Learning to Form Dynamic Committees

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

Learning agents can improve performance when they cooperate with other agents. Specifically, learning agents forming a committee outperform individual agents. This "ensemble effect" is well know for multi-classifier systems in Machine Learning. However, multi-classifier systems assume all data is know to all classifiers while we focus on agents that learn from cases (examples) that are owned and stored individually. In this article we focus on the selection of the agents that join a committee for solving a problem. Our approach is to frame committee membership as a learning task for the convener agent. The committee convener agent learns to form a committee in a dynamic way: at each point in time the convener agent decides whether it is better to invite a new member to join the committee (and which agent to invite) or to close the membership. The convener agent performs learning in the space of voting situations, i.e. learns when the current committee voting situation is likely to solve correctly (or not) a problem. The learning process allows an agent to decide when to individually solve a problem, when it is better to convene a committee, and which individual agents to be invited to join the committee. Our experiments show that learning to form dynamic committees results in smaller committees while maintaining (and sometimes improving) the problem solving accuracy.

Categories and Subject Descriptors

I.2.11 [**Computing Methodologies**]: Artificial Intelligence—*Distributed Artificial Intelligence*; G.3 [**Mathematics of Computing**]: Distribution functions; I.2.11 [**Computing Methodologies**]: Artificial Intelligence—*Learning*

General Terms

Experimentation

[*]Main author, student

Keywords

Cooperative CBR, Multiagent CBR, Collaboration Policies.

1. INTRODUCTION

A main issue in multiagent systems is how an agent decides when to cooperate with other agents. Specifically we focus on the issue of an agent that has to decide whether it is able to individually solve a problem or asks others to help it to solve the problem forming a Committee. For our purpose, a Committee is a collection of agents that cooperate in solving a problem by casting a vote on a (individually endorsed) solution where the overall solution is that with maximum votes. The voting can have several schemes, majority voting or approval voting—we'll see we will be using bounded weighted approval voting (BWAV).

Concerning the incentive of agents to cooperate in the form of a Committee, the basic reason is that they can improve their performance in solving problems—since we focus on classification tasks, the Committee organization improves (in general) the classification accuracy with respect to individual agents. This called "ensemble effect" is well know for multi-classifier systems in Machine Learning. However, multi-classifier systems assume all data is known to all classifiers while we focus on agents that learn from cases (examples) that are owned and stored individually. The *ensemble effect* essentially means that when the individual classifiers error is not correlated to other classifiers, the Committee improves with respect to all individual classifiers.

A second issue on multiagent cooperation involves the selection of which agents we want to cooperate with. In terms of our current framework, this involves the selection (by a *convener agent*) of the agents invited to join a Committee. Because of the *ensemble effect*, the default selection policy is to invite all available and capable agents to join a Committee. However, this process can be expensive or slow if the committee is big, and it is not evident that this policy is the best on all situations.

We present a learning framework that unifies both the "when" and "who" issues: learning a decision procedure for when to collaborate and selecting which agents are better suited for collaboration. In this framework the convener agent learns to assess the likelihood that the current committee will give a correct solution. If the likelihood is not high, the convener agent has to invite a new agent to join the Committee and has to decide which agent to invite. The initial situation of this framework starts with the convener agent that receives a problem forming a "Committee of one" and then deciding whether the problem can be solved in isolation or it is better to convene a Committee.

We present a proactive learning approach, in which an agent performs some activity on the multiagent system in order to learn this decision procedure (or *policy*) that we call the *Dynamic Committee* policy. The agent performs learning in the space of voting situations, i.e. learns when the current committee voting situation is likely to solve correctly (or not) a problem. Specifically, an agent using our proactive learning approach induces a decision tree that classifies the current voting situation as positive (likely to be correct) or negative—in which case the Committee needs to be enlarged. We will see that the decision tree can also learn when a specific agent is recommended to be invited to join the Committee. Our experiments show that learning to form dynamic committees results in smaller committees while maintaining (and sometimes improving) the problem solving performance.

The structure of the paper is as follows. First, we define a multiagent system, a Committee and voting scheme. Then we introduce the Dynamic Committee interaction protocol with an individual *Dynamic Committee* policy. Section 4 explains proactive learning of the *Dynamic Committee* policy and section 5 shows the empirical results of several experiments using this policy. The paper closes with a section on related work and some conclusions.

2. MULTIAGENT CBR SYSTEMS

We focus on agents that use Case Based Reasoning (CBR) to solve problems. CBR techniques suit perfectly into multiagent systems and give the agents the capability of autonomously learn from experience by just retaining new cases (problems with known solution). Therefore, we talk about *Multiagent CBR Systems* ($\mathcal{M}$AC).

The agents in a $\mathcal{M}$AC system are able to solve problems individually, i.e. agents can apply a CBR method using only their local case base to solve a new problem. Problems to be solved can arrive to an agent by an external user, or by another agent. By allowing an agent to send problems to another agent, $\mathcal{M}$AC systems gain a lot of flexibility. For instance, if an agent A_i receives a problem from an user and decides that there is not enough information to solve the problem in the local case base, the problem can be sent to another agent A_j expecting that A_j can solve the problem. Therefore, agents in a $\mathcal{M}$AC system can collaborate with other agents in the process of finding a solution for a problem.

Formally, a $\mathcal{M}$ACsystem $\mathcal{M} = \{(A_i, C_i)\}_{i=1...n}$ is composed on *n* agents, where each agent A_i has a case base C_i. In 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. Each agent A_i is autonomous and has learning capabilities, i.e. each agent is able to collect autonomously new cases that can be incorporated to its local case base.

Moreover, since we focus in analytical tasks, there is no obvious decomposition of the problem in subtasks. Therefore, when an agent A_i wants to cooperate with another agent A_j for solving a problem P, they cannot decompose the problem and each one solve a particular part. The only way is to send the complete problem P to A_j. After A_j answers A_i with the solution found for P, A_i can do anything with this solution. A simple way for A_i to use this solution is to compare it with the solution found by himself, if both solutions agree A_i can increase the degree of confidence on the solution found, and if both solutions disagree, maybe it's interesting to send the problem to some other agent to have a third opinion.

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.

In our framework, agents use a voting mechanism ibn order to aggregate the information contained in various SERs coming from other agents. This voting scheme is explained in the next section.

2.1 Voting Scheme

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 the 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.

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

This voting scheme can be seen as a variation of *Approval Voting* [1]. In *Approval Voting* each agent vote for all the candidates they consider as possible solutions without giving any weight to its votes. In our scheme, *Approval Voting* can be implemented making $Vote(S_k, A_j) = 1$ if $E_k^j \neq 0$ and 0 otherwise.

There are two differences between the standard *Approval Voting* and our voting scheme. The first one is that in our voting scheme agents can give a weight to each one of its votes. The second difference is that the sum of the votes of an agent is bounded to 1. Thus we can call it *Bounded-Weighted Approval Voting* (BWAV).

The next section presents the *Committee* collaboration strategy, that uses this voting scheme.

2.2 Fixed Committee Strategy

In this collaboration strategy the agent members 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 problem's solution is the class with maximum number of votes.

Since all the agents in a $\mathcal{M}$AC system are autonomous CBR agents, they will not have got the same problem solving experience. Therefore, the cases in their case bases will not be the same. This ensures that the errors that each agent make in the solution of problems will not be very correlated, i.e. each agent will not make errors in the same problems. Thus, using the committee collaboration policy an agent can increase its problem solving accuracy because it matches the preconditions of the ''ensemble effect".

3. DYNAMIC COMMITTEES

Using the *Committee* collaboration policy, all the agent members of a $\mathcal{M}$AC take part in the solution of every problem. However, this policy can have several drawbacks. First of all, if the $\mathcal{M}$AC system is composed of a large collection of agents, making every agent to take part part in the solution can have a great cost. Moreover, there can be some agents not willing to take part in the solution of every problem. But even if the cost is not a problem and all the agents are willing to take part in the solution of each problem it is not clear that the best strategy to solve a problem is by aggregating always the solution of all the agents in the system. For example, if a group of agents inside the $\mathcal{M}$AC have a error correlation (i.e. they fail to solve almost the same problems), it would be better to ask only one agent of that group, since asking the others will only report redundant information. For these reasons, it would be interesting to have dynamic committees, i.e. each time an agent needs to solve a new problem, this agent should decide which is the best subcommittee to solve that problem.

Choosing an optimum committee given a new problem can be a difficult task, and for that reason we propose an incremental protocol to build the committee based on learning techniques. This protocol is called the *Dynamic Committee* (DC) Protocol.

Let A_i be an agent that wants to solve a problem P. A_i wants to choose the best committee to solve that particular problem P, thus we will call A_i the *committee convener*. Let $\mathcal{A} = \{A_1, \ldots, A_n\}$ be the set of all the agents in the $\mathcal{M}$AC and $\mathcal{A}_t^r$ be the set of all the agent in the $\mathcal{M}$AC that still have not been invited to join the committee at a time t. The DC protocol works as follows:

1. Initially $t = 0$ and $\mathcal{A}_t^r = \mathcal{A} - \{A_i\}$.

2. A_i solves the problem P individually. $\mathcal{R}_t = \{s_i^P\}$, where s_i^P is the SER obtained by A_i for the problem P. We can consider that initially A_i has created a committee consisting on only one member, himself.

3. Unless A_r is emptyA_i has to decide whether to invite another agent to join the committee or not. If A_r is empty the protocol moves to the step 6.

   - If A_i decides not to invide another agent to join the committee the protocol moves to the step 6.
   - Otherwise, A_i has to decide which agent A_j from the remaining agents $\mathcal{A}_t^r$ to invite to join the committee.

4. A_j is removed from the set of remaining agents: $\mathcal{A}_{t+1}^r = \mathcal{A}_t^r - \{A_j\}$ and A_i waits for the answer of A_j to the invitation to join the committee.

   - If A_j refuses to join the committee $\mathcal{R}_{t+1} = \mathcal{R}_t$ and the protocol moves to the step 3.
   - Otherwise, the problem P is sent to A_j.

5. When the SER from A_j is received we add it to the current list of SERs: $\mathcal{R}_{t+1} = \mathcal{R}_t \cup \{s_j^P\}$. Then the protocol moves to the step 3.

6. The final solution is obtained using the voting process among the set of SERs $\mathcal{R}_t$ of all the agents of the current committee for the problem P. The protocol ends.

In order to use the DC protocol, an agent needs to have a policy to decide when to invite new agents to join the committee and which will be these agents. This is called the *Dynamic Committee* (DC) policy. The DC policy is the responsible of of keeping small committees and of choosing which agents will be part of the committee. Also notice, that the first time that the DC policy is used, is to decide wether to collaborate with other agents or not. Thus, if at the first iteration the DC policy decides that there is no need of inviting any more agent to join the committee, the agent A_i will solve the problem in isolation.

In the following sections we will formally define the DC policy, then we will present a learning technique that allows each agent to proactively learn its own DC policy. After that, we will show how to integrate this learned DC policy into the DC protocol.

3.1 Dynamic Committee Policy

Let us define in more detail the task of the DC policy, given an agent A_i that wants to solve a problem P. At a time t the agent A_i has build a committee composed of a set of agents $\mathcal{A}_t$ and have received a set of SERs $\mathcal{R}_t$ coming from the agents of the current committee. The set of all the remaining agents to be invited to join to committee will be noted by $\mathcal{A}_t^r$. The task of the DC policy is to predict whether the solution obtained by using the voting process among the SERs in $\mathcal{R}_t$ is likely to be correct or not. In the second case, the agent has to choose one agent of the remaining ones to invite it to join the committee. There's situations where it is not clear which one of the other agents is the best to join the committee, in these cases the DC policy can just say that a new agent should join the committee but without giving any preference to anyone.

We can now formally define the DC policy. The DC policy is a function $DCPolicy(\mathcal{A}_t^r, \mathcal{R}_t)$ that can output values in the following set: $\{+, *\} \cup \mathcal{A}_t^r$, i.e. the output can be $+$ (meaning that there is no need to include more agent in the current committee), $*$ (meaning that a new agent has to join the committee but without giving preference to anyone), or the name of one of the remaining agents. We will call positive to the $+$ output, because it means that the current candidate solution is likely to be correct, and negative to any other output, because they mean that the current candidate solution isn't likely to be correct. The following sections explain how each agent can learn its own DC policy using a proactive learning strategy.

4. PROACTIVE LEARNING

Since the *Dynamic Committee* policy is a function $DCPolicy$ $(\mathcal{A}_t^r)$ with output $\{+, *\} \cup \mathcal{A}_t^r$, the it can be seen as a classification task where we have $n + 2$ classes ($\{+, *, A_1, \ldots, A_n\}$).

We will call a *voting situation* to the set of current SERs R_t received from the current members of the committee because they completely characterize the voting situation. Thus, the learning task is to assign a class to each voting situation. In order to learn the DC policy, each agent needs experience on the situations where the policy should be applied, i.e. examples from where to learn.

The next section defines these examples. Then, we will explain the proactive strategy that the agents can use to obtain the examples, and finally how to learn the DC policy from those examples.

4.1 Defining the Training Examples

Each *voting situation* $\mathcal{R}_t$ will be characterized by several attributes in order to be represented as a attribute-value vector:

- The attributes $A_1^t \dots A_n^t$ are boolean. $A_i^t = 1$ means that at the time t the agent A_i is a member of the committee, and a value 0 means that that agent still is not a member of the committee.
- S_c^t, the candidate solution class.
- V_{max}^t, the votes for the candidate solution class.
- V_r^t, the sum of the votes for the rest of classes.
- $\rho^t = V_{max}^t/(V_{max}^t + V_r^t)$, the ratio of votes for the candidate solution.

We will use these attributes to define our examples. Specifically, we define a υ-example as the tuple $\langle A_1^t ... A_n^t, S_c^t, V_{max}^t, V_r^t, \rho^t \rangle$. Each υ-example belongs to one of these classes $\{+, *, A_1, ..., A_n\}$. Therefore, we can have positive υ-examples: eg. $(+, \langle A_1^t \dots A_n^t, S_c^t, V_{max}^t, V_r^t, \rho^t \rangle)$, characterizing voting situations where the candidate solution S_c is likely to be the correct one, and also negative υ-examples: eg. $(*, \langle A_1^t \dots A_n^t, S_c^t, V_{max}^t, V_r^t, \rho^t \rangle)$, characterizing voting situations where the candidate solution S_c is not likely to be correct, and that it would be better to ask a new agent to join the committee.

4.2 Obtaining the Training Examples

In this section we are going to present the proactive process that an agent follows in order to obtain υ-examples from where to learn.

Since every agent A_i has a case-base (collections of problems with known solution), A_i can obtain υ-examples of voting situations from which to learn the termination check. An agent A_i can send its own problems to the other agents and 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 learning the DC policy 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 the other agents, they send back a SER.
4. A_i solves P by itself by a *leave-one-out* method, i.e. it solves P using $C_i - P$ as the case base, creating its own SER.
5. With the set R_t of SERs obtained in steps 3 and 4, A_i builds υ-examples of voting situations.

Note that from the collection R_t of SERs obtained in the step 5 we can build more than one υ-example. In fact we can build a υ-example for any possible non empty subset $\mathcal{R}_t^x \subseteq \mathcal{R}_t$. Thus, step 5 is decomposed in 3 subsetps:

1. Choose a collection $\mathcal{R}$ of non empty subsets of $\mathcal{R}_t$, i.e. $\mathcal{R} \subseteq \mathcal{P}(\mathcal{R}_t)$.
2. For each voting situation $R_t^x \in \mathcal{R}$ let $\upsilon^x = \langle A_1^t ... A_n^t, S_c^t, V_{max}^t, V_r^t, \rho^t \rangle$ be the example characterizing that situation.
3. If the most voted solution is the correct class, build a positive example $(+, \langle A_1^t \dots A_n^t, S_c^t, V_{max}^t, V_r^t, \rho^t \rangle)$ otherwise build a negative example.

In the step 3, when a negative υ-example has to be built, the specific negative class (from $*, \{A_1, \dots, A_n\}$) has yet to be chosen. A υ-example belonging to a class A_j will mean that A_j is the next agent to be asked to join the committee in the voting situation characterized by the υ-example. To choose this specific negative class, the agent has to find if there is any agent A_j (from the subset of agents that still haven't joined the committee) that if added to the committee, the outcome of the voting process can change to be the right solution class. This can be done just by adding the SER sent by the agent A_j to the set R_t^x and testing the outcome of the voting process with the right solution class. If there is no such an agent A_j that added to the committee can change the result of the voting process, the υ-example will be considered simply to belong to the $*$ negative class. Now it is clear that the reason to include this new negative class is to be able to express the lack of knowledge about which agent to choose in an explicit way. When using the learned DC policy, the interpretation is that a $*$ class means that we have to ask another agent to join in the committee (because the current candidate solution is not right), but we don't have evidence on which agent to add.

Notice that the size of the set $\mathcal{R}$ depends on the number of agents involved. In our experiments we have chosen all the possible subsets of $\mathcal{R}$ to build υ-examples. But it is not feasible when the number of agents is not small, and a sample of all possible υ-examples must be chosen. A good sample B_i of the cases in the agent's case base also has to be chosen, in our experiments we use all the case base, i.e. $B_i = C_i$.

The result of the process explained in this section is a collection of υ-examples that will constitute the training set Υ_i from where to learn the DC policy.

4.3 Induction of a decision tree

Once that an agent A_i has enough υ-examples, a good DC policy can be learnt. The agents learn the DC policy using a decision tree learning algorithm with a discretization technique for the numeric attributes.

To build the decision tree T_i, each agent A_i use a standard ID3 algorithm ([12]), but with the following considerations:

- Discretization of numerical attributes: Each numeric attribute a is discretized to have 2 classes. To do this, the best cutpoint κ (in the sense of maximizing the information gain) that divides Υ_i in two subsets is obtained. On one leaf of the decision tree we will have all the examples where $value(a) < \kappa$, and in the other one all the examples wher $value(a) \geq \kappa$.
- Pruning: If in some moment of creating the tree, the number of examples is very low (less than 20), that node is not splitted any more, and is considered as a leaf. This is done to avoid overfitting.

As we have very few attributes, it is likely that in each leaf of the tree there is a mix of examples of several classes. This situation os shown in Figure 1. The tree shown is the learned tree by an agent called A_1 in a system composed of 5 agents. Notice that the class ocurring more times is the positive class $+$. This is as expected, since the agents have an individual accuracy of about 75%. By looking at the tree, we can see that the most discriminant attribute is ρ. This is expected, since ρ represents the fraction of votes for the candidate solution. Notice that when ρ is lower than 0.517 there are very few positive examples and that the leaf where most of the positive examples are is the leaf where $\rho > 0.517$, $V_{rest} < 0.875$ and $V_{max} > 1.875$, i.e. when there is a big fraction of votes for the candidate solution and few votes for the rest of solutions.

In order to use the decision tree, each agent transform the leaves of the tree into leaves containing a single class. To do this transformation in a leaf l, let p_l be the number of positive examples belonging to that leaf and n_l the number of examples not belonging

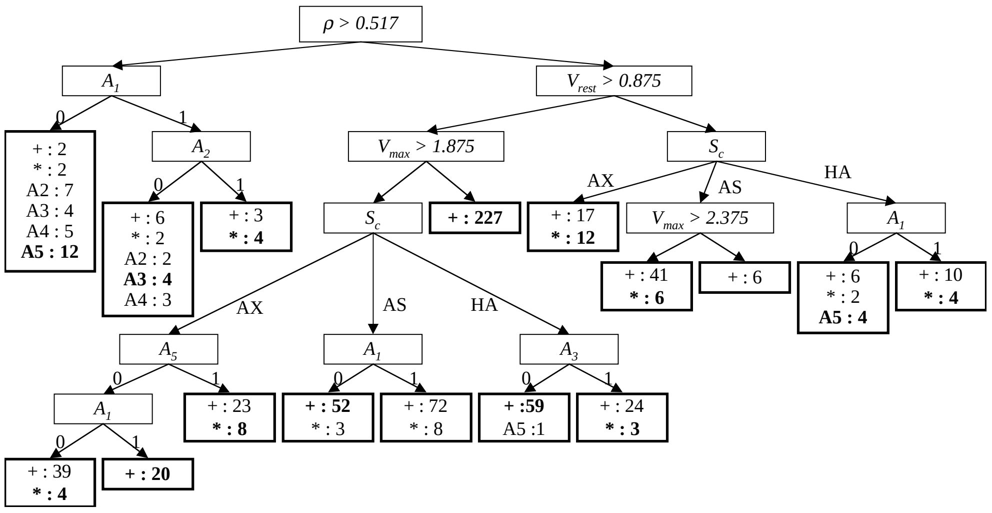

Figure 1: Learned tree for an agent working in a 5 agent $\mathcal{M}$AC. The labels AX, AS and HA in some branches mean that S_c equals that solution class in that branch. The nodes labelled with A_i test it the agent A_i is a member of the current committee or not.

the the positive class (negative examples). Then, we can consider $N_l = \frac{n_l}{(n_l+p_l)}$ as the predicted probability of an example to be in the positive class. This probability can be interpreted as the probability of the candidate solution derived from the voting situation characterized by an example falling in the leaf l of being the right one. In other words, if we characterize a voting situation, we classify it using the decision tree T_i, and it falls in a leaf l, the predicted probability of obtaining the right solution class with the current members of the committee is N_l. If we asusme that we want to obtain dynamic committees that achieve high accuracies, we want to maximize N_l. If N_l for a given voting situation is not above a fixed threshold ν, the agent will consider that the current committee is not good enough. Therefore, the leaves of the decision tree T_i are transformed in the following way:

For each leaf l:

- If $N_l > \nu$, the leaf is considered positive, and is labelled $+$.
- If $N_l \leq \nu$, the leaf is considered negative, and it is labelled with the negative class with more examples on that leaf, eg. if we have in one leaf 25 examples of the class $+$, 14 of the class $-$, and 9 of the class A_1, the leaf will be labelled as $-$, because is the negative class with more examples (14 versus the 9 examples of the class A_1.
- If there are very few examples in one leaf (i.e. less than 10 in the experiments), we consider that the estimation N_l of the probability of the candidate solution to be the right one has not enough confidence, and thus we consider the leaf as negative. If we look again at the Figure 1, this happens in the third leaf starting from the right.

The Figure 2 shows the tree resulting from transforming that on Figure 1. Notie, that all the positive leafs are under the conditions $\rho > 0.517$ and $V_{rest} < 0.875$, i.e. when there is a big fraction of votes for the candidate solution and few votes for the rest of solutions. Notice also that the third leaf starting from the right has been labelled as negative because it had less than 10 examples.

In Figure 2, the threshold ν has been fixed to 0.9, and this will also be the value used in the experiments presented in the next section. After this transformation, the decision tree is ready to be used.

In the following we explain how to integrate this learned tree into the DC protocol to be used as the DC policy.

4.4 Using the Learned Policy

Once an agent A_i has learned its own tree T_i it can be used directly as the DC policy for be used inside the DC protocol. Each time A_i has to decide whether to ask a new agent to join the committee or not, the current voting situation is characterized by the corresponding attributes $\upsilon^t = \langle A_1^t ... A_n^t, S_c^t, V_{max}^t, V_r^t, \rho^t \rangle$. Then this characterization υ^t is classified using the learned tree T_i. If it is classified as belonging to de $+$ class, no more agents will be asked to join the committee, and the current candidate solution will be considered as the final solution. If υ^t is classified in any of the classes $A_i, \ldots, A_n$, the agent with the same name will be asked to join the committee. Finally, if the predicted class is $*$ one of the remaining agents will be chosen randomly to be asked to join the committee.

5. EXPERIMENTAL RESULTS

In this section we want to test if the learned Dynamic Committee policy is able to properly decide when to collaborate with other agents and whether it is able to properly select which agents to invite to join the committee. To assess the correctness of the learned policy we have compared the behaviour of agents using Dynamic Committee strategies with agents using Fixed Committee strategies and agents that solve problems in isolation (without collaborating with any other agent). We have made experiments with $\mathcal{M}$AC systems composed of 3, 5, 7, 9 and up to 10 agents. The purpose of varying the number of agents is to assess that the learning process is general enough to be able to work under any configuration of a $\mathcal{M}$AC system.

We use the marine sponge classification problem as our test bed. 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 in different benthos that spawn species with different characteristics due to the local habitat conditions.

We have designed an experimental suite with a case base of 280

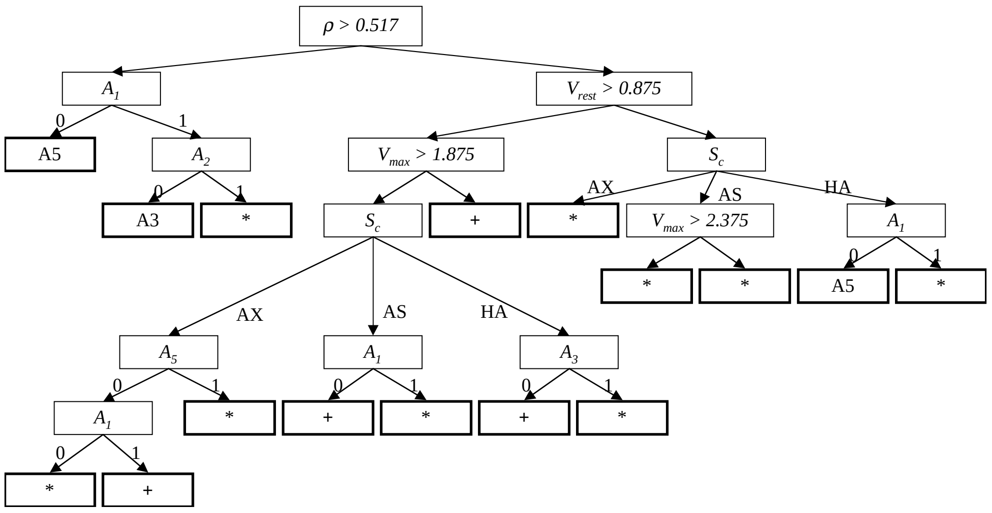

Figure 2: Learned tree for an agent working in a 5 agent $\mathcal{M}$AC after being processed to have only one class per leaf.

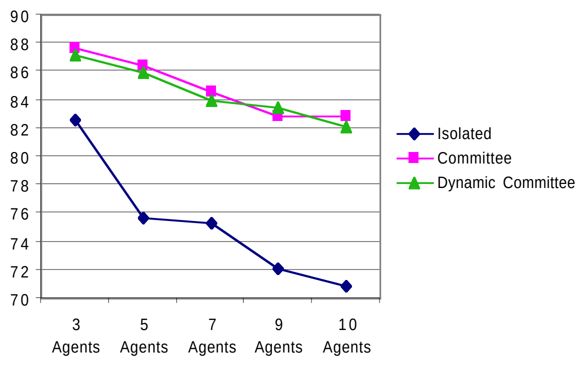

Figure 3: Accuracy comparison of several $\mathcal{M}$AC systems using committee and dynamic committee collaboration policies.

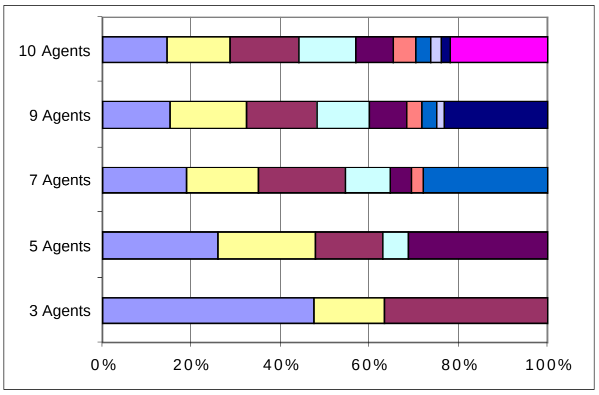

Figure 4: Percent of times that the convener agent has convened committees of different sizes.

marine sponges pertaining to three different orders of the *Demospongiae* class (*Astrophorida*, *Hadromerida* and *Axinellida*). In an experimental run, training cases are randomly distributed among the agents without replication, i.e. there is no case shared by two agents. In the testing stage unknown 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 agents use a 3 nearest neighbour algorithm to solve problems and the results presented here are the result of the average of 5 10-fold cross validation runs.

For each $\mathcal{M}$AC system configuration tested, each agent receives a different number of traning cases. For instance, in the 3 agent scenario, the training set has only to be splitted into three parts (one per agent), and therefore, each agent receives about 84 cases. But in the 10 agent scenario, the training set is split into ten parts, and thus the agents only receive 25.2 cases in average. Therefore, we must have in mind that in out experiments, as more agents we have, each agent have less information, and thus the individual classification accuracy diminishes, and the incentives to collaborate increases accordingly.

Figure 3 shows the accuracy obtained by agents using the Dynamic Committee strategy. Accuracy of agents using the Fixed Committee strategy and agents olving problems in isolation are also shown for comparison purposes. The first thing we see is that by using Dynamic Committees, the agents obtain much greater accuracy than solving problems in isolation, showing that the learned Dynamic Committee policy is able to detect when there is a need for convening a committee. Figure 3 also shows that the accuracy obtained by agents using the Dynamnic Committee strategy and agents using the Fixed Committee strategy is almost undistinguishable. This result shows that when the Dynamic Committee policy decides not to invite any more agent to join the committee is because really there is no need to. Notice also that the difference between the isolated agents scenario and the agents using Dynamic Committees is greater in the experiments with a greater number of agents. But, as explained before, this is because in the experiments with a great number of agents, each agent has fewer examples, and thus individual accuracy drops. Dynamic Committee strategy shows to be more robust under this conditions where the data is very fragmented among the agents.

To show the effectiveness of the Dynamic Committee policy, Figure 4 shows the how many times the convener agent builds a committee of different sizes. For each configuration of the mul-

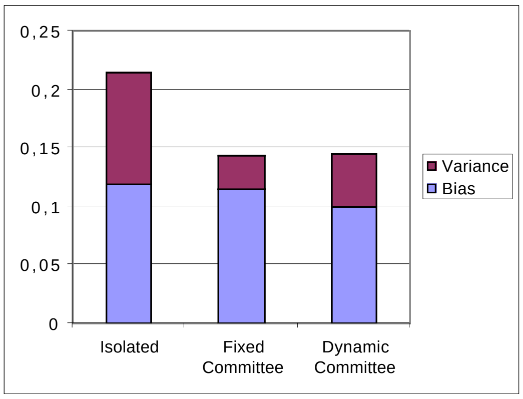

Figure 5: Bias plus variance decomposition of the classification error for a system with 5 agents.

tiagent system an horizontal bar is shown. This bar is divided in several intervals: 3 intervals for a 3 agents system, 5 for a 5 agents system, etc. Each interval means the percent of times that the convener agent has build a committee of each size. The left most interval represents the agent solving the problem in isolation, the second interval represents a committee of size 2, etc. For instance, for the 9 agents system, we can see that in a 15.0% of the times the convener agent has solved the problem in isolation, and that in the 48.3% of the times, the commitee convened has been smaller or equal to 3 agents. However, we can see also that there is a 23.4% of the times where the committee convened contains all the agents in the system. Analyzing the results, we observed that those 23.4% of problems were those where the agents had a great degree of disagreement in the classification of the problem. That is to say, there was never a big majority of votes for any solution class and the Dynamic Committee policy tried to invite more agents trying to solve that situation. We have also observed that most of the errors of the committee ocurred in that last case where the convener invited all the agents to join the committee. Therefore, we can say that the convener only invites all the agent to join the committee for really hard problems. In all the other configurations, the results are very similar, and for instance in the 5 agents scenario the convener agent works in isolation a 26.0% of the times, and only invites all the agents to join the committee a 31.3% of the times.

All the experiments reported in this paper have been done without replication of cases, i.e. there are no case shared by two agents. Therefore, every agent in the system can report interesting information to solve a problem. Clearly this is the best situation for the standard committees and the worst for the dynamic committees, since the ensemble effect gurantees us that we will obtain better predictions by combining the predictions from all them. But even that, we have shown that the dynamic committees can perform as well as the standard committees in terms of accuracy, and that the can do it at a much lower cost. As a future work, we plan to perform experiments with replication of cases to confirm that the dynamic committee policy will be able to deal with thr situation as well as (or better) than the Fixed Committee.

Next section presents the bias plus variance analysis of the results.

5.1 Bias plus variance analysis

Bias plus Variance decomposition of the error [9] is a useful tool to provide an insight of learning methods. Bias plus variance analysis breaks the expected error as the sum of three non-negative quantities:

- Intrinsic target noise: this is the expected error of the Bayes optimal classifier (lower bound on the expected error of any classifier).
- Squared bias: measures how closely the learning algorithm's prediction matches the target (averaged over all the possible training sets of a given size).
- Variance: this is the variance of the algorithm's prediction for the different training sets of a given size.

Since the first quantity cannot be measured, the bias plus variance decomposition estimates the quantities of the squared bias and variance. In order to estimate these quantities we are using the model presented in [9]. Figure 5 shows the bias plus variance decomposition of the error for a system composed of 5 agents.

Comparing the Fixed and Dynamic Committee strategies with the individual solution of prolblems, we see that the error reduction obtained is mainly due to a reduction in the variance component. This result is expected since a general result of machine learning tells that we can reduce the classification error of any classifier by averaging the prediction of several classifiers when they make uncorrelated errors due to a reduction in the variance term [6].

However, comparing the Fixed Committee with the Dynamic Committee results, we can see that with the Dynamic Committee strategy the reduction of the error is not only in the variance term, but there is a sensible reduction in the bias term of the error. This can be explained by two facts: as the committees are smaller with the Dynamic Committee strategy, the reduction of the variance term is smaller, and the convener agent control over the outcome of the voting process by using its capability to select which agents invite to join the committee is the responsible of the reduction of the bias term.

Therefore, we can conclude that the Dynamic Committee strategy can reduce the error of an agent working in isolation not only by the Committee effect of reducing the variance term but also by a reduction of the bias term.

6. RELATED WORK

The "ensemble effect" is a general result on multiple model learning [8], that 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 [11] as a basic way to combine continuous estimators, and since then many other methods have been proposed: *Stacking generalization* [14], *Cascade generalization* [7], *Bagging* [2] or *Boosting* [5] 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 multiplicity of classifiers to increase accuracy of existing classification methods. Our intention 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. A more similar approach is the one proposed in [13], where a MAS is proposed for pattern recognition. Each agent is a specialist recognizing only a subset of all the patterns, and the predictions are combined dynamically.

The meta-learning approach in [3] 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.

A more similar approach is that taken in [10], where a distributed CBR system is considered for personalized route planning. They present collaborative case-based reasoning (CCBR) as a framework where experience is distributed among multiple CBR agents. Their individual agents, are only capable of solve problems that fall within their area of expertise. When an agent cannot solve a problem, it broadcasts the problem to the other agents, and if there is some agent capable of solving it, if will return the relevant retrieved cases to the initial agent. This approach differs from ours in the sense that they only perform case retrieval in a distributed way. The initiating agent receives all the relevant cases contained in all the case bases of the other agents, and the it solves the problem locally. In our approach, an agent con only work with its local case base.

Also relevant there is a work on learning to from groups or coalitions of agents. Sarathi and Sen [4] propose a framework for agents that learn who are the best agents to collaborate with in the form of stable coalitions. However, they focus on the assignment of tasks to the agents that can perform them in a more efficiend way, and do not deal with committees as we do.

7. CONCLUSIONS AND FUTURE WORK

We have presented a framework for cooperative multiagent CBR systems called $\mathcal{M}$ACs. The framework is cooperative in the sense that the agents cooperate with other agents if this can report some improvement in performance. This article addresses two main issues on collaboration: when to collaborate, and with who to collaborate. We have presented a strategy called the Dynamic Committee strategy, that unifies both questions. We have also presented a learning technique that allows an agent to learn its own decision procedure to deal with the "when" and the "who" issues.

In section 5 we have presented experimental results showing that the learned Dynamic Committee policy can effectively decide when to convene a committee and when not. Using this learned policy, an agent can obtain much better results than working in isolation by convening a committee when there is the need to. We have also shown that using this technique, the results obtained match Fixed Committee accuracy (where all the agents are convened to solve every problem).

The effectiveness of the learned Dynamic Committee policy is also endorsed by the fact that the for most of the misclasified problems the convener agent had invited all the agents in the system to join the committee, showing that the Dynamic Committee policy is able to notice that a problem is difficult and the agent will convene all the agents in order to be able to solve it. Moreover, the fact that the classification accuracy is not lower than the accuracy of the Fixed Committee also tells us that the Dynamic Committee policy can effectively decide when there is no need for inviting more agents to join the committee.

In the experiments section, all the results reported from $\mathcal{M}$AC systems where the data is uniformly distributed among the agents, but we plan to study the behaviour of the Dynamic Committee strategy in multagent systems where the the "ensemble effect" preconditions do not hold. This can happen in two sitations: the first one is where there are some agents making random guessing or giving allways wrong answers (breaking the condition of having an error rate lower than 0.5); and the second situation is where we have groups of agents whose errors are correlated.

We also plan to perform incremental learning, i.e. the Dynamic Committee policy could be updated with each new problem solved. In this case, the learned policy can be more adatptive if more agents enter of die in 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".

8. REFERENCES

[1] S. J. Brams and P. C. Fishburn. *Approval Voting*. Birkhauser, Boston, 1983.

[2] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[3] P. K. Chan and S. 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.

[4] P. S. Dutta and S. Sen. Emergence of stable coalitions via task exchanges. In C. Castelfranchi and W. L. Johnson, editors, *Proceedings of the first international conference on Automous Agents and Multiagent Systems*, pages 312–313. ACM press, 2002.

[5] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Proc. 13th ICML*, pages 148–146. Morgan Kaufmann, 1996.

[6] J. H. Friedman. On bias, variance, 0/1 - loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery*, 1(1):55–77, 1997.

[7] J. Gama. Local cascade generalization. In *Proc. 15th International Conf. on Machine Learning*, pages 206–214. Morgan Kaufmann, San Francisco, CA, 1998.

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

[9] R. Kohavi and D. H. Wolpert. Bias plus variance decomposition for zero-one loss functions. In L. Saitta, editor, *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 275–283. Morgan Kaufmann, 1996.

[10] L. McGinty and B. smyth. Collaborative case-based reasoning: Applications in personalized route planning. In *ICCBR*, pages 362–376, 2001.

[11] 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.

[12] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[13] L. Vuurpijl and L. Schomaker. A framework for using multiple classifiers in a multiple-agent architecture. In *Third International Workshop on Handwriting Analysis and Recognition*, 1998.

[14] D. H. Wolpert. Stacked generalization. Technical Report LA-UR-90-3460, Los Alamos, NM, 1990.