Partitioned Multiagent Systems in Information Oriented Domains

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

Multiagent systems often operate in environments where access to, and manipulation of, information is paramount. We formally characterize a subset of these domains as *Information Oriented Domains*, and consider a novel organizational approach to handling multiagent problem solving in these environments.

The initial step is to partition the information domain, using an evolutionary algorithm and a similarity criterion. This partitioning step results 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 arise naturally from the information content of the environment, and are not determined *a priori*. This organization of agents, spanning the information space, can then be exploited in problem solving and information retrieval tasks. The partitioning of the information space leads to greater efficiency at run time.

In general, these partitioned systems are appropriate for information retrieval systems, or (more specifically) for libraries of software tools or partial plans. Moreover, these organizations can also be used as the basis for domain-specific problem solvers, where the partitioning leads to efficiency in searching the solution space.

As an illustration of this last advantage of dynamically partitioned systems, we describe an organization-based multiagent planner that we have designed and implemented. The agent expertise covers a set of HTML documents, and makes use of semantic tags added to these documents. A user can query the system, retrieving both the answer desired as well as additional related (and useful) information held by each of the agents in the organization.

Content Areas: Distributed AI, Information Gathering, Organization of Agents

1 Introduction

The organization of multiagent systems (MAS) can potentially be quite complex, and difficult to design appropriately. Given a problem that needs to be solved by an MAS, it is challenging to know ahead of time how many agents are appropriate, the role each agent should play in the organization, and the necessary interactions among agents.

There are several directions that one can take in 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) and cooperative problem solving 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 can also serve as a preliminary step to the distribution of tasks among agents.

Our approach is that in certain domains, knowing the *structure of information content* can guide us in the design of an appropriate agent system. MAS often operate in environments where access to, and manipulation of, information is paramount. We formally characterize a subset of these domains as *Information Oriented Domains* (IODs). Our first step is to partition the information domain, using an evolutionary algorithm and a similarity criterion.

In general, these partitioned systems are appropriate for information retrieval systems, or (more specifically)

*Supported by the Eshkol Fellowship, Israeli Ministry of Science

as an organizational structure for libraries of reusable plans [3] and as a means to organize different recipes [8] for plans in a given domain.

We discuss the evolutionary algorithm (Section 2 and 2.2) that leads to different emergent patterns of organization. We focus on its application to IODs, presenting their formal definition and specifics in Section 3.

One applied use of such an evolutionarily-derived organization, namely an organization-based multiagent planner in the information retrieval domain, is presented in Section 5. Each agent holds information that it has been assigned through the evolutionary algorithm. This information, together with semantic operators, comprises the agent's expertise. The organization can be approached by a user who poses a query (couched in the appropriate query language). The agents try to answer the query, both directly and by providing the user with information that might be relevant to the solution. The agents' overall strategy is to get to the most "useful" information state, based on the user's query.

2 The Emergence of an Organization of Agents

A multiagent testbed was designed and implemented to investigate the emergence of an organization in an MAS. In [7], we presented experimental results, showing interesting patterns of organization that might evolve, along the lines of the game of Life [4].[1]

Our evolutionary model is based on a Cellular Automata architecture [17, 13]. We also borrow from other related areas, such as clustering and indexing, although we could not have handled our partitioning task using techniques from these domains alone.

The population of agents, and the information those agents hold, evolve over time. Agents are run as autonomous threads, developing their own expertise (in contrast to a system that does clustering and then attaches agents to each cluster). Each agent collects documents from a given set of information; the information contained in these documents comprises the agent's knowledge. This knowledge may vary as the agent collects more documents, or passes part of its documents to a newly-spawned child agent.

A set of simple rules was designed that can control the birth, death, and survival of agents in the system (we kept close to the original rules of *Life*, though taking into account the addition of an information environment). Given that the following parameters have their values set, we present the main algorithm with the currently implemented rules.

[1]Three main characteristics of forms that can evolve in the game of Life are whether the pattern is stable (i.e., fixed through time), periodic (i.e., oscillating among different forms with a fixed period), or fades with time (i.e., the population dies out). We derived all three kinds of patterns in our experiments, for different sites and parameter settings.

The parameters include: the number of neighbor documents the agent will maximally collect at each iteration (NumDocs), the number of nearest documents the agent might add to its current cluster (NumAdded), the threshold that determines when the population is overcrowded (HighPopDensity), the maximum and minimum number of documents that each agent might hold in its cluster (MaxDocs and MinDocs), the set of rules that comprise the main evolutionary algorithm, the group of documents currently owned by agent i ($C[i]$ and its size $ClusterSize_i$), the set of documents that are not owned by any agent (Free), the number of documents agent A_i is currently holding ($MyDocs_i$), and the ratio between the number of agents assigned to the documents that are neighbors of the documents to which A_i was assigned (call it AgentsOnDocs), and the number of documents that are neighbors of the documents to which A_i was assigned (i.e., NumDocs).

The main algorithm consists of a loop that each agent performs in turn; the algorithm stops when no agent has performed any action. When the algorithm stops, the agents have reached a stable pattern. Each surviving agent is holding a group of documents that are very close to one another, and distant from the other documents held by the other agents.

Each agent collects NumDocs documents that are the closest neighbors of the documents already held at $C[i]$. It computes the density of the population given by $\frac{AgentsOnDocs}{NumDocs}$. Then, the agent checks what its current situation is regarding the number of documents it holds and the density of the population, and decides what action to take: if the number of documents the agent is currently holding (i.e., $MyDocs_i$) is less than MinDocs and the population density is larger than the threshold HighPopDensity, then the agent returns its documents in $C[i]$ to Free, and dies. If $MyDocs_i$ is larger than MaxDocs, and the population density is less than the threshold LowPopDensity then the agent spawns a new agent A'_i, and divides $C[i]$ into $C[i']$ and $C[i'']$. The third case happens when $MyDocs_i$ is less than MaxDocs and the population density is less than the HighPopDensity threshold; then the agent adds to its current $C[i]$ the NumAdded closest documents from $NumDocs$ that are not owned by any other agent. The set Free is updated accordingly. The agent does not do anything if none of these three cases is true. Each agent also outputs its current documents in $C[i]$.

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

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,[2] 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.

On the one hand, one can consider our algorithm as evolving a population of agents. The agent organization changes as long as agents die or spawn children that are added to this organization. In practice, we get different patterns of organization starting from different sets of initial parameters. We have a structure that changes over time, with agents that die, agents that survive, and new agents that are born.

On the other hand, we can also look at the *information* the agents are holding. These are the same agents that participate in the dynamically changing organization, who might die, survive, or spawn. This information changes as the evolutionary process progresses. Each agent "knows more" as long as it collects more documents. This knowledge decreases when it spawns a child and transmits to it part of its knowledge.

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

2.1 Related Work

Two other software agent systems have been developed to retrieve information based on an evolutionary approach [11, 12]. Although [11] is also targeted at searching for information, that research is not concerned with the organization of the information and how it can be distributed among a society of agents. The agents move from location to location without storing the knowledge discovered. The second system, Amalthea [12], assists a user in finding sites that are of interest to him. The system can be viewed as a single agent, and thus the organizational pattern of its elements (in our case, the evolving agents) is not the system's main concern.

Other multiagent systems designed for information gathering include the InfoSleuth project [14], Retsina [2, 16], and MACRON [1].[3]

[2] The similarity between the documents was computed based on the TFIDF formula [15].

[3] We mention here only information gathering agent-based systems. For a more complete overview of related systems, see [6].

2.2 Domain Considerations for an Organization-based Multiagent System

We have shown how an agent organization might be evolved for a given information domain. Whenever this organization is approached for real use (e.g., presented with a query), it might be the case that not all the agents' information is relevant to the task. Therefore, given an organization, and a subdomain of the overall domain of the organization, we might be able to find a suborganization of the original organization that is more suitable given the specific goals. We have designed an algorithm to find a suborganization O* of a given organization that is minimal in size (i.e., minimal number of agents necessary), and maximal in information.[4]

Once an agent organization has been evolved for a given domain, and the relevant suborganization O* is found, it can be approached for two primary uses. The organization can be regarded as *an organized source of information*. The organization can also be considered *a multiagent problem solver*.

In the first case, for example, the domain in question might consist of recipes (as Kraus and Grosz denoted the notion of partial/final plans [8]), coordination mechanisms (Decker [3]), cases (in the context of case-based reasoning (Kolodner [9]), or software tools. Then, agents can be approached to retrieve a partial/final plan for a situation presented by a user. The situation can be one that is already known by the agent, or it can be similar to one that the agent knows.

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). These recipes for plans have been clustered based on a similarity criterion that the designer of the evolutionary algorithm (as explained in Section 2) has determined.

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. The agents are approached with a goal to achieve. 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.

In our system implementation, the agents know the information they hold, and in addition they possess operators to manipulate their knowledge (and can thus actively achieve certain goals).

[4] The description of this algorithm is beyond the scope of the present paper, but can be found in [6].

3 Information Oriented Domains

Although the evolutionary approach to deriving agent organizations is general, we focus in this paper on Information Oriented Domains.[5] Formally:

Definition 1 (Information Oriented Domains (IODs)) *An IOD is a tuple $< IS, S, \mathcal{A}, \{op\}, c, v, w >$, where: IS is the information set, e.g., a set of documents, S is the set of possible states of the world. In an IOD, we define a state at time t as pair $[Q, PA_t]$, where Q is the query a user submits, and PA_t is the partial answer gotten up to time t. In this case a partial answer is measured by the worth of the state, which represents how much the derived information is useful and related to Q. $\mathcal{A}$ is the set of agents, $|\mathcal{A}| = n$. $\{op\}$ is the set of operators that the agents can apply. c is the cost of getting to the most useful state relative to the given query Q that was submitted to $\mathcal{A}$ in the context of IS. v expresses how much information the agents hold. w expresses how much information is in the state relative to the information of the agents, IS, and the query Q.*

The main points to notice in these kinds of domains are the information for which the agents are responsible, and the definition of a goal state (or, more precisely, the lack of a simple definition). The agents' aim is to get to useful states (the more useful, the better). If the agents fail in achieving the goal posed to them, they might nevertheless hold additional information that can serve the user even if this is not exactly the information the user was looking for. Moreover, if the agents do succeed in getting the exact answer the user was interested in, the agents might hold additional information that can also be useful to the user. Agents might contribute to the answer with their own points of view, or partial views based on their expertise.

4 Planning in Information Oriented Domains

The model of IODs, where utility is attributed to aggregating information in response to a user query (and the notion of goal satisfaction is "fuzzy") can serve well for an anytime planner, or for planning under constraints. Each of the agents in the organization holds information that can be useful to the query posed by the user. If the time limit is reached, or any other resource that constrains the running of the system is exhausted, the agents can be approached to retrieve their current information, and its usefulness relative to the problem in question.

When dealing with an organization-based multiagent system, there are three main steps to handle:

1. Evolve the population — This refers to running the evolutionary algorithm presented in Section 2. The organization we get consists of a number of agents, each holding a disjoint part of the domain of action of the agents.

2. Prepare the organization for planning (or any other problem solving task) — This is the step in which the HTML files are processed; semantics are manually or automatically added to the files. Currently, we make use of no automatic tool to solve this problem, using instead manual insertion of tags from the Simple Html Ontologies Extensions Language [10].[6] At this step, each file that each agent holds is processed, and SHOE-like semantics are added to them. An ontology file describing the world of the agents is also built. The semantics added to the agents' knowledge include also the operators the agents will use in the third step. If the agents are to learn adaptive operators, then this is the step where the agents process their own knowledge. In addition, this step might include the running of another algorithm to choose a suborganization out of the organization developed in the first step, which will focus on the specific domain of interest.

3. Let the organization plan (or solve any other task) — The user approaches the organization and poses a query (a goal) to be solved by the organization (there are other scenarios where the user, or his personal agent, approaches a specific agent or a specific group of agents to solve a query). In our case, we implement an interface agent that handles the user query, by communicating with the entire organization.

5 An Organization-based Multiagent Planner

We have implemented an organization-based multiagent planner in Java. The specific agent organization with which we worked emerged after running the algorithm presented in Section 2; the domain the organization was formed over consists of HTML documents. We also add semantics to the agents' files based on SHOE [10].

Each agent is implemented as a Java thread, allowing each agent to be autonomous. The language in which the user poses his queries is also based on the relations described in the SHOE-based ontology file [5]. We define a general agent class, and then two more specific agents that inherit from this class (the planner agent, and the interface agent, A_0).

A_0 mediates between the user and the agents in the organization. Currently A_0 sends every subquery to each one of the agents in the organization. Another approach to the solution of the query is to let A_0 solve the *assignment problem*, i.e., the agent A_0 assigns a subquery to another agent in the organization or to a subgroup of agents, based on the agents' expertise, and in

[5] For a more complete discussion of Information Oriented Domains, see [6].

[6] Were the files to be marked up with XML tags prior to running the evolutionary algorithm, this semantic processing step might not be required.

such a way that the agents' partial solutions will overlap as little as possible. Agents might reason in different ways; they might know different operators, and even if they share the same language of operators, they might have different knowledge (i.e., the content on which they apply these operators is different). Applying the same operators on different values might result in different points of view.

After having evolved an agent organization for a given domain, we continue with the second step as explained in Section 4. We first build an ontology containing the entities and their categories in the given world. We have chosen to implement the semantics of the agents' knowledge and the ontology file in the SHOE [10] language.

The ontologies additionally include the relations that exist in the given world. The ontology holds the relations and rules with their parameters that will be instantiated by each of the agents in their personal pages. We also added in the description field of each relation a number representing the "level of usefulness" of the relevant relation to the entire ontology.

When the multiagent planning system is initialized, each agent is also given the corresponding ontology file so that it can build its data base, together with general information about the world. Each agent holds three hash tables containing information about the relation objects and their usefulness, and the rules the agent knows. Each agent also holds two blackboards containing information about the current and the accumulated solutions this agent has found.

The interface agent A_0's data include: the query submitted by the user, the size of the planning organization, the prefix that relates the ontology file to the world the agents know, and a blackboard that holds the solution that is given to the user (each new partial solution for each subquery is appended to this board).

The blackboards consisting of the solutions include assignments to variables, and the relations and rules that have been instantiated.

Agents send to each other KQML-like messages consisting of the following fields: a performative (e.g., startQ, finished, failed, partial, stop), the content of a message is given by a blackboard that represents the current solution submitted by an agent, the current subquery to be solved, the receiver of the message, the sender of the message, the usefulness level of the solution in the content field, the entire blackboard of the agent, including all the information it has gathered up to the time of this message, the prefix of the relevant ontology, and the time of the message.

The user interacts with the multiagent planner through a graphical user interface. The system first needs to know the size of the organization and the name of the file that describes the ontology. The user is then requested to enter all the files held by each of the agents in the organization and their locations. Finally the user is asked to enter his query (composed of the conjunction of relations or rules described in the ontology). The output of the planner includes the solution found to the entire query, and the additional information held by all the agents in the organization.

After this data has been input to the system, the interface agent A_0 enters the main loop of the planner. A_0 sends a startQ message to all the agents with the current subquery (q_{curr}). It collects all the agents' answers, and chooses the best answer (i.e., a "finished" answer or a partial answer with the highest level of usefulness). If the best answer found is not "failed" then A_0 updates its blackboard, and prints it for the user. This loop continues until there are no more subqueries to solve.

Each planning agent gets a subquery at a time, and tries to solve it using the knowledge at its disposal. Specifically, either the agent has a relation whose parameter values are consistent with the values given to the same variables in the solution found so far, or it can find a partial solution or a rule whose consequence induces the current subquery. When the agent looks for a partial solution, it checks the subquery level of usefulness, and it looks for other relations or rules that have the same level of usefulness, or have other levels incrementally distant (i.e., check relations or rules at increasing distances). The algorithm of each planner agent after it gets a startQ message for q_{curr} follows. Each agent checks whether it knows q_{curr} as a relation or a rule. If so, it checks the consistency of the parameters for each relation with the same name. If the agent knows the rule, it tests all its antecedents (ifs) (again as relations/rules). If the parameters in the relation are variables, and these variables already have values assigned that are consistent with the agents' instantiations, the agent assigns them to the variables. If the parameters in the relation are constants and these are consistent with the agents' instantiations, the agent considers this relation OK. If the parameters are variables and they do not appear in the current blackboard, the agent assigns them the agent's values. Finally for this case the agent sends a finished message. If q_{curr} was not known by the agent as a relation nor as a rule, the agent checks whether it can be partially established. The agent checks the relations and rules from the given level, and the other levels incrementally distant from this level until no level exists or a relation or a rule has been found that is OK. Then, the agent sends a partial message. In any other case the agent sends a failed message. The agent adds its current assignments to its blackboard and prints it for the user. The agent initializes a new blackboard for the solution of the next subquery.

5.1 Experiments

We analyze the implemented system in both a quantitative and qualitative manner. We first run the evolutionary algorithm (described in Section 2) on the documents belonging to the site of the Institute of Computer Science at the Hebrew University (whose root is located at www.cs.huji.ac.il) so that an organization of agents emerges. Given the parameter setting we chose, we derived a stable organization composed of (only) two agents. Currently the evolutionary algorithm leaves the documents that have not been assigned to any agents in a pool (called Free).[7]

We prepared the agents as explained in the second step in Section 4. First, an ontology file describing the specific domain was built [5].

Each relation has a level of usefulness attached. The true worth function for all states, representing all the implicit goals in which a user might be interested, is obviously very complicated to compute. We try, instead, to approximate it by using an induced function generated from the ontology. We also added rules, composed of a list of antecedent "if" terms and a consequence, the "then" term. Based on the specifications of SHOE, the if parts could consist of other relations. In our implementation we let these parts also be references to other rules. This will enable the agents to apply more complex operators.

Then, each agent's HTML files were embedded with semantic SHOE tags. Instead of reading the entire text, each time the agent needs to look for information it only has to check the data structures it holds containing the information described in SHOE. That is, the agent will only need to process structured and concise data.[8]

5.1.1 Qualitative Results

We ran the multiagent planning system for several queries (first without any rules, but with binary relations). Beyond the answer to the query, we also examine the additional information held by each agent in the organization. We also can evaluate the solution that the mediator agent chooses to show to the user.

The first, simple query posed to the system was `teacherAssistantOf(?x,DAI)`. Agent A_2 is the expert in DAI, Java and CAD, and A_1 is expert in theory and holds two general documents from the Institute of Computer Science. A_2 found a complete answer to the query (i.e., it assigned `Claudia Goldman` to `?x`). A_1 found a partial solution to the query that gives additional information to the user. The relation in the query has a level of usefulness of 8. A_1 found another relation at the same level; `cs.advisor(Dorit Aharonov, Michael BenOr)`.

The second query presented to the planner was `research(http://www.cs.huji.ac.il/,?x) & download(Tcl/Tk,?y)`. `research` is at level 9 and `download` is at level 7. Both agents knew an instantiation of the relation "research." One of these was chosen as the final answer. A_1 suggested assigning `http://www.cs.huji.ac.il/CS/labs.html` to `?x`, and A_2 suggested `http://www.cs.huji.ac.il/labs/dai`. These suggestions correspond to each of the agents' expertise. A_0 chose A_1's suggestion. A_2 gave the user more focused information as to where to look for DAI research in the Institute of Computer Science. `Tcl/Tk` is taught as part of the CAD course. A_2 could give a complete answer to the second subquery, assigning `http://www.cs.huji.ac.il/course/tibam` (tibam is the Hebrew name for CAD) to `?y`. A_1's knowledge has nothing in common with Tcl/Tk. A_1 could only find a partial answer concerning another relation at level 8 (distant by only 1 from the level of `download`). In this case the relation `advisor` was chosen, with parameters corresponding to the name of an advisor in theory and a PhD student of his.

The third query was `courseExplains(?x, Software Agents) & teacherOf(?y, Software Agents) & download(Java,?z)`. `courseExplains` and `teacherOf` were given a level of 8, and `download`, a level of 7. A_2's knowledge was more relevant to all of these subqueries than A_1's. Nevertheless, since both agents gave partial answers to all of these subqueries, A_0 chose A_1's suggestions. In this case, the user will get more benefit if it looks at the additional information chosen by A_2. A_1 suggests the relation `advisor` at level 8 with its parameters assigned to a theory advisor and his student. A_2 suggested two relations. The first was `givesCourse` at level 8, with the parameters `CS` and `DAI`. The second was `relatedCourse` that relates the page on readings in DAI to the DAI course page. The additional information given in A_2's blackboard can guide a user to other pages related to the subject of his original query.

After running the above experiments using only relations, we added three rules to the ontology [5]. The first rule states that a person is a member of a department X if he does research that is done in the department X. The second and third rules help in finding the e-mail address of a person. If he is a teaching assistant of a course, and the course has an e-mail address, then this address can be retrieved as the person's. If a person is a member of an organization, then the organization's e-mail can be returned as that of the person's.

[7] The files with the knowledge each agent holds are located under www.cs.huji.ac.il/~ clag/OPlan/agents2.1/Agent1.html and Agent2.html. All the files the agents link to should be suffixed with html.

[8] At the same location (www.cs.huji.ac.il/~ clag/OPlan/agents2.1/), agent1.html and agent2.html hold the list of files to which were attached the SHOE semantic tags. Although each of these files is an html file, Netscape does not parse SHOE syntax; it's necessary to look at these files as text to see the SHOE tags. All the files together with their semantics are also located at the same address.

In one experiment, we looked for `emailAddress(Claudia Goldman,?e)`. A_1 suggests a partial solution, that is the e-mail of the theory research group at the CS department. A_2 finds a final answer, since `Claudia Goldman` is the teaching assistant of DAI, and this course has her e-mail address.

In the second experiment, the query was email(D,?e). D is a member of the theory group at CS. A_1 knows the e-mail address of the Institute, and this is returned as D's e-mail (the organization e-mail webmaster@cs.huji.ac.il is returned). A_2 holds the same solution as before: the e-mail address of the TA of DAI; this is additional information. Given for example, that A_1's information was not sufficient, we could approach another person at the Institute in which we are interested, and ask for help about another member.

There is a tradeoff between the speed of the planning system, and the quality of the system's answer. In our current implementation, we prefer faster responses to more complete ones. Whenever a user has time constraints, he would rather look for an answer that is as complete as possible which can be achieved as fast as possible. In other words, a partial solution with a high level of usefulness will be preferred over a complete solution with lower usefulness.

5.1.2 Quantitative Results

It is known that planning is an exponential task [18]. The point we have been emphasizing in this paper is that having knowledge partitioned and organized reduces the work of planning agents. Although the work to be performed by each agent remains exponential, its magnitude is smaller. In [6] we present a detailed analysis of when it is worthwhile to use the agent organization-based approach instead of having a single agent that solves the same planning task by itself. We summarize these results below.

We will use the following notation: $Q = \bigwedge_{i=1}^{|Q|} q_i$ is the query posed by the user. All our computations were done assuming that all of the q_is are independent or, as in our implementation, whenever a variable gets a value this constrains the following q_is that need the same variable. We did not implement backtracking. We analyze the case in which backtracking is possible later in this section. R_O refers to the relations in the ontology file. We count every possible instantiation of each relation as one member in R_O. R_O is the total number of relations instantiated. x_i, $1 \leq x_i \leq R_O$, is the number of relations that agent A_i holds. s will denote the upper bound on the length of the solution of each q_i. We denote by A the upper bound on x_i such that $A < R_O$. We denote by γ the gain of implementing an organization instead of having a single agent solving a given query Q (that is, the numerator is the single-agent complexity, and the denominator is the agent-organization complexity). γ_1 represents this value when A_0 knows how to solve the assignment problem, and γ_2 expresses the value in the second case.

We analyze four cases where we distinguish between the cases when A_0 knows to solve the assignment problem, and whether the agents are allowed to backtrack:

1. A_0 **knows which agent can solve each one of the subqueries, the agents do not backtrack**
   $\gamma_1 = \frac{|Q| \times R_O^s}{x_1^s + \ldots + x_{|Q|}^s} \geq 1$ when $\forall i, x_i \leq R_O$.
   If all the agents hold $x_i = R_O$ relations, then there is no gain.
2. **All the agents try to solve each one of the subqueries, the agents do not backtrack**
   $\gamma_2 = \frac{|Q| \times R_O^s}{|Q| \times (x_1^s + \ldots + x_{OrgSize}^s)} = \frac{R_O^s}{x_1^s + \ldots + x_{OrgSize}^s}$
   (a) If at least one agent A_i holds $x_i = R_O$ then there is no gain in having the organization (i.e., $\gamma 2 < 1$).
   (b) if $\Sigma_{i=1}^{OrgSize} x_i \leq R_O$ *then* $\gamma_2 = \frac{R_O^s}{x_1^s + \ldots + x_{OrgSize}^s} \geq 1$
   (c) If $\Sigma_{i=1}^{OrgSize} x_i > R_O$ (i.e., different agents might know the same instantiations of the relations), and $A < \frac{R_O}{\sqrt[s]{OrgSize}}$, then $\gamma_2 \geq 1$. However, if $s = 1$ it is not worthwhile to apply the agent organization approach.
3. A_0 **knows which agent can solve each one of the subqueries, the agents do backtrack**
   $\gamma_1 = \frac{R_O^{s \times |Q|}}{x_1^{s \times |Q|} + \ldots + x_{|Q|}^{s \times |Q|}}$. If $\Sigma_{i=1}^{|Q|} x_i \leq R_O$ then $\gamma_1 \geq 1$. If $\Sigma_{i=1}^{|Q|} x_i > R_O$ then $\gamma_1 \geq 1$ when $A < \frac{R_O}{\sqrt[s \times |Q|]{|Q|}}$.
4. **All the agents try to solve each one of the subqueries, the agents do backtrack**
   $\gamma_2 = \frac{R_O^{s \times |Q|}}{|Q| \times (x_1^{s \times |Q|} + \ldots + x_{OrgSize}^{s \times |Q|})}$. If $|Q| \times \Sigma_{i=1}^{OrgSize} x_i \leq R_O$ then $\gamma_2 \geq 1$. If $|Q| \times \Sigma_{i=1}^{OrgSize} x_i > R_O$ then $\gamma_2 \geq 1$ when $A < \frac{R_O}{\sqrt[s \times |Q|]{|Q| \times OrgSize}}$.

6 Conclusions

We have proposed an organization-based approach for multiagent system design, describing an evolutionary algorithm that dynamically partitions an information space among agents. Each of these agents (whose number is also determined dynamically) assumes the "expertise" found in its part of the overall space, leading to greater efficiency when doing problem solving.

We focused on a particular subclass of domains, called Information Oriented Domains.

The model presented as a testbed was implemented and can be run on Web sites. First, it computes a matrix of distances among the documents at the site. Then, the evolutionary engine applies the rules designed to spawn an agent, to kill it, or to let it collect additional related documents. These rules were implemented along the lines of the rules of the Game of Life.

This is a general framework that organizes the agents into experts in a given domain. Therefore, the same organization can be helpful for solving many tasks related to the same domain (in contrast to the DAI organization related research that deals with task-dependent organizations, instead of information dependent organizations).

These organizations can be approached by users or other agents as catalogued libraries of information about a given domain, as a library of cases (in the case based reasoning context), or as a library of partial plans. Moreover, an agent organization can serve as a multiagent system for solving problems.

A specific and practical implementation of an organization multiagent planner for IODs has been presented. We also present results from experiments we ran with the planner, whose domain was HTML documents taken from the site of the Institute of Computer Science at the Hebrew University.

Semantics were added to the agent files using the SHOE language. An ontology file was introduced that describes the entities and the relations and rules that exist in the given world.

Agents were programmed to plan given their expertise accumulated during the organization's evolution, enriched with the semantic knowledge that was then added. Agents are capable of finding the most useful solution to a user's query. Agents in the organization also hold additional information that can benefit the user.

We analyzed the worth of the multiagent organization-based approach when the agents can backtrack, and when they cannot. We also distinguish between the case in which the solution to the assignment problem is known, and the case in which all agents work on each of the subqueries, comparing these results.

References

[1] K. Decker, V. Lesser, M.V. Nagendra Prasad, and T. Wagner. Macron:an architecture for multi-agent cooperative information gathering. In *Procs. of the CIKM workshop on Intelligent Information Agents*, Baltimore, Maryland, 1995.

[2] K. S. Decker and K. Sycara. Intelligent adaptive information agents. *Journal of Intelligent Information Systems*, 9:239–260, 1997.

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

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

[5] C. V. Goldman. A shoe-based ontology file for the institute of cs, 1998. http://www.cs.huji.ac.il/∼clag/OPlan/agents2.1/ontology.html.

[6] C. V. Goldman. *Multiagent Learning Systems*. PhD thesis, The Hebrew University, 1999.

[7] C. V. Goldman and J. S. Rosenschein. Evolving organizations of agents. Technical Report WS-97-03, AAAI, Providence, Rhode Island, 1997. Multiagent Learning Workshop at AAAI97.

[8] B. Grosz and S. Kraus. Collaborative plans for group activities. In *Procs. of IJCAI93*, pages 367–373, Chambery, France, 1993.

[9] J.L. Kolodner. *Case Based Reasoning*. Morgan Kaufmann, 1993.

[10] S. Luke, L. Spector, D. Rager, and J. Hendler. Ontology-based web agents. In *Proceedings of AA-97*, Marina del Rey, Feb. 1997.

[11] F. Menczer and R. K. Belew. Adaptive information agents in distributed textual environments. In *Proceedings of AA-98*, pages 157–164, Minneapolis, USA, May 1998.

[12] A. Moukas and P. Maes. Amalthea: an evolving multiagent information filtering and discovery system for the www. *Journal of Autonomous Agents and Multiagent Systems*, To appear.

[13] J. Von Neumann. *Theory of Self Reproducing Automata*. University of Illinois Press, 1966. Ed. and Completed by A. Burks.

[14] M. Nodine, B. Perry, and A. Unruh. Experience with the InfoSleuth agent architecture. In *Procs. of AAAI-98 Workshop on Software Tools for Developing Agents*, 1998.

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

[16] K. Sycara, K. Decker, A. Pannu, M. Williamson, and D. Zeng. Distributed intelligent agents. *IEEE Expert*, 11(6), 1996.

[17] S. Ulam. Random processes and transformations. In *Procs. of the International Congress on Mathematics*, 1952.

[18] D. E. Wilkins. *Practical Planning*. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1988.