Multiagent Learning through Collective Memory

Andrew Garland and **Richard Alterman**
Computer Science Department
Brandeis University
Waltham, MA 02254
{aeg, alterman}@cs.brandeis.edu

Abstract

Collective memory is a mechanism through which a community of agents can learn improved patterns of cooperative activity to take advantage of regularities in their domain of problem-solving activity. Collective memory is defined as the breadth of procedural knowledge the community acquires through experience when interacting with each other and the world. This knowledge may address many aspects of multiagent problem solving, but we will focus on two aspects: learning cooperative procedures and learning agents' capabilities. We demonstrate how collective memory reduces the amount of total effort and communication needed by a community of agents to solve problems. Finally, we discuss how this model of behavior is better suited for our domain of interest than related work.

Introduction

We are interested in technical issues in building multiagent systems where agent performance can improve from learning. We call our solution to this problem *collective memory*. Collective Memory (CM) is the resource agents gain through experience and use to improve their performance when interacting to solve collaborative problems. CM can be stored in a centralized memory, in the distributed memories of the individual agents, or in a hybrid such as institutional memory. This work addresses a purely distributed implementation.

It has been argued (Cole & Engeström 1993) that CM plays a central role in the development of distributed cooperative behavior. CM can be a source for the dissemination of community procedural knowledge. When a novice agent collaborates with more experienced agents, the requests and responses of the more experienced agents guide the novice into more efficient patterns of activity. There can be a benefit to the experienced agents as well, since the novice may, in her ignorance, force some innovation to occur by refusing to follow the old way.

We are currently investigating two aspects of learning that CM facilitates:

Learning Cooperative Procedures. Agents maintain a procedural knowledge CM – a casebase of execution traces of cooperative problem solving episodes used as plan skeletons in the future. These execution traces are analyzed and, when possible, improved before storage into the casebase.

Learning Agent Capabilities. Agents maintain an operator probabilities CM – a tree structure used to estimate the probabilities of successfully executing operators. The data underlying this tree are simple, yet powerful enough to improve planner performance and, thus, reduce communication.

The penultimate section discusses how the characteristics of our domain of interest influenced our design of CM. In short, CM must enable learning despite the following constraints of the task environment:

1. Agents have limited knowledge about each other.
2. Communication is necessary.
3. Operators and plans will sometimes fail.
4. Communicate while executing, not planning.

Since communication is considered an action, it will be recorded in the execution trace, and thus agents can learn to reduce the amount of communication in the same manner they learn to reduce their use of other operators. Our thesis is that the mechanisms of collective memory lead to efficient long-term behavior even if short-term behavior is suboptimal.

Collective Memory Task Environment

We consider the task environment to be a problem solving activity populated with multiple adaptive agents (Alterman 1988) of differing abilities and very limited knowledge about each other. The cycle of activity for the community is given below. The difference between steps 3c and 4 is that an agent is assumed to have very little spare time to update her memory while solving the problem, but afterwards she has time to reflect upon her activity in more detail.

1. Give the community a set of goals to achieve.
2. Allocate the goals among the agents.
3. Until all agents have achieved their goals:
   (a) Active agents use CM to create/adapt a plan.
   (b) Agents with plans attempt an operator.
   (c) Make on-line adjustments to CM.
4. Make off-line adjustments to CM.

It is impossible to guarantee that agents will find efficient solutions without using previous experience, since there is no mandated cooperation or coordination strategy. The agents have common top-level goals, but they all have their own point of view on how best to proceed. Through the use of CM, agents start having a "common viewpoint" on how to best achieve a solution. That is, an agent remembers successful patterns of cooperation in which she was involved, and can use them as a basis for future interactions. In novel situations, where no such common viewpoint has been established, there is no mechanism to force the agents to do the most sensible thing; in other words, agents might refuse to assist when they should, or agree to help when they should not.

MOVERS-WORLD

Consider the problem of CM in the context of MOVERS-WORLD. In MOVERS-WORLD, the task is to move furniture and/or boxes from a house into a truck (or vice-versa). MOVERS-WORLD has several agents who are differentiated by their types: some are "lifters" and some are "hand-truck operators". Agents with the same type still have differing capabilities due to attributes such as strength. There are type-specific operators that the agents can use and there are general purpose operators that any agent can perform. The agents in MOVERS-WORLD do not engage in any communication at planning time. Rather, they plan independently, act independently, and only communicate to establish some cooperation when necessary.

CM can impact the performance of agents in MOVERS-WORLD in two ways during step 3a above:

1. An agent can use her past successful problem solving experience to guide her in similar situations rather than to have to plan from scratch to find a solution. An example of a cooperative routine useful for future reuse is for a lifter to learn to load boxes onto available hand-trucks before leaving a room.
2. When an agent has to plan from scratch, her experience in interacting with the domain will allow her to construct a plan that is more likely to succeed, since she has a better idea of both her and other agents' capabilities. An example of improving planner performance based on experience is that if agent A requests help from agent B and receives a negative response, then A's planner will be less likely to construct plans requiring B's assistance in similar circumstances.

Planning and Acting with CM

We propose a structure for CM composed of two parts. The first is a casebase of previous problem solving experiences, which the agents can rely on in lieu of their planner. The second is a tree structure, which can be used to estimate the probability of success for operators (the probabilities are initially 50%).

When using this structure of CM for planning (step 3a above), the agents refer to their casebase of procedural knowledge CM whenever there is a change in their top-level goals or a significant change to their task environment (e.g., changing locations). When either there are no such changes or no relevant plan is found in the casebase, the agent adapts the current plan or plans from scratch.

When planning from scratch, the agent uses the operator probabilities CM to guide it through the search space in a number of ways:

1. Individual agents use a hierarchical, best-first, adaptive planner that sorts plans by their likelihood of success; the probability of success for a plan composed of several operators is the product of the operators' probabilities.
2. Operator success probabilities are used to guide role-binding selections, i.e. the planner selects role bindings that seem most likely to lead to a successful operator. More specifically, the role-bindings are also selected via a separate best-first search, which is ordered by:

   $$\text{P (planner satisfying preconditions)} \; * \; \text{P (agent can execute the operator)}$$

   The former probabilities are difficult to estimate precisely, and currently are only non-unity for preconditions that require interaction with other agents.
3. Probabilities are also used to constrain the search space of the planner. Each agent has a frustration level (or progress estimator), and the planner throws away plans that are less likely to succeed than the threshold associated with the frustration level. So,

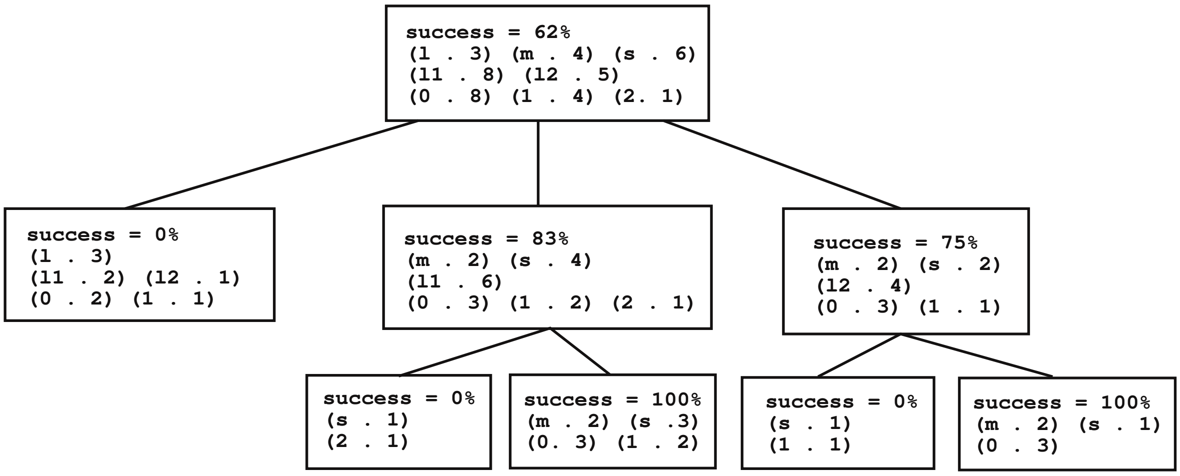

Figure 1: COBWEB tree for calculating operator probabilities.

as she becomes more frustrated, the agent is willing to attempt to undertake plans that are less likely to succeed.

We consider the use of operator probabilities be a form of CM, yet we do not so consider traditional methods of improving planner performance such as those used in SOAR (Laird, Rosenbloom, & Newell 1986), PRODIGY (Minton *et al.* 1989), and DAEDALUS (Langley & Allen 1991). This is because we are encoding execution time knowledge, not planner knowledge. The output of the planner can be different even with identical state and problem descriptions.

Learning Cooperative Procedures

In the interest of brevity, the following is a summary of how the casebase of previous problem-solving knowledge can be maintained. The details of how CM uses prior problem-solving experience to learn cooperative procedures are given in (Garland & Alterman 1995).

Updating a procedural knowledge casebase is too ambitious an undertaking for the agents to accomplish on-line. Off-line, agents have the time to analyze their performance in order to update their procedural knowledge casebase. Therefore, after the community of agents solves a problem, each agent evaluates her execution trace and stores it in CM after a three step process:

1. Summarize the execution trace.
2. Improve the summarized trace.
3. Fragment the improved trace.

The purpose of the first step is to reduce the amount of information that is stored in memory (and thus simplify the remaining steps and later adaptive effort). The second step attempts to remove inefficiencies in agent behavior. The third breaks the execution trace into useful pieces of information. Those pieces of information are then placed in collective memory in a manner that makes them retrievable under similar circumstances in future problem-solving situations.

Cased based reasoning (Kolodner 1993) can be a powerful mechanism for reducing communication in a multiagent domain, as has been previously demonstrated in (Sugawara 1995; Ohko, Hiraki, & Anzai 1995). We expect that the reductions in communication shown in those works will carry over to our domain even though their benchmark systems used more communication. Furthermore, we expect CM to learn more efficient behavior for other operators as well.

CBR does have potential drawbacks, especially in a distributed setting, but they are manageable. First, there is a potential scaling problem, i.e. as the casebase gets larger, performance degrades because retrieval time increases. However, there are techniques to constrain the size of a casebase such as (Minton 1990; Smyth & Keane 1995). Second, in a multiagent setting, global difficulties can arise when individuals use local criteria to determine the best case to retrieve as discussed in (NagendraPrasad, Lesser, & Lander 1995). In MOVERS-WORLD, any such inconsistencies are resolved in the exact same manner as when the plans were generated from first principles, i.e. discrepancies between these competing versions of cooperative behavior are resolved via communication.

Learning Agent Capabilities

It is a question for CM to determine how to best organize information about the capabilities of the agents. In this paper, we present a distributed model where each agent learns what she can through her own interactions with other agents and the world. We might also consider other organizations, such as a centralized or institutionalized CM to handle this. For example, in MOVERS-WORLD there is no explicit representation of a task assigner. In future research, a task assigner might be explicitly represented, and she would also use a form of CM to learn agent capabilities in order to better allocate tasks among the agents.

Agent capabilities are modeled very simply in MOVERS-WORLD – by estimating the probability of success for operators that might be attempted. The successes and failures of attempted operators are stored in a COBWEB (Fisher 1987) tree associated with all the observable features of the various role fillers for the operator. We assume that operators might fail for reasons not explicitly considered (c.f., the qualification problem) and the COBWEB tree can handle this noisy data.

Another advantage of COBWEB is that the incremental nature of updating the tree allows the agents to update it on-line. To help combat COBWEB's dependence on the order of data that it receives, the tree is reclustered off-line at the end of the problem-solving episode.

The planner can then determine success probability by either:

1. Direct past experience if that same operator with the same observable features has been attempted before.

2. Indirect past experience otherwise. This means that the current operator is classified in the COBWEB tree by the features for current role fillers, and the probability is determined by the algorithm in Figure 2.

See Figure 1 for a sample COBWEB tree that can be used to predict the probability of success for agent A, who is attempting to load an object onto a hand-truck with another agent[1]. The three features shown in the figure are object size ({large(l), medium(m), small(s)}), agent to assist ({lifter1(l1), lifter2(l2)}) and the number of objects already on the hand-truck ({0, 1, 2}). In a given node, each feature value is paired with the number of observations in the sub-tree rooted at that node that have the same feature value. In the domain underlying the data in this tree, large objects cannot be loaded onto a hand-truck and lifter2 is incapable of stacking two boxes onto a hand-truck, whereas lifter1 has no problem doing so.

So, if the tree shown in Figure 1 were used to estimate the probability of success for an operator that

[1] Lower branches have been trimmed from this figure.

```
factor = 1
probability = 0
node = COBWEB-node(operator,features)

if (is-leaf(node)==TRUE) then
  count = node-size(node)
  while (count > 0)
    factor /= 2
    probability += success(node) * factor
    --count

node = parent(node)
while (node != NULL)
  factor /= 2
  probability += success(node) * factor
  node = parent(node)

probability += factor * 0.5;
return probability
```

Figure 2: Algorithm to Calculate Operator Probability

was classified into the lower right hand node shown, the algorithm would compute the probability to be 80%. (Even though no children are shown, that node is not a leaf since it has heterogeneous values for the object size feature. Had the node been a leaf with a count of 1,2, 3, or 5, the probability would have been 80%, 90%, 95%, or 99% respectively.) The error of that prediction can be computed based upon whether the attempt was successful or not. For example, if the operator succeeded, the error would be 20%. On the other hand, if it failed the error would be 80%.

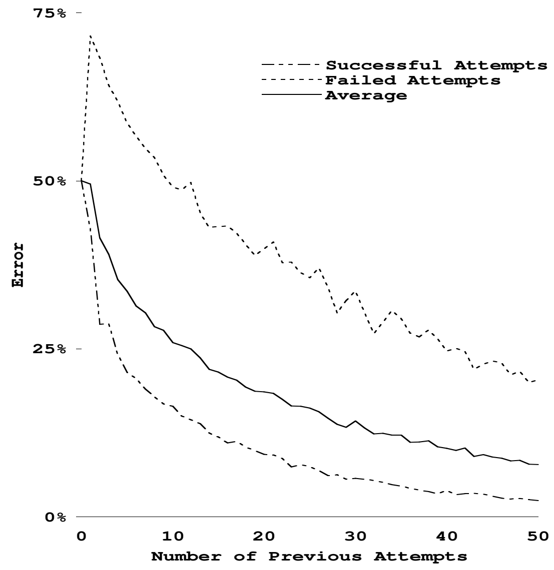

Figure 3: Predicting operator success with reclustering

Figure 3 shows a graph of the errors associated with using a COBWEB tree to predict operator success for 50 successive, randomly generated attempts (averaged over 2500 trials). The system was quite accurate at predicting successful attempts, with an error of under 10% after 16 attempts; however, even after 50 attempts, failed attempts had an error over 21%. For these problems, 70% of the attempts were successful, which is why the average values are closer to the successful attempts' values.

The results in Figure 3 are determined using periodic reclustering of the cobweb tree to simulate the end of the problem solving sessions. The number of attempts an agent made before reclustering was selected from (1, 2, 3, 4, 6, 7, 8) with probabilities (6.2%, 11.4%, 20.6%, 22.5%, 17.6%, 12.0%, 5.9%, 2.9%). The reclustering algorithm used was quite simple: remove all intermediate nodes and then rebuild the tree using the same methods that COBWEB would normally use. As Figure 4 shows, this type of reclustering did not lower the error rates significantly. Two subjects for future research are to investigate whether this is because the example was oversimplified (by only using three features) and to measure the performance of more sophisticated reclustering techniques.

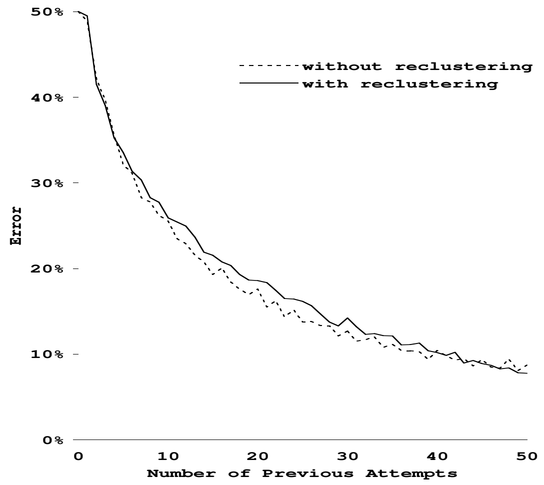

Figure 4: Effect of reclustering

As a result of her experience, A is less likely to create plans that contain failure-prone operators. For example, she is unlikely to request assistance from lifter2 when there are already objects on the hand-truck, and she is unlikely to request assistance from either lifter1 or lifter2 for large boxes. A will, therefore, initiate wasteful communication less often.

Discussion

We will now discuss the constraints of the task environment that CM must accommodate.

Agents have limited knowledge about each other. Agents have no *a priori* knowledge of other agents' capabilities; instead, the other agents are considered black boxes. Since the only way to characterize black boxes is to construct a mapping between inputs and outputs, an agent can only assess abilities based upon the success or failure (the output) of attempted operators (the input).

Communication is necessary. Communication is essential in a domain such as MOVERS-WORLD for two reasons. First, the impoverished model of agent capabilities is inadequate to use communication-free techniques, such as the analytic approach of (Genesereth, Ginsberg, & Rosenschien 1986) and the observation-based approach of (Huber & Durfee 1995). Second, there is neither an alternative to communication to allocate more agents to a goal, nor is there an incentive for agents to take on extra goals. So, when the set of agents working on a goal is inadequate, communication must be used to enlist help. Thus, an approach that treats other agents as just another object in the environment, such as (Sen, Sekeran, & Hale 1994), is not appropriate.

Operators and plans will sometimes fail. Agent operators may not work as expected for a variety of reasons including an incomplete domain model (c.f., the qualification problem) and an incorrect assessment of agent abilities. In addition to operator failure, the reasons that a plan might be abandoned include inaccurate or outdated situation knowledge.

Communicate while executing, not planning. If communication must occur, it is desirable to limit the burden it places on the agents. If communication is fast relative to cogitation, then we should freely incorporate it into the planner since the benefits would outweigh a marginal slowdown in planning. However, we feel that this is not a reasonable assumption for many domains since processor technology historically increases at a faster rate than communication technology. We therefore limit our communication to very simple requests and responses. We further assume that this limited communication and action take roughly the same amount of time, which is appropriate for the types of actions we consider in our domain.

Since agents might have to replan frequently due to operator or plan failure, and communication is expensive, and any communication establishing cooperation for steps of a plan that are never executed is wasted, communication should occur while executing rather than while planning. Standard DAI planners, even a functionally accurate planner such as (Corkill 1979), that use communication during planning are therefore not appropriate. It is possible that if a single agent fails, these planners would involve the whole community in replanning and recommunicating. Likewise, a contract net approach (Davis & Smith 1983) would potentially have to constantly revise the bid and task announcements as new information becomes available.

Conclusion

Collective memory is a mechanism through which a community of agents can learn improved patterns of cooperative activity to take advantage of regularities in their domain of problem-solving activity. We have previously shown how a collective memory of cooperative procedures can be used to reduce the effort, and, in this paper, we demonstrated how a CM of COBWEB trees can be used to predict operator probabilities as an impoverished model of agent capabilities. Finally, we discussed how this model of behavior suits our domain of interest better than related work.

References

Alterman, R. 1988. Adaptive planning. *Cognitive Science* 12:393–421.

Cole, M., and Engeström, Y. 1993. A cultural-historical approach to distributed cognitition. In Salomon, G., ed., *Distributed Cognitions*. Cambridge University Press. 1–46.

Corkill, D. D. 1979. Hierarchical planning in a distributed environment. In *Proc. Sixth International Joint Conference on Artificial Intelligence*, 168–175.

Davis, R., and Smith, R. 1983. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence* 20:63–109.

Fisher, D. H. 1987. Knowledge acquisition via incremental conceptual clustering. *Machine Learning* 2:139–172.

Garland, A., and Alterman, R. 1995. Preparation of multi-agent knowledge for reuse. In *1995 AAAI Fall Symposium on Adaptation of Knowledge for Reuse*, 26–33.

Genesereth, M.; Ginsberg, M.; and Rosenschien, J. 1986. Cooperation without communication. In *Proc. Fifth National Conference on Artificial Intelligence*, 51–57.

Huber, M. J., and Durfee, E. H. 1995. Deciding when to commit to action during observation-based coordination. In *Proc. First International Conference on Multiagent Systems*, 163–170.

Kolodner, J. L. 1993. *Case-Based Reasoning*. San Mateo, CA: Morgan Kaufmann Publishers.

Laird, J. E.; Rosenbloom, P. S.; and Newell, A. 1986. Chunking in SOAR: The anatomy of a general learning mechanism. *Machine Learning* 1:11–46.

Langley, P., and Allen, J. A. 1991. Learning, memory, and search in planning. In *Proc. Thirteenth Annual Conference of the Cognitive Science Society*, 364–369.

Minton, S.; Carbonell, J. G.; Knoblock, C. A.; Kuokka, D. R.; Etzioni, O.; and Gil, Y. 1989. Explanation-based learning: A problem-solving perspective. *Artificial Intelligence* 40:63–118.

Minton, S. 1990. Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence* 42:363–392.

NagendraPrasad, M. V.; Lesser, V. R.; and Lander, S. 1995. Retrieval and reasoning in distributed case bases. Technical Report CS TR 95-27, University of Massachusetts.

Ohko, T.; Hiraki, K.; and Anzai, Y. 1995. Learning to reduce communication cost on task negotiation among multiple autonomous mobile robots. In *Working Notes on Adaptation and Learning in Multiagent Systems*, 65–70.

Sen, S.; Sekeran, M.; and Hale, J. 1994. Learning to coordinate without sharing information. In *Proc. Twelfth National Conference on Artificial Intelligence*, 426–431.

Smyth, B., and Keane, M. T. 1995. Remembering to forget. In *Proc. Fourteenth International Joint Conference on Artificial Intelligence*, 377–382.

Sugawara, T. 1995. Reusing past plans in distributed planning. In *Proc. First International Conference on Multiagent Systems*, 360–367.