Organization Detection for Dynamic Load Balancing in Individual-Based Simulations

Cyrille Bertelle, Antoine Dutot, Frédéric Guinand, Damien Olivier

Organization Detection for Dynamic Load Balancing in Individual-Based Simulations

Cyrille Bertelle Antoine Dutot Frédéric Guinand Damien Olivier

Université du Havre

25 rue Philippe Lebon

BP 1123, 76063 Le Havre Cedex

antoine.dutot@univ-lehavre.fr

Tel: +33 (0) 2 3274 4373

Fax: +33 (0) 2 3274 4314

August 8, 2006

Abstract

Large-scale individual-based simulations can benefit a lot from high performance computing environments. The benefit that can be hopped depends greatly on a good load distribution among the processing ressources together with the minimization of the communication overhead. However, minimizing both idle time and communication overhead requires the search for a trade-off.

Inspired by complex systems, the approach described in this paper aims at minimizing the volume of data exchanged over the network between tasks of a distributed application, while balancing the load between available computing

ressources. The method lies on the trail-laying trail-following paradigm used in algorithms based on artificial ants.

Keywords: Complex Systems, Self-Organization, Artificial Ants, Dynamic Graph, Community Structures, Dynamic Load Balancing, Individual-Based Model, Ecological Simulations.

1 Introduction

There exist many application classes whose structural as well as numerical characteristics may change during their execution. Simulations are one of these classes, they cover a wide range of domains: from life and Earth sciences to social sciences. Many applications rest on a mathematical analysis and formulation of the global process to be simulated with systems of differential equations. An alternative or a complementary way of modeling problems for simulation purposes consists in modeling the environment, each entity moving and evolving within this environment, and the mechanisms needed for computing entities evolution, their activities and their interactions. Entities may be living beings, particles, or may represent non-material elements (vortices, winds, streams...). Individual-based models (IBM) belong to this category. Within such models, interactions between the environment and the individuals are explicit as well as interactions among individuals themselves. During the simulation, some individuals may appear and/or disappear without prior notice, and interactions may change, leading to the emergence of unpredictable organizations at different description levels.

Many recent studies have adopted this approach. The motivation is that "*simulating the actions of single organisms allows to study how the properties of higher level ecological entities like swarms, populations, trophic networks and regional distribution patterns emerge*" [4]. Instead of using systems of differential equations, an individual based model provides a representation of each entity,

interactions between the environment and the individuals and among individuals themselves. From the computer science point of view, multi-agent systems coupled with discrete-time simulation constitutes an obvious way of implementing IBMs. Several tools and frameworks have already been proposed for that purpose as reported in [14]. What motivates our work comes from a series of observations. In order to observe emergent properties in IBMs, many individuals have to be considered, the use of discrete classes instead of individuals may lead to wrong results [6]: "*If IBMs treat individuals as members of classes, rather than as unique organisms, predictions of population size and variability may be wrong*". In addition, "*a successful application of the approach requires detailed biological information about the represented species.*" [4]. As a consequence, since the number of entities may reach thousands to millions, distributed simulations running on high performance computing environments is unavoidable in order to obtain results in a reasonable time. However, the benefit that can be hopped depends greatly on a good load distribution among the processing ressources together with the minimization of the communication overhead. In order to limit/avoid performance degradations due to the dynamics of such applications, the possible re-allocation of existing entities and the allocation of new ones have to be continuously managed. But that's not enough, for observing emergent organizations and structures in the system, the scientists need another software for detecting such properties. The tool presented in this paper, AntCO^2, advises for both: dynamic allocation and re-allocation of individuals and detection of application-level organizations. AntCO^2 is a bio-inspired algorithm based on numerical ants. It gives global information in a decentralized way that indicate a possible better location of some existing entities or a good location for new ones, according to processors load, inter-entities communications and existing application-level organizations. But it also gives global information for detecting new application-level organizations.

The next section presents the general context of ecological simulations and specifies the problem of dynamic load balancing, in particular for individual-based ecological models. In section 3, the problem is formalized and the method is presented. Some experiments are reported in Section 4 and the last Section discusses open perspectives for this work.

2 General Context

The problem we are interested in is the distribution of entities composing a system conceived for simulating ecosystems using individual-based models. Before entering in the details of the distribution strategy, the following section presents the target applications.

2.1 Individual-Based Ecological Simulation

Mechanisms leading to the global behavior of ecological systems often remain unclear, due to the large number of interactions and feedbacks processes. Differential equation-based models may be sometimes unsuited since the system is often characterized by different organization levels and by changing structures. For many years, it has been accepted that some high level properties of ecological systems emerge from interactions among lower level components, but it has recently gain an important effort in the community [27, 25, 21, 20, 24, 4]. In order to better understand the mechanisms at the origin of the emergence of these properties and to capture ecological dynamics over several levels of organization, ecologists use individual-based models [8, 18]. These models allow the examination of explicit spatial and temporal variation on the scale of the individual. This approach allows scientists to build virtual laboratories to test different scenarii of the population evolution, and help them to understand the variations of

some specific parameters of individuals (length, weight...) depending or not on environmental characteristics.

For example, in [20], the authors present a spatiotemporal IBM of roach to study emergent properties at the individual and the population level. The fish model includes swimming mode that may lead to shoal formation, habitat selection according to food availability, temperature, light and age, bioenergetics, responsible for growth and that specifies energy storage, food consumption, and extra metabolic functions (respiration...). They also provide a model of the environment (Lake Belau). Their study shows that some interesting emergent properties can be obtained from the simulation. At the individual level, they notice that spatial behavior, as well as growth and food consumption fit with empirical studies. At the population level, among other properties, they show that the fishes self-sort themselves in shoals according to their length, they also remark a length variability into the population, and they show that this variability depends on the lake morphology.

Our motivation for that work comes from a series of remarks. At first, a noticeable point is that some processes occurring in IBMs are non linear and very sensitive to initial conditions. As a consequence, two runs of the same simulation, with similar initial conditions, may lead to different global situations. Secondly, some parts of the simulation depend on random choices, and this leads ecologists to perform many runs starting from the same parameters set but with different values of the random seed. Thirdly, running individual-based simulations may require a lot of time with respect to the temporal as well as the space scales considered.

Finally, as said by Dunstan and Johnson in [24]: "*in real ecologies, space does matter: real assemblages typically manifest non-random spatial pattern at a range of scales, which underpins patterns of spatial variability, and individual organisms are more likely to interact with neighbouring than with distant indi-*

viduals".

We can summarize all these remarks saying that individual-based ecological applications are time-consuming, characterized by dynamically changing structures, and made of sets of locally interacting entities.

So, in the context of a distributed/parallel implementation of such applications, a desirable feature of a dynamic load-balancing method should be the detection of emerging structures based on interactions in order to map these structures on different computing resources with the aim of balancing the load. The remaining part of this paper is dedicated to the description and analysis of such a method.

2.2 Dynamic Load Balancing

The execution of an application in a parallel environment is efficient if the inactivity periods of the different computing resources are small with respect to the total execution duration. The two main sources of overhead for a given processor are idleness due to a lack of work, and time spent waiting for data from another processor (a.k.a. communication overhead). When the application is fully defined before the execution (precedence task graph, tasks durations and communication delays), an *a priori* computation of a schedule is possible. But, when the application is partially known or undefined before the execution, an on-line strategy with the aim of reducing both idleness and communication overhead has to be considered. This is dynamic load balancing (DLB). The problem may be described as the on-line allocation or migration of tasks to computing resources. It is usually assumed that communications between tasks located on the same processor may be neglected. The communications that may entail overhead occur between tasks located on distinct computing resources. Thus, the quality of a dynamic load balancing strategy rests on its capacity of combining local decisions (communicating

tasks gathered into clusters) with global ones (load balancing). Partitioning of the application graph seems to be a suitable starting point for that purpose, since clustering may be driven by the objective of minimizing the communications, and the load balancing may be achieved by fixing the number of clusters.

There are two main approaches to the problem. In the first one, relevant information is gathered and the load balancing is computed by only one processor: this is the centralized scheme. Advantadges: the decision is taken according to the full knowledge of the system and classical quality-proven algorithms can be used. But, keeping and/or gathering updated information generates an important network traffic and the decision process (collect of information and decision making) has to be faster than changes in the application. The decentralized scheme is the second approach. Most of distributed DLB strategies rely on load exchanges between neighbor computing resources. For instance, in the method proposed by Heiss and Schmitz [19], each computing resource possesses a load balancing procedure which is activated when a change of the load occurs in the neighborhood. They address the problem of dynamically load balancing several applications, and their approach is based on an analogy with physics. Each application is considered as a fluid characterized by a viscosity factor modelling communication density between tasks forming the application. Fluids are added to a flat container (corresponding to the computing resources). Two kinds of forces are considered: gravitational forces that entails a spreading of the fluid, and frictional resistance and cohesion forces (related to the viscosity) that keep the fluid dense. The former force leads to more parallelism while the latter aim at minimizing inter-processors communications. However, in their work, the behavior of each application is supposed fixed during the application execution, so for them, "*load balancing should take place when the load situation has changed. The load situation changes when tasks are generated or finished*", while for us, load balancing should take place when either the load situation changes or when the communi-

cation characteristics (numerical as well as structural) change.

Another interesting method was proposed by Fenet and Hassas [26] who use numerical ants. Ants are used for process migration, according to pheromones concentration on the arcs of the computing environment topology. Pheromones code for the load of the processors and different kinds of pheromones are attributed to different groups of processes, given that processes inside a group aim at communicate with each other.

Our goal is to implement such a DLB method that is additionally able to detect interacting groups instead of identifying them *a priori*.

3 Organization Detection and Load Balancing

Large-scale ecological simulations are natural candidates for distributed discrete event simulation [15]. However, such simulations may also be performed through a series of discrete time steps [28]. In both cases, the same graph model can be considered: a dynamic interaction graph, that is a graph varying along time.

Individual-based ecological models are made of an environment and many entities interacting with the environment and each other. We are in front of complex systems in which there are numerous interactions. As the system evolves, communications between entities change. Entities and their interactions appear and disappear creating stable or unstable organizations. Each entity, in IBMs, is simulated at the level of the individual organism rather than the population level, and the model allows individual variation to be simulated [1]. Thus, during the simulation process some entities may appear or disappear without prior notice, groups of entities with weak interactions may meet, leading to an important increase of the number of communications between them.

3.1 Problem formulation

While highly dynamic, the whole system may be modeled using a classical undirected graph whose structural and numerical characteristics may change during the simulation.

Within such a graph, each vertex is associated to an element of the simulation that may be a biological entity or any environmental element, and an edge materializes an interaction between two elements of the simulation, mapped to a communication in lower levels of the implementation.

More formally, the considered simulations are composed of a number n at time t of interacting entities, which we wish to distribute over a number of p processing resources. Entities do not necessarily communicate with all the others and their numbers and the communications may vary during the application execution. The pattern of communications can be represented by a dynamic graph (network) in which entities are mapped one-to-one with vertices and communications with edges.

We distinguish communications occuring inside a single computing resource, that is between entities in executing on the same computer, from communications that must cross the network between distinct computing resources. We call the later *actual communications*.

Dynamic graph: A dynamic graph $G(t) = (\mathcal{V}(t), \mathcal{E}(t))$ is a weighted undirected graph such that:

- $\mathcal{V}(t)$ is the set of vertices at time t.
- $\mathcal{E}(t)$ is the set of edges at time t. Each edge (u, v) is characterized by:
  - a weight $w(t, u, v) \in \mathbb{N} - \{0\}$.

The problem is to distribute entities of the application at anytime in such a way to balance the load on each processing resources and at the same time to minimize the actual communications. To solve this problem, we search a partition

of the graph $G(t)$ defined as follows:

Let $G(t) = (\mathcal{V}(t), \mathcal{E}(t))$ a dynamic graph, a partition $\mathcal{D}(t)$ of G is composed by k disjointed subsets $\mathcal{D}_i(t)$ of $\mathcal{V}(t)$ called domains with :

$$k > 0, \bigcup_{i=1..k} \mathcal{D}_i(t) = \mathcal{V}(t)$$

Where the set of edges connecting the domains of a partition (i.e. edges cut by the partition) $\mathcal{D}(t)$ is called an edge-cut denoted by $\mathcal{B}(t)$.

The objective is therefore to find a k-partition, at anytime, which evenly balances the vertex weight among the processing resources while minimizing the total weight of $\mathcal{B}(t)$. The number of domains k must be greater or equal than p the number of processing resources. For example, in figure 1, we have 4-partitions and three domains identied by three colors.

3.2 AntCO^2, an ant algorithm

Ant algorithms are a class of meta-heuristics based on a population of agents exhibiting a cooperative behaviour [23]. Ants are social insects which manifest a collective problem-solving ability [9]. They continuously forage their territories to find food [16] visiting paths, creating bridges, constructing nests, etc. This form of self-organization appears from interactions that can be either direct (e.g. mandibular, visual) or indirect. Indirect communications arise from individuals changing the environment and other responding to these changes: this is called *stigmergy*[1]. There are two forms of stigmergy, sematectonic stigmergy produces changes in the physical environment - building the nest for example - and stigmergy based on signal which uses environment as support. Thus, for example, ants perform such indirect communications using chemical signals called

[1]PP. Grassé, in Insectes Sociaux, 6, (1959), p. 41-80, introduced this notion to describe termite building activity.

pheromones. The larger the quantity of pheromones on a path, the larger the number of ants visit this path. As pheromone evaporates, long paths tend to have less pheromone than short ones, and therefore are less used than others (binary bridge experiment [17])). Such an approach is robust and well supports parameter changes in the problem. Besides, it is intrinsically distributed and scalable. It uses only local information (required for a continuously changing environment), and find near-optimal solutions. Ant algorithms have been applied successfully to various combinatorial optimization problems like the Travelling Salesman Problem [10], routing in networks [5, 29], clustering [13], graph coloring [7], graph partitioning [22], and more recently DNA sequencing [2], dynamic load balancing falls into the category of optimization problems.

Dorigo et al. [12], define an approach named Ant Colony Optimization (ACO) that may be applied to a given problem if it is possible to define: the problem representation as a graph, the heuristic desirability of edges, a constraint satisfaction, a pheromone updating rule and a probabilistic transition rule.

Our approach, AntCO^2, is an ant algorithm that exhibits great similarities with ACO, excepted that it has no satisfaction constraint, and detects organizations with several colonies in competition, each of a distinct color.

Organizations are seen as a sub-graph, set of vertices that are more densely connected each other, both in number of links, and eventually in the importance of the links, while being slightly connected with the other vertices that do not belong to the considered sub-graph.

We search such organizations since they concentrate communications inside themselves. As we want to reduce communication costs, identifying high communication areas allows to better distribute them. Furthermore the graph being dynamic, such organizations will present a more stable form than edges and vertices taken alone.

Indeed, communications on a single host are often considered negligible,

whereas communications between hosts are more costly. Therefore placing organizations on hosts may provide a good communication cost reduction. We must also take into account the possible overhead of the different computing resources. To reach this objective we have to determine a good load balancing between computing resources. Unfortunately, this objective is in conflict with the other which is to minimize the communication costs. A good trade-off must be find. Thus if to find organizations with collaborative ants offer a good solution to reduce the communication costs, competition with colored ants is introduced to obtain a good load balancing.

3.3 Colored ants

The method proposed by Kuntz et al. [22] for graph partitioning is able to detect 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 intercluster 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 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 similar (considering the resource computing capacity),
3. and their number as well as the graph structure may change.

For the first issue, in [12], the authors mentioned the possibility to parametrize their 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 1, where elements of the two opposite and not directly linked clusters should be allocated to the same processing resource.

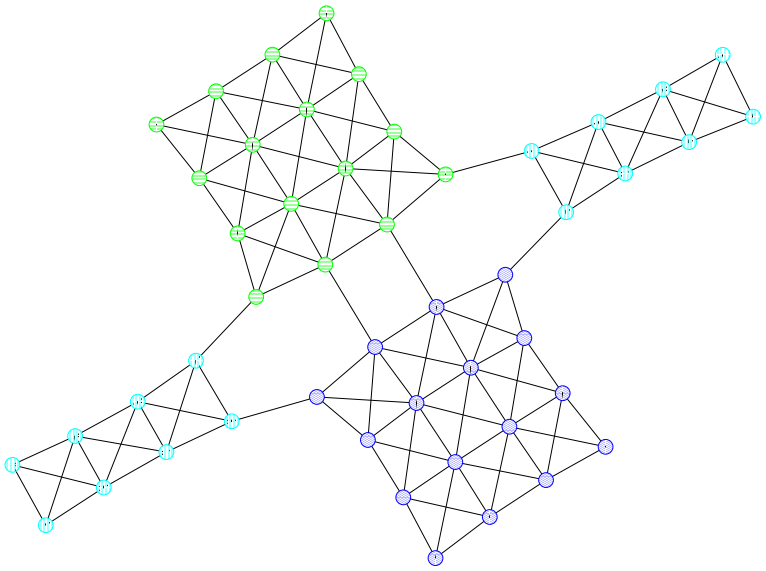


Figure 1: *Example for which the best number of clusters is different of the number of processing resources.*

As the number of processing resources available at a given moment is known, this problem may be identified as a competition between processing resources for computing as many elements as possible. This may be directly included in the ant approach by considering competing ant colonies. In our approach, a distinct color is associated to each computing resource, and an ant colony is attached to each resource. Each ant is also colored, and drops pheromones of its color.

So, in addition to the classical collaborative nature of ant colonies we have added the competition aspect to the method. In fact, this reflects exactly the trade-off previously discussed about dynamic load-balancing. On one hand, the collaborative aspect of ants allows the minimization of communications by gathering into clusters elements that communicate a lot, while, on the other hand, the competition aspect of colored ants allows the balancing of the load between

computing resources.

The technical issues about the management of ants, colors and pheromones are described in detail in the next sections.

3.4 Graph description and notations

As previously mentioned, we consider an undirected graph which structural as well as numerical characteristics may be subject to changes during the execution. Colored ants walk within this graph, crossing edges and dropping colored pheromones on them.

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

- $\mathcal{C}(t)$ is a set of p colors where p is the number of available processing resources of the distributed system at time t.
- $\mathcal{V}(t)$ is the set of vertices at time t. Each vertex v is characterized by:
  - a color $c \in \mathcal{C}(t)$,
- $\mathcal{E}(t)$ is the set of edges at time t. Each edge (u, v) is characterized by:
  - a weight $w^{(t)}(u, v) \in \mathbb{N} - \{0\}$ that corresponds to the volume and/or the frequency and/or the delay of communications between the elements associated to vertices u and v,
  - a quantity of pheromones of each color.

The figure 1 shows an example of a dynamic communication colored graph where the proposed method described in the following, changes the color of vertices if this 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. On the figure we can see three colors only,

one for each computing resource, and four clusters. Indeed it can be necessary, in order to maintain a good load balancing to have several distinct clusters of the same color.

3.5 Pheromones management

We denote by $\mathcal{F}(t)$ the population of ants at time t, and $\mathcal{F}_c(t)$ the set of ants of color c at time t. Pheromones are dropped on edges by ants crossing them. Pheromones are colored. An ant x of color c crossing an edge (u, v) between steps $t-1$ and t will drop a given quantity of pheromone of color c. This quantity is denoted by $\Delta_x^{(t)}(u, v, c)$, and the quantity of pheromones of color c dropped by ants when they crossed edge (u, v) during time interval $]t-1, t]$ is equal to:

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

The total quantity of pheromones of all colors dropped by ants on edge (u, v) during $]t-1, t]$ is equal to

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

If $\Delta^{(t)}(u, v) \neq 0$, the rate of dropped pheromones of color c on (u, v) during $]t-1, t]$ is equal to:

$$K_c^{(t)}(u, v) = \frac{\Delta^{(t)}(u, v, c)}{\Delta^{(t)}(u, v)} \text{ with } K_c^{(t)}(u, v) \in [0, 1] \tag{3}$$

The quantity of pheromone of color c present on the edge (u, v) at time t is denoted by $\tau^{(t)}(u, v, c)$. At the beginning $\tau^{(0)}(u, v) = 0$ and this value changes

according to the following recurrent equation:

$$\tau^{(t)}(u,v,c) = \rho\tau^{(t-1)}(u,v,c) + \Delta^{(t)}(u,v,c) \tag{4}$$

Where $\rho \in]0,1]$ denotes the persistence of the pheromones on the edges, that is the proportion of pheromones which has not been removed by the evaporation phenomenon.

$\tau^{(t)}(u,v,c)$ may be considered as a reinforcement factor for clustering vertices based on colored paths. However, due to the presence of several colors, this reinforcement factor is corrected according to $K_c^{(t)}(u,v)$ that represents the relative importance of the considered color with respect to all colors. This corrected reinforcement factor is noted:

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

Then, if we denote by $\mathcal{V}_u$ the set of vertices adjacent to u at t, the color $\xi^{(t)}(u)$ of this vertex is obtained from the main color of its incident edges:

$$\xi^{(t)}(u) = \arg\max_{c\in\mathcal{C}(t)} \sum_{v\in\mathcal{V}_u} \tau^{(t)}(u,v,c) \tag{5}$$

3.6 Ants moving and population management

Ants move according to local information present in the graph. Each processing resource is assigned to a color. Each vertex gets its initial color from the processing resource it was allocated to. Initially the number of ants of a given color is proportional to the processing resource power that they represent.

3.6.1 Population management

The process is iterative, between two steps, each ant crosses one edge and reaches a vertex. When there are too few ants, evaporation makes pheromones disappear and the method behaves as a greedy algorithm. If there are too many ants, pheromones play a predominant role and the system efficiency may decreases. Furthermore, the population is regulated with respect to the number of processing resources and to the number of entities.

Initially, our algorithm creates a fixed number of ants per vertex, which depends on processing ressources power. Vertices are, in the same way, proportionally associated to processing resources. Then, during the execution, our method tries to keep the population density constant in the graph. When some new vertices are created, new ants are created in order to maintain the population density, and some ants are removed from the population when some vertices disappear. When one new processing resource becomes available, the number of colors increases by one. However, the population has to be modified in order to take into account this new color. This is done by changing the color of an equal number of ants of each current color into the new color. Symmetrically, when a processing resource, associated to color c, disappears from the environment, either because of a failure or because this resource is out of reach in case of wireless environments, all ants of color c change their color into the remaining ones. The creation and the removing of edges have no effect on the population.

3.6.2 Ants moves

The moving decision of one ant located on vertex v_k is taken according to its color and to the concentration of the corresponding colored pheromones on adjacent edges of u. Let us define $p^{(t)}(u, v_k, c)$ the probability for one arbitrary ant of color c, located on the vertex u, to cross edge v_k during the next time interval

$[t, t+1[$. If we denote $w^{(t)}(u, v_k)$ the weight associated to this edge at time t, then:

$$\begin{cases} p^{(t)}(u, v_k, c) = \dfrac{w^{(0)}(u, v_k)}{\sum\limits_{v \in \mathcal{V}_u} w^{(0)}(u, v)} & \text{if } t = 0 \\ p^{(t)}(u, v_k, c) = \dfrac{(\Omega^{(t)}(u, v_k, c))^{\alpha} (w^{(t)}(u, v_k))^{\beta}}{\sum\limits_{v \in \mathcal{V}_u} (\Omega^{(t)}(u, v, c))^{\alpha} (w^{(t)}(u, v))^{\beta}} & \text{if } t \neq 0 \end{cases} \tag{6}$$

The parameters α and β (both > 0) allow the weighting of the relative importance of pheromones and respectively weights. However, if ants choices were only driven by this probability formula, there would be no way for avoiding oscillatory moves. So, we introduce a penalisation factor in equation (6) $\eta \in]0, 1]$ aiming at preventing ants from using previously crossed edges. The idea is very similar to the tabu list used in tabu search heuristic, but we add this constraint directly into the probability formula. Each ant has the ability to remember the k last vertices it has crossed. These vertices are stored into a list: $\mathcal{W}_x$ with card$(\mathcal{W}_x) < M$ (M constant). Then, the value of η for an ant x considering an edge (u, v) is equal to:

$$\eta_x(v) = \begin{cases} 1 & \text{if } v \notin \mathcal{W}_x \\ \eta & \text{if } v \in \mathcal{W}_x \end{cases} \tag{7}$$

Then, for the ant x, the probability of choosing edge (u, v) during time interval $[t, t+1[$ is equal to:

$$p_x^{(t)}(u, v_k, c) = \frac{(\Omega^{(t)}(u, v_k, c))^{\alpha} (w^{(t)}(u, v_k))^{\beta} \eta_x(v_k)}{\sum\limits_{v_q \in \mathcal{V}_u} (\Omega^{(t)}(u, v_q, c))^{\alpha} (w^{(t)}(u, v_q))^{\beta} \eta_x(v_q)} \tag{8}$$

To complete this, we introduce a demographic pressure to avoid vertices that already contain too many ants of another color. This better spread ants in the

graph to balance the load. It is modeled by an another penalisation factor $\gamma(v)$ on vertices that have a population greater than a given threshold. This factor is introduced in the formula (8). Given $N(v)$ the ant count on the vertex v and N^* the threshold.

$$\gamma(v) = \begin{cases} 1 & \text{if } N(v) \leq N^* \\ \gamma \in]0,1] & \text{else} \end{cases} \tag{9}$$

The formula (8) becomes :

$$p_x^{(t)}(u, v_k, c) = \frac{(\Omega^{(t)}(u, v_k, c))^\alpha (w^{(t)}(u, v_k))^\beta \eta_x(v_k)\gamma(v_k)}{\sum_{v_q \in \mathcal{V}_u} (\Omega^{(t)}(u, v_q, c))^\alpha (w^{(t)}(u, v_q))^\beta \eta_x(v_q)\gamma(v_q)} \tag{10}$$

3.6.3 Ants way of life

The algorithm is based on an iterative process. During the time interval $[t, t+1[$, each ant may either, hatch, move, or die.

An ant of color c, located on a vertex u dies if the proportion of color c on adjacent edges is under a threshold. If $\phi \in [0, 1]$ is the threshold, then, the ant of color c located on u dies if:

$$\frac{\sum_{v \in \mathcal{V}_u} \tau^{(t)}(u, v, c)}{\sum_{\underline{c} \in \mathcal{C}(t)} \left(\sum_{v \in \mathcal{V}_u} \tau^{(t)}(u, v, \underline{c}) \right)} < \phi \tag{11}$$

A new ant is then created in another location. This "jumping" mechanism improves the global algorithm behavior. For instance, when the graph becomes unconnected, some ants may be prisoners of isolated clusters, and the die-and-hatch sequence allow them to escape.

The mechanism, while keeping population constant, also avoids locked situations : grabs, overpopulation, starvation as shown on figure 2. These problems occur when the system meets local minima. For example, when a set of red

ants are circled by a set of blue ants inside the same organization, or when too many ants of the same color are captured in an area because of a too large volume pheromones at this place. This phenomenon is amplified by retroactions with ants attracted by pheromone, depositing more pheromones, that attracts more ants etc. It also allows to avoid areas without any ants, that are the counterparts of over-populated areas. Moreover, this improves the reactivity of our algorithm that runs continuously, not to find the best solution for a static graph but, for providing any-time solutions to a continuously changing environment.

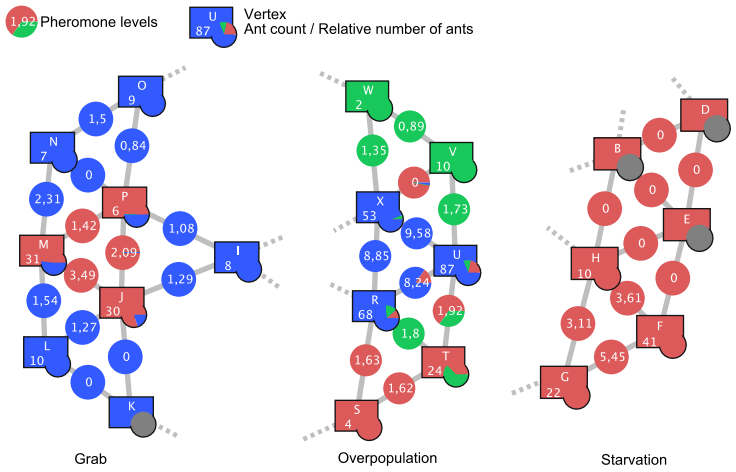


Figure 2: *Problems solved by jumping mechanism.*

3.7 AntCO^2 architecture an implementation

Figure 3 shows the AntCO^2 architecture. In this diagram AntCO^2 is seen both as a service inside a middleware and as a user of this middleware. The middleware allows communications between AntCO^2 and the distributed application, providing entity migration and communication measurement. It also serves as an interface to the computing environment providing event notification like the apparition or disappearance of resources. For the application, AntCO^2 is a service of the middleware that offers entity migration suggestions.

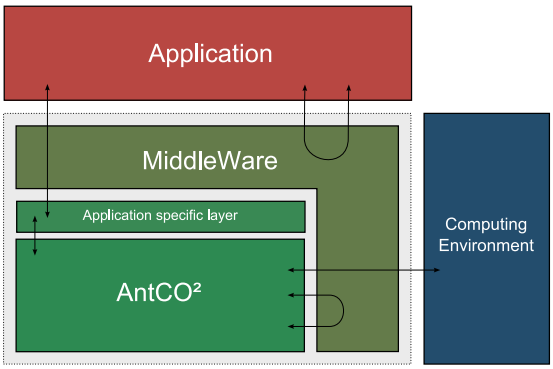


Figure 3: *AntCO*^{2} *architecture.*

As the application is distributed, AntCO^2 is also distributed, possibly with an instance on each computing resource used by the application. Each instance of AntCO^2 has a set of ants and a sub-graph representing the part of the application it is serving. Across iterations ants are exchanged between AntCO^2 instances, information events are received from the application (node or edge creation/deletion, edge valuations, etc.). Events are also received from the computing environment (computing resources added or removed for example).

At regular intervals, or when queryed by the application, AntCO^2 instances send migration advices to the application. A migration is advised, when an vertex color changes, therefore indicating that the corresponding entity in the application should be on another computing resource. The application is free to follow or ignore this advice according to specific information or constraint it has.

Several methods can be used to distribute AntCO^2 as shown on figures 4, 5 and 6. The first figure depicts an environment where the application uses the whole cluster of computers for its own use, excepted on computer dedicated to AntCO^2. The second figure shows the same architecture but with several computers for AntCO^2. Finally, the last figure, shows a possible better distribution architecture, where each AntCO^2 instance would be located on the same computer as each application part it serves.

Algorithm 1 show the ant behavior and ant environment evolution. Algorithm 2 shows the application interactions through the middleware with AntCO^2.

Algorithm 1: AntCO^2 instance

```
forever
    for all ants do
        if the ant does not die (eq. 11) then
            Choose the next vertex to visit according to pheromone level
            and edge weight (eq. 10)
    for each edge do
        Apply persistence factor ρ on pheromones τ^(t−1) (eq. 4)
    for each node do
        Search dominant pheromone color on all incident edges (eq. 5)
        and color the node with it;
    Look at application and computing environment events arrived and
    modify the sub-graph and the ant population accordingly
    Send migration suggestions to the application
end
```

Algorithm 2: Application instance

```
forever
    ...
    Look at suggestions from AntCO² and migrate entities accordingly
    Send measures, events and eventual migration decisions to AntCO²
end
```

Here is an example of an usage scenario. We take a distribution architecture as depicted in figure 6, where each computing resource holds both a part of the application and an instance of AntCO^2. To simplify explanations we will use only two computing resources, R_1 and R_2.

The application starts its two instances on App_1 and App_2 (Figure 7). Two corresponding instances of AntCO^2 are created accordingly Ant_1 and Ant_2. En-

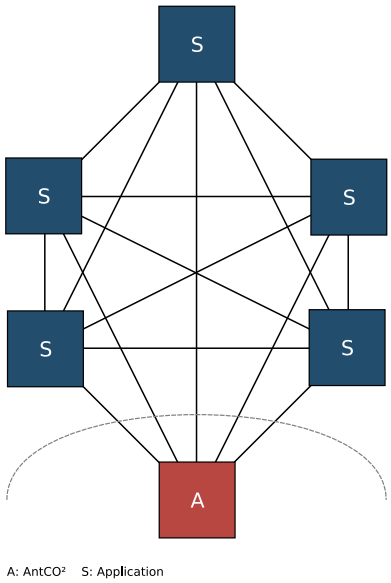


Figure 4: *Distribution model, one computer dedicated to AntCO*^{2}.

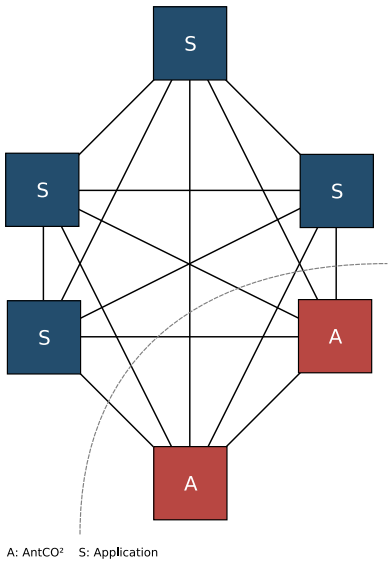


Figure 5: *Distribution model, a set of computers dedicated to AntCO*^{2}.

tities of the application start to appear on instance App_1 and notifies Ant_1 of this. The color theses entities is therefore known directly. Identically, entities appear on App_2.

Each time entities of App_1 interact one whith another, Ant_1 is notified and

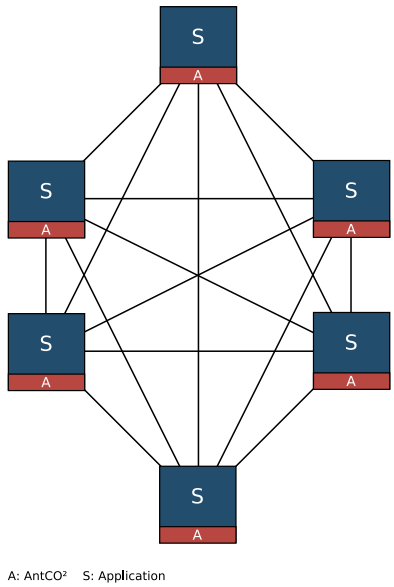


Figure 6: *Distribution model, an AntCO^2 instance runs on all computers.*

an edge is created between this entities. Several organizations may appear in the application, with entities more closely interacting with other entities of the organization than with other entities.

If an entity of App_1 in an organization interact with an entity of App_2 in an other organization, both Ant_1 and Ant_2 are notifyed. If this interaction becomes more an more important, the edge on both Ant_1 and Ant_2 will be notified and its importance will be updated. More an more ants will cross it. It is possible thant other links appear between these two organizations, leading to a colonization by ants of one color/computing resource of the two organizations. At this point a set of entities on one computing resource will be colored by the color identifying the other resource. This color change will indicate that these entities should be migrated. Indeed, they are in high interaction, they should then probably run on the same computing resource.

Identically an organization, running for example on R_1, may be split in two since the the application notifies AntCO^2 that the importance of set of edge (in-

teractions) inside the organization decrease, or disapear. If one part of this split organization has some weak relations with another organization on R_2, it may be colonized by ants of R_2 and again, AntCO^2 will detect a color change and advice the application App_1 to migrate the entities on R_2.

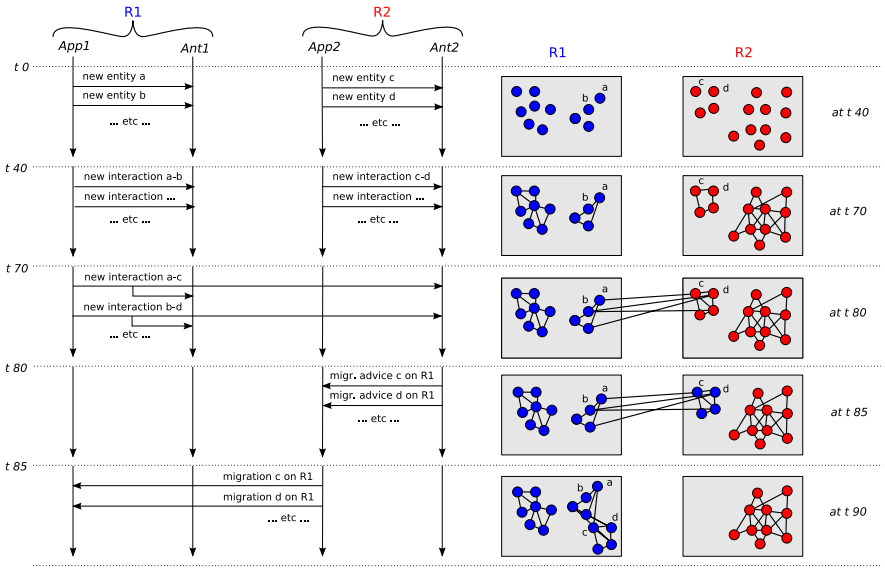


Figure 7: *Sequence diagram of AntCO^2 and an application running in parallel.*

4 Experiments and results

Dynamic load balancing falls into the category of distributed time-varying problems. It seems difficult, unless impossible, to perform a comparison to optimal solutions on dynamic graphs, because of our incapacity to compute such optimal solutions. Indeed, suppose that we were able to divide the dynamic problem into a series of static statements. Finding an optimal solution for each static statement is by itself a difficult problem, but let us assume that it would be possible. It may happen that applying the optimal solutions for two consecutive steps is not optimal because of the amount of migrations entailed by the static choices. Moreover, the structures should be kept from one step to the next one.

A dynamic graph should take into account at once also inter-steps This is why the first part of the performance analysis is dedicated to the comparison of allocations computed by our method for some classes of static graphs with optimal allocations. The second part of the analysis focuses on the reactivity and on the adaptability of our algorithm for some relevant dynamic graphs.

Before entering into details, some performance measures are defined in the next section.

4.1 Quality analysis

As previously said, two measures are relevant for qualifying the quality of a dynamic load balancing:

- The global costs of communications;
- The load-balancing of the application.

They are antagonist. So, in order to evaluate our solution we first define two quality criteria r_1 and r_2.

The first criterion $r_1 \geq 0$ corresponds to the ratio of actual communications (see section 3.1) over the total number of communications occurring in the graph. Solutions are better when r_1 is smaller.

$$r_1 = \frac{\sum_{\mathcal{B}(t)} w(u,v,t)}{\sum_{\mathcal{E}(t)} w(u,v,t)}$$

The second criterion r_2 measures how the load is balanced among the available processing ressources, independently of the communications. For each color c, we have $V_c(t)$ the set of vertices having color c at time t. Then we have:

$$r_2 = \frac{\min \mathcal{K}}{\max \mathcal{K}} \quad \text{where} \quad \mathcal{K} = \text{card}(V_c(t))$$

The load-balancing is better when r_2 is close to 1.

In case of static graphs, these criteria, enable us to store the best solutions obtained so far, that makes the method anytime.

Since we seek to find organizations, r_1 is considered to be a more important criterion. Indeed, it is explicitly defined in ant behavior, unlike r_2 that is implicitly optimized by competition mechanisms.

4.2 Dynamic Graphs

Here are two experiments we made with dynamic graphs. For these tests, we used program that simulate the application by creating a graph and then applying events to it. Events are the appearance or disappearance of an edge, a vertex or a processing resource, but also functions that change weights on edges.

The first dynamic graph represents a static grid above which another smaller grid moves. The smaller grid continuously connects and disconnects to the lower one as it moves. Typically this represents a small organization exploring a larger set of interacting entities (see figure 8)[2].

The small graph keeps the same color along the experiment while it crosses different domains of the grid having a distinct color because communications in the small graph are stronger and communication with the larger graph are less durable. Therefore, the organization formed by the small graph is not perturbed by its interactions with the grid. This graph could represent an aquatic simulation application where a fish school passes in an environment. Interactions in the fish school are based on the fish vision area and are more important, and durable, inside the school than with the environment.

It must be noted that the dynamics of the graph helps creating correct clusters, indeed this is the dynamics, and the short durability of interactions between the

[2]You can see a video of this experiment at the following URL http://litis.univ-lehavre.fr/~dutot/videos/MovingStruct2.avi.

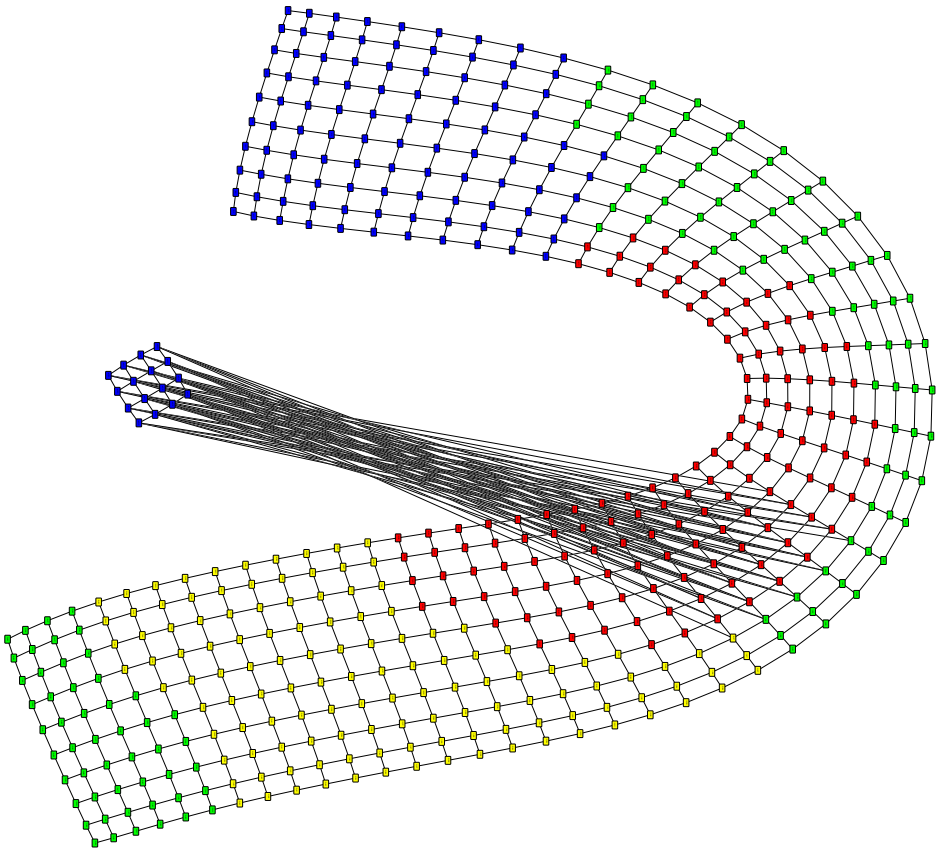


Figure 8: *A small organization browsing a larger set of interacting entities.*

larger graph and the smaller that allow to define the organization of the smaller graph. Not only the graph topology is taken into account, but the evolution of the graph in time is used.

Here are the parameters used:

Parameter	α	β	ρ	N^*	M	ϕ	ants per vertex
Value	1	4	0.8	5	4	0.3	10

he second graph is not dynamic in its topology, however, the application it represents is executed on a set of processors whose number varies. As said previously, there are several forms of dynamics and the dynamic of the computational environment must also be taken into account. The application entities are mapped as a grid (for example, many numerical physics simulations that use a mesh).

In figure 9, a 30×30 static grid is used, introducing new colonies, that is computing resources, at regular intervals every 1000 steps. We can compare this approach to a recursive bisection, but using a non power of two number of partitions. This test simulates a computing environment inside which more and more computing resources become available. The figure also shows the evolution of the two quality criteria r_1 and r_2 during this test. We can see that besides the small size of the grid that brings solutions at a lower rate at each new computing resource added, the algorithm stabilizes on good quality solutions.

It is interesting to notice that no global reorganization of the graph : organization are globally maintained in place when a new computing resource is added. Indeed some areas are cut whereas the other parts stay in position therefore limiting the number of migrations.

The same test has been made on a quite larger graph containing 2851 vertices and 15093 edges (figure 10 that only shows the evolution of criteria r_1 and r_2).

The parameters used where :

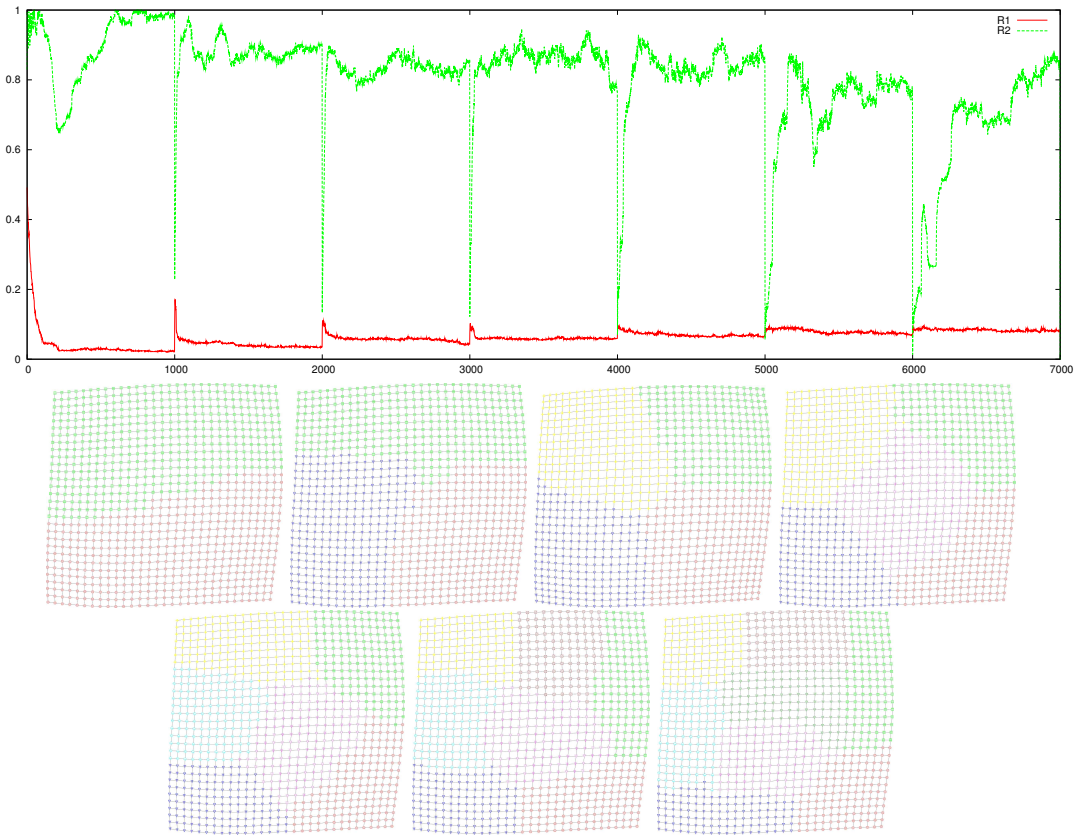


Figure 9: Adding a colony every 1000 steps in a 30×30 grid.

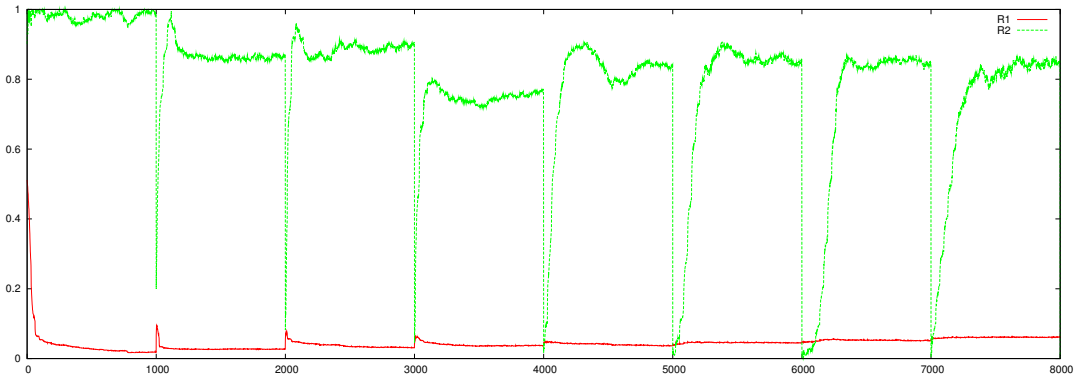


Figure 10: Adding a colony every 1000 steps on a large graph (2851 vertices and 15093 edges).

Parameter	α	β	ρ	N^*	M	ϕ	ants per vertex
Value	1	3	0.8	10	3	0.3	8

4.3 Distributed simulations

Our algorithm is particularly suited for dynamic graphs. The relevance of our method lies in the incremental computation of the solution. A change in the input (the dynamic graph, the number of computing resources) can interrupt the computation at any time, it will continue from here to compute a new solution (as seen in the previous section).

Indeed, at the base component level, the dynamic graph is constantly being reconfigured. At the contrary, taken as a whole, long lasting organizations appear. They are the image of organizations appearing in the distributed application whose the graph is a representation. Such organizations often have a larger time to live than the average duration of edges or vertices of the graph. Inside these organizations, communication is higher both in terms of volume and connectivity. And these two last points are criteria used by ants to form clusters.

We tested our algorithm on graphs deduced from an ecosystem simulation where entities have a boid-like behavior, made of three rules:

- avoidance: they try to stay at a small distance of perceived boids,
- cohesion: they try to fly toward the average position of all perceived boids,
- alignment: they try to match velocity with perceived boids.

These rules create one or several groups of boids. Furthermore, boids try to avoid predators introduced in the simulation. Predators destroy some boids, and cut boids groups in sub-groups.

Each boid is modeled by a vertex in the dynamic graph. When a boids comes into the field of view of another this create an interaction (boids reacting to others in their field of view) and therefore an edge in the graph. Figures 11 and 12 show

both the boids simulation and the corresponding colored dynamic graph. On figure 12 boids group formed, and the graph shows the corresponding clusters, detected by AntCO^2 as shown by colors.

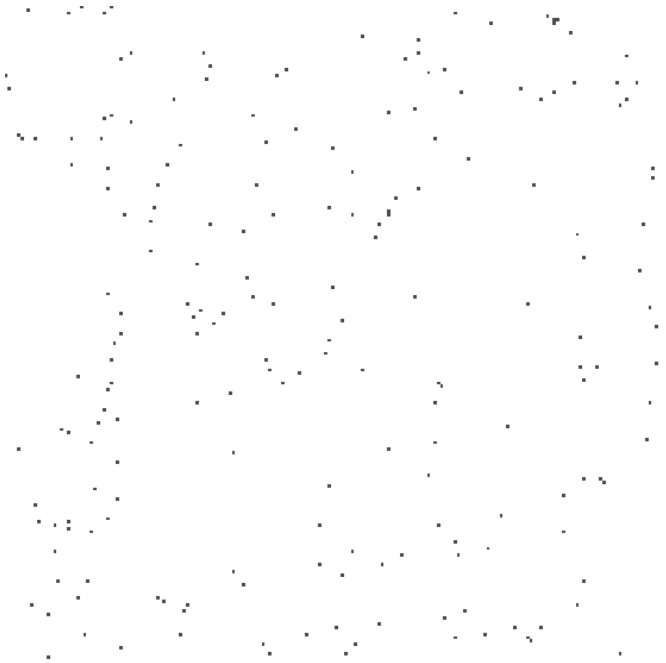
We compared the distribution found by AntCO^2 with the two other methods:

1. random,
2. and using a grid-like environment.

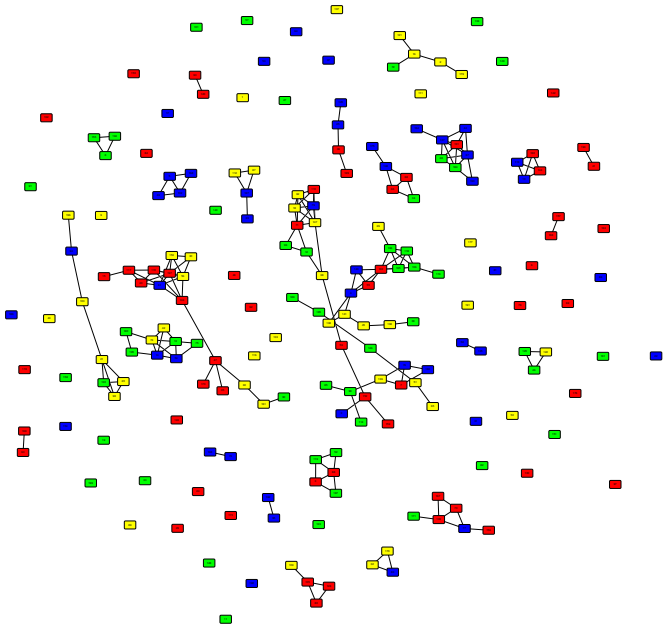
In random mode, as soon as a boid appear, a processor is assigned to it and it will never change afterward. In the simulations we used, the boid count is fixed once and for all, therefore this load balancing mode is at any time optimal. Each processor is charged the same (depending on respective powers), however communication costs are not taken into account. Inside an organization there are chances that two communicating boids are on two distinct processors. This distribution mode is therefore the worst concerning the network load.

The grid-like environment cuts the space in rectangular cells, and each cell is assigned to a computing resource. Boids execute on the resource of the cell they cross. This mode improve communication minimization but degrades load balancing compared to the random mode. Indeed, with this mode, as boids travel without constraints, it is possible that all individuals migrate to the same grid cell, hence on the same processor. Moreover, concerning communications, it is possible that an organization stay on the frontier of two cells handled by two distinct processors hence forcing high communications to use the network.

These two methods have been compared to the results produced by AntCO^2. We see that this last one gives a good trade-off between load balancing and communication minimization. Concerning quality criterion r_1, communications, results are better than the two other approaches. For r_2, load balancing, AntCO^2 is better than the grid-like distribution mode. It cannot be better than the random distribution mode that is optimal when considering only load balancing.



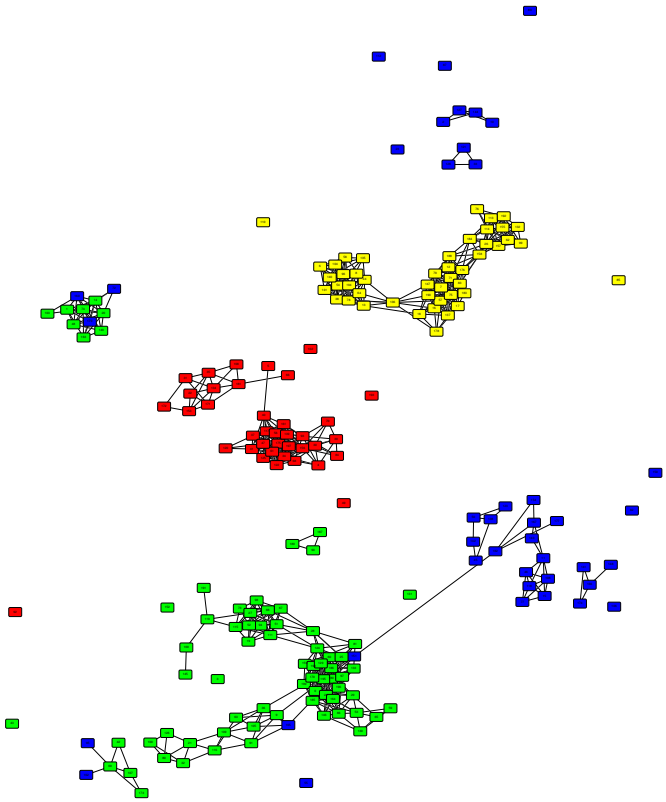
(a) Simulation



(b) Colored relation graph between entities of the simulation

Figure 11: *At the beginning of the simulation.*

(a) Simulation



(b) Graph

Figure 12: *Later on in the simulation.*

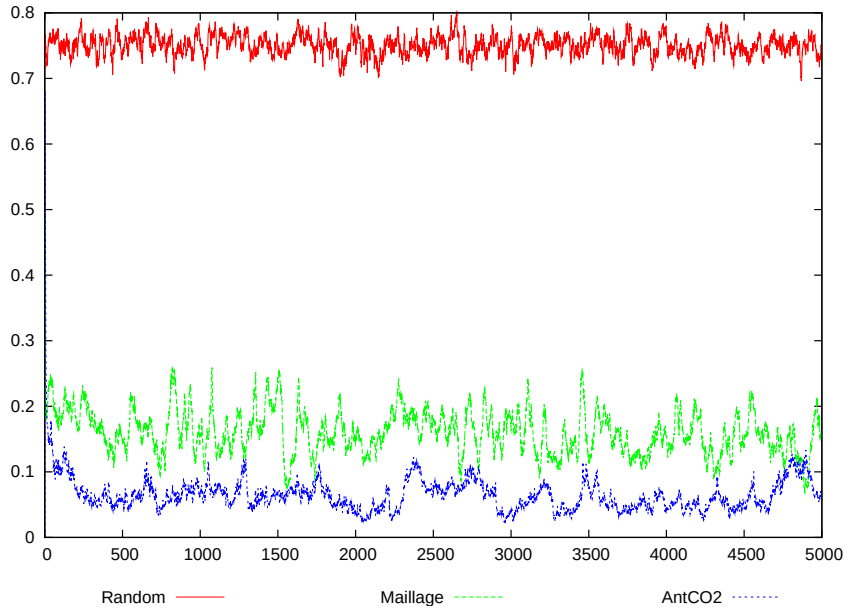


Figure 13: Comparison on criterion r_1 between distribution modes "random", "grid-like" and AntCO^2 with 200 boids.

Figures 13 and 14 show the evolution of criteria r_1 and r_2 respectively on a test with 200 boids spread in 4 species during 5000 time steps. These figures compare the three distribution modes described above. For r_2 the random model, always optimal, is not shown.

The parameters that where used for AntCO^2 are:

Parameter	α	β	ρ	N^*	M	ϕ	ants per vertex
Value	1	1	0.6	5	3	0.25	4

5 Conclusion

In this paper we presented an colored ant algorithm allowing to detect and distribute dynamic organizations. The algorithm offers advices for entity migration in a distributed system taking care of the load and communication balancing. We described a base colored algorithm, observed its behaviour with static and dynamic graphs and provided methods to handle them.

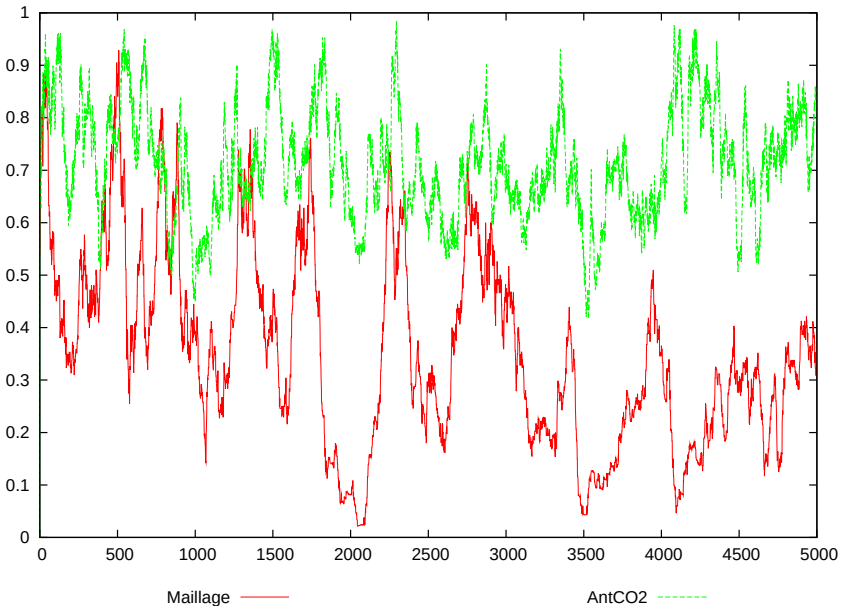


Figure 14: Comparison on criterion r_2 between distribution modes "grid-like" and AntCO^2 with 200 boids.

Our algorithm handles dynamic graphs. Two properties of the algorithm allow this: *positive feedback* maintain paths in the graph between highly correlated vertices, *negative feedback* isolate these communities. The first is controlled by ants (pheromone drops), while the other is completely driven by the environment (evaporation, edge deletion, weights). Negative feedback is what makes our algorithm truly adaptive to dynamic graphs, allowing to forget bad communities introduced by dynamics.

Organizations emerge from the ant behavior. Such an emergence is not explicitly implemented. These organizations make up solutions, which is the reason why we don't need an objective function at the contrary of traditional ant systems [11].

We favor the r_1 criterion, communication minimisation, above the r_2 criterion, load balancing, since the algorithm searches for organizations. The r_1 criterion is explicitly defined in ant behavior whereas r_2 is only implicitly expressed

by colored ant competition.

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 information peculiar to the application.

This work takes place within the context of aquatic ecosystem models[3], where we are faced to a very large number of heterogeneous auto-organizing entities, from fluids representatives to living creatures presenting a peculiar behaviour.

References

[1] D. J. Barnes and T. R. Hopkins. The impact of programming paradigms on the efficiency of an individual-based simulation model. *Simulation Modelling Practice and Theory*, 11(7-8):557–569, 2003.

[2] C. Bertelle, A. Dutot, F. Guinand, and D. Olivier. DIMANTS: a distributed multi-castes ant system for DNA sequencing by hybridization. In *in NETTAB 2002*, pages 1–7, Bologna (Italy), 15 july 2002.

[3] C. Bertelle, V. Jay, and D. Olivier. Distributed multi-agents systems used for dynamic aquatic simulations. In D.P.F. Müller, editor, *ESS'2000 Congress*, pages 504–508, Hambourg, September 2000.

[4] B. Breckling, U. Middlehoff, and H. Reuter. Individual-based models as tools for ecological theory and application: Understanding the emergence of organisational properties in ecological systems. *Ecological Modelling*, 194(1-3):102–113, 2006.

[5] G. Di Caro and M. Dorigo. Antnet: A mobile agents approach to adaptive routing. Technical report, IRIDIA, Université libre de Bruxelles, Belgium, 1997.

[6] David R. Cope. Individuality in modelling: a simplifying assumption too far ? *Nonlinear Analysis: Real World Applications*, 6(4):691–704, 2005.

[7] D. Costa and A. Hertz. Ant can colour graphs. *Journal of Operation Research Society*, 48(3):295–305, 1997.

[8] L. J. Gross D. L. DeAngelis. *Individual-based models and approaches in ecology: populations, communities and ecosystems*. Chapman and Hall - New York, 1992.

[9] J.-L. Deneubourg and S. Goss. Collective patterns and decision making. *Ethology Ecology and Evolution*, 1(4):295–311, 1989.

[10] M. Dorigo and L.M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[11] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: optimization by a colony of cooperating agents. *IEEE Transactions Systems Man Cybernetics*, 26:29–41, 1996.

[12] G. Theraulaz E. Bonabeau, M. Dorigo. *Swarm Intelligence: from natural to artificial systems*. Oxford University Press, 1999.

[13] B. Faieta and E. Lumer. Diversity and adaptation in populations of clustering ants. In *Conference on Simulation of Adaptive Behaviour*, Brighton, 1994.

[14] V. Ginot, C. Le Page, and S. Souissi. A multi-agents architecture to enhance end-user individual-based modelling. *Ecological Modelling*, 157(1):23–41, 2002.

[15] K. Glass, M. Livingston, and J. Conery. Distributed simulation of spatially explicit ecological models. In *Proceedings of the 11th Workshop on Parallel*

and Distributed Computing (PADS'97). Vienna (Austria), june 10-13, pages 60–63, 1997.

[16] D. M. Gordon. The expandable network of ant exploration. *Animal Behaviour*, 50:995–1007, 1995.

[17] S. Goss, S. Aron, J.-L. Deneubourg, and J. M. Pasteels. Self-organized shortcuts in the argentine ant. *Naturwissenchaften*, 76:579–581, 1989.

[18] V. Grimm. Ten years of individual-based modelling in ecology: what have we learned and what could we lear in the future ? *Ecological Modelling*, 115(2-3):129–148, 1999.

[19] H.-U. Heiss and M. Schmitz. Decentralized dynamic load balancing: The particles approach. *Information Sciences*, 84:115–128, 1995.

[20] F. Hölker and B. Breckling. A spatiotemporal individual-based fish model to investigate emergent properties at the organismal and the population level. *Ecological Modelling*, 186(4):406–426, 2005.

[21] F. Jopp and H. Reuter. Dispersal of carabid beetles - emergence of distribution patterns. *Ecological Modelling*, 186(4):389–405, 2005.

[22] 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.

[23] C. G. Langton. *Artificial Life*. Addison Wesley, 1987.

[24] C. R. Johnson P. K. Dunstan. Predicting global dynamics from local interactions: individual-based models predict complex features of marine epibenthic communities. *Ecological Modelling*, 186(2):221–233, 2005.

[25] H. Reuter, F. Hölker, U. Middelhoff, F. Jopp, C. Eschenbach, and B. Breckling. The concept of emergent and collective properties in individual-based

models - summary and outlook of the Bornhöved case studies. *Ecological Modelling*, 186(4):489–501, 2005.

[26] S. Hassas S. Fenet. Ant based system for dynamic multiple criteria balancing. In *First International Workshop on Ant Algorithms (ANTS'98)*, October 15-18, Brussels (Belgium), 1998.

[27] F. Müller S. N. Nielsen. *Joergensen and Müller (eds.): Handbook of ecosystems theories and management.*, chapter Emergent properties of ecosystems, pages 195–216. CRC publishers - New York, 2000.

[28] Mark Scahill. Distributed individual-based environmental simulation. In Ralf Denzer, David A. Swayne, and Gerald Schimak, editors, *Environmental Software Systems*, volume 2, pages 269–276. Chapman & Hall, May 1997. Presented at ISESS97 in Whistler, Canada.

[29] T. White. Routing with swarm intelligence. Technical Report SCE-97-15, Systems and Computer Engineering Department, Carleton University, September 1997.

6 Notations

Notation	*Signification*
$G(t) = (\mathcal{V}(t), \mathcal{E}(t))$	Dynamic communication graph at time t.
$G(t) = (\mathcal{V}(t), \mathcal{E}(t), \mathcal{C}(t))$	Dynamic colored communication graph at time t.
$u, v \in \mathcal{V}(t)$	Vertices.
$e = (u, v) \in \mathcal{E}(t)$	An undirected edge.
$c \in \mathcal{C}(t)$	A color.
$n \in \mathbb{N}^+$	Vertex count.
$p \in \mathbb{N}^+$	Colors / computing resource count.
$t \in \mathbb{N}^+$	Time.
x	Usually an ant.
$N(v) \in \mathbb{N}$	Ant count for vertex v.
$w^{(t)}(e) \in \mathbb{N}^+$	Weight of edge e at time t.
$\mathcal{V}(t)$	Set of vertex at time t.
$\mathcal{V}_c(t)$	Set of vertices having color c at time t.
$\mathcal{E}(t)$	Set of edges at time t.
$\mathcal{E}_u(t)$	Set of edges adjacent to vertex u at time t.
$\mathcal{A}(t)$	Set of actual communication edges at time t.
$\mathcal{C}(t)$	Set of colors at time t.
$\mathcal{B}(t)$	Edge-cut at time t.
$\mathcal{D}(t)$	Partition of $G(t)$ at time t.
$\mathcal{D}_i(t)$	Domain making up partition $\mathcal{D}(t)$.
$\mathcal{F}(t)$	Ant population at time t.
$\mathcal{F}_c(t)$	Ant population having color c at time t.
$\Delta_x^{(t)}(e, c)$	Pheromone quantity of color c dropped by ant x on edge e during time interval $]t-1, t]$.
$\Delta^{(t)}(e, c)$	Pheromone quantity of color c dropped by all (c colored) ants on edge e during time interval $]t-1, t]$.
$\Delta^{(t)}(e)$	Pheromone quantity dropped by all ants on edge e during time interval $]t-1, t]$.
$K_c^{(t)}(e)$	Pheromone of color c rate on edge e at time t.
$\tau^{(t)}(e, c)$	Total pheromone quantity of color c on edge e at time t.
$\tau^{(t)}(e)$	Total pheromone quantity on edge e at time t.

Notation	*Signification*
$\Omega^{(t)}(e, c)$	Corrected reinforcement factor.
$\xi^{(t)}(u)$	Color of vertex u at time t.
$p_x^{(t)}(e, c)$	Probability for an ant of color c to cross edge e at time t.
α	Pheromone importance.
β	Weight importance.
$\rho \in]0, 1]$	Pheromone persistence factor.
$\gamma(v)$	Vertex v demographical pressure.
N^*	Demographic pressure threshold.
W_x	Tabu list for ant x.
M	Tabu list size.
$\eta_x(u)$	Penalisation factor for ant x to visit vertex u again.
ϕ	Jump threshold.
r_1	Actual communications proportion over all communications.
r_2	Minimal partition size over maximal partition size.