privileged paths that form *loops*. This is due to the fact communications in such areas are mostly the same and pheromone take a too large importance in ant path choices. Such paths exclude some nodes that could be settled and leads to three problems: *grabs*, *starvation* and *overpopulation* as shown in figures 2(a), 2(b) and 2(c). In these figures, the graph representation is as follows. Vertices are rectangles. Edges are shown with a pie chart in the middle that indicates relative levels of pheromones with the maximum pheromone level numbered. Vertices are labeled by their name at the top with under at the left the total number of ants they host and at the right a pie chart indicating the relative number of ants of each color present on this vertex.

grabs Small ants group of a given color are locked in an area of the graph by a larger colony of another color that surround them. In this case they cannot escape due to repulsion factors and cannot help consolidating correct clusters of their own color elsewhere (figure 2(a)).

starvation Whole parts of the graph get less and less pheromone trails and are left unoccupied by ants. As pheromone evaporate, less and less ants are attracted by them (figure 2(b)).

overpopulation Some parts of the graph get too much ants as the total number of ants on a vertex is not bounded. In the worse case, ants of all colors can appear in such high density populations without the repulsion factors to operate correctly (figure 2(c)).



(a) Grab (b) Starvation (c) Overpopulation

Fig. 2. Problems encontered by our algorithm

The three next sections deal with methods we used to solve these problems.

3.1 Death and hatching to escape local minima

Control on the ant population as been changed to take into account the three problems given above. Before, this control was used only to handle the dynamic graph (vertex and edge appearance or disappearance). Now we add some death and hatching mechanisms. We perturb the ants repartition generating small stable clusters which are the result of local minima. Furthermore this procedure makes senses since our algorithm runs continuously not to find a static solution as the standard Ant System, but to provide anytime solutions to a continuously changing environment.
Including death and hatching asks a question about population: for a given number of vertices is it constant or not? Indeed we want to avoid cases where all ants disappear or too many ants appear. Therefore, we resolved to make one hatch for one death.
Additions to the original Colored Ant Algorithm are:

1. We define the following positive numbers:
   - $\tau^{(t)}(u, c)$ is the quantity of pheromone of color c present on all edges connected to vertex u:

$$\tau^{(t)}(u, c) = \sum_{v \in \mathcal{V}_u} \tau^{(t)}(u, v, c) \tag{9}$$

   - $\tau^{(t)}(u)$ is the quantity of pheromone of all colors dropped on all edges connected to vertex u:

$$\tau^{(t)}(u) = \sum_{c \in \mathcal{C}} \tau^{(t)}(u, c) \tag{10}$$

- $\varphi_c(u) \in [0,1]$:
$$\varphi_c(u) = \frac{\tau^{(t)}(u,c)}{\tau^{(t)}(u)} \tag{11}$$
the relative importance of pheromones of color c compared to pheromones of all colors on edges leading to vertex u.

2. Then, at each step, before the ant chooses an arc to cross (equations 6 to 13), we must choose either the ant will die or not. We determine this using a threshold parameter $\phi \in [0,1]$[1] for an ant of color c on vertex u:
   - if $\varphi_c(u) < \phi$ we make the ant die and create a new ant choosing a new location for it as follows. We select randomly a set $\mathcal{V}_n$ of n vertices. Let card($\mathcal{F}_v$) be the number of ants on vertex v. Then we select a vertex u in $\mathcal{V}_n$ using:
   $$u = \arg\min_{v \in \mathcal{V}_n} (\text{card}(\mathcal{F}_v)) \tag{12}$$
   and make the new ant hatch on it.
   - else, we proceed as specified in the original algorithm choosing a new edge using probabilities (equation 6 and following).

This procedure eliminates grabs and starvation. Grabbed ants die, and hatch in starvation areas. However it does not eliminate loops, and sometimes loops tend to reappear. In order to break them, we introduce more memory in ants: instead of being able to memorize only one vertex, ants can memorize three or more vertices as explainer under.

3.2 Ant memory for two vertices oscillation suppression

To avoid ants that form loops, going infinitely from one vertex to another we introduce the ability to remember one or more vertices the ant comes from.
We therefore introduce in the formula 7 a penalisation factor $\eta \in]0,1]$. Given $\mathcal{W}_x$ the set of the last visited vertices by ant x with $|\mathcal{W}_x| < M$, the new probability formula for the specific ant x is:

$$p_x(u,v_k,c) = \frac{(\Omega^{(t)}(u,v_k,c))^\alpha (w(t,u,v_k))^\beta \eta_x(v_k)}{\sum_{v_q \in \mathcal{V}_u} (\Omega^{(t)}(u,v_q,c))^\alpha (w(t,u,v_q))^\beta \eta_x(v)} \tag{13}$$

Where

$$\eta_x(v) = \begin{cases} 1 \text{ if } v \notin \mathcal{W}_x \\ \eta \text{ if } v \in \mathcal{W}_x \end{cases} \tag{14}$$

3.3 Demographical pressure

Demographical pressure is the ability for ants to avoid edges that already contain too many other ants (of any color). (This can be compared to the way individuals react inside swarms, herds or fish schools.) We use it to better spread ants in the graph. Demographical pressure acts in the same direction as ant memory. It helps spreading ants and avoids loops. It therefore minimises starvation and suppresses overpopulation. Furthermore, it allows an isolated group of ants (a grab) to escape its confined position (local minima), being pushed by other ants or because the confined group is overpopulated to go in another location. (This can be compared to percolation, under a given threshold, ants cannot escape from a confined part of the graph, if demographical pressure is activated, other colonies tend to "press" the grab and isolated ants escape forming a path to a new location.)
Like for ant memory, we introduce in the formula 13 a penalisation factor $\pi(u,v)$. Given $N(u,v)$ the ant number on the vertex (u,v) and $N^*(u,v)$ the threshold.

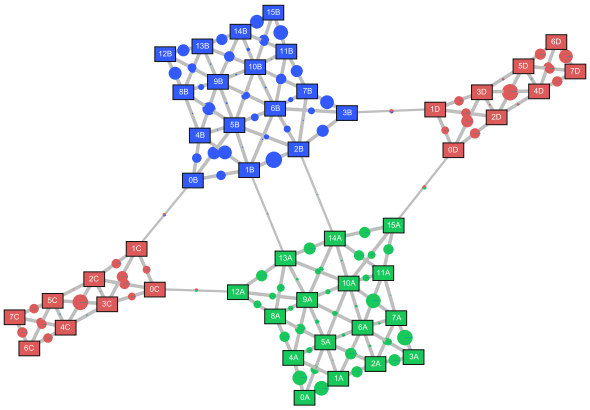
$$\pi(u,v) = \begin{cases} 1 & \text{if } N(u,v) \leq N^*(u,v) \\ \pi \in]0,1] & else \end{cases} \tag{15}$$

[1] Preferably small, under 0.1.

The corrected formula 13 becomes :

$$p_x(u, v_k, c) = \frac{(\Omega^{(t)}(u, v_k, c))^{\alpha}(w(t, u, v_k))^{\beta}\eta_x(v_k)\pi(u, v_k)}{\sum_{v_q \in \mathcal{V}_u}(\Omega^{(t)}(u, v_q, c))^{\alpha}(w(t, u, v_q))^{\beta}\eta_x(v)\pi(u, v)} \quad (16)$$

As illustration, we tested it on a graph with two big and two small clusters, a small one being twice smaller than a big as shown in figure 3. We process it with three ants colonies, the result being that the two small clusters are of the same color. In this configuration, without demographical pressure, ants in a small cluster stay confined, the other small cluster being colonised by the color of the two large one. With demographical pressure exceeding ants in one small cluster escape from it, and as they cannot stay in big clusters, they tend to colonize the other small cluster.



-

Fig. 3. *Demographical pressure*

4 Experimentations

Experiments have been made with different static and dynamic graphs with different properties as detailed under. Some are representative of possible simulations, like grids for regular applications with for example distributed domains or meshes, or like scale-free graphs for interaction networks... But we also tested the algorithm on random graphs as well as hierarchical graphs.
It is necessary to measure the quality of the solution found on these graphs. There are two antagonist aspects to take into account :

- The global costs of communications, to which we map a criterion r_1;
- The application load-balancing, to which we map a criterion r_2.

The first, r_1, identifies between two solutions which has proportionally less actual communications. Thus we compute the sum of all actual communication costs, noted a. Then we compute the ratio r_1 among the total volume of communications, noted s, on the graph and we have:

$$r_1 = a/s$$

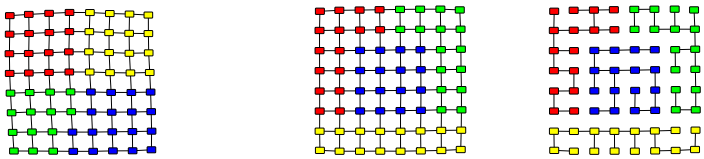
The more r_1 is close to 0, the more actual communications are low, as expected.
The second criterion r_2 considers the load-balancing. For each color c, we have v_c the number of vertices of color c and p_c the power of processing resource affected to c. Then we have:

$$r_2 = \frac{min\mathcal{K}}{max\mathcal{K}} \quad \text{where} \quad \mathcal{K} = \left\{ \frac{v_c}{p_c}; c \in C \right\}$$

The load balancing it better when r_2 is close to 1. These criteria are used to compare different solutions obtained during the computation, essentially to verify if we improve the solution during the steps. These cretins, enable us to store the best solution obtained so far.

4.1 Grids

In this kind of regular graph, communications play a major role. Here are two examples with a 8×8 grid, one with uniform communications across the grid (figure 4), another with four clusters of high communication. In the second one, ants find clusters of high communications exactly with $r_1 = 0.04$ and $r_2 = 0.88$, as shown by creating a maximum spanning tree on the graph and cutting the less communicating edges. Parameters used were $\alpha = 1$, $\beta = 1$, $\rho = 0.86$, $\eta = 0.0001$, $N = 7$.



Uniform communications ($r_1 = 0.16$, $r_2 = 0.88$)

Specific communication layout ($r_1 = 0.04$, $r_2 = 0.88$)

Fig. 4. Grids

4.2 Random graphs

Random graphs[6] have a degree distribution that follows a normal law. We tested our algorithm with a random graph of 300 vertices following a Poisson distribution. Inside this graph, we have four large clusters of highly communicating entities and several other small clusters. The structure of this graph only appears in communications. Figures 5 shows the graph with highly communicating vertices highlighted. Figure 6, shows figure 5 with low communication edges removed. The algorithm finds communication organizations in 847 steps with $r_1 = 0.18$ and $r_2 = 0.91$ and parameters $\alpha = 1$, $\beta = 1$, $\rho = 0.86$, $\eta = 0.0001$, $N = 8$. Some errors are present due to the large number of low communication edges that still impact the algorithm.

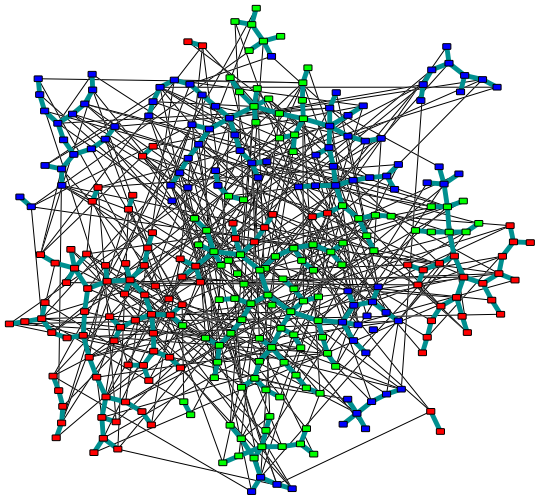


Fig. 5. *A random graphs with highly communicating edges highlighted* ($r_1 = 0.18$, $r_2 = 0.91$)

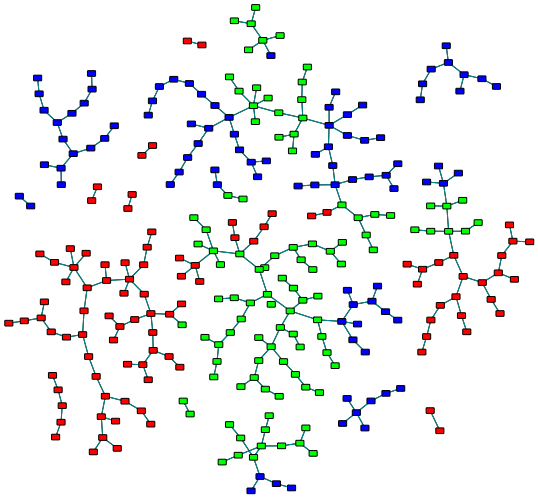


Fig. 6. *The figure 5 with low communicating edges removed.*

4.3 Scale-Free graphs

Scale-free graphs[11,1] have a degree distribution that follows a power-law. Such graphs are often generated as a growing network using preferential attachment. The graph shown in figure 7 has 400 vertices, it was generated following this approach. It has been colored using four ant colonies in 249 steps with $r_1 = 0.06$ and $r_2 = 0.8$ and parameters $\alpha = 1$, $\beta = 1$, $\rho = 0.86$, $\eta = 0.0001$, $N = 5$.

4.4 Dynamic graphs

Dynamic aspects have been tested, for that we used a program that simulate the application by creating and then applying events to it. Events are the appearance and disappearance of a vertex or processing resources, and weight modifications on edges. The graph presented in figure 8 is a dynamic graph of 32 vertices that continuously switch between four configurations. The figure shows it a eight steps of its evolution (from left to right then top to bottom). For this graph we used parameters $\alpha = 1$, $\beta = 4$, $\rho = 0.8$, $\eta = 0.0001$, $N = 10$ and four colors. At each step criteria r_1 stayed between 0.12 and 0.2 and r_2 at 0.77.

5 Conclusion

In this paper we have presented an algorithm based on numerical ants that offers advices for entity migration in a distributed system taking care of the load and communication balancing. We have described a base colored ant algorithm, observed its behaviour with dynamic graphs and provided methods to handle them. We have shown several experiments with different graphs (dynamic, scale-free and random) of this system. We develop actually an heuristic layer allowing to handle some constraints tied to the application, like entities that cannot migrate (e.g. bound to a database), but also informations peculiar to the application. This work takes place within the context of aquatic ecosystem models, where we are faced to a very large number of heterogeneous auto-organizing entities, from fluids representatives to living creatures presenting a peculiar behaviour.

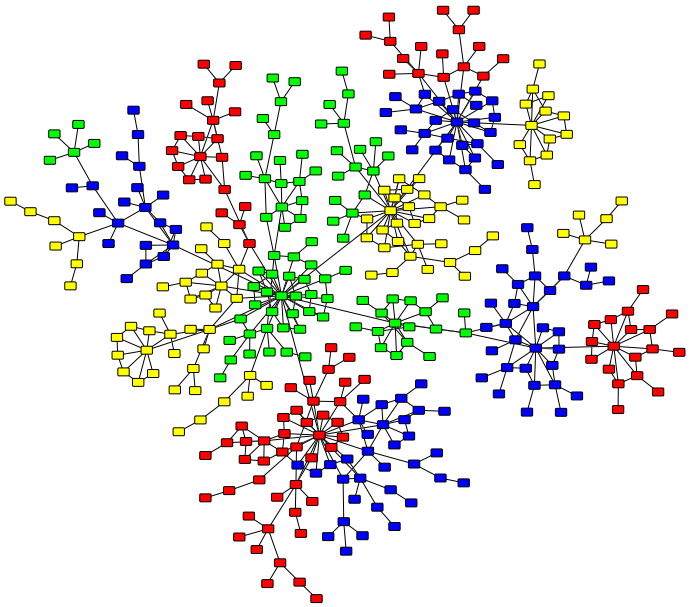


Fig. 7. *A scale-free graph of 400 vertices colored with 4 colonies ($r_1 = 0.06$, $r_2 = 0.8$)*

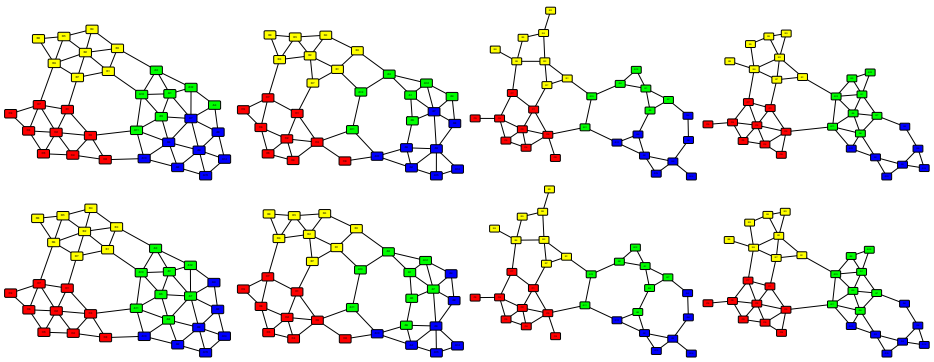


Fig. 8. *A dynamic graph*

References

1. R. Albert and A.L. Barabaśi. Statistical mechanics of complex networks. *Reviews of modern physics*, 74:47–97, 2002.
2. S. H. Bokhari. On the Mapping Problem. *IEEE Transactions on Computers*, 30:207–214, March 1981.
3. E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence - From natural to Artificial Systems*. Oxford University Press, 1999.
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. L. Eager, E. D. Lazowska, and J. Zahorjan. A comparison of receiver-initiated and sender-initiated adaptive load sharing. *Performance evaluation*, 6:53–68, 1986.
6. P. Erdös and A. Rényi. On random graphs. *Pubiones Mathematicaelicat*, 6:290–297, 1959.
7. H.-U. Heiss and M. Schmitz. Decentralized dynamic load balancing: the particle approach. *Information Sciences*, 84:115–128, 1995.
8. Z. Jovanovic and S. Maric. Heuristic algorithm for dynamic task scheduling in highly parallel computing systems. *Future Generation Computer Systems*, 17:721–732, 2001.
9. P. Kuntz, P. Layzell, and D. Snyers. A colony of ant-like agents for partitioning in vlsi technology. In *Fourth European Conference on Artificial Life*, pages 417–424, Cambridge, MA:MIT Press, 1997.
10. F. C. H. Lin and R. M. Keller. The gradient model load balancing method. *IEEE TOSE*, 13:32–38, 1987.
11. M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, March 2003.
12. M. H. Willebeek-LeMair and 1. P. Reeves. Strategies for dynamic load balancing on highly parallel computers. *IEEE Transactions on parallel and distributed systems*, 4(9):979–993, 1993.