Learning When to Collaborate among Learning Agents

Santiago Ontañón and Enric Plaza

IIIA, Artificial Intelligence Research Institute
CSIC, Spanish Council for Scientific Research
Campus UAB, 08193 Bellaterra, Catalonia (Spain).
{santi,enric}@iiia.csic.es, http://www.iiia.csic.es

Abstract. Multiagent systems offer a new paradigm where learning techniques can be useful. We focus on the application of lazy learning to multiagent systems where each agents learns individually and also learns when to cooperate in order to improve its performance. We show some experiments in which CBR agents use an adapted version of LID (Lazy Induction of Descriptions), a CBR method for classification. We discuss a collaboration policy (called Bounded Counsel) among agents that improves the agents' performance with respect to their isolated performance. Later, we use decision tree induction and discretization techniques to learn how to tune the Bounded Counsel policy to a specific multiagent system—preserving always the individual autonomy of agents and the privacy of their case-bases. Empirical results concerning accuracy, cost, and robustness with respect to number of agents and case base size are presented. Moreover, comparisons with the Committee collaboration policy (where all agents collaborate always) are also presented.

1 Introduction

Multiagent systems offer a new paradigm to organize AI applications. Our goal is to develop techniques to integrate lazy learning into applications that are developed as multiagent systems. Learning is a capability that together with autonomy is always defined as a feature needed for full-fledged agents. Lazy learning offers the multiagent systems paradigm the capability of autonomously learning from experience. In this paper we present a framework for collaboration among agents that use CBR and some experiments illustrating the framework.

A distributed approach for lazy learning in agents that use CBR (case-based reasoning) makes sense in different scenarios. Our purpose in this paper is to present a multiagent system approach for distributed case bases that can support these different scenarios. A first scenario is one where cases themselves are owned by different partners or organizations. These organizations can consider their cases as assets and they may not be willing to give them to a centralized "case repository" where CBR can be used. In our approach each organization keeps their private cases while providing a CBR agent that works with them. Moreover, the agents can collaborate with other agents if they keep the case

L. De Raedt and P. Flach (Eds.): ECML 2001, LNAI 2167, pp. 394–405, 2001.

privacy intact and they can improve their performance by cooperating. Another scenario involves scalability: it might be impractical to have a centralized case base when the data is too big.

Our research focuses on the scenario of separate case bases that we want to use in a decentralized fashion by means of a multiagent system, that is to say a collection of CBR agents that manage individual case bases and can communicate (and collaborate) with other CBR agents. In this paper we focus on a collaboration policy (*Bounded Counsel* policy) that improve the individual performance of CBR agents without compromising the agent's autonomy and the privacy of the case bases. We also present later the *Committee* policy for comparison purposes. These collaboration policies are a refinement of the general multiagent scenario of *Cooperative CBR* proposed in [10]. The collaboration policies presented here are strategies that CBR agents can follow to improve their individual performance in this framework.

The structure of the paper is as follows. Section 2 presents the collaboration policies the CBR agents can follow to improve their performance cooperating with other agents in a multiagent system. Then, section 3 presents the CBR method that the agents use in our current experiments. Section 4 presents the proactive learning process that allows the agents to generate the examples that characterize the collaboration states in the multiagent system and then learn to tune their individual collaboration policies. The experiments themselves are explained in section 5. The paper closes with related work and conclusion sections.

2 Policies for Cooperative CBR

A multiagent CBR ($\mathcal{M}$AC) system $\mathcal{M} = \{(A_i, C_i)\}_{i=1...n}$ is composed on n agents, where each agent A_i has a case base C_i. In the experiments reported here we assume the case bases are disjunct ($\forall A_i, A_j \in \mathcal{M}\mathsf{AC} : C_i \cap C_j = \emptyset$), i.e. there is no case shared by two agent's case bases. This is just an experimental option and not a restriction on our model. In this framework we restrict ourselves to analytical tasks, i.e. tasks (like classification) where the solution is achieved by selecting from an enumerated set of solutions $K = \{S_1 \dots S_K\}$. A case base $C_i = \{(P_j, S_k)\}_{j=1...N}$ is a collection of pairs problem/solution.

When an agent A_i asks another agent A_j help to solve a problem the interaction protocol is as follows. First, A_i sends a problem description P to A_j. Second, after A_j has tried to solve P using its case base C_j, it sends back a message that is either `:sorry` (if it cannot solve P) or a solution endorsement record (SER). A SER has the form $\langle\{(S_k, E_k^j)\}, P, A_j\rangle$, where the collection of *endorsing pairs* (S_k, E_k^j) mean that the agent A_j has found E_k^j cases in case base C_j endorsing solution S_k—i.e. there are a number E_k^j of cases that are relevant (similar) for endorsing S_k as a solution for P. Each agent A_j is free to send one or more endorsing pairs in a SER record.

Before presenting the two policies for cooperative CBR, *Committee* and *Bounded Counsel* policies, we will introduce the voting mechanism.

2.1 Voting Scheme

The voting scheme defines the mechanism by which an agent reaches an aggregate solution from a collection of SERs coming from other agents. The principle behind the voting scheme is that the agents vote for solution classes depending on the number of cases they found endorsing those classes. However, we want to prevent an agent having an unbounded number of votes. Thus, we will define a normalization function so that each agent has one vote that can be for a unique solution class or fractionally assigned to a number of classes depending on the number of endorsing cases.

Formally, let $\mathcal{A}^t$ the set of agents that have submitted their SERs to agent A_i for problem P. We will consider that $A_i \in \mathcal{A}^t$ and the result of A_i trying to solve P is also reified as a SER. The vote of an agent $A_j \in \mathcal{A}^t$ for class S_k is

$$Vote(S_k, A_j) = \frac{E_k^j}{c + \sum_{r=1...K} E_r^j}$$

where c is a constant that on our experiments is set to 1. It is easy to see that an agent can cast a fractional vote that is always less than 1. Aggregating the votes from different agents for a class S_k we have ballot

$$Ballot^t(S_k, \mathcal{A}^t) = \sum_{A_j \in \mathcal{A}^t} Vote(S_k, A_j)$$

and therefore the winning solution class is the class with more votes in total, i.e.

$$Sol^t(P, \mathcal{A}^t) = arg \max_{k=1...K} Ballot(S_k, \mathcal{A}^t)$$

. We will show now two collaboration policies that use this voting scheme.

2.2 Committee Policy

In this collaboration policy the agents member of a $\mathcal{M}$AC system $\mathcal{M}$ are viewed as a committee. An agent A_i that has to solve a problem P, sends it to all the other agents in $\mathcal{M}$. Each agent A_j that has received P sends a solution endorsement record $\langle\{(S_k, E_k^j)\}, P, A_j\rangle$ to A_i. The initiating agent A_i uses the voting scheme above upon all SERs, i.e. its own SER and the SERs of all the other agents in the multiagent system. The final solution is the class with maximum number of votes.

2.3 Bounded Counsel Policy

In this policy the agents member of a $\mathcal{M}$AC system $\mathcal{M}$ try first to solve the problems they receive by themselves. Thus, if agent A_i receives a problem P and finds a solution that is satisfactory according to a *termination check* predicate, the solution found is the final solution. However, when an agent A_i assesses that

its own solution is not reliable, the Bounded Counsel Policy tries to minimize the number of questions asked to other agents in $\mathcal{M}$. Specifically, agent A_i asks counsel only to one agent, say agent A_j. When the answer of A_j arrives the agent A_i uses the termination check. If the termination check is true the result of the voting scheme is the global result, otherwise A_i asks counsel to another agent—if there is one left to ask, if not the process terminates and the voting scheme determines the global solution.

The termination check works, at any point in time t of the Bounded Counsel Policy process, upon the collection of solution endorsement records (SER) received by the initiating agent A_i at time t. Using the same voting scheme as before, agent A_i has at any point in time t a plausible solution given by the winner class of the votes cast so far. Let V^t_{max} be the votes cast for the current plausible solution, $V^t_{max} = Ballot^t(Sol^t(P, \mathcal{A}^t), \mathcal{A}^t)$ and $V^t_r = \left(\sum_{S_k \in K} Ballot(S_k, \mathcal{A}^t)\right) - V^t_{max}$. The termination check is a boolean function $TermCheck(P, \mathcal{A}^t)$ that determines whether there is enough difference between the majority votes and the rest to stop and obtain a final solution. In the experiments reported here the termination check function (applied when there are votes for more than one solution class) is the following

$$TermCheck(P, \mathcal{A}^t) = \frac{V^t_{max}}{V^t_r} \geq \eta$$

i.e. it checks whether the majority vote V^t_{max} is η times bigger than the rest of the ballots. After termination the global solution is the class with maximum number of votes at that time.

Later in §4 we will show how the *TermCheck* predicate can be learnt by each individual agent. The results of using this "fixed" *TermCheck* predicate (with value $\eta = 3$) will be compared with the results of the learnt predicate in §5.

The collaboration policies described here have been implemented on the Noos Agent Platform [8]. NAP consists of Noos, a representation language with support for case management and retrieval [1], and FIPA-compliant utilities for agent interaction. A multiagent system in NAP consists on the individual agents capabilities (like CBR) plus a specification of the agent roles and interaction protocols in the framework of agent-mediated institutions [8]. Cases are represented as feature terms in Noos and the next section introduces the CBR method used in our CBR agents.

3 Case-Based Reasoning Agents

In this section we present LID, the CBR method used by agents. LID (Lazy Induction of Descriptions) builds a symbolic description $\mathcal{D}_P$ for a problem P to one or more cases in the case base [3]. In this framework, cases are structured and they are represented in the formalism of feature terms and symbolic descriptions are also built as generalizations [2]. We can consider $\mathcal{D}_P$ as a *similitude term* [9], i.e. a symbolic description of the similarities between a problem P and the

retrieved cases $\mathcal{S}_{\mathcal{D}_P}$. Also notice that a new similitude term is generated for each new problem.

An agent has a case base $C_i = \{(P_j, S_k)\}_{j=1...N_i}$ of classified cases that is used by LID. In order to classify a problem P in one of those classes, LID builds a description $\mathcal{D}_P$ such that

- $\mathcal{D}_P$ is a partial description of P, i.e. $\mathcal{D}_P \sqsubseteq P$ ($\mathcal{D}_P$ subsumes P).
- $\mathcal{D}_P$ contains the most relevant features of P.
- $\mathcal{D}_P$ induces a subset of the case base that satisfies that description: $\mathcal{S}_{\mathcal{D}_P} = \{(P_j, S_k) \in C_i | \mathcal{D}_P \sqsubseteq P_j\}$; we call $\mathcal{S}_{\mathcal{D}_P}$ the *discriminatory set* of $\mathcal{D}_P$.

LID uses a top-down heuristic strategy to build the description $\mathcal{D}_P$. LID uses an heuristic to determine which of the features present in P are more relevant for the purpose of classifying P correctly into a solution class in K. LID uses an heuristic[1] that determines which feature f is more discriminating with respect to the solution classes K. Then it adds f to $\mathcal{D}_P$ with the value $P.f = v$ (the value that P has in feature f). Then LID only considers the subset of the case base defined by the discriminatory set $\mathcal{S}_{\mathcal{D}_P}$ —the other cases are discarded. Using the new case base $\mathcal{S}_{\mathcal{D}_P}$ LID uses the heuristic to determine which of the remaining features present in P is most discriminatory and adds it to $\mathcal{D}_P$. This process continues adding features to $\mathcal{D}_P$ until the termination criterion is met.

The termination criterion is met a) if all cases in $\mathcal{S}_{\mathcal{D}_P}$ are classified into a unique solution class S_k or b) adding further features of P into $\mathcal{D}_P$ does not reduce the discriminatory set $\mathcal{S}_{\mathcal{D}_P}$ to a set that has a unique solution class S_k.

When the termination is due to the second condition it means that the retrieved cases belong to more than one solution class, say problem P may belong to a subset of solution classes ($S_P \subset K$). The answer of LID is the following:

- the solution to problem P is one of the classes $S_k \in S_P$,
- the explanation of solution classes S_P is that problem P satisfies the description $\mathcal{D}_P$,
- there are a number of cases endorsing each solution class $S_k \in S_P$ —namely the cases in $\mathcal{S}_{\mathcal{D}_P}$ with solution S_k. All this cases have description $\mathcal{D}_P$, in common with P.

In the framework of a single agent the multiplicity solutions can be resolved by adopting a majority criterion and then the CBR system gives as solution the solution class $S_k \in K$ with more number of endorsing cases. In the framework of multiagent CBR system the multiplicity solutions is managed by the cooperation policies explained in §2.

4 Proactive Learning

We have seen that in the Bounded Counsel policy the agents need a definition for the termination check predicate. In the section 2.3 we have defined the *Term-*

[1] See [3] for a full explanation and evaluation of LID. The heuristic used is the RLM distance [6], also used in [2] as a heuristic to select the most discriminatory features.

Check predicate with a user defined parameter η, but probably a better approach is to let each agent to learn its own termination check predicates.

We are going to present an approach where each agent will take actions in order to obtain examples from which to learn its individual *TermCheck* predicate. A new stage is needed before the agents are ready to cooperate. In this stage, called the *proactive learning stage*, the agents will actively obtain the experience they need (a training set) through sending problems to some other agents and evaluating their results. From this experience each agent will learn a concrete definition of *TermCheck* (each agent learning it from its own training set). Specifically, each agent will learn a decision tree that will be used to assess when to terminate.

4.1 Defining the Examples

In order to learn the *TermCheck* predicate the agents need experience on the situations where *TermCheck* would be applicable. That is to say, each agent wants to learn when the result of the voting scheme will lead to the correct solution depending on the collection of SERs (solution endorsement records) that take part in the voting process.

Let's define the situations where *TermCheck* is applicable: Given an agent A_i that wants to solve a problem P, and at a time t has asked counsel to a subset of agents $\mathcal{A}^t$. A_i will have a set R_P^t of known SERs for the problem P at time t, that includes all the SERs received from the agents in $\mathcal{A}^t$ and the SER obtained by trying to solve the problem by A_i. The agent A_i can obtain a winning class with the voting scheme of section 2.1. The termination check has to predict if $Sol^t(P, \mathcal{A}^t)$ is the correct solution (and thus A_i won't need the counsel of more agents) or not.

We will call this a *voting situation*, and it is defined by the set of endorsing pairs $E_P^t = \{(S_k, E_k^j)\}$ that take part in the voting process. We will characterize a voting situation E_P^t by several attributes: V_{max}^t, the votes for the most voted solution, V_r^t, the votes for the rest of solutions, and $\rho = V_{max}^t/(V_{max}^t + V_r^t)$, the ratio between the most voted solution and the total number of votes.

We will use these three attributes to define our examples. Specifically, a v-example is defined by the three attributes above $v = \langle V_{max}^t, V_r^t, \rho \rangle$. Each v-example belongs to the positive class $(+)$ when $Sol^t(P, \mathcal{A}^t)$, the most voted solution, is equal to the correct solution —otherwise it belongs to the negative class $(-)$. In this way we have a classification problem with two classes: $(+, \langle V_{max}^t, V_r^t, \rho \rangle)$ and $(-, \langle V_{max}^t, V_r^t, \rho \rangle)$. In other words, a positive v-example characterizes a voting situation where there is no need of asking counsel to more agents and a negative v-example characterizes a voting situation where the agent do need the counsel of more agents.

4.2 Obtaining the Training Examples

Since every A_i has a case-base (collection of problems with known solution), A_i can obtain v-examples of voting situations from which to learn the termination

check. Sending those problems to the other agents an agent A_i can then assess the correctness of the voting processes derived from the SERs received from those agents. Thus, an agent A_i obtains a training set for *TermCheck* as follows:

1. Choose a subset B_i of cases from its own case-base $B_i \subseteq C_i$.
2. For each problem P in B_i
3. A_i sends P to a subset $\mathcal{A}^S$ of the other agents.
4. A_i solves P by itself by a *leave-one-out* method, i.e. it solves P using $C_i - P$ as case-base.
5. With the set R_P^S of SERs obtained in steps 3 and 4, A_i builds υ-examples of voting situations.

Note that from the collection R_P^S of SERs obtained in step 5 we can build more than one υ-example. In fact we can build a υ-example for any possible non empty subset $R_P^E \subseteq R_P^S$. Thus, step 5 is decomposed in 3 substeps:

1. Choose a collection $\mathcal{R}$ of non empty subsets of R_P^S, i.e. $\mathcal{R} \subseteq \mathbb{P}(R_P^S)$
2. For each voting situation $R_P^x \in \mathcal{R}$ let $\upsilon^x = \langle V_{max}^x, V_r^x, \rho \rangle$ be the example characterizing that situation.
3. If the most voted solution class is the correct class, build a positive example $(+, \langle V_{max}^x, V_r^x, \rho \rangle)$ otherwise build a negative example $(-, \langle V_{max}^x, V_r^x, \rho \rangle)$.

The collection $\mathcal{R}$ chosen in step 1 depends on the number of agents involved. In our experiments we have chosen collections that amount to generate a number of approximately 5,000 υ-examples. The result of this process on all $P \in B_i$ is a collection of examples that form the training set Υ_i for learning *TermCheck*.

4.3 Learning the Termination Check

Once an agent A_i has enough υ-examples, it can learn a good *TermCheck* predicate. In our experiments we have used all cases $B_i = C_i$ and all agents $\mathcal{A}^S = \{A_1, \ldots A_n\}$. The agents learn *TermCheck* using a decision tree algorithm with a discretization technique for the numeric attributes.

To build the decision tree T_i, each agent A_i does the following with its own training set Υ_i:

1. For each attribute $a \in \{V_c, V_r, \rho\}$ find the best cut point κ (in the sense of maximizing the information gain) that divides Υ_i in two subsets, one with the examples that have $Value(a) < \kappa$ and another with the examples that have $Value(a) \geq \kappa$.
2. Select the attribute a that has obtained the best information gain, and repeat the process for each one of the two resulting subsets of examples and the remaining attributes.

Note that with only three attributes characterizing the examples the tree will have at most depth 3. In this condition it is unavoidable to have a mix of positive and negative examples in each leave. In the Figure 1.a shows a decision

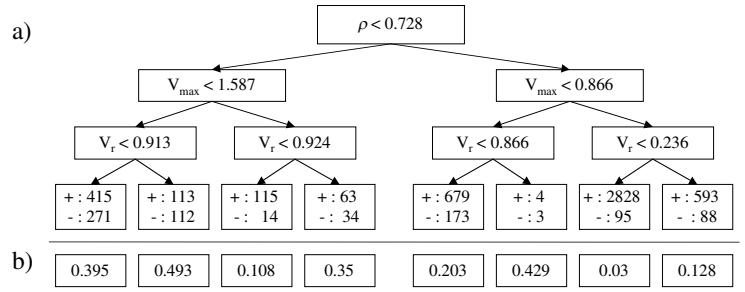

Fig. 1. Example of a learned tree for the termination check predicate.

tree where each leaf l has a number of positive and negative examples belonging to l, e.g. first leaf on the left has 415 positive and 271 negative examples.

In order to use the T_i, each agent A_i transforms the leaves of T_i into leaves that contain the probability for an example to be in the negative class. Let p_l be the number of positive examples and n_l negative examples belonging to leaf l, then probability of an example to be in the negative class is $N_l = \frac{n_l}{(n_l+p_l)}$. Figure 1.b shows the result of this transformation, e.g. first leaf on the left has $N_l = 0.395$.

4.4 Proactive Bounded Counsel

An agent A_i solving a problem P uses the decision tree T_i as *TermCheck* every time it asks the counsel to a new agent. If *TermCheck* returns true A_i terminates and yields as result the solution class with maximum votes in the current voting situation—otherwise A_i will ask a new agent for counsel. A_i assess every voting situation building its corresponding characterization $\upsilon^t = \langle V^t_{max}, V^t_r, \rho \rangle$ and then uses T_i to classify it. The result is that T_i will classify υ^t in some leaf l with negative class probability N_l.

Now A_i would want to ask counsel of another agent if the probability of the current result of the voting process to be incorrect is high. Thus, each A_i has a threshold ν such that if $N_l \geq \nu$ it considers that the probability of being incorrect is too high and decides to ask counsel to another agent, and if $N_l < \nu$ it considers that the current solution is good enough and the process terminates. Thus, ν determines the degree of confidence that A_i wants to have about a solution before answering it without asking counsel to more agents. That is to say, we want at most a predicted probability of error of ν. Constant ν has been fixed to 0.05 for all the agents in our experiments. This means that *TermCheck* predicts that the solution given will be the correct with a probability of 0.95.

If we look at Figure 1 we can see that with $\nu = 0.05$ only one leaf has $N_l < \nu$ (the seventh leaf with $N_l = 0.03$). However, notice that most of the examples (2923 of the 5600, i.e. 52.2%) belong to that leaf.

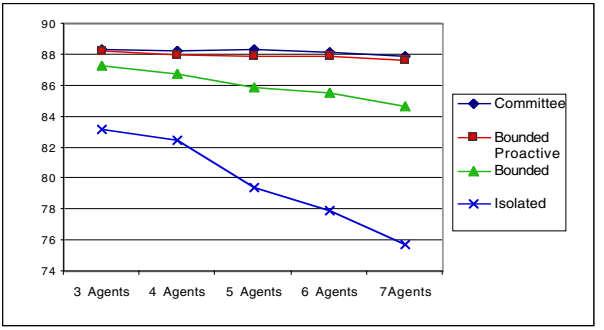

Fig. 2. Accuracy comparison between proactive bounded, bounded, and committee policies.

5 Experimental Results

In this section we want to analyze how the collaboration policies improve the agents performance with respect to the isolated agents scenario, where the agents do not cooperate and each agent tries to solve the problem by itself. We experiment with with multiagent systems composed of 3 to 7 agents. The purpose of varying the number of agents is to assess how well behave the collaboration policies when data is more fragmented.

We use the marine sponge identification (classification) problem as our test bed —this data set was also used to assess the relational inductive method INDIE in [2]. Sponge classification is interesting because the difficulties arise from the morphological plasticity of the species, and from the incomplete knowledge of many of their biological and cytological features. Moreover, benthology specialists are distributed around the world and they have experience different benthos that spawn species with different characteristics due to the local habitat conditions.

In order to compare the performance of the three policies, we have designed an experimental suite with a case base of 280 marine sponges pertaining to three different orders of the *Demospongiae* class (*Astrophorida*, *Hadromerida* and *Axinellida*). In an experimental run, training cases are randomly distributed to the agents (without repetitions, i.e. each case will belong to only one agent case base). In the training phase, problems arrive randomly to one of the agents. The goal of the agent receiving a problem is to identify the correct biological order given the description of a new sponge.

The experimental setting allows us to compare the collaboration policies behavior with different number of agents. For instance, in the 3 agents scenario, an agent has about 84 cases (about 28 cases per class)—while in the 7 agents scenario, an agent has about 36 cases (about 12 cases per class). The results presented here are the result of the average of 5 10-fold cross validation runs.

Figure 2 plots the accuracy of the three collaboration policies compared with respect to the accuracy of isolated agents. Clearly, any collaboration policy is

Table 1. Cost comparison of Bounded Counsel, Proactive Bounded Counsel, and Committee policies.

	3 Agents	*4 Agents*	*5 Agents*	*6 Agents*	*7 Agents*
Isolated Agents	1.00	1.00	1.00	1.00	1.00
Bounded	1.43	1.59	1.71	1.89	2.01
Proactive Bounded	2.11	2.66	3.41	4.11	4.80
Committee	3.00	4.00	5.00	6.00	7.00

more desirable for the agents than working alone with its own data. Let us consider the accuracies of *Proactive Bounded Counsel* and fixed *Bounded Counsel*. Figure 2 shows that the agents with proactive learning always have better accuracy. Moreover, *Proactive Bounded Counsel* is more robust when the number of agents is greater (i.e. the size of the case-bases is smaller) because they can learn an adequate *TermCheck* predicate for each situation.

Let us consider now the *Committee* policy. This collaboration policy has better accuracy than the fixed *Bounded Counsel* policy. However, *Proactive Bounded Counsel* is always very near to the *Committee* policy—and the difference is not statistically significant (99% confidence in a signed rank test).

The difference between the *Committee* and the *Proactive Bounded Counsel* policies is in terms of cost. The *Committee* policy always ask all agents while *Proactive Bounded Counsel* is intended to minimize those questions. Table 1 shows the costs involved in different policies for a situation where each time an agent asks counsel to another has a fixed cost, 1 euro in this example. The isolated agents never ask counsel so cost is constant, while the *Committee* policy always asks counsel to all agents and cost increases with the number of agents in the system. The cost of *Bounded Counsel* policies are between these two extremes. Notice that the *Proactive Bounded Counsel* policy is more expensive (asks more counsels) than the fixed *Bounded Counsel* policy. However, the accuracy results in Figure 2 shows that the Proactive Bounded policy asks for counsel when it's really needed, achieving an increased accuracy that matches that of the expensive Committee policy. Summarizing, the *Proactive Bounded Counsel* policy has the same accuracy that the *Committee* policy at a lower cost.

In order to assess that this results are not dependent on the assumption that case bases are disjunct we performed a specific experiment with overlapping cases. Specifically, each agent has a case base with a 10% of duplicated cases (in the sense that they are already present in another case base). The results are very close to the ones in Fig. 2, showing that the collaboration policies work well with some overlapping. Clearly, in a setting where overlapping grows to higher values the incentives for the agents to collaborate will diminish.

6 Related Work

A general result on multiple model learning [5] demonstrated that if uncorrelated classifiers with error rate lower than 0.5 are combined then the resulting

error rate must be lower than the one made by the individual classifiers. The BEM (*Basic Ensemble Method*) is presented in [7] as a basic way to combine continuous estimators, and since then many other methods have been proposed: *Stacking generalization*, *Cascade generalization*, *Bagging* or *Boosting* are some examples. However, all these methods do not deal with the issue of "partitioned examples" among different classifiers as we do—they rely on aggregating results from multiple classifiers that have access to *all* data. Their goal is to use a multiplicity of classifiers to increase accuracy of existing classification methods. Our goal is to combine the decisions of autonomous classifiers (each one corresponding to one agent), and to see how can they cooperate to achieve a better behavior than when they work alone.

The meta-learning approach in [4] is applied to partitioned data. They experiment with a collection of classifiers which have only a subset of the whole case base and they learn new meta-classifiers whose training data are based on predictions of the collection of (base) classifiers. They compare their meta-learning approach results with weighted voting techniques. The final result is an *arbitrator tree*, a centralized and complex method whose goal is to improve classification accuracy. We also work on "partitioned examples" but we assume no central method that aggregates results; moreover we assume a multiagent approach where communication and cooperation may have a cost that has to be taken into account.

DRL [11] is a distributed technique that learns rules from partitioned data; DRL's goal is to achieve scalability for large data sets. Rule induction in a workstation follows a top-down strategy in each data set, finding rules that are *satisfactory* for a specific data set. These rules —termed *candidate rules*— are sent to an additional workstation reviewing them over the entire data set; when a rule is satisfactory for the entire data set it is *accepted* by the algorithm. We work on a multiagent setting instead of a distributed one, but it seems DRL could be easily adapted to a multiagent setting. The main differences are i) that DRL is an inductive learning technique (designed to speed up class descriptions) while we use lazy learning techniques and ii) DRL has a centralized stage where the entire data set is available to the algorithm.

7 Conclusions and Future Work

We have presented a framework for cooperative CBR in multiagent systems. The framework is cooperative in that the CBR agents help each other to improve their individual performance. Since the agents improve with respect to their performance as isolated individual, cooperating is also in their individual interest—specially since the framework allows them to keep confidential their own cases. A major theme in multiagent systems is the *autonomy* of the agents. In our framework the agent autonomy is mainly insured by two facts: i) the capability of each agent to determine whether or not itself is competent to solve a problem, and ii) the capability of each agent to integrate into a global solution for a problem the counsels given by other agents.

Another issue is the generality of the cooperation policies and their dependence upon the CBR agents using LID. The cooperation policies depend only on the CBR agents being able to provide SERs (Solution Endorsement Records), so any CBR method that can provide that is compatible.

Finally, we plan to lift the restriction of the case bases of the agents in a $\mathcal{M}$AC system being disjunct. Basically, our idea is that agents could incorporate in their case bases some cases originally owned by other agents. The interesting question here is this: what strategy of case sharing can improve the overall $\mathcal{M}$AC system performance —without every agent having in their case base every case known to the $\mathcal{M}$AC system.

Acknowledgements. The authors thank Josep-Lluís Arcos and Eva Armengol of the IIIA-CSIC for their support and for the development of the Noos agent platform and the LID CBR method respectively. Support for this work came from CIRIT FI/FAP 2001 grant and projects TIC2000-1414 "eInstitutor" and IST-1999-19005 "IBROW".

References

[1] J. L. Arcos and R. López de Mántaras. Perspectives: a declarative bias mechanism for case retrieval. In David Leake and Enric Plaza, editors, *Case-Based Reasoning. Research and Development*, number 1266 in Lecture Notes in Artificial Intelligence, pages 279–290. Springer-Verlag, 1997.

[2] E. Armengol and E. Plaza. Bottom-up induction of feature terms. *Machine Learning Journal*, 41(1):259–294, 2000.

[3] E. Armengol and E. Plaza. Lazy induction of descriptions for relational case-based learning. In *ECML-2001*, Lecture Notes in Artificial Intelligence, 2001.

[4] Philip K. Chan and Salvatore J. Stolfo. A comparative evaluation of voting and meta-learning on partitioned data. In *Proc. 12th International Conference on Machine Learning*, pages 90–98. Morgan Kaufmann, 1995.

[5] L. K. Hansen and P. Salamon. Neural networks ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.

[6] Ramon López de Mántaras. A distance-based attribute selection measure for decision tree induction. *Machine Learning*, 6:81–92, 1991.

[7] M. P. Perrone and L. N. Cooper. When networks disagree: Ensemble methods for hydrid neural networks. In *Artificial Neural Networks for Speech and Vision*. Chapman-Hall, 1993.

[8] E. Plaza, J. L. Arcos, P. Noriega, and C. Sierra. Competing agents in agent-mediated institutions. *Journal of Personal Technologies*, 2:212–220, 1998.

[9] E. Plaza, R. López de Mántaras, and E. Armengol. On the importance of similitude: An entropy-based assessment. In I. Smith and B. Saltings, editors, *Advances in Case-Based Reasoning*, number 1168 in Lecture Notes in Artificial Intelligence, pages 324–338. Springer-Verlag, 1996.

[10] Enric Plaza, Josep Lluís Arcos, and Francisco Martín. Cooperative case-based reasoning. In Gerhard Weiss, editor, *Distributed Artificial Intelligence Meets Machine Learning. Learning in Multi-Agent Environments*, number 1221 in Lecture Notes in Artificial Intelligence, pages 180–201. Springer-Verlag, 1997.

[11] F. J. Provost and D. Hennessy. Scaling up: Distributed machine learning with cooperation. In *Proc. 13th AAAI Conference*, 1996.