A Cooperative Approach to Large Knowledge Based Systems

C. V. Ramamoorthy
S. Shekhar
Computer Science Division
University of California, Berkeley

Abstract
Conventional Expert Systems, based on single flat knowledge base, have fared well in solving a narrow range of problems. However their architectures do not extend to large applications like automated factories, which require multidisciplinary knowledge and which are geographically distributed. To solve multidisciplinary problems, a knowledge-based system must be able to (1) reason about the need for cooperation, (2) understand global knowledge to locate relevant expert systems and (3) select appropriate cooperation plans. Contemporary approaches like Blackboard NII 1986 Contract-net Smith contract 1989 and Distributed problem-solving Lesser Corkill 1983 explore alternative cooperation plans without any reasoning about the need for cooperation and understanding of global knowledge. We propose Cooperation models to characterize three essential decisions in the cooperation process. We devise a computational method to decide if an expert system has enough knowledge to solve a given problem or if it needs to consult with other expert systems. We propose Yellow pages technique to represent global knowledge and to select appropriate cooperation plans. Our approach lets expert systems autonomously resolve the three fundamental decisions in cooperation at runtime in contrast to contemporary approaches where the decisions are made at design time by the programmers.

1. Introduction

Although conventional expert systems are useful in solving certain problems, they do not have the ability to solve sizable problems, especially those requiring a team of people with multidisciplinary skills Expert systems like Mycin[4, 5], which are based on a single flat knowledge-base, have fared well in solving a narrow range of problems. However, their architectures do not extend well to large applications like Automated Factories[6, 7]. Presently, a set of expert systems and knowledge bases are used in different divisions (e.g. product design, machine control, inventory control, management, marketing) of a factory. The component systems have knowledge about individual divisions, which may be located in different parts of the country. These systems are often neither compatible nor linked via a network. The lack of integration makes it difficult to minimize the time in designing and marketing a new product[8]. For example one can not easily check if a new product design is reasonable under the constraints of material and tooling available with the factory. In general, present expert system architectures do not extend to problems which require multidisciplinary or geographically distributed knowledge.

There have been a number of attempts to extend expert systems in order to solve these problems: (i) Hierarchical knowledge-base as in Caduceus[9], (ii) Collection of knowledge-bases as in Genesis[10], and (iii) Cooperating expert systems.

The knowledge-base of Caduceus[9] is organized in a hierarchical fashion. The reasoning process determines the part of the knowledge-base relevant to the problem at hand before going into detailed reasoning. This approach improves the performance of expert systems and allows the use of a large knowledge-base. It does not, however, provide any help for geographically distributed applications or multidisciplinary applications. The task of understanding, debugging and maintaining such an expert system is quite difficult, because the system is large, complex and monolithic.

Another approach is to structure a large AI system as a collection of expert systems, each of which specializes in a specific domain of knowledge. The approach in Genesis[10], allows multiple expert systems to co-exist. The user can invoke any of the expert systems, and switch from one to another. This architecture can be used for multidisciplinary applications by creating distinct expert systems for each discipline of knowledge. In Genesis, however, the user has the burden of selecting the right expert systems to be invoked and to combine their results. This is tedious and difficult for end-users.

Integration of the collection of AI programs into a cooperating group of expert systems is needed, so that each expert system knows about other systems in its environ-
ment and can interact with them. Large applications which require knowledge from several disciplines may be structured as a collection of cooperating expert systems. This architecture also extends easily to geographically distributed applications since the expert systems in the collection can easily be distributed over a network of machines.

Our Coop project[11] provides the primary ingredients of Cooperating Expert Systems (CES) including- (a) the ability to introspect and decide whether one can solve a certain problem or needs help, (b) the ability to organize a large body of knowledge into domains [12] for efficient access to relevant knowledge and reasoning methods, and (c) the ability to coordinate and control the collaborations among several expert systems With the addition of the ability of cooperation at run-time, expert systems will be able to help in large applications like automated factories.

This paper discusses the cooperating expert systems in a distributed environment. We identify the important issues in design and implementation of cooperating expert systems applications on a network. Section 2 presents a taxonomy of AI system structures and bring out the issues in cooperation. Section 3 lists the previous approaches and section 4 and 5 defines our approach to cooperation among intelligent programs. Section 6 describe the planned tools to resolve AI issues in cooperation among expert systems. We finally discuss our approaches in building the Coop, an environment we have developed for CES applications, and some of the unresolved issues.

2. Cooperating Expert Systems

The difference between current AI systems and cooperating expert systems is illustrated in Figure 1. Traditionally, an AI program runs as a single process. This process may run on a single processor (class-I), or it may use several processors (class-IV) with the help of a network-transparent operating system. Alternatively, a large AI system may have many processes. These processes may run on a central processor (class-II) or be distributed over several processors (class-III).

Distributed Cooperating Expert Systems (Class-III) provide the most general structure for knowledge-based systems. We focus on class-III structure for CES in this proposal.

2.1. Required Capabilities for Cooperation

Consider an example from the aerospace domain to bring out the critical decisions in cooperation. Assume the commander off a space station observes an anomaly in the thermal environment control system of the space station. He examines the symptoms from sensors and tries to identify the problem. The station commander may diagnose the anomaly by himself or he may decide to consult a system specialist on board or on ground. The decision to consult a specialist is a decision to cooperate, i.e., he needs cooperation from others to solve the problem. Then the station commander has to select an appropriate specialist. This step requires the decision: with whom should system cooperate? Next, the station commander has to decide on the mode of consultation with the specialist and discuss the problem. This step in general requires a decision about how to cooperate. The last issue in cooperation concerns the services that can facilitate cooperation. For example, the electronic directory off specialist on board can help in locating appropriate specialists on ground.

One can model station commander and specialists as individual expert systems or agents in the AI world. These agents should be capable of resolving the cooperation issues without human intervention. Then the three major decisions in cooperation are:

1. Decision to cooperate: Each agent must know the limitations of its knowledge and decide when help is needed from other agents to solve the problem at hand. We call this decision the introspection problem.

<table>
<thead>
<tr>
<th>Software</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unprocessor</td>
</tr>
<tr>
<td>One Process</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>Traditional AI Systems e.g. MYCN</td>
</tr>
<tr>
<td>Multiple processes</td>
<td></td>
</tr>
<tr>
<td>II</td>
<td>Cooperating Expert Systems e.g. HEARSAY-II</td>
</tr>
</tbody>
</table>

Figure 1 Classification of AI systems
2. With Whom to cooperate: After realizing the need to consult other experts, the agent needs to find an appropriate expert which can help it. Also, the agent has to figure out a way to convey its question to the expert and interpret the advice from the expert. This requires a global knowledge about the available AI systems in the environment, their special expertise, and their knowledge representations. The **global knowledge** provides a method to specify the information about the roles of various agents and their knowledge.

3. How to cooperate: Since multiple agents are cooperating with each other to solve a particular problem, it is necessary to coordinate and coordinate their interaction and cooperation. A **cooperation policy** specifies the division of work among different members of a team and a method to control the team.

3. Alternate Approaches

One possible approach to cooperative problem solving is based on **blackboard paradigm**[1, 13]. The current hypothesis is made available in a shared memory called "blackboard". Applicable rules from each knowledge base are brought into the blackboard, scheduled and executed. This provides a way to use multi-disciplinary knowledge to solve speech recognition problem. However, the shared blackboard cannot easily be distributed over the network of computers. HEARSAY-II speech recognition system[14] can be considered to be the first system using the C.E.S. architecture. This system has a set of knowledge bases - one for each aspect (phonemes, syllables, words) of speech recognition.

Another approach to cooperative problem solving focuses on utilizing a network of computers. A collection of expert systems to track moving vehicles[3] experimented with various ways of dividing up the problem, and degree of dependence of one system on other systems to get good performance. This approach provides a way for programmers to design multi-disciplinary AI systems for problems, which can decompose into independent subproblems. This approach does not provide a model of cooperation and has extremely limited global knowledge. This approach does not help if there are subtle interaction (e.g. resource contention, dependency) between the subproblems.

Contract net protocol[2] provides still another paradigm to cooperative problem solving in distributed environment. In this paradigm, the AI system needing help from others divides up the problem and invites bids from other systems to solve the sub-problems. The solutions returned by various systems are finally combined. Section D.4 describes the mile-stone projects in the development of cooperating systems architecture.

Our approach to cooperating expert systems is based on cooperation models to identify the critical decisions in cooperation. This approach utilizes the reasoning capability of knowledge based systems to reason about the need of cooperation, to locate relevant expert system and to select appropriate cooperation plans (e.g. contract net). This approach lets expert systems decide at runtime to seek cooperation from other specialists in contrast to most distributed AI models where the decision is made at design time. It provides a novel computational method to decide if an expert system has enough knowledge to solve a given problem or if it needs to consult other expert systems. **Annotated prolog**, an extension of Prolog language can be used to support introspective expert systems. **Alternative Cooperation models** and tools are proposed for supporting fundamental decisions in cooperation.

4. Our Approach

Cooperation decision based on the lack of sufficient knowledge has two dimensions: breadth and depth. Breadth of knowledge refers to how many domains an expert knows about, whereas the depth of knowledge refers to the range of problems an expert can solve within its domains of expertise.

In many domains (e.g. fault diagnosis) the knowledge is structured hierarchically. The specialists have deeper knowledge about one set of malfunctions, whereas a generalist possesses shallow knowledge about several sets of system mal-functions. We use confidence factors and support pairs [15] as a metric of the depth of knowledge in this domain. Each fact and rule in the knowledge base is annotated with a support pair to denote the fraction of relevant cases where it gives correct results. Typically specialists possess detailed rules about few system malfunctions with high confidence factors, and the general practitioner possess general rules about several system anomalies with lower confidence factors. We propose to use the Support Logic Programming (SLP)[15] to give the power of
depth-related introspection to the agents. This problemsolving process produces a confidence factor along with the solution. If the confidence factor is low, the solution is rejected and one seeks cooperation from other agents in the environment.

SLP is implemented as an extension of Prolog. A query is answered with the Prolog style of procedural proof, but a support pair is computed and associated with the final answer. Each fact and rule in Prolog is annotated with a support pair \([S_1, S_2]\), representing the uncertainty about the belief in the hypothesis. \(S_1\) represents the degree of support for the hypothesis and \(1-S_2\) the degree of support for the negation of the hypothesis. \(S_2\) represents the degree to which the hypothesis is necessarily supported and \(S_1\) as the possible degree to which the hypothesis may get supported. In general, \(S_1 \times (1-S_2) \geq 1\) is satisfied. If \(1-S_2 \times (1-S_1) = S_2 - S_1\) represents the ignorance or non-commitment towards disjunction of hypothesis and its negation. This gives a way to represent the degree of ignorance or uncertainty about a hypothesis, not possible in the probabilistic framework. A true fact has a support pair \([1,1]\), a false fact \([0,0]\), and a completely uncertain fact, \([0,1]\).

A support logic program is a set of program clauses of the form

\[ A \leftarrow B_1, B_2, \ldots, B_n : [S_1, S_2] \]

where \(A, B_1, B_2, \ldots, B_n\) are atoms. This may be a Prolog clause [15] with the addition of the support pair \([S_1, S_2]\). A procedural interpretation for program clauses is as follows: For each assignment of variables, if \(B_1, B_2, \ldots, B_n\) are all true, then the hypothesis \(A\) is necessarily supported to the degree \(S_1\) and \(\neg A\) to the degree \(S_2\). The interpretation of program clauses for the degenerate case of program clauses without one of (antecedent, consequent) can be easily derived.

The combination of evidence for a hypothesis can be done by extending the Prolog semantics. Since a theorem in Prolog can be proved in more than one way, there are multiple proof paths associated with the theorem. In the support logic programming, each proof path for a hypothesis corresponds to a certain evidence supporting the theorem. The support pairs of facts and rules used in the proof path are combined to give the support pair for the theorem. The rules of combination for support pairs are derived from a model of interpretation.

We propose computational ways to resolve the fundamental decisions in cooperations using proposed techniques in following paragraphs:

1. **Decision to Cooperate:** The introspection problem can be solved in the support logic programming as follows. The expert is assumed to reason by logic programming. There is a support pair associated with each rule and fact in the knowledge base. Given a query, the expert system associates a support pair of \([0,1]\) to represent complete uncertainty or initial ignorance about the truth-value of query before the reasoning. Then, the Prolog interpreter tries to prove or disprove the query and in the process generates a proof path and variable bindings. In this process the interpreter also computes a support pair for the query using the support pairs of the rules and facts used in the proof path. If the derived supports for the answers to the query indicates high uncertainty or ignorance (for example, \(S_1 - S_2 > 0.7\)), then the expert system decides to seek cooperation.

2. **Locating relevant expertise:** To locate the specialists to consult, we use yellow pages. Yellow Pages map from the attribute of a problem to the identity of an expert system, which has the relevant expertise and map from the expert system identity to the network address of the system, and help in establishing communications with the system. Both services include databases and search algorithms. The search algorithm in Yellow Pages, computes a mapping from the attributes of problem to a problem domain, and then the mapping from the domain to the identity of the expert system in the domain. Since this method is application dependent, Yellow pages also provide search heuristics for mapping from problem domain to relevant of expert systems.

3. **Cooperation Plan:** A library of routines is planned to support popular cooperation plans like contract net and blackboard. For example, to support the Contract Net policy, we provide routines like - broadcast contract, bidsubmission, award contract, and select-best-bid. These libraries would be linked to an AI language, e.g. prolog. Various expert systems would use the library to implement
specific cooperation policies they choose to use for a given problem.

6. The Coop Develop Environment

Since the implementation of cooperation policies is not trivial, it is desirable to have tools supporting the major decisions in the development of CES applications. A generic blackboard development system (GBB), a multi-agent computing environment (MACE), and the Coop environment fall in this category.

GBB[17] is a tool for constructing Blackboard-based AI systems (class-II). A blackboard is a shared object among multiple agents. Each agent stores its hypothesis on the blackboard. Then, applicable rules are brought into the blackboard, scheduled, and executed. GBB supports efficient management (insertion and retrieval) of blackboard objects. From the specifications of the structure of the blackboard and its access methods, the system generates an application-specific blackboard database kernel which is tailored to the application system. While GBB only supports blackboard-based class-II systems, MACE can support the development of class-II systems as well as class-III systems. MACE provides basic facilities for creation of multiple agents, knowledge representation for the world to which the agent belongs (e.g., roles of other agents and their locations), and a description language to specify the specialties of each agent. The Coop environment is similar to MACE except that MACE provides basic facilities like communication implicitly while the Coop environment provides them explicitly with servers. The use of servers enhances the flexibility in the system design and improves the scalability of the system.

The Coop environment at Berkeley is aimed at supporting several decisions in cooperating expert system application design in an application independent manner. It supports the decision to seek cooperation via annotated Prolog. It provides a service (yellow pages) to locate experts with knowledge about a domain. However, the application has to figure out the domain of the problem at hand. The environment provides services for communication, synchronization and global state estimation for utilizing a network of machines effectively. The application has to divide up the work and decide the cooperation policy using the primitives provided. The application also has to resolve the possible differences in the knowledge representation used by different expert systems. With the environment, one can explore various cooperation strategies for specific applications and develop prototype systems quickly.

The Coop environment structures the society of agents as a set of organizations providing essential infrastructure and services needed by individual expert systems at runtime or needed by an application programmer in designing cooperating expert systems. An organization is a group of agents providing a unique service. The Coop environment provides a set of services, and mechanisms to create new organizations and to publicize their services to the community of expert systems so that more services can be added. Initial services provided in the Coop environment include a service builder, yellow pages, network event manager, synchronization and coordination primitives.

<table>
<thead>
<tr>
<th>Specification language &amp; interpreter (introspection, cooperation,...)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Services</strong></td>
</tr>
<tr>
<td>Yellow pages and White pages</td>
</tr>
<tr>
<td>Interface between Services and Unix (Naming, clock synchronization, status maintenance)</td>
</tr>
<tr>
<td>UNIX 4.2 on a SUN cluster</td>
</tr>
</tbody>
</table>

Figure 2 The Coop Environment for cooperating expert systems

As shown in Figure 2, we provide the following set of services:

(a) **Annotated Prolog for Introspection**

Each rule in the Prolog rule base is annotated with a pair of numbers which quantify the confidence in that rule. This pair of numbers provide the "support" for the rule. We have extended the 'C'-prolog to represent support pairs and to combine the pairs in the course of theorem proving and compute the support for the final outcome. When the...
support for a solution is less than a certain threshold, the expert system decides to seek cooperation. We also added network communication primitives to Prolog for distributed computing. The extended interpreter is about 10,000 lines of C code.

(b) Yellow pages and White pages

Agents need a way to locate the specialists to consult. The yellow and white pages assist in this task. Yellow Pages map from the attribute of a problem to the identity of an expert system, which has the relevant expertise. White Pages map from the expert system identity to the network address of the system, and helps in establishing communications with the system. Both services include databases and search algorithms. The search algorithm in Yellow Pages, computes a mapping from the attributes of problem to a problem domain, and then the mapping from the domain to the the identity of the expert system in the domain. Since this method is application dependent, Yel-

lowpages allow the clients to define their own search heuristics for mapping from problem domain to relevant of expert systems.

(c) Network Event Manager and Interface with Unix:

Unix provides a basic communication mechanism but does not provide a naming service. We added a Naming server (White Pages) to the unix IPC package. We provide a status maintenance service to collect global state of the network. This server collects local status of individual machines on the network and merges them. It supports queries about the values of state variables (e.g. load average, up-time of machines etc.) describing the network status. The Network Event Manager provides a network wide demon. It monitors the status database and detects the occurrence of different events in the event table. Once an event occurs it takes the appropriate action. The interface with Unix and Network Event Manager 10,000 lines of C code.

(c) Library of Cooperation Policies

A library of routines is provided to support four cooperation policies and mechanisms, discussed in section 5. For example, to support the Contract Net policy, we provide routines like - broadcast contract, bid-submission, award contract, and select-best-bid. These libraries can be linked to any language, e.g. prolog. Various expert systems can use the library to implement specific cooperation policies they choose to use for a given problem.

(c) Worm mechanism

This service helps in initiating, controlling and terminating a group of processes on the SUN-cluster connected by ethernet. The Worm program uses the UNIX command ‘rsh’ to start a process group on remote machines and uses the network event manager to monitor the process group. It provides multi-cast facility for communication in the process group. We also support protocols for agreement and consensus problems among the group of processes. It also supports fault tolerance process groups by automatically restarting process on the failed processors. The worm program consists of about 5,000 lines of C and lisp code.

7. Conclusions and Future Work

Expert systems have been successful in solving a wide range of specialized problems, requiring high levels of human intelligence. They have successfully been used in commercial applications. They must now be useful for larger problems which are multidisciplinary and often geographically distributed.

The rule based architecture of present expert systems do not scale up to large distributed applications. Hierarchi-

cal knowledge bases and Cooperating expert systems are alternate ways to solve the problem of scalability. We find that the cooperate expert systems provide a comprehensive model for structuring large and distributed knowledge based systems.

Modeling and implementation of CES raise several interesting issues in AI and in distributed systems. The AI issues include the model of cooperation, adding ability of introspection to expert systems, locating relevant expertise for a problem, and setting up a cooperation plan to solve the plan. Some of these issues are resolved by tools in Coop environment, while other issues remain open. To fully automate cooperation, AI systems need ways to choose cooperation policy at run-time.

Distributed system issues relate to efficient management of a large cooperating expert system as a set of interacting processes and knowledge bases scattered over a large inter-network. This requires facilities for communica-
tions, clock synchronization, global state maintenance, event management and run-time support for processes over the network. These problems become difficult when different parts of networks follow different conventions for network management. While communication across machines on different networks has been made possible by TCP-IP, there is no clock-synchronization algorithm available for such environment. We are still experimenting with the mechanisms for initiating a group of processes on the internetwork (Worm). These are not trouble-free as illustrated by the catastrophic virus infections in major networks, which occurred across US computing facilities in November 1988. We need to work out an agreement between foreign worm processes and the host operating programs running on a common network. We have not explored interactions between different worm processes, for providing up-to-date status information, and monitoring processes in real-time. Thus cooperating expert systems provide several interesting challenges to distributed computing.

8. Acknowledgement

The Coop project has been supported by Air Force (RADC), National Science Foundation and Micro grants from the state of California. Y.-F. Chen and A. Prakash designed and implemented the Network Event Manager and Ramachandran implemented Annotated Prolog and Service Builder. We have benefited from discussions with Professors L. A. Zadeh, S. Russell, R. J. Lauber, and R. Ashany. Finally we thank the anonymous referees and our colleagues, J. Srivastava, A. Purshottam, A. Hung, M.T.Raghunath and V. Garg for their valuable comments, which greatly improved the readability and organization of this article.

References
4. Feigenbaum, E., "Key-note speech," Nat'l Conf. on AI (AAAI), 1983, Minneapolis, MN.