Swarm Intelligence role in Multi Agent Systems coordination

Danilo Pianini

February 28, 2010

Abstract

This work is made in the sphere of the course of Multi Agent Systems L-S 2009.

The main focus is on the motivations and theoretical bases of the research project developed at the Florida Institute of Technology during the summer 2009 in collaboration with Sascia Virruso and prof. Ronaldo Menesez.

Contents

1 Motivations **4**

2 Main concepts **4**
2.1 Agents and Multi Agents Systems 4
2.2 Coordination models 5
2.3 Linda coordination system and tuple centres 6
2.4 Swarm Intelligence 6
2.5 Stigmergy 7

3 Scale free networks **7**

4 Entropy **8**

5 Clustering **9**

6 State of the art **9**

7 Toward a new framework **10**

8 Experimental results **13**

9 Conclusion and future works **20**

1 Motivations

Due to the growth of communication technologies and the diffusion of devices with good computation capabilities, the software is becoming more and more widespread. The main features of such software are distribution (the software is anywhere), communication (every device is connected to the others) and activity (the software should take care automagically of the most of the user tasks). This situation brings the new systems to a growing complexity, in a situation which is becoming more and more difficult to manage relying on classical software engineering meta-models, like Object Oriented Programming (OOP). In order to bring the engineering principles of locality and encapsulation at the desired level of abstraction, a new paradigm has been proposed, relying on the concept of *software agent*.

This paradigm shift has been proved to be really effective for problems integrating social and spatial aspects (like simulation and collective decision making support) [5] and it's very promising for all those problems which require an open and scalable software, often with self-organizing features; e.g. noisy phased array radars [25] the design of a control system for constellations of communication satellites, the control of large and distributed chemical plants, electronic commerce and financial softwares [20] and many others.

2 Main concepts

2.1 Agents and Multi Agents Systems

The key concept of *Agent* has been argument of discussion for a long time in the computer science community. In 1995 Woolridge and Jennings [26] proposed two ways to define an agent. While the strong notion (which includes mentalist characteristics typical of intentional systems [14]), is clearly addressed towards those researchers who mostly apply the agent concept to artificial intelligence, the weak notion defines an agent as a a software entity exhibiting four properties:

1. Autonomy: agents operate without the direct intervention of humans or others
2. Social ability: agents interact with other agents
3. Reactivity: agents perceive their environment
4. Pro-activeness: agents are able to exhibit a goal directed behaviour

In [27] this definition is took back and it is resumed in three main characteristics that an agent must have in pursuit of its design objectives:

1. Autonomy: an agent has its own thread of execution
2. Situatedness: an agent performs its action situated in a particular environment
3. Pro-activity: an agent need to act to ensure that its set of goals are achieved

Among all the concepts listed before, the main one is probably the autonomy, which is always present in every definition. Under a certain point of view, the other proprieties might be seen as consequences of this one. The relevance of autonomy as main concept is clear in [10]. A software component can be defined autonomous when it is implemented as a threaded application logically detached from the other components of the application, with a specific task or a specific goal [27] as, for instance, a UNIX-like software process [26].

2.2 Coordination models

An autonomous software component, in order to reach its goals, must be *situated* in an environment where to act. The notion of environment, absent in the OOP perspective [4], is often considered to be a primary abstraction [24] [27].

This abstraction brings the focus of the software engineering process from the structure dimension to the orthogonal dimension of the interactions among agents [5] as it's clearly possible to see in Gaia methodology [27].

A useful abstraction to describe the coordination systems is the coordination medium [12]. In this point of view, we distinguish between the coordinables, namely the entities which behaviour must be coordinated (in this case study the agents), and the coordination medium, which fills the interaction space managing the interactions among coordinables.

There are two main ways to let agents inter communicate, according to according to [18]: control driven models and data driven models. In the control driven perspective [19], the coordinables typically open themselves to the external world and interact with it through events occurring on well-defined input/output ports. The observable behaviour of the coordinables, from the point of view of the coordination media, is a state change or an event occurring on these ports. The coordination laws establish how events and state changes can occur and how they propagate. Therefore, the coordination media basically handle the topology of the interaction space, with no concern for the data exchanged between coordinables [13]. In data-driven coordination models, coordinables interact with the external world by exchanging data structures through the coordination media, which basically act as shared data spaces. The coordination laws determine how data structures are represented and how they are stored, accessed, and consumed. Unlike control-driven coordination models, the coordination medium has no perception of the changes in the state of coordinables, and does not provide any control over the communication topology [13].

In the environments in study characterized by openness and unpredictability, data driven models have some strong points compared to the control driven ones because of their ability of time-space decoupling. The main weak spot of these lasts is due to the intrinsic lack of predetermined information about events, which naturally brings to the adoption of models capable to easily decouple time and space. Agents may join and leave the system dynamically, they could change their behaviour depending on their specific current goal and environment status. The data driven model, having not to take care of this kind of interactions, downloads to the agents the responsibility to sense the situation of their world and act consequently.

2.3 Linda coordination system and tuple centres

Linda is an example of a strongly data driven coordination model [15]. It relies on the concept of *tuple*, which could be defined as an ordered list of elements, usually represented as:

tuple_name(tuple_argument_1, tuple_argument_2, ..., tuple_argument_N)

where each tuple argument could also be a tuple itself. One of the advantages of the tuple notion to represent information is the possibility to exploit the unification mechanism typical of the logic languages, which makes possible to represent a set of possible tuples using a template, usually representing a variable value starting with an uppercase character (in Prolog style).

In Linda, the coordination medium is represented by the *tuple space*, namely a place where agents can put, read and retrieve tuples using five basic primitives:

- *out(t)* - inserts tuple t in the tuple space;
- *in(T)* - retrieves a tuple matching the template T from the tuple space, waiting until it is available;
- *rd(T)* - reads a tuple matching the template T from the tuple space, waiting until it is available;
- *inp(T)* - retrieves a tuple matching the template T from the tuple space if already present;
- *rdp(T)* - reads a tuple matching the template T from the tuple space if already present;

Coordination models like LINDA were first conceived in the context of closed systems [6], but their use is now widespread as data driven coordination models.

The main limitation of Linda is the lack of computational capacity: as-is, a tuple space is little more than a tuple container. It would be interesting to melt both advantages of data-driven and control-driven models, extending tuple spaces with the ability of react to events. Working in this direction, Omicini and Denti introduced the concept of *tuple centre*. A tuple centre is defined as a tuple space whose behaviour can be defined by means of reactions to communication events [17].

2.4 Swarm Intelligence

Swarm intelligence describes the collective behaviour of decentralized, self organized systems, natural or artificial. The concept is employed in work on artificial intelligence [2].

The interest in swarm intelligence when discussing about coordination systems for MAS is due to the property to generate emerging self-organizing behaviours without any explicit programming and without any leader. The resulting behaviour of the whole system is due to the interactions among its components, so that "the whole is more than the sum of the parts".

A great feature of swarm intelligence systems involves the absence of leaders. This means there is no agent or component knowing the current state of the system, as a consequence there is no need to find out a way to store and retrieve

global information about the system. While in classical software systems the access to the global information is not a great problem, when we talk about distributed system this question assumes a huge dimension. Even if there are algorithms which allow to take screenshots of a distributed application [11], this strategy is in most cases inadequate, especially when openness and scalability become key features. For a quick example of the complexity explosion of this problem, try to think about taking a snapshot of the whole World Wide Web. The approach of swarm intelligence is radically different and relies on top of the concept of *locality*: every interaction that occurs involves neighboring agents or an agent and the part of the environment it is able to sense. In that way we can go over the problem of the knowledge of the global state just adopting techniques which do not require this information.

There are many examples of swarm intelligence phenomena in the natural world. Particularly interesting under this point of view are the social insects: fireflies show a way to synchronize their flashing, termites are able to build extremely complex liars and ants can find the shortest paths to bring food to the anthill and are able to precisely cluster every thing in the right anthill room.

2.5 Stigmergy

One of the most important ways to communicate between swarm agents is through the stigmergy. The concept of stigmergy was introduced by Pierre-Paul Grassé in the 1950s to describe the indirect communication taking place among individuals in a social insects colony [3]. Stigmergy is a mechanism of spontaneous, indirect coordination between agents or actions, where the trace left in the environment by an action stimulates the performance of a subsequent action, by the same or a different agent.

The key element of stigmergy is the environment modification: agents have no more to directly communicate among themselves, they can modify the environment, leaving a trace of their activity, so that other agent can sense and react to to modification. As a consequence stigmergy allows complex social behaviour decoupling interacting agents. It has surprisingly good properties in terms of flexibility and adaptability [22].

An example of stigmergic coordination in nature is given by the ants communicating together using pheromone: this chemicals are spread in the ground, signaling the others the presence of a kind of information.

A data-driven coordination model is the the perfect substrate where stigmergy can take place: the coordination medium naturally acts as environment and its content can be sensed and modified by agents. Moreover, in a hybrid model with tuple centres it can also have its own rules, acting in the same way a real environment acts.

3 Scale free networks

A scale-free network is a network whose degree distribution follows a power law, at least asymptotically. That is, the fraction $P(k)$ of nodes in the network having k connections to other nodes goes for large values of k as $P(k) \simeq k^{-\lambda}$ where λ is a constant whose value is typically in the range $2 < \lambda < 3$, although occasionally it may lie outside these bounds.

Systems as diverse as genetic networks or the world wide web are best described as networks with complex topology. A common property of many large networks is that the vertex connectivities follow a scale-free power-law distribution. This feature is found to be a consequence of the two generic mechanisms that networks expand continuously by the addition of new vertices, and new vertices attach preferentially to already well connected sites. A model based on these two ingredients reproduces the observed stationary scale-free distributions, indicating that the development of large networks is governed by robust self-organizing phenomena that go beyond the particulars of the individual systems [1].

In [1], Barabasi and Albert also present an algorithm whose effectiveness has already been tested in [9].

4 Entropy

Emergent self-organization in multi-agent systems appears to contradict the second law of thermodynamics. This paradox has been explained in [23] in terms of a coupling between the macro level that hosts self-organization (and an apparent reduction in entropy), and the micro level (where random processes greatly increase entropy). Metaphorically, the micro level serves as an entropy "sink", permitting overall system entropy to increase while sequestering this increase from the interactions where self organization is desired.

An good way to evaluate the goodness of a self organizing application is to find a way to measure the entropy at the desired level of abstraction. If this parameter is decreasing by time we can deduce our application effectively shows self organizing behaviour.

In order to exploit the entropy measure in experiments, we must have a definition which allows to calculate it. Information entropy is defined as [21]:

$$S = -\sum_i p_i \log p_i$$

where i ranges over the possible states of the system and p_i is the probability of find the system in state i. Being the possible states of a large, distributed application absolutely too many to enumerate, this kind of definition is not applicable.

In [9] is defined the concept of spatial entropy. Denoting by q_{ij} the amount of tuples matching template i within tuple space j, n_j the total number of tuples within tuple space j and s the number of templates, the entropy associated with tuple template i within tuple space j is:

$$H_{ij} = \frac{q_{ij}}{n_j} \log_2 \frac{n_j}{q_{ij}}$$

The entropy associated with a single tuple space is then:

$$H_j = \frac{\sum_{i=0}^{s} H_{ij}}{\log_2 s}$$

where $\log_2 s$ is introduced in order to normalize the values so that $0 \leq H_j \leq 1$. For r tuple space, the spatial entropy of the network became:

$$H = \frac{1}{r} \sum_{j=0}^{r} H_j$$

where the division by r is introduced in order to obtain $0 \leq H \leq 1$.

Note that this kind of measure do not takes in account the entropy of the pheromone system or of other components of the system than tuples, which are the level we are really interested in. In that way, forgetting about the entropy of the lower levels in the formula, we can evaluate just the entropy of the level we want to organize.

5 Clustering

To cluster means to group a number of similar things together. It's a basic form of organization: everybody tends to group together similar or related things. Nobody puts books and dishes together, for instance, but everybody keeps dishes and glasses in the same place. Supposing to have a function able to relate two objects semantically, an ordered situation is to have similar things stored in the same place.

Conveying this concept to the tuple based coordination systems world, we can obtain an ordered situation when tuples matching the same template are clustered in the same tuple space: this situation minimizes the spatial entropy described before. However, if all the related tuples are in the same tuple space and this one fails, the whole system may be compromised. That is the reason because over clustering is considered to be a problem.

In [7] a solution to avoid overclustering in SwarmLinda is proposed, introducing a parameter MaxSize which does not allow tuple spaces to store too much tuples. The same solution has been taken back in [8], where the main weak spot is exposed: the parameter MaxSize should be dynamic in order to deal with an open and unpredictable system.

6 State of the art

There are in literature some works in which the already exposed principles has been implemented. In [9] a mechanism relying on ants and pheromone has been proposed, suggesting to let the tuples move at the out time, binding them to an "out-ant". The main weak spots of this kind of approach is that once a tuple is dropped by the out-ant, it's no more able to move automatically according to the changes on the system. In [16] has been proposed a model of coordination relying on the top of TOTA middleware. The main lacks of this approaches are that TOTA does not take care of recognize connection and disconnection events and that the TOTA model does not rely on a simple data content inside the tuples, but injects inside them also the movement rules, mixing data driven and control driven coordination models into the same structure.

7 Toward a new framework

What we want to obtain is a new framework for agents coordination which can take over the problems of the previous explained techniques, adding also the ability to detect semantic relationships between tuples clustering them accordingly.

The virtual network building and managing is integrated in the framework and its main features are an high fault tolerance and the possibility to take care of connection and disconnection events. In order to build a network with these characteristics we used the BA model [1] in which nodes are added to the network according to a preferential attachment mechanism. Each node tries to connect to other two nodes using preferential attachment, namely the probability to connect to a node is higher if the degree of the node is high. The choice to link every new node with two others is merely an issue of fault tolerance. The network becomes more resilient to disconnections because a single node failure never breaks the network. Indeed in case of fault, when a node senses that the number of its connections is less than two, tries to recover the double connection.

The choice of the node to connect is made also evaluating other parameters: *degree of mobility*, *numbers of connection* and its *location*.

The chosen architecture is made of three layers. The lower level is called *Node Research Manager* (NoRMan)and manages the low-level aspects related to the real network structure, ensuring each node to have an up to date list of all the physically reachable nodes, each with its related informations that are useful to the upper layers such as position (in terms of longitude and latitude), degree of mobility and number of active virtual connections. In order to accomplish, NoRMan automatically scans all the IP addresses, starting from the nearer to the one in use. Once another node is found, each one exchanges the informations about all the known nodes, if there are new nodes or updated informations about a known node, the list is modified accordingly. It is also possible to manually contact a node specifying an IP address if it's already known. NoRMan has its own "keep alive", which ensures the known nodes are still online and up to date. NoRMan has support for geolocalization through a specific interface. An external agent stub is already ready in order to support implementation of GPS geolocalization. Currently, the agent only does a table lookup in a IP address to location database. If the lookup fails, it generates random position. Over NoRMan runs the *Topology Abstractor* (TopA) which is made of a server and a *Barabasi Network Builder* (BaNBu). Its target is to exploit the informations which NoRMan carries to realize and keep updated the virtual Barabasi Network. The third layer is the core system, which exploits the TuCSoN 1.9.1 framework in order to build distributed tuple spaces and to implement the following behavior.

The final result we want to get is to have a self-made, fault tolerant virtual Barabasi Network where each node represents a tuple space, and where the tuples which are tagged as "semantic" are able to move in the network without any user intervention. Tuples which are semantically related must cluster themselves avoiding overclustering [**?**]. In addition, the cluster must move following the users needs. More precisely, we expect that if someone is producing data and someone else is searching for semantically related data, such data will move toward the searcher as fast as its related. In that way, data self-organizes itself, being near to who really asks for data.

In order to get a scalable self-organizing system, we took cue from the biological systems. In particular, we decided to create a pheromone system which, exploiting the two base mechanisms of diffusion and evaporation, makes each node able to excite the neighbors and to have a sort of memory of the actions. The pheromone is emitted every time a tuple is searched, ensuring in that way the data to be attracted to the searchers as soon as they start. In order to avoid the creation of multiple pheromones with the same "meaning" which would have lead to performance issues, every time the system needs to emit pheromone the tuple is broken and a pheromone for every part is emitted. For example, the search for *a(b,c,d)* is supposed to be semantically the same of *a(d,b,c)*. With the adopted system, four pheromones are generated: *pheromone(a,I)*, *pheromone(b,J)*, *pheromone(c,J)*, *pheromone(d,J)* instead of *pheromone(a(b,c,d),K)*, *pheromone(a(d,b,c),L)*, where I, J, K, L are the amounts of pheromone which calculation will be explained later. If we try to apply the matching algorithm, with a tuple *f(g)*, we need to make eight semantic comparisons instead of sixteen. Each time a search is performed, a total amount of 100 pheromone is emitted. This amount is spread among the tuple name and all the arguments, supposing the number of arguments is n and the total amount is t, the tuple name has an amount of

$$2 * \frac{t}{n+2} \tag{1}$$

and each argument has an amount of

$$\frac{t}{n+2} \tag{2}$$

It's easy to demonstrate that the total amount is always t. Every five seconds, each node checks for the presence of pheromones. If there is any found, the spread and evaporation process starts. Supposing I is the total amount of pheromone of kind k, the new amount of pheromone is 90% of I. If the new amount is under a threshold (currently 1), the pheromone of kind k is deleted. The remaining 10% (S) is then spread to the neighbors. Supposing the node has j neighbors, each neighbor receive a quantity q of pheromone such as:

$$q = \frac{S}{j+1} \tag{3}$$

which means that every time a quantity q of pheromone does not spread and evaporates. This process is done for every pheromone kind. After that, the process sleeps for a time t which is, supposing w is the maximum wait time in milliseconds (currently 25000) and c is the total amount of pheromone in the system:

$$t = max(0, w - 1.0001^c) \tag{4}$$

This function has the quality of keep the wait time almost constant for low amounts of pheromone and to make the evaporation system very excited when the total amount of pheromone overtakes the threshold of 100000, avoiding the system to be overburden.

The pheromone system is meant to let the tuples cluster in the nodes that searches for related concepts. In parallel with this, in order to avoid overclustering and to cluster tuples even if nobody searches, we need a system able to

dynamically detect the maximum number of tuple allowed for each tuple centre and able to excite and move the tuples which are not related to the others present in the tuple centre.

First of all, we defined a measure of local entropy extending the formula of M. Casadei. Given a tuple centre with N tuples, denoting with $tuple_i$ the i-th tuple and with $match(a, b)$ the value of semantic match between the tuples a and b, the local entropy value is:

$$E_{local} = 1 - \frac{\sum_{i=0}^{N} \sum_{j=0}^{N} match(tuple_i, tuple_j)}{N} \tag{5}$$

It is possible to obtain a formula in order to calculate the entropy of the whole system simply using the mean. Supposing the system to be composed of K nodes and denoting with E_i the local entropy of the i-th tuple space:

$$E_{system} = \frac{\sum_{i=0}^{K} E_i}{K} \tag{6}$$

Each node exploits the local entropy value in order to evaluate how many tuples it's able to contain, evolving the concept of MaxSize with an eye to the system dynamics. The basic concept is that the more the tuple centre is ordered, the higher is the number of tuples it is disposed to contain. If a node has too many tuples, it became excited and tries to move them towards the neighbors. Naming max_size this number, it can be calculated as:

$$max\_size = \frac{C}{E_{local}} \tag{7}$$

where C is a constant which depends on the performances of the system, in the current framework version $C = 50$

If the number of tuples N inside the tuple space with local entropy E_{local} is lower than max_size, a tuple is chosen for moving every T milliseconds, where:

$$T = N^{1-E_{local}} * 1000 \tag{8}$$

Which means that the higher the entropy, the faster the tuples move and the higher the number of tuples, the slower they move. This mechanism allows the tuples to be clustered even if there are no nodes searching: the system is more quiet if it reaches a situation where there are many tuples which are related that is, substantially, a cluster.

The tuple which may move is chosen randomly. For each neighbor and for the node itself it's calculated an affinity value. Naming $pheromone_k$ the intensity of the pheromone of kind k, the affinity $A(T, N)$ between the tuple T and the node N is defined as:

$$A(T, N) = \sum_k pheromone_k^{match(k,tuple)*5} \tag{9}$$

The tuple chooses its destination with a probability proportional to the affinity value.

Node	k1	k2	k3	k1	k2	k3	k1	k2	k3	Search
2	25	25	25	42	16	22	70	49	44	Bird
11	25	25	25	52	13	14	67	8	5	Cat
14	25	25	25	22	15	11	39	8	0	Turtle
17	25	25	25	20	7	32	6	0	47	Avenue
20	25	25	25	15	11	43	7	11	52	Highway
69	25	25	25	11	15	53	6	5	63	Road
123	25	25	25	15	51	9	13	62	1	Pear
157	25	25	25	7	48	9	0	27	0	Apple
237	25	25	25	18	49	29	17	55	8	Orange

Table 1: Situation in terms of kind of tuple contained by each node after half an hour and at the end of the first run of the first experiment

8 Experimental results

Two experiments were made in order to verify the self organizing properties of the framework. Both the experiments uses the same set of tuples and the same initial configuration. As soon as starts, the framework builds a free scale network using the BA model explained above. Each node starts containing the same 75 tuples, five for each of the 15 types, split in 3 kinds:

- Kind k1:
  - *animal*, *dog*, *cat*, *bird*, *turtle*
- Kind k2:
  - *apple*, *banana*, *pear*, *orange*, *food*
- Kind k3:
  - *avenue*, *highway*, *street*, *plaza*, *road*

Both the experiments were run on the nine nodes.

In the first experiment each node starts searching for a specific kind of tuple, as described in the table. What we expect is to see the tuples move and reach the node which is searching for them, avoiding overclustering. Having nine nodes and fifteen types of tuple, we expect that the tuples which are not searched will cluster in the nodes that are searching for tuples of the same kind. We ran the experiment three times, recording for each node the local entropy and the tuple content. The number associated to each node is an unique identifier and does not concern anything important for the experiment. The experiments lasted about two hours. For the first run we made the table 1 and the graphs in image 1. As we can see, from a situation of disorder there are clusters clearly appearing after few minutes. Clusters in nodes 2, 17, 237 are however not well formed, we can assert it because the color of these nodes in the second graph in image 1 is not saturated. At the end of the experiment all the nodes but 2 have a very good cluster. Looking at the table 1, we can deduce from the high number of tuples of kind k2 and k3 that the node 2 is subject to an high traffic. More interesting is the image 2: we can see that the level of entropy of the whole system decreases with time, which is the main objective of the framework. Looking attentively

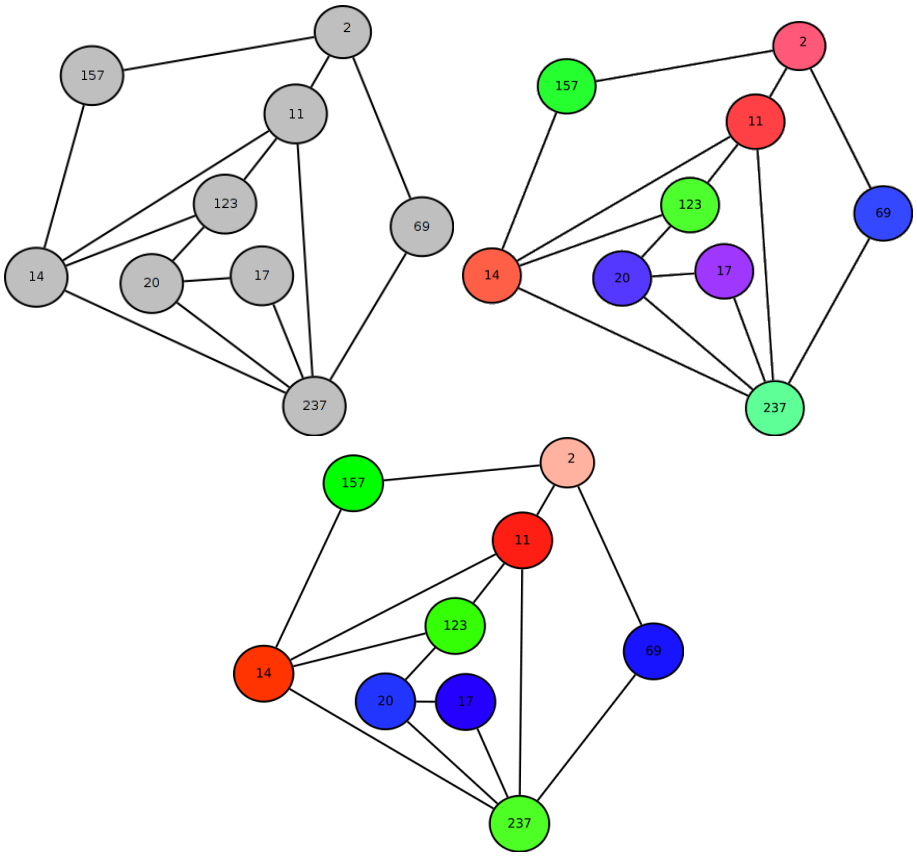

Figure 1: Cluster formation for the first run of the first experiment. The more the color is red/green/blue the more the tuples are k1/k2/k3 related

the first graph, which is related to the first run, we can notice that there are some nodes with a very low entropy level and others which have an entropy that remains quite high. These nodes are the most connected, they have to support the highest amount of traffic and as a consequence they often contain tuples which are not related between them. The second and the third graphs of the image 2 refers to other two runs. The behavior is always similar, even if not identical because of the unpredictability of the system, and shows an entropy decreasing with time for the whole system.

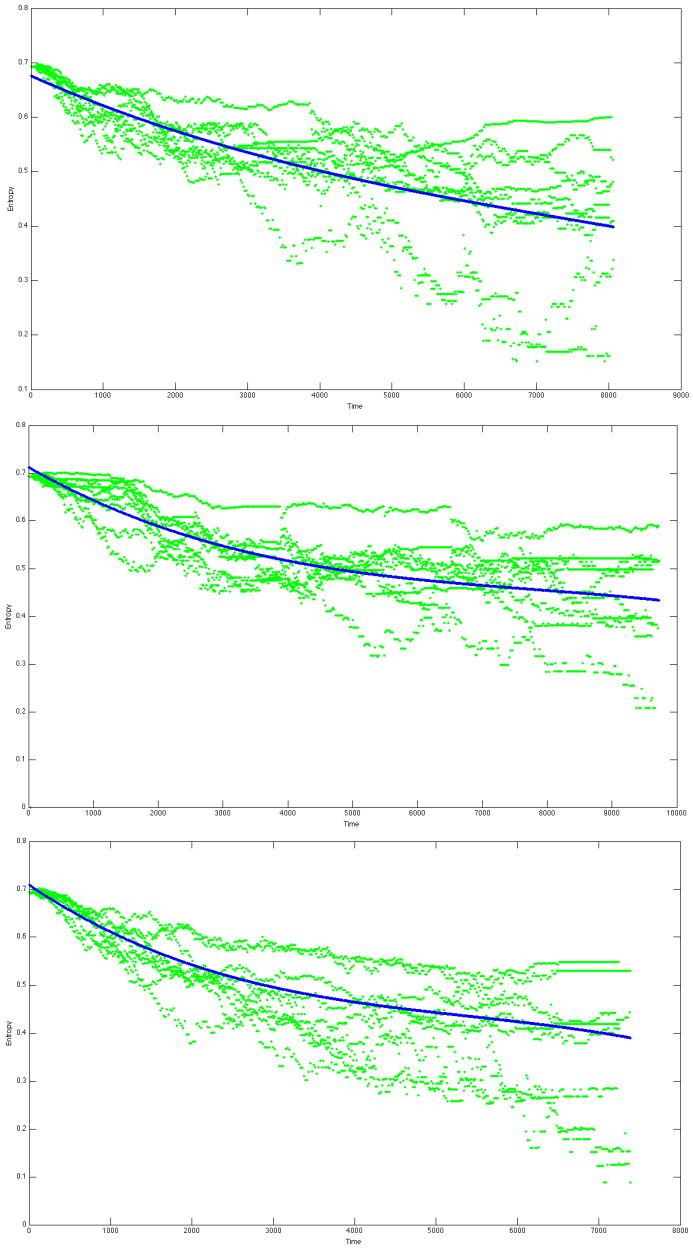

Figure 2: Entropy measurement in three runs of the first experiment. The green points are the local entropy of each node, the blue line is the entropy of the system.

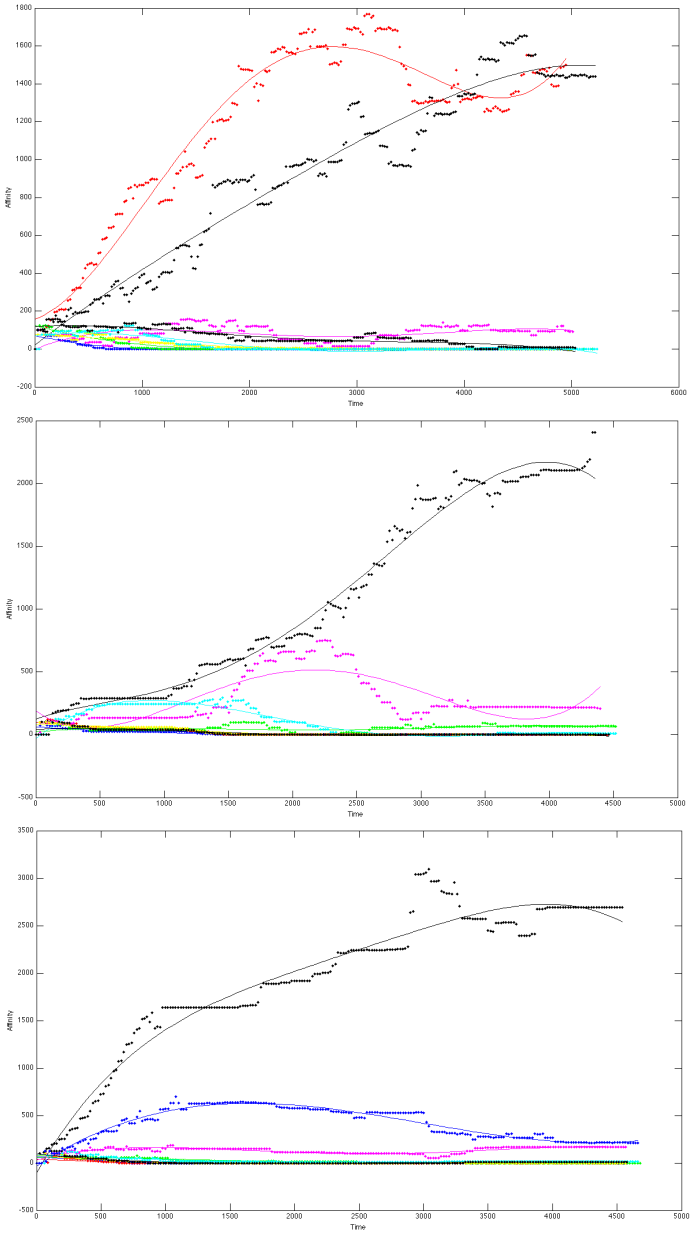

Figure 3: Affinity with tuple *dog* in three runs of the second experiment

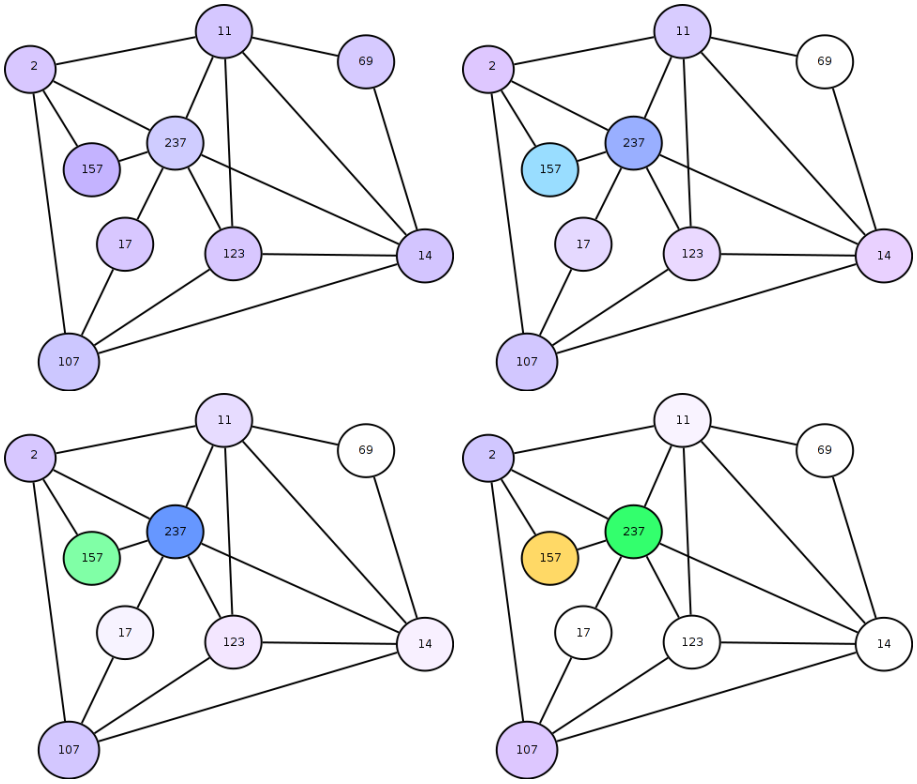

Figure 4: Affinity with tuple *dog* in the second run of the second experiment

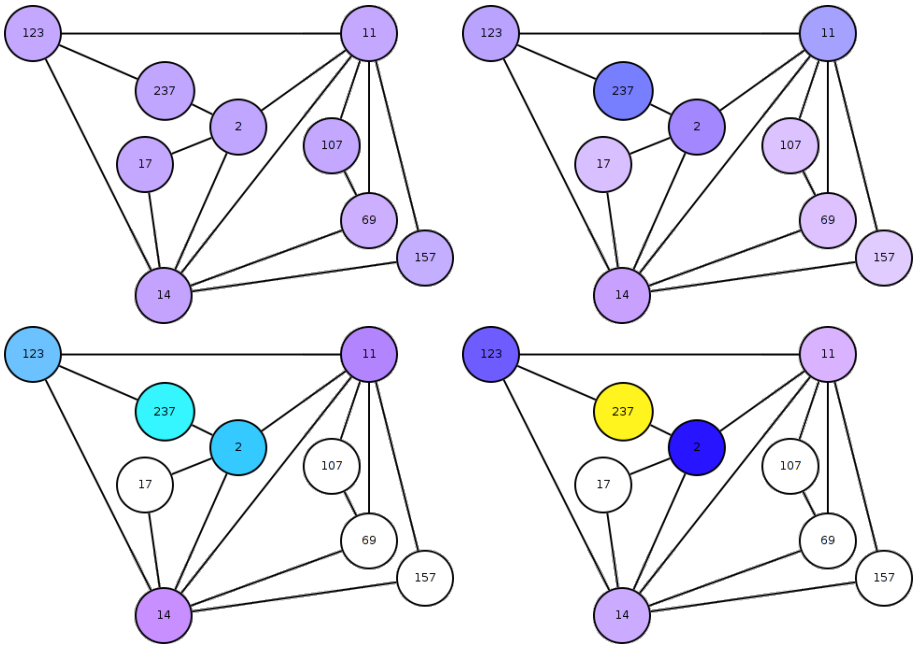

Figure 5: Affinity with tuple *dog* in the second run of the second experiment

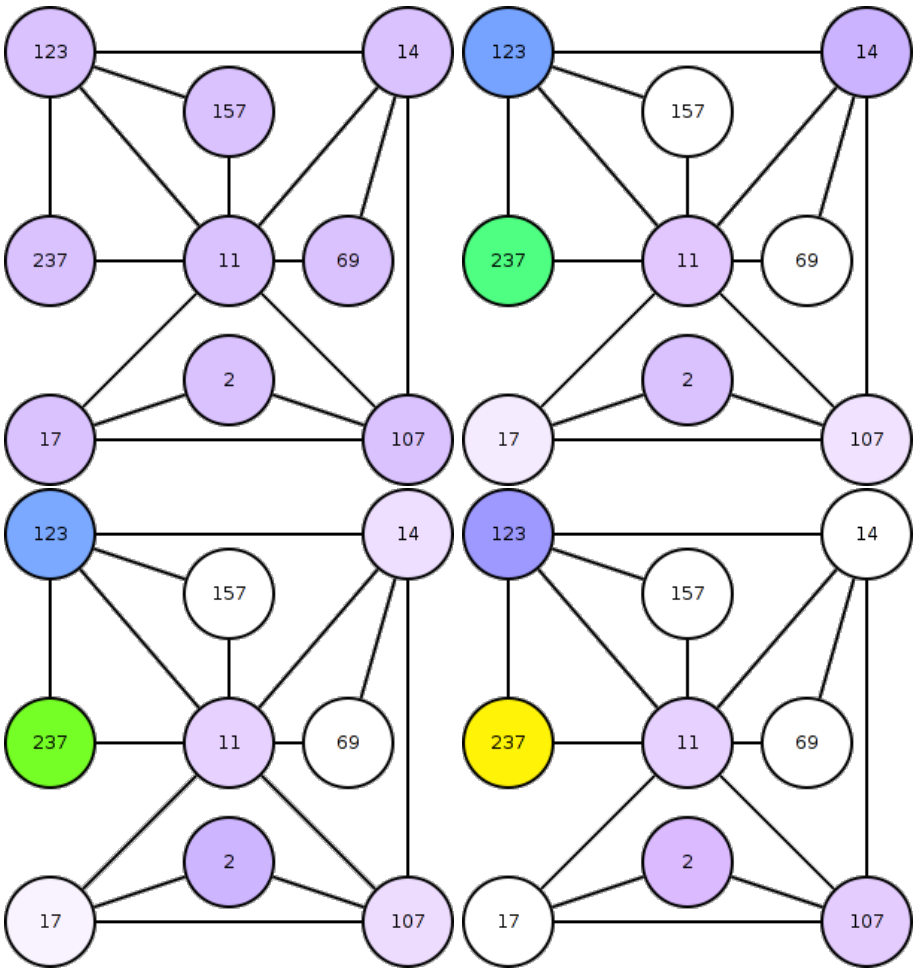

Figure 6: Affinity with tuple *dog* in the third run of the second experiment

In the second experiment, a single node searches continuously for the tuple *dog*. What we expect to see is that the tuples of kind dog and the related ones cluster on the node which searches. We also expect that the near nodes will contain tuples similar between them but not related to *dog*. This is a good test for the overclustering avoidance algorithm: if it doesn't work well, we'll see a single big cluster in the node which searches. In order to stress as much as possible the pheromone system, we decided to search from the less connected node (node 157 for the first run and node 237 for the second and third runs). In order to measure how much the cluster is related with the tuple searched we used the concept of affinity between a tuple and a tuple centre. In the figure 3 we can see the affinity value for each node. The node which searches is the red one in the first graph and the black one in the others two. The figures 4, 5, 6 show the clusters formation in four steps. The warmer is the color, the more the cluster is affine with *dog*, the more the color is saturated, the higher is the number of tuples that contains. A white node means there are no tuples, a purple or blue node means a cluster of tuple not related to *dog*, a yellow or green cluster means a cluster of tuples sightly related and a red cluster means a cluster of tuples very related to *dog*. As it's possible to see, in all the cases there are clusters forming, there are some nodes which are empty, one or two that contains tuples related with the one which is searched and other nodes where tuples unrelated with the one searched clusters. Analyzing the results we can assert which, while the second and third runs have a similar behavior, in the first test there are two clusters forming and not only one. The reason of this behavior is the proximity of the node 237, which is very connected, to the node 157 that searches. This situation means that the node 237 has an high amount of pheromone (because of the proximity) and an high traffic (because of the number of connections). These two factors causes the formation of a cluster which is quite related with the searched tuple. As it's possible to see from figure 4, however, the searching node maintains an higher affinity.

9 Conclusion and future works

Experimental results witness that the adopted strategies are able to bring the system toward self organization, semantically organizing tuples inside a network.

There are two interesting topics to explore in future. The first one is a sort of evaporation mechanism for tuples, such that an information which is never required could, by time, disappear. Another interesting point to explore could be the diffusion of the tuples: currently the framework just takes care to move the information (namely the tuples) toward the points where is required. I could be interesting to spread the information, diffusing the tuples instead of moving them.

References

[1] A. L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[2] Gerardo Beni. From swarm intelligence to swarm robotics. pages 1–9. 2005.

[3] Eric Bonabeau. Editor's introduction: Stigmergy. *Artificial Life*, 5(2):95–96, 1999.

[4] Grady Booch. *Object-Oriented Analysis and Design with Applications (3rd Edition)*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004.

[5] F. Bousquet and Le C. Page. Multi-agent simulations and ecosystem management: A review. *Ecological Modelling*, 2004.

[6] Nicholas Carriero and David Gelernter. How to write parallel programs: a guide to the perplexed. *ACM Comput. Surv.*, 21(3):323–357, 1989.

[7] Matteo Casadei, Ronaldo Menezes, Robert Tolksdorf, and Mirko Viroli. On the problem of over-clustering in tuple-based coordination systems. In Ozalp Babaoglu and Howard Shrobe, editors, *1st IEEE Conference on Self-Adaptive and Self-Organizing Systems (SASO 2007)*, pages 303–306, Boston, Massachusetts, USA, 9–11 2007. IEEE Computer Society.

[8] Matteo Casadei, Ronaldo Menezes, Mirko Viroli, and Robert Tolksdorf. Self-organized over-clustering avoidance in tuple-space systems. In *IEEE Congress on Evolutionary Computation, 2007 (CEC 2007).*, pages 1408–1415. IEEE Computer Society, 25–28 September 2007.

[9] Matteo Casadei, Ronaldo Menezes, Mirko Viroli, and Robert Tolksdorf. A self-organizing approach to tuple distribution in large-scale tuple-space systems. In Davis Hutchison and Randy Katz, editors, *Self-Organizing Systems*, volume 4725 of *LNCS*, pages 146–160. Springer, August 2007. 2nd International Workshop on Self-Organizing Systems (IWSOS 2007), The Lake District, UK, 11–13 September 2007. Proceedings.

[10] Cristiano Castelfranchi. Guarantees for autonomy in cognitive agent architecture. In *ECAI-94: Proceedings of the workshop on agent theories, architectures, and languages on Intelligent agents*, pages 56–70, New York, NY, USA, 1995. Springer-Verlag New York, Inc.

[11] K. Mani Chandy and Leslie Lamport. Distributed snapshots: determining global states of distributed systems. *ACM Trans. Comput. Syst.*, 3(1):63–75, February 1985.

[12] Paolo Ciancarini. Coordination models and languages as software integrators. *ACM Comput. Surv.*, 28(2):300–302, 1996.

[13] Paolo Ciancarini, Andrea Omicini, and Franco Zambonelli. Multiagent system engineering: the coordination viewpoint. In *Intelligent Agents VI — Agent Theories, Architectures, and Languages, volume 1767 of LNAI*, pages 250–259. Springer-Verlag, 2000.

[14] Daniel Dennett. Intentional systems theory. In Ansgar Beckermann and Sven Walter, editors, *Oxford Handbook of the Philosophy of Mind*, chapter 19. Oxford University Press, 2007.

[15] David Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems*, 7:80–112, 1985.

[16] Marco Mamei and Franco Zambonelli. Programming stigmergic coordination with the tota middleware. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 415–422, New York, NY, USA, 2005. ACM.

[17] Andrea Omicini and Enrico Denti. From tuple spaces to tuple centres. *Science of Computer Programming*, 41(3):277 – 294, 2001.

[18] George A. Papadopoulos and Farhad Arbab. Coordination models and languages. In *Advances in Computers*, pages 329–400. Academic Press, 1998.

[19] George A. Papadopoulos and Farhad Arbab. Configuration and dynamic reconfiguration of components using the coordination paradigm. *Future Generation Computer Systems*, 17(8):1023–1038, 2001.

[20] Omer F. Rana and Kate Stout. What is scalability in multi-agent systems? In *AGENTS '00: Proceedings of the fourth international conference on Autonomous agents*, pages 56–63, New York, NY, USA, 2000. ACM.

[21] CE Shannon and W Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, Urbana, Illinois, 1949.

[22] Paul Valckenaers, Martin Kollingbaum, Hendrik Van Brussel, and Olaf Bochmann. The design of multi-agent coordination and control systems using stigmergy. In *In Proceedings of the third International Workshop on Emergent Synthesis (IWES01*, 2001.

[23] H. Van Dyke Parunak and Sven Brueckner. Entropy and self-organization in multi-agent systems. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 124–130, New York, NY, USA, 2001. ACM.

[24] Danny Weyns, Andrea Omicini, and James Odell. Environment as a first class abstraction in multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):5–30, 2007.

[25] David H. Wolpert and Kagan Tumer. An introduction to collective intelligence. Technical report, NASA, 1999. NASA-ARC-IC-99-63.

[26] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10:115–152, 1995.

[27] Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge. Developing multiagent systems: The gaia methodology. *ACM Trans. Softw. Eng. Methodol.*, 12(3):317–370, July 2003.