Evolving Organizations of Agents

Claudia V. Goldman Jeffrey S. Rosenschein
Institute of Computer Science
The Hebrew University
Givat Ram, Jerusalem, Israel
clag@cs.huji.ac.il, jeff@cs.huji.ac.il

Abstract

We investigate how agents can learn to become experts, and eventually organize themselves appropriately for a range of tasks. Our aim is to look at evolutionary processes that lead to organizations of experts.

The distributed artificial intelligence (DAI) community has dealt with multiagent systems that organize themselves in order to achieve a specific shared goal. Various organizations can arise that will effectively balance the load on the agents and improve their performance. We here look at the process of emergence of an organization as a step that takes place prior to the execution of a task, and as a general process related to a range of problems in a domain.

To explore the ideas set forward, we designed and implemented a testbed based on the idea of the game of *Life*. We present experimental results that show different patterns of organizations that might evolve in a multiagent system.

Keywords: organization of agents, evolutionary model, software agents

Introduction

The topic of organization and self-organization of agents has been dealt with in the Distributed Artificial Intelligence (DAI) community with regard to the division of labor among agents who are trying to achieve some shared goal. We are interested in how agents can learn to organize themselves and in particular, how the agents in a multiagent environment can learn to become experts. We suggest an algorithm that influences the agent population in the system, and the system evolves expert agents in a domain. The proposed algorithm is performed as a step prior to problem solving in the domain. After the agents have become experts, other applications or the agents themselves can address these experts to get assistance or tasks resolved.

We have designed and implemented a testbed to experiment with evolving organizations of agents. The implementation of this testbed is along the lines of the Game of *Life* (Gardner 1983). This game was invented by John Horton Conway in 1970. It takes place in a two dimensional grid (assumed to be an infinite plane). Each cell in this board has eight neighboring cells. The game was designed as a one person game, where an initial configuration of the grid is submitted (i.e., specifying which cells are empty and which cells are occupied by a counter). From then on, the rules of the game will control and change the configurations of the grid, leading to interesting patterns of organization. There are three rules that define when an occupied cell remains occupied (i.e., it survives), when an occupied cell becomes empty (i.e., it dies), and when an empty cell becomes occupied (i.e., a birth occurs). Every organism with two or three neighboring organisms survives for the next generation. Each organism with four or more neighbors dies (i.e., it is removed) due to overpopulation. Every organism with one neighbor or none dies from isolation. Each empty cell adjacent to exactly three neighbors is a birth cell. An organism is placed on it at the next move. There were three known behaviors that could emerge in different configurations with these rules. There were stable populations, that did not change their structure once they got to it. There were configurations that faded away, i.e., the grid remained totally empty after a number of iterations of the game. And, a configuration could be periodic or oscillating (i.e., it is composed of subconfigurations that change among them in a definite cycle [larger or equal to one]). These examples stress the unpredictability of the results in *Life* even though the rules of the game are very simple, and seem predictable.

In this paper, we present experimental results, that show interesting patterns of organization of agents that might evolve, with similar characteristics to patterns evolved in the game of *Life*.

The Organization of Agents

The organization and self-organization of agents operating in multiagent systems has been studied in the

DAI community, in relation to the agents' tasks. A *good* organization will balance the load on the agents, and will cause the agents to improve their performance. Most work published on this topic considers the organization of the agents as a means that enables the agents to better adapt to their environment.

Ishida (Ishida 1992) proposes a model in which agents organize themselves by decomposing the problem they have to solve to achieve a shared goal. He calls this approach *organization centered problem solving.* Each agent always evaluates its actions based on how each action contributes to the agent's goal, and on how each action contributes to the other agent's goal. The process by which agents reorganize themselves is by changing their current goals, i.e., refining their knowledge, or partitioning a goal into subgoals.

Another model for self organization was presented by Guichard et al. (Guichard & Ayel 1994). They introduced two primitives of reorganization: the composition (i.e., regrouping several agents into one), and the decomposition (i.e., the creation of several new agents). An agent decomposes itself when it reasons by applying its mechanism of introspection and it *concludes* that it is unable to perform its tasks adequately. They didn't explain formally any specific mechanism, but they mention several examples of it based on the number of tasks, the importance of tasks, or the required time for resolving the problem.

Huberman and Hogg (Huberman & Hogg 1995) look into communities of practice, i.e., *informal networks that generate their own norms and interaction patterns.* They consider a group of individuals that are trying to solve a problem. Individuals can interact with one another, and they might do so with an interaction strength proportional to the frequency with which they exchange information with each other. At each step individuals can choose to work on their own or use information or other help from others in the community. They suggest characterizing the structure of a community by counting the number of neighbors an individual has, weighted by how frequently they interact. They then study the general evolution of such networks and find the conditions that allow the community to achieve the optimal structure purely through adjustments. This requires each agent to learn about the hints it might get from other agents and to learn to use them effectively.

Nagendra Prasad et al. (Nagendra Prasad, Lesser, & Lander 1996) investigated the usefulness of having heterogeneous agents learning their organizational role in a multiagent parametric design system. The problem the agents are faced with is to find a design, i.e., to find a set of values for the set of parameters of the problem. The agents can initiate designs, extend or critique them until they get to a final design that is mutually accepted by all of them. In the learning study they considered only the first two roles. Each of these roles is defined as a set of tasks that the agent has to perform in a composite solution that solves a given problem. In (Nagendra Prasad, Lesser, & Lander 1996), it is noted that the designer of the system cannot know beforehand the best assignments of roles to the agents because he lacks knowledge about the solution distribution in the space.

The agents went through a training period in which they evaluate their possible roles at each state while searching for a solution to specific problems. Then, the agents would select the role that maximized their evaluation. This involves communicating with the other agents in the system to obtain information required to calculate the values of the evaluation function. In addition the actual performance measure of each agent's choice is only known when the search for the solution is completed. Then, the agents backtrace their evaluations and assigned them to the roles. The agents are trained for all possible states of a search.

In all the cases reviewed in this section, the reorganization of a group of agents is strictly related to a specific problem the agents are trying to solve. The study presented in (Nagendra Prasad, Lesser, & Lander 1996) is more general, in the sense that a still predefined number of agents is trained for all the possible states that could occur while searching for a solution for a multiparametric design.

The changes produced in these organizational structures are induced by a decrease in performance (i.e., they will reorganize because their performance has decreased, in order to improve it). In summary, the agents search for an internal organization while achieving a shared goal, balancing the load imposed on them as a group, and improving their performance.

The Multiagent Testbed

We have designed and implemented a testbed to experiment with evolving organizations of agents. The rules of our testbed were designed along the lines of the game of *Life.* The main addition to the game is the consideration of the environment in which the agents grow, die, or live. In our current implementation we choose the information domain, consisting of documents from a given source (e.g., a site on the Internet, files from a directory, from an archive, etc.). Therefore, the rules of death, birth, and survival were defined based on the resources (i.e., the information in the documents) that the agents hold. In this testbed, we deal with homogeneous agents coming from a general class of agents.

There are two main concepts that need to be defined in the game of *Life*: the neighborhood relation and the rules of the game. In our case, we compute the nearness among the documents in the set on which the simulation is run. Each agent can access a matrix consisting of the proximity values calculated for any pair of documents found in the collection.

For our implementation, we download all the documents at the corresponding site. We then build an inverted index of all the words that appear in these documents. We store, for each word, the total number of times the word appeared in all the documents together, the total number of documents in which the word appeared, and for each document that the word appeared in, the total number of appearances in that document. Afterwards, we compute the proximity values among all the documents collected, and build the neighborhood matrix out of these values. Currently, we have implemented two relations that decide when two documents are considered neighbors. In both cases, the neighborhood relation implies a gradual relation (in contrast to a binary relation, in which we could only say whether two documents are neighbors or not).

One neighborhood relation considers the words that appear in the documents, that is, two documents are neighbors when they are *semantically similar.* The other relation considers the links each document points to. That means that two documents will be considered neighbors if both of them point to mutual links. The matrix is a square matrix with rows and columns equivalent to the number of documents collected. Each cell is initialized with zero. For each word that appears in the inverted index, and documents Doc_i, and Doc_j in which the word appears, the value of the cell $[i, j]$ is incremented by an amount $Incr[word, Doc_i, Doc_j]$ based on the Inverse Document Frequency (IDF) weight formula (Salton & McGill 1983). The IDF formula assumes that the word's importance is proportional to the occurrence frequency of each word in each document and inversely proportional to the total number of documents in which the same word appears.

The evolutionary engine program simulates a system populated with agents that gather resources (e.g., documents). The main algorithm for evolving an organization of experts is presented in Fig. 1. $C[i]$ denotes the current set of documents owned by agent i. AgentsOnDocs is the number of agents already holding documents that are neighbors of the documents that agent i is holding. Free is the set of documents that are not owned yet by any agent in the current population.

Any simulation starts with an initial number of agents set by the user. The user also assigns an initial document to each agent. Then, the agents collect more documents that are close enough to their seed document, based on the neighborhood matrix. The number of additional documents is set by the user. From then on, the agents will be involved in a loop in which they might be able to collect more documents from the given collection, they might die when they have very few documents in relation to the other agents, they might split into two agents if they have too many documents in relation to the other agents, and they might do nothing. This main loop ends after an iteration in which all the agents do nothing, i.e., no changes were made to the sets of documents the agents hold so far. During the simulation, each agent is described by a personal window in which the current addresses of the documents it was assigned are presented. At the end of the simulation, all the sets of documents held by each agent, together with the words that characterize them, are copied to HTML files. A set of parameters can be set by the user to test different scenarios, initial conditions of the system, and different thresholds needed by the rules that will influence the behavior of the agents.

```
For each agent $A_i$ in the agents list:
  Find NumDocs documents that are the closest
  neighbors to the documents already in $C[i]$
  Find out the size of AgentsOnDocs
  Compute PopulationDensity = $\frac{AgentsOnDocs}{NumDocs}$
  If ($MyDocs_i$ < MinDocs) and
      (PopulationDensity > HighPopDensity)
    $A_i$ returns its documents in $C[i]$ to Free
    $A_i$ dies
  If ($MyDocs_i \geq$ MaxDocs) and
      (PopulationDensity < LowPopDensity)
    $A_i$ spawns a new agent $A'_i$
    $A_i$ divides $C[i]$ into $C[i']$ and $C[i'']$
  If ($MyDocs_i$ < MaxDocs) and
      (PopulationDensity < HighPopDensity)
    $A_i$ adds to its current $C[i]$,
    the NumAdded closest documents from $NumDocs$
    that are not owned by any other agent.
    Free is updated accordingly
  else do nothing
  Output $A_i$'s current documents in $C[i]$
```

Figure 1: The main loop of the evolutionary engine

Experimental Results

In the game of *Life* four main behaviors were reported and described (Gardner 1983): stable, periodic or oscillating, fading, and ever growing patterns of organization. We will show similar results in our testbed, considering also the domain we choose. We built a matrix of the neighborhood relation among 166 documents taken from the Computer Science site at the Hebrew University (the URL of the root of the site is http://www.cs.huji.ac.il).

The parameters that can be set by the user include NumDocs (i.e., the number of neighbor documents the agent will collect at most at each iteration), NumAdded (i.e., the number of the nearest documents the agent might add to its current cluster [chosen from NumDocs]), HighPopDensity (it determines when the population is overcrowded), MaxDocs (i.e., the maximum number of documents that each agent might hold in its cluster), MinDocs (i.e., the minimum number of documents that an agent can have).

Stable Patterns The stop condition of our algorithm is a state in which every agent has nothing to do. At this point, the agents got to a stable state, i.e., we found a stable pattern of organization. Each agent holds a defined cluster of documents.

The system stabilized in a pattern of two agents for the following parameters: the number of documents added at each iteration was three, there were two agents in the initial population, the threshold for high population was 0.2, the maximal number of documents was 16 and the minimum number of documents was set as 10. The same result was gotten for a different initial number of documents that each agent held (i.e., for 2, 5, and 8 initial documents).

When the minimal number of documents was changed to 5, a stable pattern of 3 agents was gotten for an initial number of documents ranging from 1 to 8. These three agents in the system were responsible for three topics respectively: material about writing in HTML, pages about a specific project developed in the Distributed Systems Laboratory, and pages about the learning group at the Hebrew University.

When the threshold for the high population was increased to 0.6, we got two different sets of stable patterns. One was reached with 8 agents when the initial number of documents was 1, 2, 4, 5, 7. A pattern of ten agents was reached for 3 or 6 initial documents. The eight agents were responsible for pages on HTML, the programming laboratory, the Data Base group and information for students, the Distributed Systems group, three agents for three different exercises material in a course about learning, material about the learning group and additional material about the Distributed Systems group, respectively.

When the threshold for high population density remained at 0.6 and the minimal number of documents was raised to 10, we got three different sets of stable patterns: seven agents composed the final structure starting from 1, 4, or 7 initial number of documents, five agents were the result starting from 2 or 5 initial documents, and six agents resulted when the initial two agents started with 3 or 6 initial number of documents.

In Figures 2,3,4,5, the number of agents in the organization is presented as a function of the initial number of seed documents given to each of the two agents in the initial population. We denote a loop between x agents by a value of -x. For example, in Fig. 2, when there were three documents given to each agent, the organization arrived at a group of one agent and a loop between two other agents. In Figures 6, 7, we present the number of agents in the organization that evolved from an initial population of two agents that were given five initial documents.

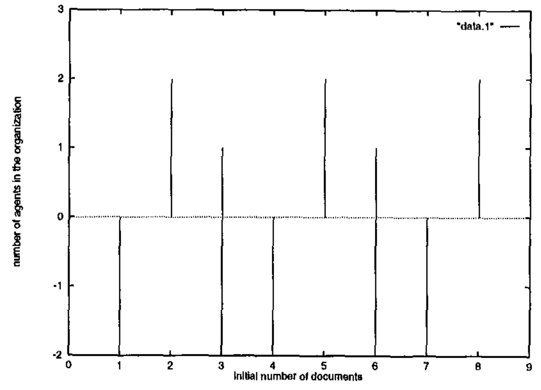

Figure 2: HighPopDensity=0.2, MaxDocs=16, MinDocs=10, 2 agents in the initial population

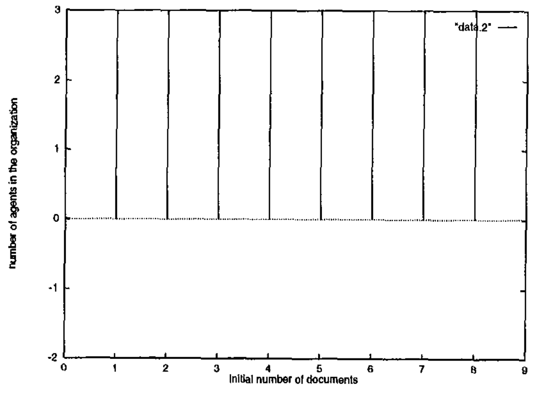

Figure 3: HighPopDensity=0.2, MaxDocs=16, MinDocs=5, 2 agents in the initial population

Ever Growing Patterns To achieve a population of agents with an unlimited size, we would need an infinite source of information. For example, for the domain we have chosen, we would need a site or a group of sites that grow all the time.

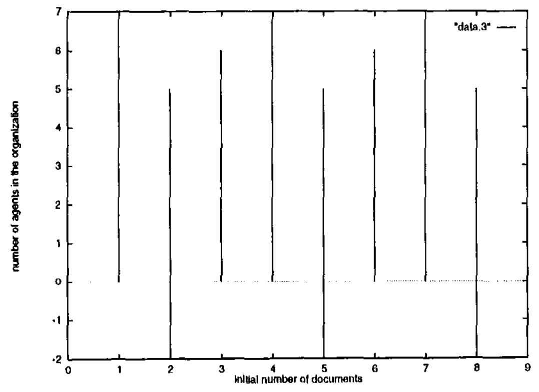

Figure 4: HighPopDensity=0.6, MaxDocs=16, MinDocs=10, 2 agents in the initial population

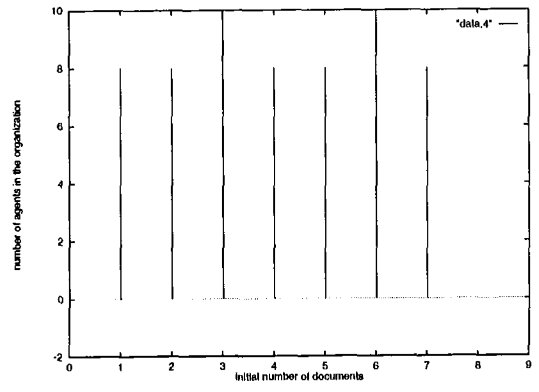

Figure 5: HighPopDensity=0.6, MaxDocs=16, MinDocs=5, 2 agents in the initial population

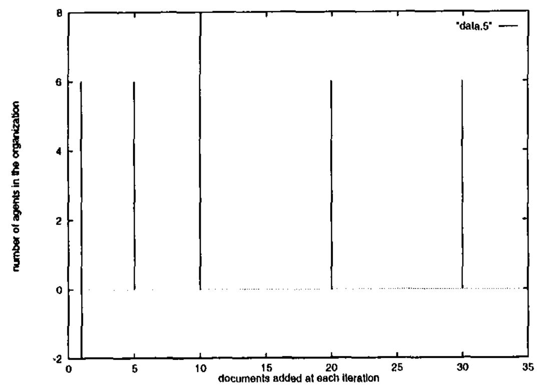

Figure 6: HighPopDensity=0.6, MaxDocs=16, MinDocs=10, 2 agents in the initial population, five initial documents for each agent

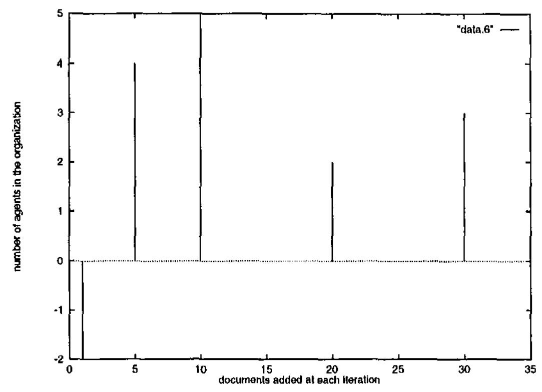

Figure 7: HighPopDensity=0.2, MaxDocs=16, MinDocs=10, 2 agents in the initial population, five initial documents for each agent

In our current simulation, the site of documents is finite. There is a possibility in which, for example, one agent splits into two agents, one of them dies, and the remaining agent repeats this process of splitting into one agent that dies, and the other agent splits again and so forth, without stopping. We will present this example as a *blinker* pattern. In the specific example, the agent that splits itself (i.e., the father) and the agent that remains alive and will reproduce exchange the clusters of documents they hold. In our implementation, each time an agent reproduces, its two sons get two new names, but we can think of an agent spawning itself into another agent, and the same agent remains alive with part of its original documents. Therefore in our current implementation and given the domain as a finite site of documents, we cannot reach an ever growing pattern of organization.

Fading Away Patterns We will consider two cases. In one, we will check whether the whole population of agents can vanish and we will also show cases where agents can die in a system that survives.

For the following parameters: threshold for the high population set as 0.2, maximal number of documents as 16, minimal number of documents as 10, three documents added at each iteration, two agents in the initial population with 39 or 40 initial documents, the agents vanished. The same result was achieved for high population threshold set to 0.6 and minimal number of documents set to five, and also for five as the minimal number of documents and the threshold set to 0.2.

We also got results that show that agents can die, when the appropriate condition is fulfilled.

Oscillating Patterns (Blinkers) Oscillating patterns in multiagent systems are those patterns consisting of agents that pass their documents from one agent to another continuously.

For example, when the number of documents added per iteration was three, the threshold for high population was 0.2, the maximal number of documents was 16 and the minimal number of documents was 10, there were two agents in the initial population, we got a blinker pattern for different initial numbers of documents (1,3,4,6,7). The system entered a loop of one agent holding documents related to the learning group and another agent holding documents related to a project in the Distributed Systems group. Every three iterations of the simulation, the agent that survived held one of these two clusters, and after three more iterations, the agent that remained alive held the other cluster.

Another case was when the threshold for the high population was 0.6, the minimal number of documents was 10, and the initial number of documents was eight. The system evolved into five agents (responsible for the Data Base group and information for the students, an exercise in a course about learning, the programming laboratory course, the learning group, the Distributed Systems group), and a loop between two other agents (one cluster included material about a specific exercise in the learning course, and the other about tools for the learning course).

Other behaviors analyzed in *Life* (Gardner 1983) concern distinct kinds of collisions among different patterns. This is also relevant in our scenario. There occurs a collision between clusters of documents (i.e., between the agents responsible for them) when one agent (or several agents) dies, and its documents are redistributed between the other agents (i.e., other agents take control over these documents like a *Life* pattern that eats another pattern, or when we get to oscillating behavior (i.e., agents pass their documents between themselves in a cycle).

Conclusions

We propose an algorithm that evolves different patterns of organization in a multiagent system considering the environment of the agents. The algorithm affects the population of agents, i.e., their growth, death and survival. Each agent collects information from the domain, that eventually will become its domain of expertise.

We show different interesting patterns of organization that agents can evolve (i.e., stable, periodic and vanishing behaviors). In our model, the agents get their expertise before they deal with any goal or task. We are solely interested in agents that learn to become experts, and not with the load balancing or solving of a specific problem.

These patterns of organization that evolve in a prior step, are useful as a source of information. Specific agents can be addressed to answer queries on the knowledge they have. For example, the organization can be approached by a user or another agent application to retrieve information related to a given query. Another example is to regard this organization as a library of reusable plans (Decker & Lesser 1994). Moreover, these organizations serve as a preparing stage before the agents divide the tasks among themselves. Each agent can achieve goals that are more related to its expertise in a more efficient way.

Acknowledgments

This work has been supported in part by the Israeli Ministry of Science and Technology (Grant 032-8284) and the Israel Science Foundation (Grant 032-7517).

Claudia V. Goldman is supported by the Eshkol Fellowship, Israeli Ministry of Science.

We'd like to thank Ofer Margoninski for implementing the system for analyzing the structure of sites on the Internet.

References

Decker, K., and Lesser, V. 1994. Designing a family of coordination algorithms. Technical Report 94-14, University of Massachusetts, Amherst.

Gardner, M. 1983. *Wheels, Life and other mathematical amusements.* W.H. Freeman and Company.

Guichard, F., and Ayel, J. 1994. Logical reorganization of dai systems. *ECAI94 Workshop on Agents, Theories, Architectures and Languages* LNAI.

Huberman, B. A., and Hogg, T. 1995. Communities of practice: Performance and evolution. *Computational and Mathematical Organization Theory* 1:73–92.

Ishida, T. 1992. The tower of babel: Towards organization centered problem solving. In *Proceedings of the Eleventh International Workshop on Distributed Artificial Intelligence.*

Nagendra Prasad, M.; Lesser, V.; and Lander, S. 1996. Learning organizational roles in a heterogeneous multi-agent system. In *Proceedings of the Second International Conference on Multiagent Systems.* Kyoto, Japan: AAAI Press.

Salton, G., and McGill, M. 1983. *Introduction to Modern Information Retrieval.* McGraw Hill International Books.