Preparation of Multi-Agent Knowledge for Reuse

Andrew Garland and **Richard Alterman**
Computer Science Department
Center for Complex Systems
Brandeis University
Waltham, MA 02254

{aeg, alterman}@cs.brandeis.edu

Abstract

It is often possible to envision the ways in which knowledge will be reused. By preparing the knowledge appropriately at storage time, we can simplify the later task of adapting the stored knowledge. Preparation can simplify, remove inefficiencies, and segment the trace data into useful ideas.

In this paper, we apply this principle to a computational model that deals with the problem of a distributed collective memory for multi-agent systems and we provide technical detail about how the experience of the agents is prepared before storage.

1 Introduction

We are interested in technical issues in building multi-agent systems that use a distributed collective memory of previous problem-solving episodes. The basic idea is that a group of heterogeneous agents solve some collective problem. The resulting episode traces are divided up and stored in the collective memory of the problem-solving agents. In future episodes, these traces can be used, and adapted, so as to serve as a basis for improved performance in a similar episode of problem-solving. The notion of storing solutions to previously solved problems is consistent with other models of case-based learning, as well as models of learning based on chunking (e.g., Laird, Newell, & Rosenbloom, 1986).

In this paper, we will describe the techniques and principles we are currently using to prepare the data before storage into collective memory. The current status of our system is that we have several running examples of multiple agents solving problems (in a simulated domain called MOVERS-WORLD) by adapting previous traces of activity. We have not conducted tests that quantify the impact of data preparation on either the efficiency of stored traces or overall community performance. Appendix C contains more information on performance, efficiency, and evaluating the system.

1.1 Collective Memory

The technology used in this paper supports the evolution and maintenance of a distributed collective memory for a community of heterogeneous agents who plan and work cooperatively. The collective memory of a community of agents is defined as the history of previous activities that will be used as a resource for future activities. It has been argued (Cole & Ergestrom,1992) that collective memory plays a central role in the development of distributed cooperative behavior.

The question of collective memory is to determine techniques for storing agent histories in the most useful manner for future problem-solving episodes, i.e. techniques for individual agents to effectively participate in the maintenance of a collective memory. We only discuss a purely distributed collective memory in this paper, but we are also interested in centralized (institutional) and hybrid organizations for collective memory.

1.2 MOVERS-WORLD

Consider the problem of collective memory 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 can be differentiated by their roles, e.g. hand-truck operators and heavy lifters. There are role-specific operators that the agents can use (e.g. PUSH-HANDTR for hand-truck operators and LIFT for heavy lifters) and there are general purpose operators that any agent can perform (i.e., MOVE, SIGN, and WAIT). The SIGN operator initiates communication between two agents and the WAIT operator is a coordination mechanism; both are discussed in more detail in Section 2.3. Hand-trucks can hold only two boxes at a time. Heavy items require two heavy lifters to carry and bulky items, like a piano, do not fit on hand-trucks. Over time, the agents learn cooperative routines for moving objects around in MOVERS-WORLD.

1.3 Execution Traces

In our model, the data prepared and stored in collective memory are segments of execution traces. Others (Veloso & Carbonell, 1993; Laird, Newell, & Rosenbloom, 1986) have argued for what Carbonell (1983) refers to as the *derivational history* model for storage in memory. The notion is that an agent stores in memory a history of the planning session that can be replayed in future planning episodes. Our system stores execution traces rather than derivational histories because in uncertain worlds a derivational history will not adequately reflect the constant tinkering that is done at runtime to accomplish a multi-agent task — and an execution trace will. At runtime, an agent's activity may not go as expected for a variety of reasons, e.g.,

Communication. She could be interrupted by requests from other agents.

Conflict. She could have to contend with other agents over a limited resource.

Infidelity. Another agent could fail to keep a commitment.

Because of this, the execution trace contains more information than any of the plans of the individual agents; the trace includes the history of agent interaction required to solve the problem. This kind of information cannot be extracted from the planning process.

1.4 Knowledge Preparation

In this paper we tackle the issue of how a trace from a completed problem-solving episode is prepared before it is stored in collective memory. There are at least three important reasons why trace data must be prepared before it is stored in collective memory.

Data may be bad. Independent agents are unpredictable. Time constraints can lead to inefficiencies in the plan for an activity. For these reasons, activity traces can reflect inefficient patterns of agent activity.

Data may be too detailed. Many details are irrelevant. Reasoning about the data, to improve and reuse it, can be simplified by removing detail.

Data may contain multiple ideas. A single trace may be used to generate a large set of hypotheses about action sequences that represent regularities in cooperation between agents for a given domain. Each hypothesis is a segment of a trace that accomplishes a set of goals. Over the generations of problem-solving, certain hypotheses, saved in collective memory, will be more useful than others.

2 An Architecture to Support Collective Memory

2.1 The Agent Community

The control structure for the community of agents is not hierarchical nor is planning centralized (Georgeff, 1983). Agents do not share an overarching plan (Durfee & Lesser, 1987), rather cooperative plans emerge from local interactions (Mataric, 1992). The capacities of other agents is not treated as a given (Genesereth, Ginsberg, & Rosenschien, 1986), but this information can be implicitly acquired through the mechanisms of collective memory.

The basic cycle for the system goes as follows:

1. Generating a problem for the community to solve.
2. Solving the problem with the community.
3. Preparing the execution trace for storage.
4. Updating the collective memory.

Agents have identities such that different combinations, or teams, of agents can be used for each cycle of the system. Agents can request advice from an external user during the problem-solving episode (although this is not necessary for the current set of problems).

Improvement in the performance of a community of agents occurs over several generations of problems. When solving initially, agents are satisfied with any solution. However, when preparing the solution for storage, attempts are made to improve the solution that was generated on the fly. In future episodes, fragments of the improved solution may be recalled and further improvements can be made. Thus, through the mechanisms of knowledge preparation and collective memory, the behavior of the community will evolve and improve for a certain class of problems.

2.2 Individual Agents

At any given point in time, an agent's knowledge of the external world is her perceivable environment and a map of the world that she constructs as she goes along. A plan is created for a given goal, in a given context, by either selecting an old plan from memory (i.e., a previous execution trace) or creating a new plan from scratch. Old plans are stored somewhat abstractly so they require some refinement before they are deployed. In either case, agents are assumed to be adaptive planners (Alterman, 1988), so if the plan does not exactly match the current circumstances, differences and changes will be teased out during the ongoing interaction between the agent and the environment. Much of the adaptation of plans (both new and

Time 11	**Plans** HL1	((SIGN HL1 HL2 ((UNLOAD-TOGETHER HL1 HL2 HBOX1 HANDTR1 ROOM2))) (UNLOAD-TOGETHER HL1 HL2 HBOX1 HANDTR1 ROOM2))
	Actions	HL1 places a call to HL2. HL1 starts talking to HL2: "HL2, help me achieve ((UNLOAD-TOGETHER HL1 HL2 HBOX1 HANDTR1 ROOM2))" "HL1, Yes, but you have to wait." "HL2, I'm hanging up."
Time 12	**Plans** HL1	((WAIT HL1 HL2 ((UNLOAD-TOGETHER HL1 HL2 HBOX1 HANDTR1 ROOM2))) (UNLOAD-TOGETHER HL1 HL2 HBOX1 HANDTR1 ROOM2))
	HL2	((MOVE HL2 ROOM1 ROOM2) (SIGN HL2 HL1 ((UNLOAD-TOGETHER HL2 HL1 HBOX1 HANDTR1 ROOM2))) (UNLOAD-TOGETHER HL2 HL1 HBOX1 HANDTR1 ROOM2))
	Actions	(MOVE HL2 ROOM1 ROOM2)
Time 13	**Plans** HL1	((WAIT HL1 HL2 ((UNLOAD-TOGETHER HL1 HL2 HBOX1 HANDTR1 ROOM2))) (UNLOAD-TOGETHER HL1 HL2 HBOX1 HANDTR1 ROOM2))
	HL2	((SIGN HL2 HL1 ((UNLOAD-TOGETHER HL2 HL1 HBOX1 HANDTR1 ROOM2))) (UNLOAD-TOGETHER HL2 HL1 HBOX1 HANDTR1 ROOM2))
	Actions	HL2 places a call to HL1. HL2 starts talking to HL1: "HL1, help me achieve ((UNLOAD-TOGETHER HL2 HL1 HBOX1 HANDTR1 ROOM2))" "HL2, You're ready now, huh? ok." "HL1, I'm hanging up."

Figure 1: Communication, cooperation and coordination between agents HL1 and HL2.

old) that occurs results from communication between agents at runtime.

An agent creates a plan from scratch using a given set of STRIPS-like operators (Fikes & Nilsson, 1971) using a hierarchical search strategy like ABSTRIPS (Sacerdoti,1974). Unlike other distributed planners (e.g., Corkill, 1979), the agents do not use communication in their planning process. This means that the plans generated may not be globally efficient or even feasible. Depending on their type, different agents have different operations that they can perform.

2.3 Adaptation, Communication, Cooperation, and Coordination

Communication is the central mechanism of cooperation, coordination and adaptation in a community of interacting agents.

The agents use a connection-oriented communication protocol, i.e. communication between agents is similar to a telephone conversation. One agent can "call" another and either establishes a connection or gets a "busy" signal (only two agents can participate in a single negotiation). Once a connection is established, communication is limited to a small number of request and response types. Either agent can terminate the conversation. Agents do not have to be in the same location to engage in communication.

Two agents are said to cooperate if they act or work together to achieve some common purpose. Communication is used to request cooperation; an agent's response depends on whether the request is anticipated or unanticipated. An agent always agrees to cooperate on an anticipated request but cooperation is not guaranteed otherwise.

Cooperation is not guaranteed for unanticipated requests since agents have their own private agendas and attitudes about cooperation. For the present, we only assume that agents are neither "lazy" nor "contrary." This means that an agent will be cooperative when she has no current plan or when she has the same plan; in other situations, it is indeterminate whether the agent will wish to cooperate. This attitude falls in the middle of the spectrum of possibilities which ranges from totally cooperative to malevolent (Rosenschien & Genesereth, 1985). Even if an agent is willing to cooperate, she may be unable to do so. However, an agent who is unwilling or unable to assist can propose an alternative.

We use the term 'coordination' to refer to an appropriate sequencing of actions for cooperating agents. Coordination is achieved via the WAIT operator and communication. An agent is in an idle state while a WAIT operator is at the top of her plan — she is waiting for one of two events: her request to be completed or communication indicating that joint action can occur. If she remains idle too long, she will become frustrated and inquire about the status of her

Time	HTO's Action	HL's Action
1	(SIGN HTO HL ((ON BOX4 HANDTR1)))	
2		(LIFT HL BOX4 ROOM1)
3		(LOAD HL BOX4 HANDTR1)
4	(TILT-HANDTR HTO HANDTR1)	(LIFT HL BOX2 ROOM1)
5	(PUSH-HANDTR HTO HANDTR1 ROOM1 STREET1)	(CARRY HL BOX2 ROOM1 STREET1)
6	(STAND-HANDTR HTO HANDTR1)	(LOAD HL BOX2 TRUCK1)
7	(SIGN HTO HL ((ON BOX4 TRUCK1)))	
8		(UNLOAD HL BOX4 HANDTR1)
9		(LOAD HL BOX4 TRUCK1)
10	(TILT-HANDTR HTO HANDTR1)	(MOVE HL STREET1 ROOM1)
11	(PUSH-HANDTR HTO HANDTR1 STREET1 ROOM1)	(LIFT HL BOX1 ROOM1)
12	(STAND-HANDTR HTO HANDTR1)	(CARRY HL BOX1 ROOM1 STREET1)
13	(SIGN HTO HL ((ON BOX1 HANDTR1)))	
14		(CARRY HL BOX1 STREET1 ROOM1)
15		(LOAD HL BOX1 HANDTR1)
16	(TILT-HANDTR HTO HANDTR1)	
17	(PUSH-HANDTR HTO HANDTR1 ROOM1 STREET1)	
18	(STAND-HANDTR HTO HANDTR1)	
19	(SIGN HTO HL ((ON BOX1 TRUCK1)))	
20		(MOVE HL ROOM1 STREET1)
21		(UNLOAD HL BOX1 HANDTR1)
22		(LOAD HL BOX1 TRUCK1)

Figure 2: An execution trace after moving three boxes onto the truck.

request. This makes the WAIT operator a means of coordinating cooperating agents. Agents use communication to determine whether they are cooperating and if so, whether they need to push a WAIT operator onto the plan. Thus, after communication and through the use of WAIT operators, the involved agents are coordinated in their current plans (or they discover that they are not cooperating).

An example of communication, cooperation and coordination between two heavy lifters, named HL1 and HL2, is shown in Figure 1. At time 11, the next operator in HL1's plan is SIGN which initiates the first communication. This is an unanticipated request for HL2, but she chooses to change her current (unshown) plan to cooperate with HL1. Since HL2 must move into the correct room first, she responds that HL1 must wait; thus, HL1 has a WAIT operator to the top of her plan at time 12. At time 13, the next operator in HL2's plan is SIGN which initiates the second dialogue. This is an anticipated request for HL1 since the request matches the WAIT operator on the top of HL1's plan. The agents are now coordinated, so HL1 removes the WAIT operator and the two agents proceed to unload the heavy box from the hand-truck.

3 Preparing Multi-Agent Knowledge

After the community of agents solves a problem, the execution trace for each agent is evaluated and then stored in collective memory 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 is an attempt 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.

Figure 2 gives streamlined execution traces for the heavy lifter (HL) and the hand-truck operator (HTO) after solving the problem of moving boxes BOX1, BOX2, and BOX4 from ROOM1 onto TRUCK1 using hand-truck HANDTR1.

The first and second boxes are loaded onto the truck in the first eight time steps: HL loads BOX4 on HANDTR1, then HL carries BOX2 to STREET1 while HTO pushes the HANDTR to the street, then HL loads the two boxes onto the truck. After this, HL goes back to ROOM1 and carries BOX1 to STREET1, intending to load it onto the truck. HTO has other ideas, though, so HTO asks HL to load BOX1 on HANDTR1. HL agrees (even though it is not very sensible) and the two agents repeat the LOAD/PUSH/UNLOAD sequence of actions.

In this episode, HL chooses to acquiesce to all requests from HTO even though it is not always more efficient to do so at the time. For this particular example, the inefficiency is removed when the trace is improved.

HTO	HL
(SIGN HTO HL ((ON BOX4 HANDTR1)))	(WAIT HL HTO ((ON BOX4 HANDTR1)))
(PUSH-HANDTR HTO HANDTR1 ROOM1 STREET1)	(LOAD HL BOX4 HANDTR1)
(STAND-HANDTR HTO HANDTR1)	(LOAD HL BOX2 TRUCK1)
(SIGN HTO HL ((ON BOX4 TRUCK1)))	(WAIT HL HTO ((ON BOX4 TRUCK1)))
(PUSH-HANDTR HTO HANDTR1 STREET1 ROOM1)	(LOAD HL BOX4 TRUCK1)
(STAND-HANDTR HTO HANDTR1)	(WAIT HL HTO ((ON BOX1 HANDTR1)))
(SIGN HTO HL ((ON BOX1 HANDTR1)))	(LOAD HL BOX1 HANDTR1)
(PUSH-HANDTR HTO HANDTR1 ROOM1 STREET1)	(WAIT HL HTO ((ON BOX1 TRUCK1)))
(STAND-HANDTR HTO HANDTR1)	(LOAD HL BOX1 TRUCK1)
(SIGN HTO HL ((ON BOX1 TRUCK1)))	

Figure 3: A summary of the execution trace after moving three boxes onto the truck.

3.1 Summarize the Execution Trace

At the end of a given problem-solving episode, there is potentially an overwhelming amount of information to process and much of it might not be relevant to future problem-solving episodes. Summarization processes allow the system to focus on the most significant aspects of a given cooperative activity, those parts that would take significant effort to reconstruct. Removing lesser details increases the generality of the resulting trace and can lessen later adaptive effort.

There are three principles that are used in summarizing the trace:

Keep conversations that establish cooperation.

Keep actions that coordinate cooperating agents.

Prune actions that a dependency structure analysis reveals are reconstructible.

Communication must be included in the summary for the agents to establish that they are cooperating in future problem solving episodes. However, some communication is reconstructible, i.e. before doing a joint operator such as UNLOAD-TOGETHER, and so will be removed by the last principle.

Remembering points of coordination simplifies the cooperative interactions between agents in future problem solving episodes. Failure to remember points of coordination can lead to future difficulties even if both agents are working from the same problem-solving fragment. For example, if a heavy lifter reused a fragment composed of (LOAD HL BOX1 HANDTR1 ROOM1) and (LOAD HL BOX1 TRUCK1 STREET1), the planner will rederive the missing UNLOAD operator. However, without a WAIT operator, she might unload HANDTR1 before the hand-truck operator has a chance to push it to the street! To achieve coordination in the summarized trace, we use the following rule:

Convert agreed-to requests into the corresponding WAIT operator.

Applying this rule to the above example will insert a WAIT operator between the two LOADs and ensure that the actions of the heavy lifter and the hand-truck operator are sensibly interleaved.

Reconstructability by the planner depends on the dependency structure implicit in the preconditions and effects of a given operator. For example, if a heavy lifter wishes to LOAD a box which is at ROOM1 onto TRUCK1 which is at STREET1, then the preceding LIFT and the CARRY operators can be reconstructed by backward chaining.

Figure 3 shows a summary of the execution trace from Figure 2 for each agent.

3.2 Improvements to the Summary

During the period of engagement, while the agents are moving things about, the agents are not concerned about finding an optimal solution — any one will do. Later, after the problem-solving episode, the agents have time to reflect on their activity and may be able to improve on their portion of the activity.

The input to this step is the summary of the activity trace. The basic idea is to go through the summary of the trace and attempt to find ways to improve the performance of the individual agent upon reuse of the summary. For example, in the case of the three boxes in the room, we would like the heavy lifter to independently discover that a better strategy is to load two boxes onto the hand-truck, rather than making an extra trip.

Two rules for improving summaries are:

Re-sequence the trace such that all possible actions that can occur do occur before leaving a given setting.

Eliminate redundant operators.

Operators may be redundant before or after the application of the former rule. For example, the second and third PUSH-HANDTR and STAND-HANDTR operators are rendered redundant after re-sequencing. On the other hand, a LOAD-TOGETHER operator

HTO	HL
(SIGN HTO HL ((ON BOX4 HANDTR1)))	(WAIT HL HTO ((ON BOX4 HANDTR1)))
(SIGN HTO HL ((ON BOX1 HANDTR1)))	(LOAD HL BOX4 HANDTR1)
(PUSH-HANDTR HTO HANDTR1 ROOM1 STREET1)	(WAIT HL HTO ((ON BOX1 HANDTR1)))
(STAND-HANDTR HTO HANDTR1)	(LOAD HL BOX1 HANDTR1)
(SIGN HTO HL ((ON BOX4 TRUCK1)))	(LOAD HL BOX2 TRUCK1)
(SIGN HTO HL ((ON BOX1 TRUCK1)))	(WAIT HL HTO ((ON BOX4 TRUCK1)))
	(LOAD HL BOX4 TRUCK1)
	(WAIT HL HTO ((ON BOX1 TRUCK1)))
	(LOAD HL BOX1 TRUCK1)

Figure 4: An improvement of the summarized execution trace after moving three boxes onto the truck.

```
HL-FRAG1: GOALS = ((ON BOX1 TRUCK1))
        (WAIT HL HTO ((ON BOX1 HANDTR1)))
        (LOAD HL BOX1 HANDTR1)
        (WAIT HL HTO ((ON BOX1 TRUCK1)))
        (LOAD HL BOX1 TRUCK1)
HL-FRAG2: GOALS = ((ON BOX2 TRUCK1))
        (LOAD HL BOX2 HANDTR1)
HL-FRAG3: GOALS = ((ON BOX4 TRUCK1)(ON BOX1 TRUCK1)(ON BOX2 TRUCK1))
        (WAIT HL HTO ((ON BOX4 HANDTR1)))
        (LOAD HL BOX4 HANDTR1)
        (WAIT HL HTO ((ON BOX1 HANDTR1)))
        (LOAD HL BOX1 HANDTR1)
        (LOAD HL BOX2 TRUCK1)
        (WAIT HL HTO ((ON BOX4 TRUCK1)))
        (LOAD HL BOX4 HANDTR1)
        (WAIT HL HTO ((ON BOX1 TRUCK1)))
        (LOAD HL BOX1 HANDTR1)
```

Figure 5: The heavy lifter's three resulting plan fragments after moving three boxes onto the truck.

is redundant when it follows a WAIT for LOAD-TOGETHER operator regardless of re-sequencing.

Figure 4 shows the improved trace, using these rules, of the summarized execution trace shown in Figure 3. Note that the plan fragment is not yet optimal since the agents expect to unload the boxes in the same order that they loaded them instead of in the reverse order. However, it is more efficient than their original solution and will require extremely little adaptation to use in similar future problem-solving episodes.

There are dangers in attempting to improve an execution trace. Namely, any changes that are introduced may or may not turn out to be improvements. In the worst case, the changes introduce failure to a pattern of activity that was previously successful. Therefore we adopted a conservative policy towards introducing "improvements." Nonetheless, our two rules were able to produce plan fragments that were more efficient than the original and were not directly producible by the planner.

3.3 Fragment the trace

The final step of preparation is to decompose the improved, summarized trace into useful pieces. In the current design of the system, a useful piece is a plan fragment that corresponds to a portion of the trace which accomplishes some set of top-level goals. The steps of the trace are sequenced. Each step is then annotated by the top-level goals that it achieves.

Plan fragments are defined in two different ways. First, all actions that are annotated with a given goal G are part of the fragment created for goal G. Second, all actions in the trace that occur in the interval between the first time when an action annotated with G appears to when G is achieved are part of the fragment created for all goals that are completed during that time period.

Thus a given action can be in more than one plan fragment. The reason for the second technique is that it captures the interleaving of both dependent and independent goals; this feature is especially important because it remembers cooperative behavior that is not directly producible by the independent hierarchical planners of the agents.

Figure 5 shows three plan fragments derived from the HL trace shown in Figure 4. Note that HL has learned two simple plans for moving one box as well as the original three boxes.

4 Storing Prepared Traces in Memory

Once the trace has been summarized, optimized, and fragmented, the last step is to store the information

Time	HTO	HL
	Execution trace	
8	(SIGN HTO HL ((ON BOX4 TRUCK1)))	(WAIT HL HTO ((ON BOX4 TRUCK1)))
9		(UNLOAD HL BOX1 HANDTR1)
10		(LOAD HL BOX1 TRUCK1)
11		(UNLOAD HL BOX4 HANDTR1)
12		(LOAD HL BOX4 TRUCK1)
	Improved trace	
	HTO	**HL**
	(SIGN HTO HL ((ON BOX4 TRUCK1)))	(WAIT HL HTO ((ON BOX4 TRUCK1)))
		(LOAD HL BOX1 TRUCK1)
		(LOAD HL BOX4 TRUCK1)

Figure 6: The results of re-using an old plan fragment to move three boxes onto the truck.

into collective memory. When plan fragments are stored in memory, they are indexed by the goal they achieve and a description of relevant role fillers. The description is an abstraction and includes a list of predicates that characterize agents, objects, and instruments relevant to the fragment. These predicates are used to determine the similarity of future settings to the stored plan.

5 Re-Using Cooperative Plans

The adaptive capabilities of the agents in the community act as a counterbalance to some of the problems of breaking up traces of cooperative behavior for storage in the distributed collective memory. When solving a problem, the agents check collective memory for plans which have indices which are "similar" to the current setting. From these, the agents select the plan fragment which accomplishes the largest number of current goals. It is possible for the agents to select different plan fragments since they each individually determine what is "similar." When this happens agreement is reached in the exact same manner as when the plans were generated from first principles; i.e., cooperation is achieved for competing versions of cooperating behavior via communication.

In another problem-solving session that both agents consider similar, HL and HTO retrieve the improved fragments produced in Section 3. In the subsequent action, HL can anticipate the requests of HTO. In this case, the plan fragment is not yet optimal since the agents expect to unload the boxes in the same order that they loaded them instead of in the reverse order. During execution, they unload them in reverse (as they must). During the subsequent analysis, collective memory is updated to reflect this so in the future the chunk will be optimal. By reusing her chunk, the heavy lifter can simplify her cooperative interaction with the hand-truck operator and forego the reasoning involved in forming other plans.

Figure 6 shows the end of the second execution trace and the corresponding improvement.

6 Appendix

A Reasoning framework

This paper presents a distributed CBR-based reasoning framework for problem-solving in a multi-agent environment. We were motivated to use preparation, adaptation and reuse in order to exploit regularities in the task domain to enhance future performance. While it is premature to state the specific benefits and limitations in our approach, we have found increased efficiency and solution quality in trial runs. At present, no invariants are guaranteed but we have attempted to maintain the efficacy of the prepared solution traces. Our techniques may not transfer directly to other domains but we expect the principles presented here will.

A.1 Knowledge

The knowledge of our system encodes the traces of previous problem-solving episodes and it is represented as a sequence of actions. The knowledge is used by independent, heterogeneous agents to solve problems in a simulated world and is acquired by the program through interaction with the world. Each agent maintains her own history and we call this knowledge the distributed collective memory of the community of agents.

A.2 Preparation

After the community of agents solve a problem, each individual agent's execution trace is prepared for storage into collective memory. Performance improvements can be gained that are not possible without preparation. The execution trace is prepared in order to remove inefficiencies and unimportant details and to segment it into ideas. A sequence of actions may be beneficially resequenced; preparation exploits this property of the knowledge representation. We will measure the benefits by showing that preparation improves upon adaptation and/or reuse alone.

A.3 Adaptation

Individual agent plans are adapted because the world is not as expected or because one agent communicates a novel plan to another. Adaptation through communication requires that the knowledge representation is uniform for all agents. The benefits of adaptation can be measured by reduced effort to achieve problem-solving tasks. Cooperative plans are gained through adaptation alone.

A.4 Reuse

Previous problem-solving experience is reused because it may be more efficient than plans derived from scratch. The reuse process does not affect either the preparation or adaptation process; adaptation works in exactly the same manner in reuse as in problem-solving from scratch. Reduced computational effort is gained through the reuse process alone if similar problem-solving episodes recur frequently.

B Task

The task is for a collection of heterogeneous agents to collectively solve problems in a domain which is an unpredictable world. The inputs to each problem-solving episode are the problem specification and a collective memory of the community of agents — the unconstrained output is a revised collective memory. The method exploits regularities in the task domain when improving the efficiency of stored plan fragments via step resequencing.

C Evaluation

The hypothesis that preparation before storage can increase the performance of a community of agents when adapting and reusing knowledge will be explored. Community performance is measured by how many time steps it takes to solve a problem. Currently, the system can be evaluated by the efficiency of the traces stored in collective memory, where the efficiency of a trace can be defined to be the performance it would yield upon reuse for the exact same problem. Although this is perhaps not as meaningful as the average expected performance in all situations where the trace could be used since it ignores adaptation costs, it is tractable and provides a reasonable basis for evaluation. When the system is more mature, an empirical evaluation of the merits of data preparation can be determined by tracking the performance of the community through a suite of test problems. No comparisons to other methods are planned at this time. We would eventually like to include both planner and memory costs in a more global evaluation of system performance. A significant contribution of our research will be highlighting the opportunity for, and benefits of, preparing knowledge before adaptation and reuse.

References

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

Carbonell, J. (1983). Derivational analogy and its role in problem solving. In *Proceedings of the Third National Conference on Artificial Intelligence*.

Cole, M. & Engestrom, Y. (1993). A cultural-historical approach to distributed cognitition. In Salomon, G. (Ed.), *Distributed Cognitions*, pp. 1–46. Cambridge University Press.

Corkill, D. (1979). Hierarchical planning in distributed environment. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, pp. 168–175.

Durfee, E. & Lesser, V. (1987). Using partial global plans to coordinate distributed problem solvers. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pp. 875–883.

Fikes, R. E. & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3):189–208.

Genesereth, M., Ginsberg, M., & Rosenschien, J. (1986). Cooperation without communications. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pp. 51–57.

Georgeff, M. P. (1983). The representation of events in multi-agent domains. In *Proceedings of the Third National Conference on Artificial Intelligence*, pp. 125–129.

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

Mataric, M. (1992). Designing emergent behaviors: From local interactions to collective intelligence. In *From Animals to Animats 2*, pp. 432–441.

Rosenschien, J. & Genesereth, M. (1985). Deals among rational agents. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pp. 91–99, Los Angeles, CA.

Sacerdoti, E. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135.

Veloso, M. & Carbonell, J. (1993). Derivational analogy in prodigy: Automating case acquisition, storage, and utilization. *Machine Learning*, 10:249–278.