Experiments in Neo-computation Based on Emergent Programming

Jean-Pierre Georgé, Marie-Pierre Gleizes, Pierre Glize

Experiments in Neo-computation Based on Emergent Programming

Jean-Pierre Georgé, Marie-Pierre Gleizes, and Pierre Glize

IRIT, Université Paul Sabatier, 118 route de Narbonne,
31062 Toulouse cedex, France
{george, gleizes, glize}@irit.fr

1 Emergent Programming

The general objective of this work is to develop a complete programming language in which each instruction is an autonomous agent trying to be in a cooperative state with the other agents of the system, as well as with the environment of the system. By endowing these *instruction-agents* with self-organizing mechanisms[2], we obtain a system able to continuously adapt to the task required by the programmer (i.e. to program and re-program itself depending on the needs). The work presented here aims at showing the feasibility of such a concept by specifying, and experimenting with, a core of *instruction-agents* needed for a subset of mathematical calculus. In its most abstract view, *Emergent Programming* is the *automated assembling of instructions of a programming language using mechanisms which are not explicitly informed of the program to be created*. We chose to rely on an adaptive multi-agent system using self-organizing mechanisms based on cooperation as it is described in the *AMAS* theory[1]. An important part of our work on *Emergent Programming* has been the exploration of the self-organization mechanisms which enable the agents to progress toward the adequate function, depending on the constraints of the environment but without knowing the organization to reach or how to do it.

2 The Elementary Example

The elementary example we choose is constituted of 6 agents: 3 "*constant*" agents, an "*addition*" agent, a "*multiplication*" agent and an "*output*" agent. A "*constant*" agent is able to provide the value which has been fixed at his creation (cf. Figure 1). The values produced by the system are results from organizations like $(A + B) * C$. *AgentOut* transmits the value he receives to the environment and is in charge of retrieving the feedback from the environment and forward it into the system. It is important to note that this information is not in any way an explicit description about the goal and *how* to reach it (it only informs that the value has to be higher or lower).

The size of the complete search space is 6^5, that is 7776 theoretically possible organizations, counting all the incomplete ones (i.e. where not every agent has all his partners). Among them, we have 6 types of different functional organization (they can actually calculate a value) (cf. Figure 1). The aim is to start without any partnerships between agents and to request that the system produces the highest value for example.

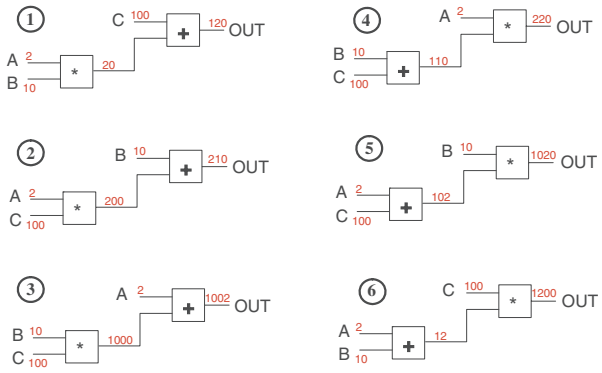

Fig. 1. The 6 different possible types of functional organizations for the elementary example

2.1 Reorganization Mechanisms

The agent's self-organizing capacity is induced by their capacity to detect *NCS* (*Non-Cooperative Situations*), react so as to resorb them and continuously act as cooperatively as possible. This last point implies in fact that the agent also has to try to resorb NCS of other agents if he is aware of them. We will illustrate this with the description of a simple NCS and how it is resorbed.

NCSNeedIn detection: the agent is missing a partner on one of his inputs. Since to be cooperative in the system he has to be useful, and to be useful he has to be able to compute his function, he has to find partners able to send values toward his input. Most NCS lead the agent to communicate so as to find a suitable (new) partner. These calls, because the agents have to take them into account, also take the shape of NCS.

NCSNeedIn resorption: this is one of the easiest NCS so resorb because the agent only has to find any agent for his missing input. The agent has simply to be able to contact some agent providing values corresponding to his own type (there could be agents handling values of different types in a system). So he generates an *NCSNeedInMessage* describing his situation and send it to his acquaintances.

NCSNeedInMessage detection: the agent receives a message informing him that another agent is in a *NCSNeedIn* situation (the sender is missing a partner on one of his inputs).

NCSNeedInMessage resorption: the agent is informed of the needs of the sender of the NCS and his cooperative attitude dictates him to act. First, he has to judge if he is relevant for the needs of the sender, and if it is the case, he has to propose himself as a potential partner. Second, even if he is not himself relevant, one of its acquaintances may be: he tries to counter this NCS by propagating the initial message to some acquaintances he thinks may be the most relevant.

It is important to note that the information which is given as a feedback is not in any way an explicit description about the goal and *how* to reach it. Indeed, this information does not exist: given a handful of values and mathematical operators, there is no explicit method to reach a specific value even for a human. They can only try and guess, and this is also what the agents do. That is why we believe the resolution we implemented to be in the frame of emergence.

3 Results

First, the internal constraints of the system are solved very quickly: in only a few reorganization moves (among the 7776 possible organizations), all the agents find their partners and a functional organization is reached. Then, because of the feedback from the environment, other NCS are produced and the system starts reorganizing toward its goal. Since the search space if of 7776 possible organizations, a blind exploration would need an average of 3888 checked organizations to reach a specific one. Since a functional organization possesses 4 identical instances for a given value (by input permutations), we would need 972 tries to get the right value. Experimentation shows that the system needs to explore less than a hundred organizations among the 7776 to reach one of the 4 producing the highest value. We consider that this self-organization strategy allows a relevant exploration of the search space. A noteworthy result is also that whatever organization receives the feedback for a better value, the next organization will indeed produce a better value.

4 Discussion

If we define all the agents needed to represent a complete programming language (with agents representing variables, allocation, control structures, ...) and if this language is extensive enough, we obtain maximal expressiveness: every program we can produce with current programming languages can be coded as an organization of *instruction-agents*. In its absolute concept, *Emergent programming* could then solve any problem, given that the problem can be solved by a computer system. Of course, this seems quite unrealistic, at least for the moment.

But if we possess some higher-level knowledges about a problem, or if the problem can be structured at a higher level than the instruction level, then it is more efficient and easier to conceive the system at a higher level. This is the case for example when we can identify entities of bigger granularity which therefore have richer competences and behaviors, maybe adapted specifically for the problem. Consequently, we will certainly be able to apply the self-organizing mechanisms developed for Emergent Programming to other ways to tackle a problem. Indeed, *instruction-agents* are very particular by the fact that they represent the most generic type of entities. The exploration of the search space, for entities possessing more information or more competences for a given problem can only be easier. For example, we think that problems like Ambient Intelligence or Autonomic Computing are ideal candidate for a problem solving by emergence approach.

References

1. M.-P. Gleizes, V. Camps, and P. Glize. A theory of emergent computation based on cooperative self-oganization for adaptive artificial systems. In *Fourth European Congress of Systems Science*, Valencia, Spain, 1999.
2. F. Heylighen. *Encyclopedia of Life Support Systems*, chapter The Science of Self-organization and Adaptivity. EOLSS Publishers Co. Ltd, 2001.