Evolutionary Patterns of Agent Organizations

Claudia V. Goldman and Jeffrey S. Rosenschein

***Abstract*—Problems approached by multiagent systems (MAS) are typically complex. It is usually difficult to know at system design stage how many agents need to be in the system, what each agent's role is, and how the agents should interact to get optimal performance out of the group. The aim of the testbed presented here is to investigate which kinds of multiagent systems could be developed to solve ranges of problems, avoiding the need to reorganize the agents from scratch for each task.**

The agent organization process explored here is based on the agents' knowledge, and not on their tasks. This opens up a new approach for Distributed Artificial Intelligence designers, to have their domain organized before the allocation of tasks among agents. These kinds of organizations should be more robust for solving different problems related to the same knowledge. We define information oriented domains (IODs) for that purpose.

An evolutionary approach to the design of a multiagent system is suggested. Our model is based on a cellular automaton whose rules of dynamics induce the formation of an organization of agents. Patterns of organization obtained empirically are presented. Our knowledge-based organization approach is analyzed both from theoretical and practical perspectives.

***Index Terms*—Organization, expert agents, information domain.**

I. Introduction

THERE are several approaches that one can take to designing multiagent organizations. One approach is to focus on the division of tasks and subtasks among agents, in order to balance the load on each of the agents, avoid redundant work, and achieve globally accurate solutions. Much research in distributed artificial intelligence (DAI) has been focused on these issues.

We are interested in a variation on this question of multiagent organization, which focuses on first organizing the knowledge that exists in the agent domain. In principle, this would enable the same agent organization to solve a range of different problems in a given information space, and could also serve as a preliminary step to the distribution of tasks among agents. Sociability then evolves from the structure of the information domain, and from the different patterns of agents' organizations.

Our approach is that in certain domains, knowing the *structure of information content* could guide us in the design of an appropriate agent system. Multiagent systems often operate in environments where access to, and manipulation of, information is paramount. We formally characterize these domains as information oriented domains.

In these domains, the information that the agents hold or are pursuing characterize their interaction, rather than the goals the agents are expected to achieve. Agents in these domains, in contrast to state oriented domains [1], do not have to deal with conflict. Information can be shared. Agents, for example, might approach the same server in parallel without preventing other agents from working on the same knowledge.

Our first step was to partition the information domain, using an evolutionary algorithm and a similarity criterion. This partitioning step resulted in a group of agents, each with a particular area of "expertise," assigned to handle the available information. Because of the nature of the evolutionary algorithm, the number of final agents and their assignments arose naturally from the information content of the environment, and were not determined *a priori*. This organization of agents, spanning the information space, could then be exploited in problem solving and information retrieval tasks. The partitioning of the information space lead to greater efficiency at run time.

In general, these partitioned systems are appropriate for information retrieval systems, or (more specifically) as an organizational structure for libraries of reusable plans [2] and as a means to organize different recipes [3] for plans in a given domain.

Possible emergent patterns within an organization of agents are demonstrated, and the evolutionary algorithm itself is discussed.

II. Information Oriented Domains and NetLife

We provide a general definition of information oriented domains, and consider the queries that might be submitted in such domains. In this paper, we focus on the agent organization that can evolve in such scenarios. In a separate paper, we studied how this organization can be approached by users and how it can solve queries [4].

An *information oriented domain* (IOD) is a tuple $\langle \mathrm{IS}, S, \mathcal{A}, \{\mathrm{op}\}, c, v, w \rangle$, where

1) IS is the information set, e.g., a set of documents;
2) S is the set of possible states of the world. In an IOD, we define a state at time t as a pair $[Q, \mathrm{PA}_t]$, where Q is the query a user submits, and PA_t is the partial answer derived at time t. In this case, a partial answer is measured by the worth of the state, which represents to what degree the information derived is useful and related to Q. The state is the agents' state of information relative to a query Q. PA_t is a vector whose components represent how much each agent knows at time t, relative to Q. There are local states that represent the information a

Manuscript received November 10, 1999; revised revised December 3, 2001. This work was supported in part by the Israeli Ministry of Science and Technology (Grant 032-8284) and the Israel Science Foundation (Grant 032-7517). This work was performed as part of the first author's Ph.D. work at the Hebrew University and was supported by the Eshkol Fellowship, Israeli Ministry of Science. This paper was recommended by Associate Editor M. Embrechts.

C. V. Goldman is with the Department of Computer Science, University of Massachusetts, Amherst, MA 01003 USA (e-mail: clag@cs.umass.edu).

J. S. Rosenschein is with The School of Computer Science and Engineering, The Hebrew University, 91904 Jerusalem, Israel (e-mail: jeff@cs.huji.ac.il).

Publisher Item Identifier S 1083-4427(02)02698-X.

single agent has relative to a given query. This local state might also represent which part of the query the single agent is working on. There is a global state that is a vector of the local states. The global state refers to the whole query. It might also represent how the query has been distributed among the agents (e.g., all the agents work on the whole query, each agent works on a different subquery, etc.). Agents move between local states, inducing the system to move between global states;

3) $\mathcal{A}$ is the set of agents, $|\mathcal{A}| = n$;
4) $\{\mathrm{op}\}$ is the set of operators agents can apply. If the agents know different operators, then this set should be interpreted as a vector of sets, such that the component i corresponds to the set of operators of agent A_i;
5) c is the cost of answering a query Q that was submitted to $\mathcal{A}$ in the context of IS. The cost is defined as the cost to get to the most useful state (not necessarily the one that completely solves Q);
6) v is the value of an agent, or a group of agents, i.e., how much information the agents hold. It is a function that quantifies the amount of information held by the agents;
7) w is the worth of a state in the context of IS, i.e., how much information there is in the agents' state with respect to IS and the query Q. This function represents the usefulness of a state relative to a query posed to the agents.

The approach in this work is to exploit domain knowledge in system design; in information oriented domains, knowing the domain's structure can guide us in the design of the agent system (i.e., the number of agents, their interactions, and roles). Different patterns of organization might influence the way in which we would interact with the agent society (e.g., exploiting specialist agents or an agent hierarchy), and the social interactions among the agents. We designed and implemented a multiagent testbed to investigate the emergence of an organization in a multiagent system, and its dynamics. In the implementation the agents were homogeneous, in the sense that they had the same capabilities, and were capable of carrying out the same group of actions (although each agent's information might differ); the agents did not have selfish interests.

The model's design was along the lines of the game of Life [5]. The main idea was to organize the agents according to the information they use for achieving the tasks to which they have been assigned. Work done on the topic of agent organization in DAI has usually dealt with different ways of dividing tasks among agents (see Section VII-A). The organization might change its structure depending on the performance of the agents; in this sense, the organization itself is *task-dependent*. In contrast, we are interested in dividing information among the agents and not in distributing tasks among them; the structure of our organizations would be *information-dependent*. In cases where the organization is task-dependent, new tasks might require new organizations for the agent society. If the organization is driven by information, it can remain constant across multiple tasks that use the same information.

The domain is considered to be given. For example, the information domain might be composed of a collection of files, a site composed of documents on the Internet, etc. Another domain might be the software domain, consisting of programs such as finger, ftp, etc. The aim of this research is to evolve an organization of agents, such that each agent will be responsible for, and eventually become an expert on, a subset of disjoint pieces of this domain. Ultimately, the organization might be approached by other agents and serve as a source of information (that could be further assisted by a system like Musag [6]), or it could itself solve problems, as a multiagent system (e.g., [4]).

Usually, memetic models (for an overview, see [7]) have considered the evolution of the information that could be transferred among a fixed number of agents in a given system. Here, the information set is fixed. The agents' population changes as long as they collect documents from a given site. Agents might die, or clone themselves, as described in the algorithm presented in Section IV-A. The knowledge of each agent is given by the documents it holds, i.e., the information in these documents.

First, the notion of a cellular automata as a basis for our model is presented. We proceed by presenting our approach, and the architecture of the testbed we have implemented, together with results from experiments that were carried out. The paper concludes by showing how our model fits the "six essentials" considered by Calvin [8] to bootstrap a Darwinian process.

III. A Cellular Automata Model for Organizing Information

Cellular automata (CA) are models for studying spatially distributed dynamic systems. They were introduced by Von Neumann in the 1940s [9], [10]. For additional historical notes about related concepts, see [10].

Time is considered discrete. The space of a CA is a grid. Each cell in the grid contains data (e.g., a string of bits). In addition, a set of rules needs to be defined on the data in the cells. These rules determine to which state each cell moves in the next period of time. The rules are uniform (i.e., the same laws apply to every cell), and they are local (i.e., the laws are applied on neighbors of the cells). The locality characteristic of the laws requires the definition of a measure of distance among the contents of the cells.

Parallel platforms enable better performance of the CA for large experiments that require the computation of many interactions among all the cells in the grid (which can be very large). Dedicated architectures were built as CA machines (e.g., the CAM architecture [11]). An extensive overview of applications of CA for studying physical systems appears in [11]. More applications related to pattern recognition and vision can be found in [12].

Complex behaviors that emerge from local interactions among simple units can be studied with CA. In our case, we chose to study the evolution of a population of software agents that became experts on a given set of information using a CA. A cell represents a single document at a given site on the Web. The rules of the system control the growth, death and actions taken by the agents in the population. These rules take into consideration the relative number of documents relevant to the agent, and the number of related documents held by other agents in the same population. The Game of Life is first described; it is one of the most popular CA, and our model and

approach which are based on it are then presented. These two models are compared, and results from experiments performed in the information space are also presented.

The Game of Life. In 1970, the game of *Life*, invented by John Horton Conway, began to attract many enthusiasts who were interested in dynamic societies of living organisms. The game [5] takes place in a two-dimensional (2-D) 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 control and change the configurations of the grid leading to interesting patterns of organization.

Conway's rules were as follows.

- *Survivals*—Every organism with two or three neighboring organisms survives to the next generation.
- *Deaths*—Each organism with four or more neighbors dies (i.e., is removed) due to overpopulation. Every organism with one neighbor or none dies from isolation.
- *Births*—Each empty cell adjacent to exactly three neighbors is a birth cell. An organism is placed on it at the next move.

Note that all births and deaths occur simultaneously. There are three known behaviors that could emerge in different configurations with the rules explained above. There are stable populations, that achieve some structure and then do not change it. There are configurations that fade away, i.e., the grid remains totally empty after a number of iterations of the game. Another pattern of behavior involves periodic or oscillating populations. These populations are composed of subconfigurations that change among them in a definite cycle (larger or equal to one).

Another kind of interesting pattern was discovered later and lead to the proof that *Life* can be universalized, i.e., it can simulate a Turing machine. Gosper at MIT and Conway at Cambridge independently showed that the gliders pattern could be used as pulses to simulate a Turing machine [13]. This new kind of behavior created a pattern that keeps growing as long as the game is played.

These examples stress the unpredictability of the results in *Life* even though the rules of the game are very simple, and seem predictable. Other characteristics of commonly evolved patterns include patterns that tend to become symmetrical although they were not symmetric in the initial configuration. The results of collisions among different forms were also analyzed (e.g., the vanishing collision, and the kickback collision [13], [5]). William Gosper, in 1971, found a *Life* form called the fishhook [13] capable of *eating* other patterns and returning to its original form.

Many variants of the original game have been worked out including playing by other rules, and playing on different geometries and dimensions of grids (e.g., [5]). The original game remained the most interesting and exciting, in particular due to the unpredictability of its simple rules. Other attempts tried to invent competitive games for two or more players, but no significant results were reported.

Dresden *et al.*. [14] developed a set of equations that enable an analytical analysis of the growth, death and survival processes that take part in *Life* games. The degree of nonlinearity is directly related to the number of interacting neighbors. They suggested studying the interesting evolving patterns by using their equations. They found that the formalism and the methods developed for problems in nonequilibrium statistical mechanics are appropriate for an analytic description of the *Life* games.

Eigen [15] referred also to the unpredictable patterns that may evolve in the game of *Life* from the simple and deterministic rules. He had, however, pointed out the lack of an environment in the standard game, i.e., in real biological processes there are stochastic elements that affect the evolution. In our work, we do take into consideration the domain of action of the agents, and observe the patterns of organization that might evolve.

Schulman *et al.* [16] investigated the properties of a dynamic system with deterministic rules such as those defined for the game of *Life* to learn about the microdynamics of a nonequilibrium system, about reproduction and evolution, and to learn about different patterns that can evolve and their relation to physical and biological formalisms. Since the analytical approach is very complex to handle, they were more concerned with understanding the parameters of the *Life* system. These parameters include temperature (i.e., a finite probability for birth or death of an organism independent of the number of its live neighbors), the average density of living entities as a function of temperature, and entropy (a measure of the increasing order in a system controlled by Conway's rules).

Some practical uses were found for the game of *Life* [5]. *Life* was applied to socioeconomic systems, to explain nebulae with spiral arms, and to identify edges in solid shapes.

As described in the following sections, we developed a model for organizing information that was based on a cellular automaton. Our grid is the information space. The "cells" of this grid are assigned to software agents, who are responsible for the information in them. That is, the agents are able to process the information, answer queries, and plan solutions alone or cooperatively. The rules that control the agents' growth, death, and behavior have been designed along the lines of the rules of the Game of Life. In addition, the environment of the agents in the definition of the rules, (i.e., the information space) has been considered.

IV. NetLife Model

We suggest finding a division of an information oriented domain D based on a division criterion (i.e., a similarity measure among the pieces of information in D) and based on a balanced number of agents. We are not dividing tasks among the agents, but rather are dividing knowledge and expertise among the agents. The quality of a partition of a domain depends also on the future performance of the experts responsible for these pieces of expertise. This division depends on a similarity criterion that induces a matrix of nearness values among the pieces of expertise in D. The *local goodness* of an emergent organization depends on the utility of the organization for a given task in D. The *global goodness* of an organization depends on the utility of the organization for a set of tasks in the same domain D.

The assumptions about the domain and agents follow.

- The organization is composed of cooperative agents.
- The domain D has a flat structure (i.e., all information pieces are equally regarded). This is in contrast to hierarchies where agents might be parents of other agents. There is no information held by one agent that includes (or is about) other information held by another agent. Dealing with hierarchies remains for future work.
- The domain is finite.
- There is a measure of similarity defined among the parts of D (i.e., the neighborhood relation).

A *domain* D consists of a finite set of pieces of information d. Each d is a basic unit of knowledge or expertise. An expert can process each unit to answer a query. A set of these is a set of pieces of expertise and will be denoted by $\{d\}$. In particular, $\{d\}$ can be of size one (i.e., one d), or it can be a cluster of related information. The d's in a specific cluster $\{d\}$ are related by a similarity criterion defined by a user. D consists of basic units of expertise (e.g., a document, a file, a picture, a plan recipe, a partial global plan, a speech act, a manual, a software program [e.g., finger, search engines, ftp]). For a given D, there are a number of tasks that can be solved in D.

A domain D is denoted as *structured* if there is a criterion of division of D into s parts $\{d\}_s$ such that these parts are disjoint and their union is D. For example, a site on the Internet with a neighborhood relation such as TFIDF (Term Frequency, Inverse Document Frequency) [17] is structured. The Postman Domain [1] is not structured: there, D is the set of all possible letters the agents might be assigned, and D is not finite. If a finite set of addresses is considered, then a division criterion might be the nearness among the addresses, and then D would be structured.

Our approach models an organization of agents with their knowledge as a Cellular Automaton. The grid in our case is the graph composed of documents taken from a given site on the Web. The edges among the nodes in this graph are given by the links that connect documents in this site. Each agent can hold a set of documents, that is one agent is responsible for several nodes in the graph. The neighborhood relation among the agents is defined based on the similarity between the documents these agents hold. The neighborhood relation is defined among these documents. The rules designed are local, since they refer to the nearness relation defined on the documents that are held by the agents (see Section IV-A1).

Formally, an organization O is a tuple: $O = \langle D, A, L(D), NeighborhoodRelation,\ Parameters,\ Division\ of\ Dintodisjoint\{d\}_s \rangle$ where A is the group of agents comprising the organization over the domain D. $L(D)$ is the common language the agents can communicate with, and with which they can "understand" their knowledge. An expert E ($E = \langle a \in A, \{d\} \subseteq D, L(D) \rangle$) is an agent responsible for a set of pieces of expertise and knows the language $L(D)$. Each expert's expertise is represented by a database described in $L(D)$. For example, when D is a site on the Internet, each agent might hold a set of words that characterize most the documents it is holding. The agents can analyze this database to decide which expert they need to consult. $\{d\}_E$ induces a range of solutions (i.e., answers) that the expert E can achieve.

For a task τ in the domain D, expert E, organization O, and domain $D, \text{Expertise}(E_\tau, O_D) = \{d\}_E \rightarrow l \in L(D)$. E_τ refers to the expert E and its knowledge about the task τ. A task is an information-based query posed to the agent. The task could request the agent to gather information, to apply a software tool, to retrieve pieces of information held by the agent, or it can also be the execution of a program (that is, the implementation of a plan that achieves τ). If τ is not in d_E then $l =$ null. If τ is partial to $\{d\}_E$ then l describes a cooperative solution among different experts.

When a task τ is given to such an organization O, each agent in O can retrieve its cluster as the answer (this is closer to the indexing approach), do some processing on the information in its cluster and retrieve a more focused answer, or cooperate with other agents to retrieve an answer. For a given query, an agent that addresses the organization can address a specific expert once, or several times, each time with a more refined query, address several experts, each one once, or address several experts several times.

The *cost* of performing a task τ by approaching the expert E is: $\text{Cost}(\tau, E) = \Sigma_{\tau' \in \tau} \text{Cost}(\tau', E) + \Sigma_{\tau'' \in \tau \wedge E\{A\}} \text{Cost}(\tau'', E\{A\})$.

The first term considers all the refinements needed to achieve τ, by addressing only E. That is, as E's cluster is bigger, then there is more chance that this term will be larger. The second term considers tasks τ that cannot be performed by E alone. That is, E needs to cooperate with other agents (a group of agents: $\{A\}$; $E\{A\}$ represents all the possible groups of agents from $\{A\}$ with which E cooperates). The tasks to be performed by several agents are denoted τ''. Thus, as E's cluster is smaller there is more chance that this term will be larger.

The *utility of an organization* O for performing a task τ, starting with expert E, is the inverse of $\text{Cost}(\tau, E)$; the values will be in the range [0, 1].

Given that agent B asks the organization O, a task τ, *the quality of the answer* of O is the distance between the goal that B wanted to achieve and the response l the organization has given for task τ.

Two organizations can also be compared by the utility they achieved starting from expert E, for task τ, and the quality of their solution.

A. The General Architecture and Algorithm

The model presented above was implemented in two separate modules (one in C, and one in Java). For a fixed information set, a population of agents evolved. Our approach combines clustering, indexing and evolutionary ideas (see related work in Section VII). The testbed enabled us to experiment with the behavior that might emerge from interactions among agents. The system was composed of homogeneous agents that were added to the system and were dropped out of the system dynamically during each experiment. This dynamic was controlled by a set of rules, that regulated the size of the system based on available resources. In particular, the domain in which the testbed was implemented consisted of a collection of documents.

Based on the rules of the game defined, and the domain of the system, the agents would develop into experts. Their expertise

is given by the information existing in the set of expertise $\{d\}$ (e.g., the documents) that each agent holds. The testbed enables us to run simulations that show how the agents become experts. Each agent collects related d's (e.g., documents) and decides to keep collecting, or decides to return its d's to a general pool (and die), or decides to split its d's among its children, based on a relative size defined by the number of agents already existing with d's that are neighbors of the agent's d's.

The rules for the testbed were designed along the lines of the game of *Life*. The main addition to the game is the consideration of the environment.

Two concepts need to be defined in the game of *Life*: the neighborhood relation and the rules of the game. Separate modules in the testbed described each one of these.

A set of parameters could 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. These parameters include

- $C[i]$—the set of documents currently owned by agent i;
- $\mathrm{ClusterSize}_i$—the size of $C[i]$;
- Free—the set of documents not owned by any agent;
- *PopulationDensity*—the ratio between the number of agents assigned to the documents that are neighbors of the documents A_i was assigned (call it AgentsOnDocs), and the number of documents that are neighbors of the documents A_i was assigned (i.e., *NumDocs*). $\mathrm{PopulationDensity} = (\mathrm{AgentsOnDocs})/(\mathrm{NumDocs})$;
- MyDocs_i—the number of documents the agent A_i is currently holding;
- *NumDocs*—the number of neighbor documents the agent will collect at most at each iteration;
- NumAdded—This is the number of nearest documents the agent might add to its current cluster (chosen from *NumDocs*);
- HighPopDensity—determines when the population is overcrowded;
- MaxDocs—the maximum number of documents that each agent might hold in its cluster;
- MinDocs—the minimum number of documents that an agent can have in a stable state;
- thresholds for the splitting procedure;
- the rule set that comprises the evolutionary algorithm.

The main algorithm for evolving an organization of experts is presented in Fig. 1.

This program was run after a matrix of the similarity values among all the pairs of d's was computed. This algorithm relates to the specific implementation. In the general case, the documents should be changed with a basic piece of expertise d. This algorithm was run on an IOD (e.g., a site on the Internet, files from a directory). The rules of death, birth, and survival were defined based on the resources (i.e., the information in the documents) that the agents held.

During the evolutionary process, the population size might decrease by one, when an agent decides to stop its process based on the rules of behavior (the rules of the game). The total knowledge of the population (given by all the documents held by these agents) changes too, since when an agent dies, it returns its documents to the free pool from which all agents pick their documents.[1]

```
For each agent A_i in the current agents list:
  1 .Find NumDocs documents that are the closest
    neighbors to the documents already in C[i]
  2 .Find out the size of AgentsOnDocs
  3 .Compute PopulationDensity = AgentsOnDocs / NumDocs
  4 .If (MyDocs_i < MinDocs) and
    (PopulationDensity > HighPopDensity)
    4.1. A_i returns its documents in C[i] to Free
    4.2. A_i dies /*Population decreases by one*/
  5 .If (MyDocs_i ≥ MaxDocs) and
    (PopulationDensity < LowPopDensity)
    5.1. A_i spawns a new agent A'_i
         /*Population increases by one*/
    5.2. A_i divides C[i] into C[i'] and C[i'']
  6 .If (MyDocs_i < MaxDocs) and
    (PopulationDensity < HighPopDensity)
    6.1. A_i adds to its current C[i],
     the NumAdded closest documents from NumDocs
     that are not owned by any other agent.
    6.2. Free is updated accordingly
  7 .else do nothing
  8 .Output A_i's current documents in C[i]
```

Fig. 1. Main loop of the evolutionary engine.

In the second case of the algorithm, the size of the population might increase by one, when an agent spawns a child. Then, the total knowledge of the population does not change but it is distributed among the agents in a different way. The agent that decided to spawn, passes part of its documents to its new child.

In the third case, the agent collects additional documents related to the ones it already holds, thus updating its knowledge. The population size remains the same. The individual knowledge of the agent, and therefore the total knowledge of the population, changes.

A testbed was developed to empirically examine different patterns of organization with their respective distributions of knowledge. Based on the rules defined, we aspired to having the system achieve some reasonable balance between the domain information (contained in documents) and the number of agents.

1) The Neighborhood Relation: The aim of this module is to build a neighborhood matrix. In the original game, the neighborhood relation was defined as a geometric relation in the grid of the game. In our case, the nearness among the documents in the set on which the simulation is run, is computed. Therefore, each time an agent needs to calculate its neighbors, it actually looks for the agents that are holding documents that are neighbors of the documents it is holding.

[1]In the implementation, the set Free of documents not assigned to any agent, might remain nonempty. The agents collected documents that were neighbors of the documents the agents already held. The number of documents that were collected depended on the maximal number of documents an agent can hold, and the population density. Therefore, the most distant documents might remain in the set Free. An interesting extension to our testbed consists in adding a new agent for a document in Free, after the actual population reached equilibrium. Then, let the whole population continue until the next equilibrium is reached, and continue in the same way of adding a new agent, until Free gets empty.

The matrix consists of the proximity values calculated for any pair of documents found in the collection. Although the process of downloading documents from a site, and building an inverted index of them can be time consuming, it should be noted that this process is done only once. Then, simulations can be run many times, by different users, using different parameters. The proximity value is a measure of how similar two documents are. The more similar two documents are, the larger their respective proximity value is. This measure should not be confused with a mathematical distance.

We deal with a site of HTML documents found on the Web. This is a finite set of documents that are connected by hypertext links. The parameters set by the user are as follows.

- HomeSite—all the documents that will be collected reside in this site.
- Root—this is the root of the site to be scanned.
- MaxPages—the maximal number of collected pages.
- MaxDepth—the maximum search depth.
- NeighborsThres—a pair of documents will be considered neighbors if their proximity value is above this threshold.
- LowThreshold—only words that appear at least LowThreshold times will be considered.
- StopFile—a file containing very common words that will be ignored when a document is analyzed.

An inverted index of all the words that appeared in these documents is then built. For each word, the total number of times the word appears in all the documents is stored together with the total number of documents in which the word appears, and for each document in which the word appears, the total number of appearances in that document is also kept. Afterwards, the proximity values among all the documents collected is computed, and the neighborhood matrix is built out of these values.

We implemented two relations to decide when two documents are considered neighbors. However, as opposed to the game of Life, the computed relations were not binary. In both cases, the neighborhood relation implied a gradual relation (in contrast to a binary relation, in which we could only say whether two documents were neighbors or not).

One neighborhood relation considered the words that appeared in the documents, that is two documents were neighbors when they were *keyword similar*. The other relation considered the links to which each document pointed. That means that two documents would be considered neighbors if both of them pointed to the same links.[2] A square matrix with rows and columns equivalent to the number of documents collected was implemented. Each cell held a similarity measure between any two documents. Each cell was initialized with zero. In the implementation presented here, the words were weighted in each document following the TFIDF (term frequency, inverse document frequency) method described in [17]. This is one example of a weighting function that can be used to compute the similarity between documents.

For each word that appeared in the inverted index, and documents Doc_i and Doc_j in which the word appeared, the value of the cell $[i, j]$ was incremented by an amount $\mathrm{Incr}[\mathrm{word}, \mathrm{Doc}_i, \mathrm{Doc}_j]$ based on the TFIDF weighting formula. This 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 word appears. To define the term $\mathrm{Incr}[\mathrm{word}, \mathrm{Doc}_i, \mathrm{Doc}_j]$, we need the following parameters:

- T—the total number of documents in the collection.
- D—the number of documents in the collection in which the word appears.
- UC—a Uniqueness Constant (usually its value is between one and three). We chose it to be two based on empirical observations.
- $\mathrm{Doc}_i, \mathrm{Doc}_j$—any two documents in the collection.
- $N[\mathrm{word}, \mathrm{Doc}_i]$—the number of appearances of word in Doc_i.
- $\mathrm{WordFactor} = \log(T) - \mathrm{UC} * \log(D)$ (notice that this factor will be large for words that appear in only a few documents).

Then, the weight of a word in a document Doc_i, $(\mathrm{weight}(\mathrm{word}, \mathrm{Doc}_i))$, was given by $\mathrm{WordFactor} * N[\mathrm{word}, \mathrm{Doc}_i]$. Now, $\mathrm{Incr}[\mathrm{word}, \mathrm{Doc}_i, \mathrm{Doc}_j]$ was the sum of $\mathrm{weight}(\mathrm{word}, \mathrm{Doc}_i)$ and $\mathrm{weight}(\mathrm{word}, \mathrm{Doc}_j)$ if both weights were not zero, and zero otherwise. The neighborhood matrix was stored in a file and used by the evolutionary engine of the testbed. The agents would check this matrix to choose documents that were neighbors of the documents they are holding.

The organization of software agents evolves together with the information. The society of agents is not fixed, nor given.

2) The Evolutionary Engine: A system populated with agents that gather resources was simulated. Eventually, each agent that *survived* until the end of the simulation was expected to hold a cluster of documents that were similar to one another.

All the information the agents had about the documents in the given collection was taken from the neighborhood matrix already built by the other component of the system. In this testbed, we dealt with homogeneous agents coming from a general class of agents. A set of rules was defined to regulate the birth, death, and survival of the agents in the system. This set was based on the number of agents relative to the number of documents that each one of them already held, and the number of documents the other agents in the system held. In the design of these rules, we tried to keep as close as we could to the original rules of *Life*, taking into consideration the addition of the environment.

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 are 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. In other words, this main loop (see Fig. 1) and the different actions that the agents follow influence the dynamics of the system.

[2]All the experiments we present were carried out with the first kind of neighborhood relation. Experiments with the second relation were performed separately.

The population of agents changes based on rules the agents follow, and the values of the parameters of the running system. The main loop of the algorithm presented ends after an iteration in which all the agents do nothing, i.e., no changes are made to the sets of documents the agents held so far. The main algorithm consists of a loop, that each agent performs in turn; this algorithm stops when no agent performs any action. At that point, the agents have reached a stable pattern. Each surviving agent is holding a group of documents that are very close to one another and quite distant from the other documents held by the other agents. We show in Section V a variety of interesting patterns of organizations that have evolved running simulations with this testbed. 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 their characterizing words, are stored.

When agent A_i has to reproduce and spawn a new agent, the same algorithm that was described is used on A_i's database (i.e., use $C[i]$ instead of *Free*). The difference here is that A_i and its new child cannot die or multiply; they will only divide the documents in $C[i]$ between them. An additional factor on which the quality of the clusters depends is the agent multiplying/splitting algorithm. Here, the program first assigns each child an initial seed document. This seed must have had enough neighboring documents on the one hand, and they must not be similar to one another, on the other hand. Then, each child proceeds to cluster documents from the parent's collection around its assigned seeds.

V. Experiments

One of the points that is most attractive in *Life* is the unpredictable behaviors that might emerge from very simple rules, that determine the next state of the system very clearly. Multiagent systems are also very complex to predict; given a problem or range of problems it is very difficult to decide on the optimal division of tasks among the agents such that the load on them will be minimal. In the case of our algorithm, a set of simple rules were defined, very similar to those of *Life*. Even though these rules also determine the next state of each of the agents in the system, the general organization pattern of the agents is unpredictable from the initial situation, or from an initial set of parameters. Therefore, we suggest running simulations with different parameter sets, understanding the possible patterns of organization that might evolve, and after a multiagent system has a known organization of experts we are able to use this expertise for further problem solving or as an information or assistance source.

The parameters that can be tuned by an agent's designer include HighPopulationDensity, MaxDocs, MinDocs, initial number of agents, what the seed documents are, and the number of documents added per iteration. Patterns of organization that were gotten along the lines of the game of *Life* (i.e., stable, periodic, and vanishing) are presented. There were different kinds of possible stable patterns, given the initial parameter values set in the simulation. In addition, there might emerge oscillating organizations. In the case of multiagent systems, this might increase the robustness of the system. For example, when D is a collection of documents distributed on different servers, one of the servers might be down, busy or performing very slowly. In that case, it is beneficial to have another agent on another computer (with the same document) that is capable of working. If different agents can hold the same area of expertise then we can achieve the same task when there is an agent available on a faster server. This should be evaluated as a tradeoff with the cycle of the pattern. If the cycle is very large it might be very expensive to wait for the available agent to do the job.

We experimented with different initial settings to see which kinds of organizations we could arrive at given a set of information (e.g., in our case we chose a site on the Web). The sites reported included the Institute of Computer Science at The Hebrew University (http://www.cs.huji.ac.il), the home page of Tel Aviv University (http://www.tau.ac.il), and the Israel Museum (http://www.imj.org.il).

When the agents were run on the Tel Aviv University site, stable organizations were gotten composed of two, three or four agents, being responsible for issues like admission, the science library, the faculty of medicine, and art studies. When testing the Israel Museum stable systems were gotten of two or three agents responsible for the shrine, archaeology, and for the museum shop information. We tested the site of the Institute of Computer Science at the Hebrew University in greater depth. A matrix of the neighborhood relation among 166 documents taken from the Computer Science site at this university was built.

Stable Patterns. The stop condition of our algorithm is triggered by a state in which every agent has nothing to do, i.e., the conditions for dying, reproducing, or collecting more documents do not hold. 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 ten. The same result was gotten for a different initial number of documents that each agent held (i.e., for two, five, and eight initial documents).

When the minimal number of documents was changed to five, a stable pattern of three agents was gotten for an initial number of documents ranging from one to eight. 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.

When the threshold for the high population was increased to 0.6, we got two different sets of stable patterns. One was reached with eight agents when the initial number of documents was one, two, four, five, orseven. A pattern of ten agents was reached for three or six 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 exercise materials in a course about learning, material about the learning group and

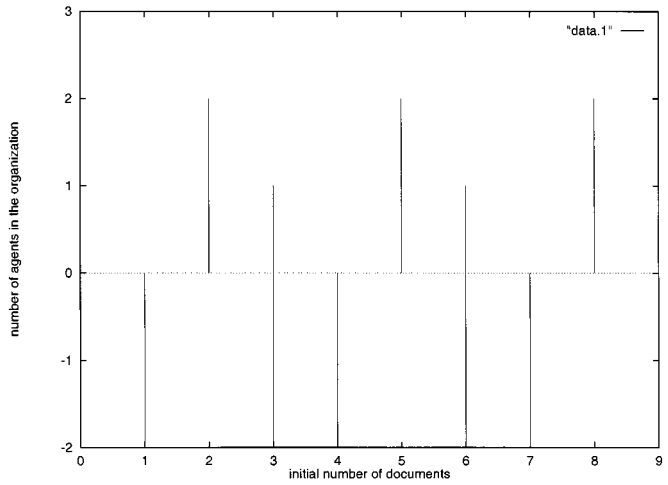

Fig. 2. $\mathrm{HighPopDensity} = 0.2, \mathrm{MaxDocs} = 16, \mathrm{MinDocs} = 10$, two agents in the initial population.

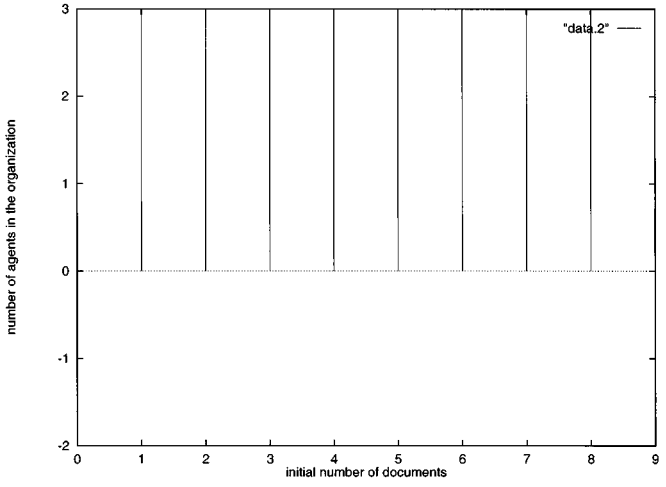

Fig. 3. $\mathrm{HighPopDensity} = 0.2, \mathrm{MaxDocs} = 16, \mathrm{MinDocs} = 5$, two agents in the initial population.

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 ten, we got three different sets of stable patterns: seven agents composed the final structure starting from one, four, or seven initial documents, five agents were the result starting from two or five initial documents, and six agents resulted when the initial two agents started with three or six initial documents.

When the threshold for high population was set to 0.6, the minimal number of documents in a cluster was ten, and the number of initial documents was five, we got two sets of stable patterns: in one the final structure was composed of six agents (when the number of documents added at each iteration was one, five, 20 or 30), in the other stable organization there were eight agents (at each iteration ten documents were added if possible for each iteration).

When the threshold for high population density was 0.8, the minimal number of documents was five, the initial number of documents for each agent was three and the number of documents added at each step was also three, the system reached a stable pattern of 11 agents (four agents for four different exercises, three agents for a project of the Distributed Systems group, two agents for the Data Base group, one agent for the programming laboratory course, and another agent held documents related to writing in HTML).

In Figs. 2–5, 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 Figs. 6 and 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.

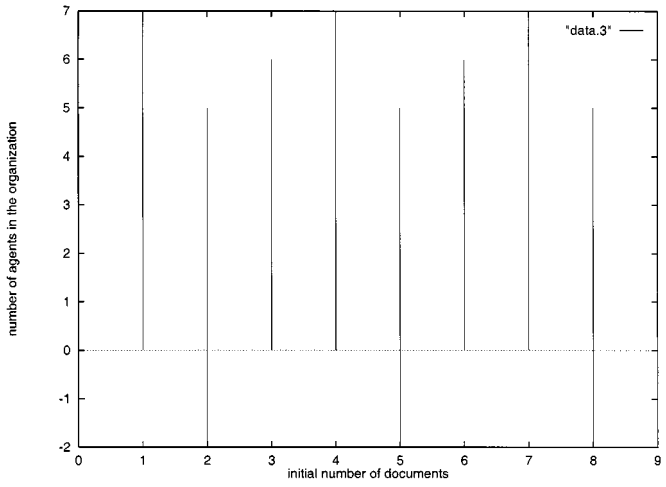

Fig. 4. $\mathrm{HighPopDensity} = 0.6, \mathrm{MaxDocs} = 16, \mathrm{MinDocs} = 10$, two agents in the initial population.

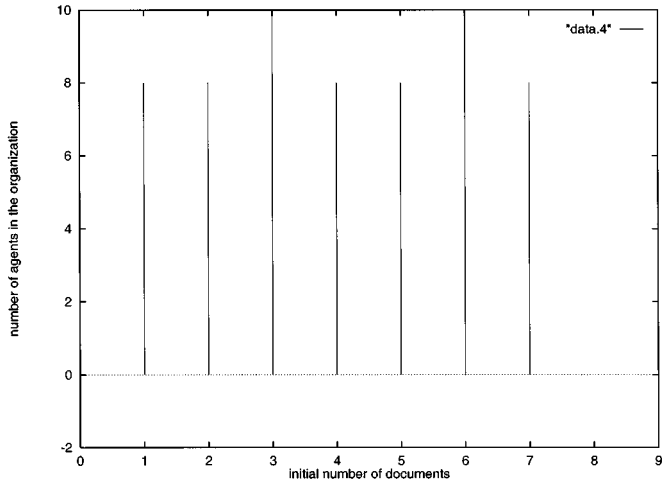

Fig. 5. $\mathrm{HighPopDensity} = 0.6, \mathrm{MaxDocs} = 16, \mathrm{MinDocs} = 5$, two agents in the initial population.

Ever-Growing Patterns. One of the results in *Life* is that the population grows without limit. To find an analogous result in our implementation, it would mean that the agent population would also grow without limit. Following our algorithm, and model, this can happen if there are an infinite number of documents, for which new agents should be cloned so that they will become responsible for them. Theoretically we might think of

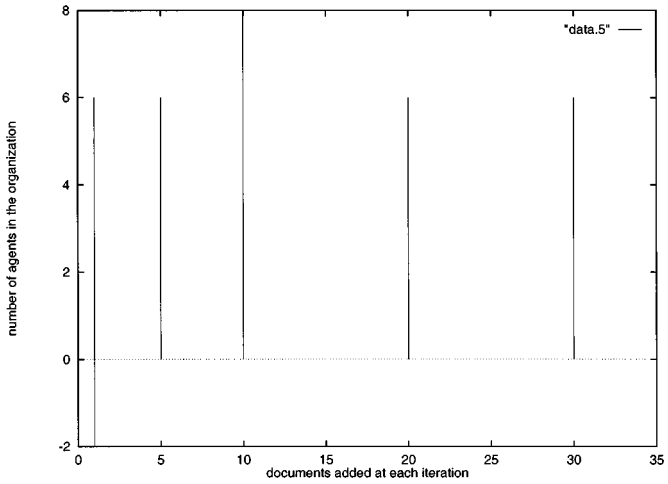

Fig. 6. $\mathrm{HighPopDensity} = 0.6, \mathrm{MaxDocs} = 16, \mathrm{MinDocs} = 10$, two agents in the initial population, five initial documents for each agent.

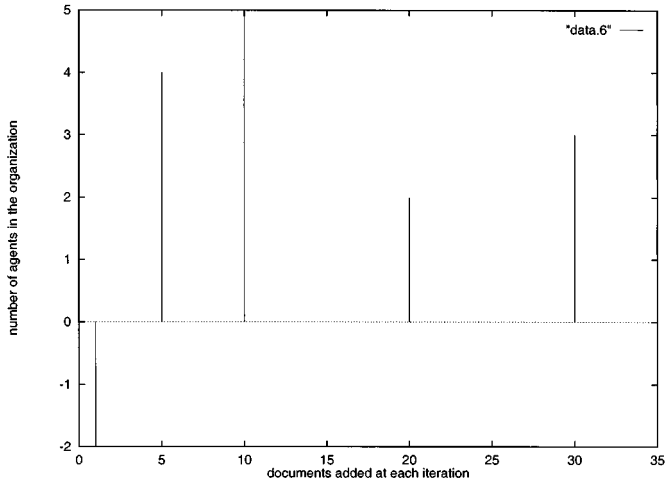

Fig. 7. $\mathrm{HighPopDensity} = 0.2, \mathrm{MaxDocs} = 16, \mathrm{MinDocs} = 10$, two agents in the initial population, five initial documents for each agent.

the set Free (of documents not assigned to any agent) as a set to which documents are added during the simulation. So it might be the case that new agents would be needed for these new documents, and in this way the pattern would keep growing. In the simulation presented here, the site of documents was finite.

There was a possibility in which, for example, one agent split into two agents, one of them died, and the remaining agent repeated this process of splitting into one agent that died, and the other agent split again and so forth, without stopping. The agent that split itself (i.e., the father) and the agent that remained alive and would reproduce, exchanged the clusters of documents they held. In the implementation, each time an agent reproduced, its two sons got two new names, but we could think of an agent spawning itself into another agent, and the same agent remained alive with part of its original documents. This example would be presented as a *blinker* pattern. Therefore in the current implementation with a finite site of documents, we could not reach an ever-growing pattern of organization.

Fading Away Patterns. Following the analogy to the results found in *Life*, settings for which the agents vanished were also tested. Agents could die, leaving other agents alive in the organization, when the appropriate condition is fulfilled.

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.

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 ten, and there were two agents in the initial population, a blinker pattern was gotten for different initial numbers of documents (one, three, four, six, and seven). 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 ten, 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).

Collisions. Other behaviors analyzed in *Life* [5], concerned distinct kinds of collisions among different patterns. This was also relevant in our scenario. There occurred a collision between clusters of documents (i.e., between the agents responsible for them) when one agent died, and its documents were redistributed among the other agents, i.e., other agents took control over these documents like a *Life* pattern that ate another pattern (see Section III), or when we got to oscillating behavior (i.e., agents passed their documents among themselves in a cycle).

VI. The Organization as a Memetic Model

The approach presented here differs from the classic memetic approach. The concept of a *meme* analogous to a gene was devised by Richard Dawkins [18] as a *unit of cultural transmission, or a unit of imitation*.[3] We are concerned about software agents and their pattern of organization. Moreover, we are also concerned about the knowledge this organization has, and how it is divided among the agents.

[3]Works on memetics are published in [19].

Usually in evolutionary memetics, the set of agents is regarded as a fixed set that is given. Then, the different approaches deal with how information evolves in this scenario. In our case, it is both that are evolving, the set of agents and the information they hold. That is, we are evolving patterns of organization that differ in their social structure, and together with this, the information known by each agent evolves. Our point is in showing a model of information transmission that is dynamic. Namely, the organizational structure also evolves.

The population of the agents together with the information they hold evolves. Each agent collects documents. The information contained in these documents comprises the agent's knowledge. This knowledge varies along the time axis, as the agent collects more documents, or passes part of them to its child. If we look at the simulation along the time axis, then looking at the same agent, all the time, we will come across different states:

1) the agent vanishes, and then the population size decreases by one. The total knowledge of the population (given by the set of all the documents held by the agents in the population) changes too, since when an agent dies, it returns its documents to a pool (from where all the agents pick their documents according to their similarity);
2) the agent spawns itself, and then a new agent is added to the population; the size of the population increases by one. At the time of the spawning, the total knowledge of the population does not change but it is distributed among the agents in a different way. The agent that decides to spawn, passes part of its documents to its new son;
3) the agent varies its knowledge. It collects more documents related to the ones it is already holding. The population size remains the same. The individual knowledge of the agent and therefore the total knowledge of the population varies.

On the one hand, our algorithm evolves a population of agents. The agent organization changes as long as the agents die, or spawn children that are added to this organization. Then, we could get different patterns of organizations starting from different sets of initial parameters. At this point we have a structure that changes over time, given the agents that are dropped out of the society, the agents that remain, and the new agents that are added.

On the other hand, we could also look at the information the agents have. These are the same agents that participate in the organization. These are the agents that might die, spawn, or live. This information varies along the evolutionary process of the organization. Each agent "knows more" as long as it collects more documents. This is true assuming the agent's knowledge is monotonic; new facts do not contradict knowledge that was acquired previously. This knowledge decreases when it spawns a child and transmits to it part of its knowledge. The agents do not mutate in the classical sense of genetics, but whenever an agent reproduces by spawning one child (in our current implementation), its knowledge varies. The new agent is born with part of its parent's knowledge (it inherits it from the agent that created it). The knowledge of the father also changes since it is responsible for the part that remained (that is the original set of documents it holds without the documents it has passed to its child).

We consider these two processes together, the evolution of a social structure together with the information distributed among the experts in the evolved population.

The forces that make children spawn or die, or remain unchanged, are the rules of the game, developed along the lines of the rules of the game of *Life*. In our case, the main difference between our rules, and *Life's* rules is that we also take into account the content of the cells in the grid. That is, the agents' decision to spawn or die depend also on the knowledge held by them, and their neighbors (defined by the similarity of their documents). These rules move the world of the agents from one state to another, where these states differ in the number of agents in the society and in the way the information held by current agents is distributed.

The forces that drive the agents reside in the will to balance the number of agents in the society, and to have them evolve as experts, namely, that nobody will know too much nor too little. We look at an effective way to develop a society of experts, and at the same time, we look at how to divide knowledge among them. Instead of doing this artificially, and off-line, we want to test how this process develops in an automatic way, e.g., by evolution.

This evolutionary process entails different patterns of organization, when we look at these evolving agents as a society. We might get stable, oscillating, and vanishing organizations.

A testbed was suggested to empirically test different patterns of organizations with their respective distributions of knowledge. Based on the rules defined, the system was expected to balance the knowledge and the number of agents.

The Organization of Agents as a Darwinian Process. A computational approach has been shown for a population that evolved dynamically, given a finite set of information. Here, it is shown that our model also fits the six essential requirements for a Darwinian bootstrapping of quality [8].

1) *There must be a pattern involved*—The pattern in our case is the structure of the agents' society, e.g., which documents are assigned to which agents, how many agents there are in the group, which patterns of organization evolve.
2) *The pattern must be copied somehow*—From one iteration of the algorithm to the other, the organization is copied and it continues to evolve following the given rules.
3) *Variant patterns must sometimes be produced by chance. Superpositions and recombinations will also suffice*—Different patterns of organizations of agents can evolve by following the rules. The agents might die or might spawn a new agent (and divide their documents appropriately).
4) *The pattern and its variant must compete with one another for occupation of a limited work space*—In our case the space is an information space: e.g., a finite site of HTML documents on the Web. The agents compete over owning the documents in the finite collection.
5) *The competition is biased by a multifaceted environment*—The rules implemented depend on the words in the respective documents. The agents' behaviors change

based on different parameters such as the number of relevant documents, and the relative number of neighbors.

6) *New variants always preferentially occur around the more successful of the current patterns*—The agents divide the documents according to the thresholds in the rules. The agents' aim is to balance their number and the number of clusters of related documents.

With respect to the catalysts and stabilizers presented in Calvin's paper, we found in our experiments that stable patterns as well as oscillating patterns might occur. We limited our algorithm to divide one agent into only one new agent (parcellation), and we also found vanishing (local as global) patterns. As stated in [8]: *When the six essentials are present, and no obvious stability or relative-rate issue seems to be precluding* process, *we are then entitled to predict that our candidate process is capable of repeatedly bootstrapping quality*.

VII. Related Work

A. The Organization of Agents in DAI

The organization and self-organization of agents operating in multiagent systems have been studied in the distributed artificial intelligence (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. Different ways of dividing tasks among the agents can influence their performance. Most work published on this topic considers the organization of the agents as a means that enables the agents to perform the tasks needed to solve a specific problem and achieve a shared goal. The system might reorganize dynamically as a response to a decrease in its performance while solving a specific problem.

Ishida [20] proposed a model in which a predetermined number of agents organized themselves by decomposing the problem they had to solve. Agents, individually, created plans and acted so that the whole group solved a given problem. If the system performance decreased, as a result of conflicting actions, for a while, the agents were triggered to reorganize themselves by changing their current goals, i.e., refining their knowledge, or partitioning a goal into subgoals. Each agent always evaluated its actions based on how each action contributed to the agent's goal, and on how each action contributed to the other agent's goal.

Guichard *et al.* [21] introduced two primitives of reorganization: composition (i.e., regrouping several agents into one), and decomposition (i.e., the creation of several new agents). The authors only mentioned several examples based on the number of tasks, the importance of tasks, or the required time for resolving the problem. The agent then must decide if delegation to another existing agent might take place, or if an agent had to be created first.

Huberman and Hogg [22] looked into communities of practice, i.e., *informal networks that generate their own norms and interaction patterns*. They considered a group of individuals that were trying to solve a problem. Individuals could interact with one another, and they might do so with an interaction strength proportional to the frequency with which they exchanged information with one another. The authors suggested characterizing the structure of a community by counting the number of neighbors an individual has, weighted by how frequently they interacted. They studied the general evolution of such networks and found the conditions that allowed the community to achieve the optimal structure purely through adjustments. Huberman and Hogg stated that as the system grew in size or in complexity, the values of the parameters that brought the system to an equilibrium caused the system to move from flat to a more clustered structure.

An interesting concept introduced by Glance and Huberman [23] was the amount of *fluidity* an organizational structure exhibits. This referred to the level of flexibility of the structure, to the readiness of the agents to modify the structure by changing the strength of their interaction with the other agents.

Nagendra Prasad *et al.* [24] investigated the usefulness of having heterogeneous agents learning their organizational role in a multiagent parametric design system. The agents need to find a set of values for the set of parameters of the problem. The agents could initiate designs, extend them, or critique them, until they got to a final design that was mutually accepted by all of them. It was noted that the designer of the system cannot know beforehand the best assignment of roles to the agents, because he lacks knowledge about the solution distribution in the space. The agents were trained to evaluate their possible roles at each one of all possible states while searching for a solution to specific problems. Then, the agents would select the role that maximized their evaluation.

In all the cases reviewed, the reorganization of a group of agents was strictly related to a specific problem the agents were trying to solve. The study presented in [24] was more general, in the sense that a still predefined number of agents was trained for all the possible states that could occur while searching for a solution to a multiparametric design. The changes produced in their organizational structure were induced by a decrease in performance, or in the latter case [24], for maximizing an evaluation function. In summary, the agents searched for an internal organization while achieving a shared goal, balancing the load imposed on them as a group, and improving their performance. The organization in these works was related to the agents' tasks and their distribution among the agents. In our work, we are interested in organizing the space in which the agents act, e.g., the information domain.

B. Populations of Information Retrieval Agents

With the growth of the Internet and the easy access to semistructured information that it provides, many approaches have been adopted for gathering information in an automated fashion. This work focuses on evolutionary algorithms that create populations of agents that inhabit and exploit the Web.

There are two known systems composed of evolving software agents that retrieve information for users. There are many other single agents applications for information retrieval. For a survey on Web mining techniques for resource discovery, and information extraction see [25]. Here, we have concentrated on evolutionary approaches that are similar to ours. Even though our work is related to societies of information retrieval agents, our approach is different.

The ARACHNID system [26] was aimed at searching for information. The authors were not concerned with the organization of the information and how it could be distributed among a society of agents. The search on the Web was performed for locating information requested by a user. In contrast, we modeled the information in a given site, for a given domain. Each agent in ARACHNID was given an energy amount needed to survive and move, that was updated when the agent got to a document. The energy was increased if the document was relevant, and decreased otherwise. Our output was the organization pattern, that eventually might be used for performing multiagent tasks such as information retrieval or planning. In the ARACHNID system the answer was given finally by one agent; the agents did not coordinate during the search process. In our system, the answer might be distributed among the agents in the society. In ARACHNID, the search was for one user. For each new user, or new query, the whole evolutionary process started again.

Amalthea [27] assisted a user in finding sites of interest to him. The system could be viewed as a single agent, and therefore the organizational pattern of its elements (in our case, the experts) was not the focal point. Amalthea was not concerned about the organization of the knowledge found nor the patterns of the multiagent system. The knowledge was not stored for further use. In our system, the answer might be distributed among all agents in the organization. In Amalthea, for each new user, or for each new request, the whole process started from the beginning.

C. NetLife—Related Implementation Areas

Our approach combines clustering, indexing, and an evolutionary algorithm for developing a dynamic multiagent system in a specific domain of action.

As a preparatory step to the organization-emerging algorithm, we needed to build a similarity matrix between the basic pieces of information of which the domain consisted. In our application, the documents from a site on the Internet were indexed using the term frequency, inverse frequency indexing method (TFIDF) [17]. These indexes then were used to measure the similarity among the documents in the given site. These similarity values induced a neighborhood relation that could induce different partitioning of the given domain.

Clustering procedures were suggested to learn without supervision about a space of data (for a survey see [28], and for file clustering, [17]). We do not know the number of agents we would like to have in the organization, therefore we do not know the number of clusters prior to the evolution of the organization. There are other clustering algorithms in which data is clustered without knowing beforehand the number of clusters [28]. In our case, we aimed at evolving the agents together with the division of information so that the number of agents will be balanced.

Classical clustering algorithms and indexing methods alone do not take into account the domain of action, and in addition they do not consider the future use of the information comprised in it for performing tasks. In our case, each cluster or set of expertise (e.g., set of textual documents) defined the range of information an expert agent had. Therefore, we were interested in the information and characteristics of individual clusters. The amount of information that each agent held would affect its future performance, i.e., whether a single expert could perform a task or answer a query, or whether a single expert needed to cooperate with other agents in the organization to achieve such a goal.

Our concern was to combine the quality of the partition of the data together with the quality of the performance of agents assigned to each cluster. Thus, the experts in our case were also a parameter of the clustering algorithm, and their organization evolved together with the partitioning of their domain of action. An example in which the difference between these approaches is relevant can be a case in which the clustering algorithm divides a domain into two main clusters, based on a similarity criterion and some testing function. In one case, we might want to have smaller clusters so that the corresponding experts will have a smaller load during their work. In another case, we might have a limit on the number of agents that can run on our system, so even if the clustering procedure produced many clusters, we might want to compose several clusters into a bigger one, to conserve the number of processes the agents run.

Moreover, the agents in our system were created and deleted dynamically through the execution of the algorithm. Although this seems to be related to ideas from evolutionary programming, e.g., genetic algorithms [29], we lack a general fitness function that could evaluate the performance of a population before the organization of agents is actually *used*.

Kohonen features map is an unsupervised learning algorithm for self-organization applied to artificial neural networks [30]. No reference was mentioned to the performance of tasks, given such a feature map. The number of nodes in the map depends on the clusters found among the input vectors. We were interested in tuning this number according to the number of agents that would need to use the information in the nodes. This is in addition to the clustering done based on the similarities among the input vectors. An application of the Kohonen feature map algorithm was used to build a semantic map of 140 documents indexed by the keywords Artificial Intelligence, for information retrieval [31].

Our idea was to evolve a multiagent system that would behave as a group of experts that could service other agents with information (e.g., help in achieving a goal) or as a system that was already preorganized and could achieve goals by itself.

VIII. Practical Aspects

Having an organization of expert agents can serve the agents themselves, other applications or users in cases where specific information is requested or when a task has to be solved. In other words, there are two main purposes or uses for an organization of experts in a given domain: as a source of information, and as preparatory step toward achieving a goal in the given domain (i.e., the agents involved in finding a solution to a given problem will not need to start planning from scratch, but they will be already organized in such a way that each of them has specific expertise in order to suggest an initial [at least] plan).

An organization as a source of information.

Information Retrieval Systems. We refer to a general class of problems that can be stated in a domain of information. The general and typical query that might be asked is to retrieve all

documents that are related to a keyword X. In such cases, the organization is approached to give an answer, given that the information is already organized according to a similarity measure.

In this case, we can use the accepted (standard) measures of recall and precision when calculating the utility and quality of the organization [17].

If we do not know who the appropriate expert is, and there are k experts in the organization, then it will take $O(K) \cdot O(\text{ProcessingTime}_i)$ steps to find expert i, and check whether it is the relevant one. If we address all the experts, it will take $K \cdot O(1) = O(K)$, and if we address N times k experts, $O(N * k)$. In addition, it is necessary to take into account the processing time of the expert or experts when they work on the query. Without having the organization, it will take $O(|D| * W)$ actions to go over all the documents in D, and process each one in W time to find the answer to the query.

Planning. Decker *et al.* mentioned in [2], their intention to develop a library of reusable coordination mechanisms. We might think of D as a domain of coordination mechanisms. Our algorithm divides this domain into groups of mechanisms that are related according to some measure of similarity. A corresponding organization of agents will be developed as well. These agents' expertise will be in the features of these mechanisms, and include the conditions under which they can be used. Other agents that need to coordinate their actions could approach the expert organization and get help as to which coordination mechanism they would be best off using.

Grosz and Kraus [3] used the term recipe, R_α to describe the actions that need to be done to achieve a goal α. These actions might have to satisfy certain constraints, and might involve a set of agents that have to perform the actions at different times (i.e., these are the variables of the recipe). The model they presented included agents that held libraries of recipes, and the agents' success in achieving their goals was also based on the integration of different recipes from different agents. When an agent was aware of a partial individual plan, and needed to find a recipe for an action in its plan, two options were proposed in [3]: contract the execution of the action and do research, i.e., ask an expert.

In our model, we suggest evolving an organization of agents that would be responsible for recipes in a given domain. Instead of having an agent facing a problem and starting to plan from scratch, it might approach an organization in its domain and ask for an initial solution. Another possibility is that agents might approach the expert agents when they need a specific recipe, and might be able to evaluate similar recipes with regard to the cost and efficiency of the different ways proposed by the expert to achieve the agent's goals. Therefore the query that an expert will be asked will be of the following form: *how do I achieve goal G? How do I get to state S?* Then the utility and quality of an organization could be computed in terms of the steps needed to perform the plan, and what is each agent's cost, and how far is the goal the system arrived at from the real goal (i.e., the one they actually intended to achieve).

Another option is to have a domain consisting of cases [32], and corresponding solutions. During the organization evolution process, the agents will learn about the similarities between the cases, and divide among them the existing experience. This kind of organization can be approached by agents willing to find prepared solutions for situations similar to those in which they find themselves.

In this case, a user that approaches the organization is acting like a person that approaches a library of reusable plans that have been organized (i.e., catalogued).

Tools and Skills. An agent might not have an ability to run a program or use a tool for achieving a goal to which it was assigned. Maybe it cannot access a specific program in its computer at that time (e.g., the mail-server is down).

The domain can consist of a set of software. For example, a set of expertise can include programs for finding people (e.g., finger), search engines, access to DBs, news, ftp, etc.

An agent can approach the organization with a query: *look for person X*. The expert that knows how to look for people suggests the use of one of its programs, or it chooses one according to the resources available, and performs the associated action.

If we did not have the organization, the cost of performing the task by an agent that does not have the appropriate skills would be essentially infinite, since the agent could not achieve its goal. If we have the organization, it will take $O(K)$ steps to find the corresponding expert out of the K in the organization.

For a more efficient division of labor. The organization of experts can also act as a multiagent system that achieves the goals assigned to it. When the organization gets a goal, each expert can suggest the best partial plan or recipe it has for achieving the closest goal to the one they were assigned. The best partial plan or recipe is chosen (e.g., by a synchronization procedure, or by known cooperative problem solving paradigms for coordination, such as voting [33], [34], negotiation [1], or communication; for an overview on coordination mechanisms see [35]). The partial plan or recipe includes what each agent needs to do and a schedule for doing it.

Another (related) use of such an agent organization is to consider the organization as a multiagent system that solves problems presented to it. Each agent's expertise or knowledge is given by the part of the domain for which it is responsible, after the evolutionary algorithm has been run. Were the agents simply approached as a source of information, they might retrieve a plan, or advice as to how it is possible to solve the goal. Here, though, the agents are also actively performing the plan instead of just suggesting it.

We have implemented a multiagent planner based on the evolution algorithm presented here [4]. This implementation, however, is beyond the scope of the current paper.

IX. Conclusion

An evolutionary approach was presented for creating agent organizations. This is a framework that can be used to test different patterns of organization that might emerge given a set of information. Since parameters such as the size of a multiagent system, and division of roles, are very difficult to determine beforehand, a model was proposed to test different organizations, and then enable a designer to choose the one that best matches the structure of his domain for future actions. First, the approach serves as a tool to analyze emerging organizations of agents. Second, the same tool can serve to analyze the structure of information domains in which agents will eventually act.

ACKNOWLEDGMENT

The authors wish to thank O. Margoninski for implementing the first version of NetLife.

REFERENCES

[1] J. S. Rosenschein and G. Zlotkin, *Rules of Encounter*. Cambridge, MA : MIT Press, 1994.
[2] K. S. Decker and V. R. Lesser, "Designing a Family of Coordination Algorithms," Univ. Massachusetts, Amherst, Tech. Rep. 94-14, 1994.
[3] B. Grosz and S. Kraus, "Collaborative plans for group activities," in *Proc. Thirteenth Int. Joint Conf. Artificial Intelligence*, Chambery, France, 1993, pp. 367–373.
[4] C. V. Goldman and J. S. Rosenschein, "Partitioned multiagent systems in information oriented domains," in *Proc. Third Int. Conf. Autonomous Agents (Agents'99)*, Seattle, WA, 1999.
[5] M. Gardner, *Wheels, Life and Other Mathematical Amusements*. New York: Freeman, 1983.
[6] C. V. Goldman, A. Langer, and J. S. Rosenschein, "Musag: An agent that learns what you mean," *J. Appl. AI, Special Issue on Practical Applications of Intelligent Agents and Multiagent Technology*, vol. 11, no. 5, 6, 1997.
[7] "A brief overview and history of memetics", B. Edmonds. (1996, Nov.). [Online]. Available: http://www.cpm.mmu.ac.uk/jom-emit/overview.html
[8] W. H. Calvin. (1997) The six essentials? minimal requirements for the darwinian bootstrapping of quality. *J. Memetics—Evol. Models Inform. Transm.* [Online]. Available: http://www.cpm.mmu.ac.uk/jom-emit/1997/vol1/
[9] S. Ulam, "Random processes and transformations," in *Proc. Int. Congr. Mathematics*, 1952.
[10] J. Von Neumann, *Theory of Self Reproducing Automata*, A. Burks, Ed. Champaign, IL: Univ. of Illinois Press, 1966.
[11] T. Toffoli and N. Margolus, *Cellular Automata Machines—A New Environment for Modeling*. Cambridge, MA: MIT Press, 1987.
[12] K. Preston, Jr. and M. J. B. Duff, *Modern Cellular Automata*. New York: Plenum, 1984.
[13] E. Berlekamp, J. Conway, and R. Guy, *Winning Ways for Your Mathematical Plays*. New York: Academic , 1982, vol. 2.
[14] M. Dresden and D. Wong, "Life games and statistical models," in *Proc. Nat. Acad. Sci.*, vol. 72, 1975, pp. 956–960.
[15] M. Eigen, "The origin of biological information," in *The Physicist's Conception of Nature*, K. Mehra, Ed, Dordrecht, The Netherlands: Reidel, 1973, pp. 594–635. Cited in "Life games and statistical models," by M. Dresden and D. Wong, in *Proc. Nat. Acad. Sci*., Vol. 72, no. 3, pp. 956–960, Mar., 1975.
[16] L. S. Schulman and P. E. Seiden, "Statistical mechanics of a dynamical system based on conway's game of life," *J. Statist. Phys.*, vol. 19, no. 3, pp. 293–314, 1978.
[17] G. Salton and M. McGill, *Introduction to Modern Information Retrieval*. New York: McGraw Hill, 1983.
[18] R. Dawkins, *The Selfish Gene*. Oxford, U.K.: Oxford Univ. Press, 1976.
[19] . *J. Memetics—Evol. Models of Inform. Transm.* [Online]. Available: http://www.cpm.mmu.ac.uk/jom-emit/
[20] T. Ishida, "The tower of babel: Toward organization centered problem solving," in *Proc. Eleventh Int. Workshop on Distributed Artificial Intelligence*. Glen Arbor, MI, Feb. 1992.
[21] F. Guichard and J. Ayel, "Logical reorganization of dai systems," in *Intelligent Agents: ECAI-94 Workshop on Agent Theories, Architectures, and Languages*, 1994, pp. 118–128.
[22] B. A. Huberman and T. Hogg, "Communities of practice: Performance and evolution," *Comput. Math. Org. Theory*, vol. 1, pp. 73–92, 1995.
[23] N. Glance and B. A. Huberman, "Social dilemmas and fluid organizations," in *Computational Organization Theory*, K. M. Carley and M. J. Prietula, Eds. Hillsdale, NJ: Lawrence Erlbaum, 1994, pp. 217–240.
[24] M. V. N. Prasad, V. R. Lesser, and S. E. Lander, "Learning organizational roles in a heterogeneous multi-agent system," in *Proceedings of the Second International Conference on Multiagent Systems*, Kyoto, Japan: AAAI Press, Dec. 1996.
[25] O. Etzioni, "The world wide web: Quagmire or gold mine?," *Commu. ACM*, vol. 39, no. 11, pp. 65–68, 1996.
[26] F. Menczer and R. K. Belew, "Adaptive information agents in distributed textual environments," in *Proc. Second Int. Conf. Autonomous Agents*, Minneapolis, MN, May 1998, pp. 157–164.
[27] A. Moukas and P. Maes, "Amalthea: An evolving multiagent information filtering and discovery system for the WWW," *J. Auton. Agents Multiagent Syst.*, vol. 1, no. 1, pp. 59–88, 1998.
[28] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. New York: Wiley, 1973.
[29] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press, 1996.
[30] T. Kohonen, *Self-Organization and Associative Memory*. Berlin, Germany: Springer Verlag, 1984.
[31] X. Lin, "A self-organizing semantic map for information retrieval," in *Proc. Fourteenth Annu. Int. ACM SIGIR Conf. Research and Development in Information Retrieval*, Chicago, IL, Oct. 1991, pp. 262–269.
[32] J. L. Kolodner, *Case Based Reasoning*. San Mateo, CA: Morgan Kaufmann, 1993.
[33] E. Ephrati and J. S. Rosenschein, "The Clarke Tax as a consensus mechanism among automated agents," in *Proc. Ninth Nat. Conf. Artificial Intelligence*, Anaheim, CA, July 1991, pp. 173–178.
[34] ——, "Reaching agreement through partial revelation of preferences," in *Proc. Tenth Eur. Conf. Artificial Intelligence*, Vienna, Austria, Aug. 1992, pp. 229–233.
[35] C. V. Goldman, "Emergent coordination through the use of cooperative state-changing rules," M.S. thesis, Hebrew Univ., Jerusalem, Israel, 1993.
[36] ——, "Multiagent learning systems," Ph.D. thesis, Hebrew Univ., Jerusalem, Israel, 1999.

Claudia V. Goldman received the B.Sc. degree in computer science in 1991, and the M.Sc. and Ph.D. degrees in computer science/distributed artificial intelligence in 1993 and 1999, respectively, all from the Hebrew University, Jerusalem, Israel. Her Ph.D. thesis was in the area of multiagent systems. In particular, she designed and analyzed algorithms for systems composed of agents who learn to coordinate their behaviors, and agents who learn to organize their knowledge.

She was a Postdoctoral Fellow at Bar Ilan University between 1999 and 2001, where she studied interactions among autonomous agents in electronic marketplaces. She is now with the Department of Computer Science, University of Massachusetts, Amherst. Her research interests are in adaptation, learning, and their complexity in multiagent systems.

Jeffrey S. Rosenschein received the A.B. degree in applied mathematics from Harvard University, Cambridge, MA, in 1979, and the M.S. and Ph.D. degrees in computer science from Stanford University, Stanford, CA, in 1982 and 1986, respectively. His Ph.D. dissertation was in the area of distributed artificial intelligence, and analyzed principles of multiple agent interactions.

His research since that time has continued to focus on issues of cooperation and competition among high-level problem solving entities. He is currently an Associate Professor in the School of Computer Science and Engineering at The Hebrew University, Jerusalem, Israel, where his interests are in distributed artificial intelligence, with an emphasis on multiagent negotiation, learning, planning, and decision making.