Dynamic Placement Using Ants for Object Based 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 distributed application may be considered as a set of interacting entities continuously evolving. Such application can be modeled as a graph with one-to-one mappings between vertices and entities and between edges and communications. 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 are achieved if and only if a good tradeoff is found. Our method for finding such a tradeoff is new and based on colored ant colonies. Each computing resource is associated to one ant colony characterized by a color, allowing an implicit consideration of the load balancing constraint. Then, using colored pheromones, ants are just seeking for communicating structures. The method operates on graphs which structural and numerical parameters may change dynamically during the execution.

Keywords: Ant algorithms, dynamic graph, clustering, auto-organization, distributed applications .

1 Introduction

In distributed application, often a very large number of entities are used to represent a complex system. The dynamics of such systems discourages a static distribution made upstream, before application execution. As the system evolves communications between entities change. Communications and entities may appear or disappear, creating organizations. As a consequence, an entity location that was correct at the beginning, can severely impact performance two hundred time steps after. 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[6] is described that advises on a possible better location of some entities according to the tradeoff between load balancing and minimization of communications overhead.
The Paper is organized as follows. Section 2 provides some background about ant algorithms and some of their applications. Section 3 details the graph representing the distributed application. Operating on this graph, our colored ant system is described in section 4. Finally, our implementation is discussed in section 5 and illustrated by some experiments, before we conclude with further expected improvements and perspectives for this system.

2 Ant Algorithms

Ant algorithms are a class of meta-heuristics based on a population of agents exhibiting a cooperative behaviour[10]. Ants continuously forage their territories to find food[8] 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].
For example, ants perform such indirect communications using chemical signals called *pheromones*. The larger the quantity of pheromones on a path, the larger the number of ants visiting this path. As pheromones evaporate, long paths tend to have less pheromone than short ones, and therefore are less used than others (binary bridge experiment).
Such an approach is robust and well supports parameter changes in the problem. Besides, it is intrinsically distributed and scalable. It uses only local informations (required for a continuously changing environment), and find near-optimal solutions. Ant algorithms has been applied successfully to various combinatorial optimization problems like the Travelling Salesman Problem[5] or routing in networks[3,11], but also to DNA sequencing[1], graph partitioning[9], coloring[4] and clustering[7].

3 Dynamic Communication Graph

3.1 Model

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 represented by vertices v_i and v_j. Communications direction being without effect on the ant algorithm, edges are undirected. Edges are labeled by weights representing 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 clusters of highly communicating entities in order to map all entities belonging to one cluster on the same computing resource. Of course, one trivial solution is obtained by mapping all entities on only one computing resource. In order to avoid this, we use several colored ant colonies that produce colored clusters, each color corresponding to one computing resource.

3.2 Dealing With a Dynamic Environment

As said above, the graph represents the application as it runs, it is therefore dynamic at several levels:

- weights change continuously;
- edges, that is communications, can appear and disappear at any time;
- vertices, that is 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.
Indeed, a monotonic approach is one way to achieve clustering on a graph. It consists in regularly applying a computation on a frozen copy of the dynamic graph, then trying to use this information, though the real graph is still evolving. This approach is problematic: the graph can have changed during computation and results may not be usable any more, creating discrepancies between the real application state and calculated migration hints. Furthermore, it is not incremental, each time the algorithm is performed anew.

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

Another way is to use an anytime algorithm. The dynamic graph is considered as a changing environment for computing entities that travel on the graph, taking into account the changes as they appear, and storing the solution directly in the graph, as an effect of their evolution. Ant algorithms are well suited for that task as it has been shown in[6]. Moreover this approach is implicitly distributed and results can be stored directly in the graph.

4 Colored Ant System

As shown above, we model large scale distributed applications by a dynamic graph $G = (\mathcal{V}, \mathcal{E})$. The ant algorithm is used to detect clusters of highly communicating 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 $G = (\mathcal{V}, \mathcal{E}, \mathcal{C})$ such that:*

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

4.1 The Colored Ant Algorithm

Our algorithm is inspired by the Ant System[6]. We consider a dynamic communication colored graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{C})$.

- 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 explained in section 4.3.
- 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. Moreover, each ant has the ability to remember one or more vertices it comes from.
  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:

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

  - The total quantity of pheromone of all colors dropped by ants on edge (u, v) between steps $t-1$ and t is noted:

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

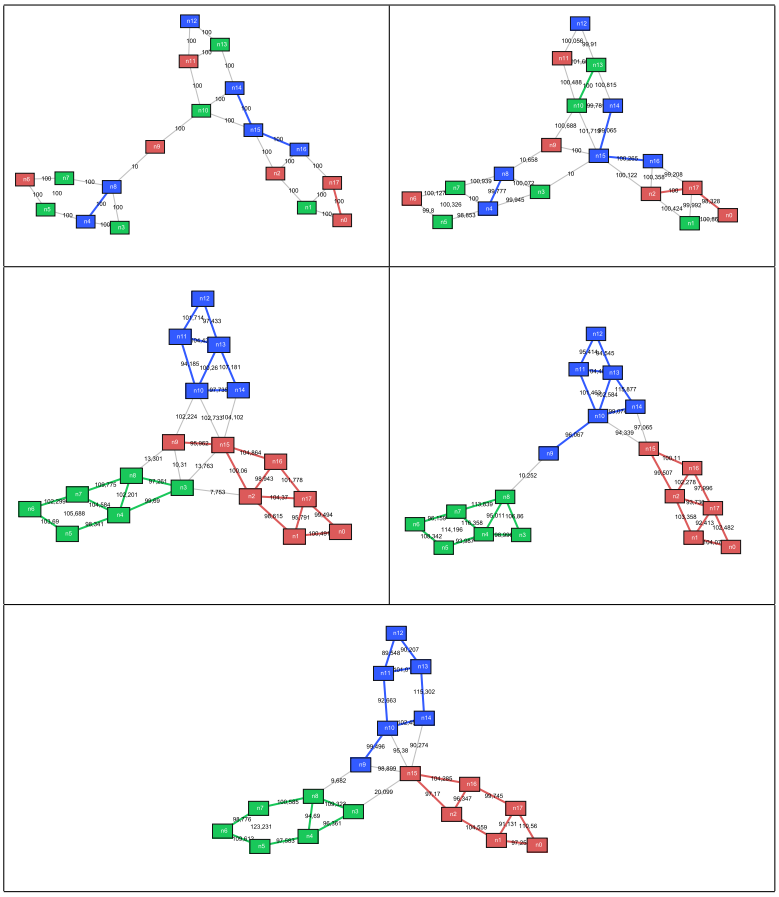

Fig. 1. *Example of a dynamic communication graph at five stages of its evolution*

- 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 auto-organization process. For this purpose, we need to balance this reinforcement factor whith $K_c^{(t)}(u,v)$, the relative importance of considered color with regard to all other colors. This corrected reinforcement factor is noted:

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

Unfortunaly, 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 $w(u,v_k)$.
  - At the initial step $(t=0)$,

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

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

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

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 choice of the next edge crossed by an ant depends on the previous probabilities. However, to avoid the ant moves to oscillate between two vertices, we introduce in the formula a penalisation factor $\eta \in [0,1]$. Given $\mathcal{W}_x$ the set of the last vertices visited ant x with $|\mathcal{W}_x| < M$, the new probability formula for the specific ant x is:

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

Where

$$\eta_{x,q} = \begin{cases} 1 \text{ if } v_q \notin \mathcal{W}_x \\ \eta \text{ if } v_q \in \mathcal{W}_x \end{cases} \tag{9}$$

- The color of a vertex u, noted $\xi(u)$ is obtained from the main color of its incident arcs:

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

4.2 Solution Quality

It is necessary to have a measure of the quality of the solution, to know if we improve the obtained solution. There are two aspects to take into account :

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

They are antagonist. So, in order to evaluate our solution we first defined two quality criterions r_1 and r_2. The first criterion r_1 identifies between two solutions which has proportionally less actual communications. Thus we compute actual communication costs, noted a, by summing actual communications on the graph (between entities located on distinct processing resources). Then we compute a 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 having color c and p_c the power of processing resource affected to c as defined in section 4.3. 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 more r_2 is close to 1, the better the load-balancing. For example, we obtain on the two graphs (Figure 1) $r_1 = 0.15$, $r_2 = 1.0$ for $t = 0$ (first graph) and $r_1 = 0.88$, $r_2 = 1.0$ for $t = 300$ (last graph).
These criterions are used to compare different solutions obtained during the computation, essentially to verify if we improve the solution during the steps. These criterions, enable us to store the best solution obtained so far.
We use also these criterions to compare communication graphs where clusters are already identified. For these graphs colors allocations are randomly shuffled. Then the algorithm tries to find the original allocation on the graph as a solution, or a solution where the criterions are closest.

4.3 Dynamic Aspects

As described above, the algorithm does not specify how it handles dynamic aspects of the graph. We indeed also need to define what happens when:

- an edge appears or disappears;
- a vertex appears or disappears;
- a processing resource appears or disappears.

We need to maintain a given level of population for the algorithm to work. When there are too few ants, evaporation makes pheromones disappear and the algorithm becomes a variant of a greedy algorithm. If there are many ants, pheromones take a too large part and the system efficiency decreases.
Furthermore, the population must take into account the number of entities to distribute and the number of processing resources. A graph with twenty entities will not need as many ants as a graph with three thousand entities.
Therefore ant allocation strategy is:

- When an entity e appears, we allocate $floor(N * p_c) * |\mathcal{C}|$ ants, with N an integer constant greater than 1 and p_c a number $\in \mathbb{R}^+$ representing the power of the processing resource characterized by color c. We place these ants on the vertex corresponding to e. This ensures that each processing resource has a number of ants dependant of its power and that the number of ants is related to the number of entities. Colors are assigned uniformly to this set of ants.
- In the same way, when an entity disappears, we remove randomly $floor(N * p_c) * |\mathcal{C}|$ ants, not necessarily on the vertex that disappear. Remaining ants that are on disappearing vertex and that are not removed die and hatch on a new vertex as explained in section 4.4.
- When an edge disappears we mererly do nothing. Indeed this only affects ants on the possible path they can follow. Identically, when an edge appears, we do nothing, ants can potentially use it immediately.
- when a processing resource of color $c \in \mathcal{C}$ appears, we allocate $N * p_c * |\mathcal{V}|$ ants of the new color c, that we spread over the graph uniformly.
- When the processing resource disappears, all ants of its color simply die.
- When the power of the processing resource of color c changes between steps $t-1$ and t (because it is more or less available) we make Δ_{ants} ants die or hatch (according to the sign of Δ_{ants}). Let $\mathcal{F}(\mathcal{C})$ be the set of ants of color $\mathcal{C}$, we compute the difference: $\Delta_{ants} = |\mathcal{F}(\mathcal{C}|) - (N * p_c^{(t)} * |\mathcal{V}|)$.

4.4 Further Improvements

Furthermore, the algorithm as stated above has several problems, mostly due to dynamics, but not only, we need to improve what happen if:

- the graph becomes a not connected graph?
- we find local minima, ants running across some preferred edges and not others, stigmergy increasing the process over and over?

To tackle these problems, we add some death and hatching mechanisms. Again, we want to maintain a stable level of population to avoid problems cited in section 4.3. Therefore, we choosed to make one hatch for one death. The goal is to perturb the ants repartition that is 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. In order to do that we try to determine when an ant is at the wrong location and unable to leave it by itself. If we detect such a case, we kill the ant and make a new one hatch at a selected position (always to keep population constant).

The algorithm is modified to detect such cases.

1. We define the following elements:
   - $\tau^{(t)}(u, c)$ is the quantity of pheromone of color c dropped on all edges connected to vertex u:
   $$\tau^{(t)}(u, c) = \sum_{v_q \in \mathcal{V}_u} \tau^{(t)}(u, v_q, c) \tag{11}$$
   - $\tau^{(t)}(u)$ is the quantity of pheromone of all colors dropped on all edges connected to vertex u:
   $$\tau^{(t)}(u) = \sum_{c \in \mathcal{C}} \tau^{(t)}(u, c) \tag{12}$$
   - $\varphi_c(u) \in [0, 1]$:
   $$\varphi_c(u) = \frac{\tau^{(t)}(u, c)}{\tau^{(t)}(u)} \tag{13}$$
   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 and 8), we must choose weither the ant will die or not. We determine this using a threshold parameter $\phi \in [0, 1]$ 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 $|\mathcal{F}(v)|$ be the number of ants on vertex v. Then we select a vertex u in $\mathcal{V}_n$ using:
   $$u = \arg \min_{v \in \mathcal{V}_n} (|\mathcal{F}(v)|) \tag{14}$$
   and make the new ant hatch on it.
   - else, we proceed as specified in the original algorithm choosing a new edge using probablities (equation 6 and following).

This mechanism brings several advantages. First this eliminate problems tied to disconnected graphs, as shown in figure 2. In this figure, the graph oscillates between a configuration where it is made of four non connected subgraphs (where ants could otherwise stay blocked), and a grid configuration. The first stage shows the initial configuration. The second is taken 30 steps after, clusters have formed with a very high level of death and hatching. At the third stage, the graph changes and becomes a grid. twenty steps after clusters reappear according to communication in the grid. In the fourth stage, as the graph changes anew, some parts of the grid clusters tend to remain.

This mechanism, while keeping population constant, allows to avoid local minima, small stable clusters inside others.

Finally, this procedure does not need a global system to observe the distributed application, all can be done locally (since hatching is random).

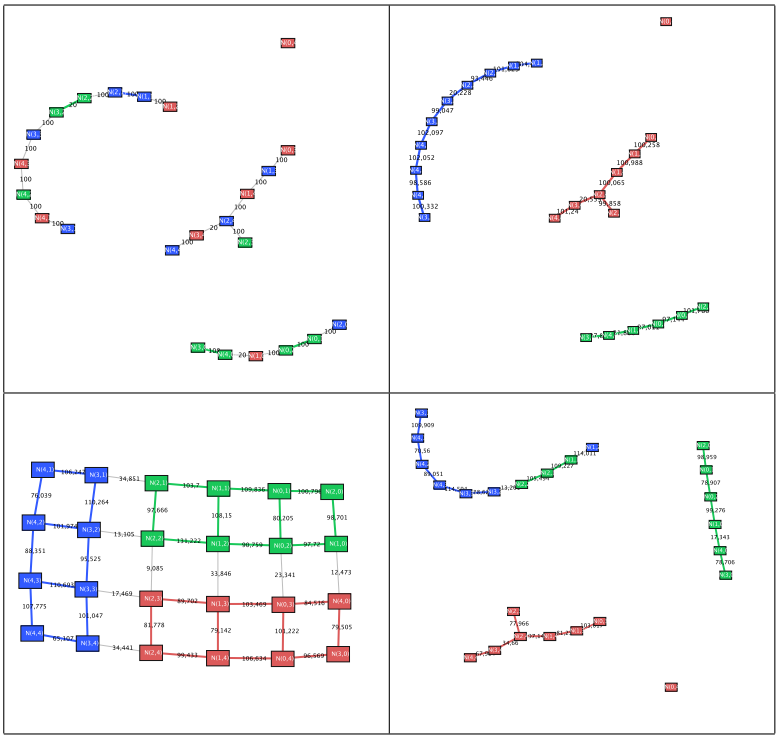

Fig. 2. *Example of a disconnected dynamic graph*

5 Implementation and Experimentation

Here are several experiments we made with two 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.

In the following 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.

The first experiment, already shown in figure 1 and detailed in figure 3 is a small graph (18 vertices), where three main communication clusters appear. These clusters are linked at the center by low communication edges that appear and disappear. Inside the clusters some edges also appear and disappear. For this experiment we used parameters $\alpha = 1.0$, $\beta = 4.0$, $\rho = 0.8$ $\phi = 0.3$, $\eta = 0.0001$, $N = 10$ and $|\mathcal{W}_x| = 4$ vertices. These parameters will be the same for all other experiments excepted when noted.

The second experiment used a bigger graph (32 vertices) that continuously switch between three configurations. Figure 4 shows two views for each snapshot of the graph. The first one, in the first column, shows all informations as explained above, the second only shows cluster relevant informations and may be easier to follow. Six snapshots of the graph are presented and show that clusters remains stable across reconfigurations.

6 Conclusion

In this paper we have presented a variant of the Ant System called Colored Ant System 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 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[2], 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. C. Bertelle, A. Dutot, F. Guinand, and D. Olivier. Dimants: a distributed multicastes ant system for dna sequencing by hybridization. In *NETTAB 2002*, pages 1–7, AAMAS 2002 Conf, Bologna (Italy), July 15th 2002.
2. 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.
3. G. Di Caro and M. Dorigo. Antnet: A mobile agents approach to adaptive routing. Technical report, IRIDIA, Université libre de Bruxelles, Belgium, 1997.
4. D. Costa and A. Hertz. Ant can colour graphs. *Journal of Operation Research Society*, (48):105–128, 1997.
5. 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.
6. 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.
7. B. Faieta and E. Lumer. Diversity and adaptation in populations of clustering ants. In *Conference on Simulation of Adaptive Behaviour*, Brighton, 1994.
8. D.M. Gordon. The expandable network of ant exploration. *Animal Behaviour*, 50:995–1007, 1995.
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. C.G. Langton, editor. *Artificial Life*. Addison Wesley, 1987.
11. T. White. Routing with swarm intelligence. Technical Report SCE-97-15, 1997.

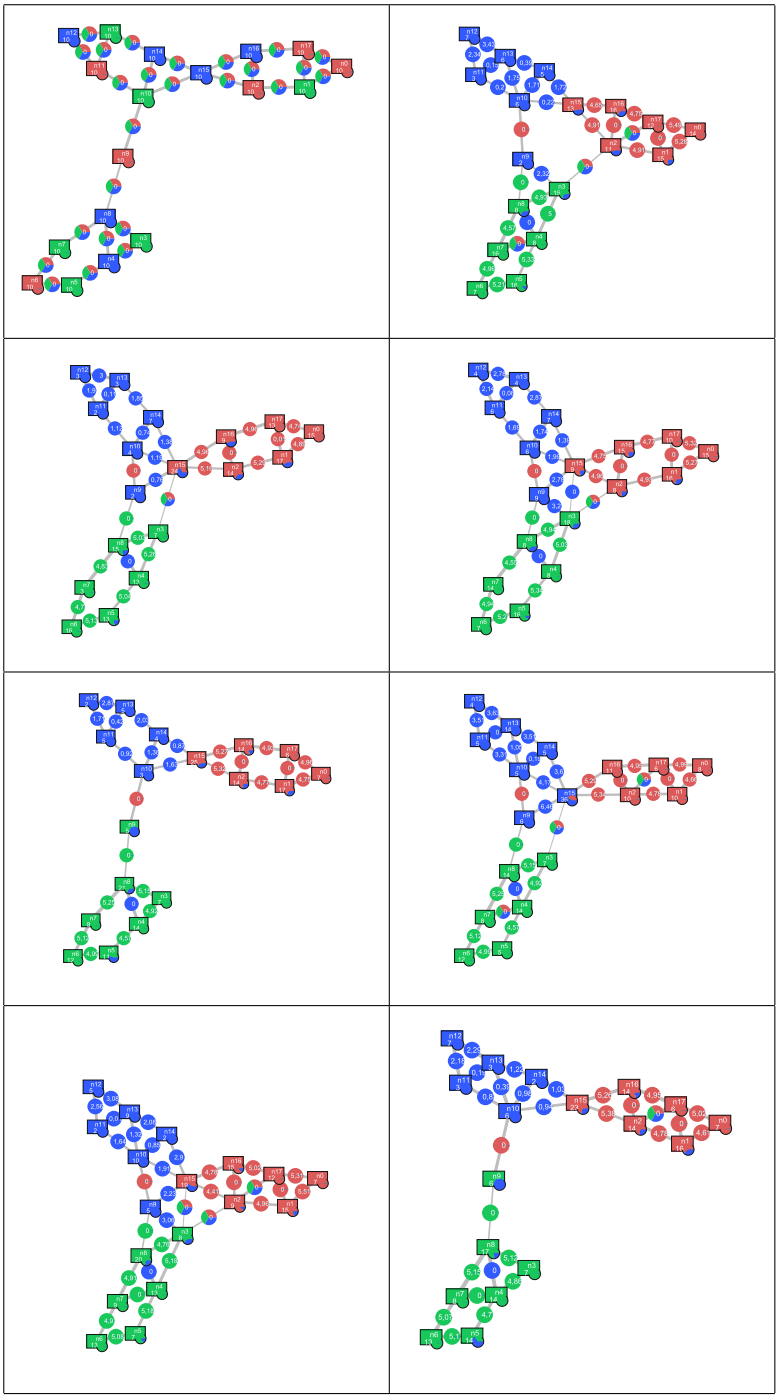

Fig. 3. *Experiment 1*

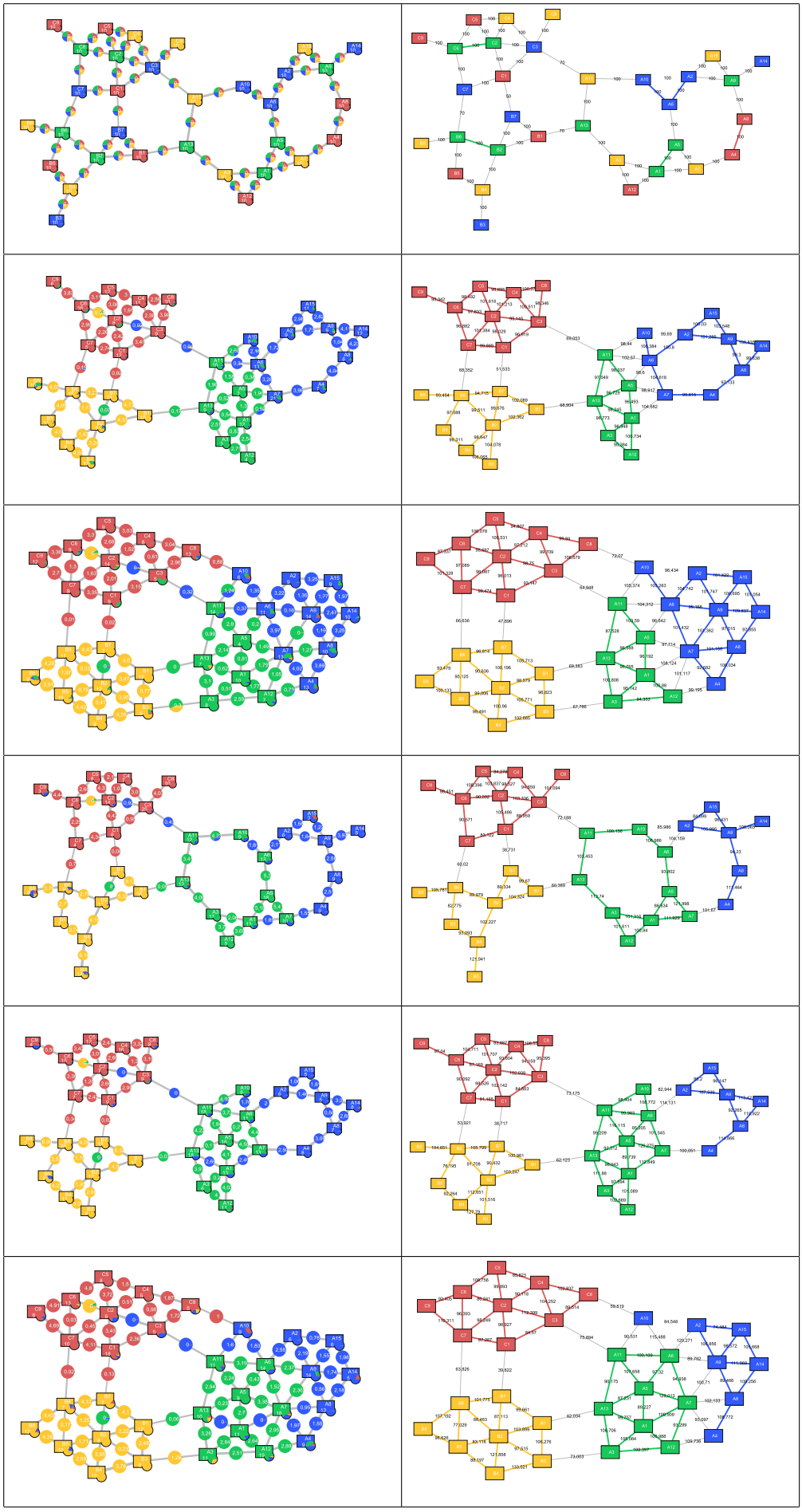

Fig. 4. *Experiment 2*