Indexing Cases for Planning and Acting in Dynamic Environments: Exploiting Hierarchical Goal Structures[1]

Stephen Robinson
Janet Kolodner
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280
E-mail: robinson@cc.gatech.edu, jlk@cc.gatech.edu

Abstract

We examine how acting in dynamic, complex, not entirely predictable environments affects the indexing, storage and retrieval of cases in a memory-based system. We discuss how a hierarchical goal structure can be exploited to provide indices for searching and storage when planning and acting in everyday environments under time pressure. The tradeoffs between the costs and utility associated with attempting to prevent repeating a failure or missing an opportunity are briefly examined. Considering these tradeoffs leads to distinguishing between when failures can be allowed to recur and when they should be anticipated and avoided. The amount of effort expended when handling failures differs for the two situations, but in both cases a hierarchical goal structure can be used to choose effective indices efficiently. This paper describes the approach taken in our EXPEDITER[2] system and briefly compares it to other approaches.

Acting in Complex, Dynamic Environments

Consider the following example of acting in a complex, dynamic, not entirely predictable environment and the difficulties for indexing and retrieving information that it highlights.

> Bob gets himself and his two children ready for work and school in the mornings the same way every day. The routine includes fixing everyone cereal for breakfast. One day, however, after putting out the bowls and spoons, Bob finds no milk in the refrigerator.

At this point Bob is faced with a number of tasks:

- he still has to fix something to eat;
- he may have to undo some work he has already done (e.g., putting away unused utensils);
- he may want to consider how to avoid running out of milk again;
- he may want to consider how he might reuse the alternative actions he chooses if they succeed.

[1]This research is supported in part by DARPA contract F49620-88-C-0058 monitored by AFOSR and in part by a President's Fellowship from Georgia Tech.

[2]EXploring Planning and Execution in Dynamic Environments employing Routines, an exploratory system currently being developed.

Bob cannot just stop what he is doing and take time to deeply analyze the situation and generate many options to proceed. He has to continue to act while he is dealing with the failure. The amount of effort he can put into analyzing the future consequences of what he chooses to do depends upon how much time he has available and how much effort it takes to repair the current failure, to analyze costs and payoffs, etc.

Continuing with the example, we see there are many factors to be considered when deciding not only what to do immediately, but how current decisions may affect future actions.

> Bob sees no suitable substitute for milk and cannot think of one. He decides that he does not have time to buy more right now. He needs something else to eat that is as easy and nutritious as cereal. Seeing bread in the refrigerator, he decides to serve toast with jelly and orange juice.

If the toast satisfies everyone and Bob decides the extra work of putting away the unused cereal utensils is not significant, he will probably not be concerned with running out of milk again. All he needs to be concerned with is remembering the "toast substitution" so he can use it if the same failure happens again. He is willing to face the failure of running out of milk in the future because it is not worth expending the effort needed to consciously alter his routine to avoid the problems that arise.

However, suppose Bob's children are upset about having to eat "boring, dry toast" in place of cereal and the ensuing commotion makes Bob late. Not being a strict disciplinarian, Bob decides that he would rather make sure that he can serve cereal than deal with upset children on another morning. Now Bob's motivation to avoid the failure is stronger so he may take some time to think of how he can alter his routines in the future to avoid this problem. He also may think of ways to remind himself to make sure he has milk available at a time when he can get more if he does not.

Indexing Cases While Acting

At the heart of memory-based systems that reuse plans and learn from experience is the storage and retrieval of cases. Storing information about experiences for later retrieval requires that the episodes be labeled in such a way that they will likely be retrieved when they are most useful in the future. The problem of labeling memory and coming up with labels during problem solving to retrieve useful past experiences is often referred to as the "indexing problem." While considerable research has been done on the indexing problem, little has been in the context of acting in dynamic environments. Planning and acting in complex, dynamic environments adds additional constraints and considerations to indexing processes.

- Time is a limiting factor since the environment may change while an agent is engaged in reasoning.
- Repair of the ongoing situation is necessary; there is no going back.
- An agent is concerned not only with accomplishing tasks and satisfying goals in the current context, but also with continued success in the future.
- The richness of complex environments may require an agent to filter through available information to get usable indices.

While systems like CHEF (Hammond, 1989a), RUNNER (Hammond *et al.*, 1990), PLEXUS (Alterman, 1988; Alterman, 1986) and FLOABN/SCAVENGER (Zito-Wolf & Alterman, 1990) are concerned with repairing failures and with success in the future (through adaptation, anticipation of failure and opportunity, etc.), they do not balance the cost of indexing memory to enable such anticipation against the utility of doing so. At first glance, it seems that an agent would always want to avoid repeating failures and to take every opportunity to optimize its activities. However, in environments where changes can occur and time to act is limited, expending effort to carefully augment plans for future use may cause additional failures or missed opportunities in the current time frame. There are a number of tradeoffs to consider.

- The time spent reorganizing memory takes time away from the current task, but
- reorganizing memory may result in improved actions in the future by contributing to
  - optimizing activity and
  - avoiding failures.
- Knowledge may be gained by adding cases to memory but the knowledge may be erroneous.
- Reorganizing memory may result in higher matching costs when searching for repairs, etc., in the future.

These tradeoffs should be considered when decisions about indexing and search are made. In particular, easy ways of doing the required indexing must be found for those situations in which the harder work of blame assignment needed to anticipate failure situations is deemed inappropriate. In this paper, we look at the ways indexing and search processes can take advantage of the hierarchical goal structures employed in planning to generate indices and annotate memory when expectation failures occur. An expectation failure can be either a failure or a previously unnoticed opportunity to improve. We show how indexing can be accomplished quickly when the cost of repeating a failure is not deemed to be high and time is scarce in the current situation. We concentrate on everyday environments[3] where activity is repetitive, allowing an agent to develop habitual ways of accomplishing goals.

One of our goals is to show how "routines" can be acquired in a memory-based planning and acting system via repairing failures and noticing opportunities for optimization. A routine is a set of actions ("primitive" steps or operators) specific to a particular situation and environment which an agent executes to achieve a set of goals. The execution order of operators in a routine emerges with experience as failing conditions and interactions are dealt with. Employing a routine helps reduce the computational load required to achieve repeated conjuncts of goals by allowing the agent to ignore preconditions, interactions, operator ordering, etc., because those details have been iteratively ironed out during the evolution of that routine. While we do not address the full set of issues associated with learning routines in this paper, it should be noted that the processes described here are part of a larger system that aims toward that capability.

It is also important to note that "routinization" does not simplify an agent's interactions with a dynamic environment. Rather, it simplifies how an agent reasons about its environment. An agent can view a familiar dynamic environment as being more stable than it really is because the agent has learned to reliably predict how the aspects it is concerned with will change and interact. An agent can assume that its expectations about an environment and its own actions will hold without checking their validity. When expectations fail, the experiences are stored to be later retrieved and employed to avoid or cope with similar failures. This is why routinization is iterative. This paper presents a method for choosing indices and organizing case memory under the time pressures of a dynamic environment that uses a hierarchical goal structure to provide readily accessible context.

EXPEDITER operates in a small simulated work area occupied by various objects such as an oven, clothes, washing machine, dryer, bread, etc. Objects, goals and operators are represented in a simple frame system patterned after the organization used in PLEXUS (Alterman, 1988). Each object, including the agent, has associated with it message handlers which govern how the object interacts with the rest of the environment at each "time" tick. The agent is limited in how much it can process at each tick, an important consideration we are working to define more carefully.

Choosing Indices Under Time Pressure

The best indices are general enough to be applicable in a sufficiently large number of situations yet concrete enough to be recognized. Choosing good indices is hard because it is difficult to determine features that will be predictive in future situations. Part of the difficulty is in not knowing the circumstances in which the case could be useful in the future. The indexing process needs predictive information at a useful level of generality from which to derive indices. An agent acting in everyday environments has some advantages. The routines, failures, repairs and opportunities it will index

[3]See Agre's discussion of everyday environments in (Agre, 1988).

are all likely to be important in the future under the same circumstances due to the cyclic nature of such environments.

The question we must address is what "the same circumstances" means. Consider Bob in the example above. He may discover he has no milk when he goes to the refrigerator, or he may remember there is no milk as he walks into the kitchen to make breakfast or any time up to the time he looks in the refrigerator. He may also discover that he is missing utensils or cereal. In a sense, all of these are the same circumstances (it is breakfast time and cereal cannot be served). Certainly, all can be repaired with the same fix (serve toast and jelly). In other senses, however, the circumstances are different. In the first situation, Bob is looking at the place where the milk should be. In the other situations, he is not. Furthermore, he has completed varying amounts of the routine leading up to getting the milk. Clearly, the first instance of not having milk (the one where he figured out that toast and jelly was a good substitute for cereal) must be indexed so that it can be recalled in all of these circumstances. And clearly, since time is of the essence, the method of choosing indices and storing cases must be fast. Search through memory needs to be efficient for the same reasons. When cases are sought to provide repairs for failures, a quick response is needed to allow action to continue.

Exploiting Hierarchical Goal Structure

The organization of a hierarchical goal structure can be exploited to meet these needs. In this section, we will illustrate how the goal structure can be used to organize the dynamic memory (Schank, 1982) of which it is a part. Consider Bob again. The circumstances under which he needs to remember substituting toast and jelly for cereal are when he has the goal of having cereal for breakfast and cannot. While he may not have been consciously considering that goal as he was going through the routine of getting spoons and bowls, getting cereal, getting milk, etc., it does organize that set of actions. Using that goal as an index to the "toast and jelly" case makes it accessible no matter when during the action sequence the agent discovers that cereal cannot be served.

Such an index can be chosen quickly if memory is organized in goal/subgoal hierarchies that also organize actions according to the subgoals they achieve. Figure 1 shows part of one of these hierarchies. When an action fails, the subgoals it is instrumental to can be found by traversing up the hierarchy. The subgoals found this way become indices for the new situation. Thus, when Bob looks in the refrigerator and finds no milk, he can quickly choose the indices "trying to `grasp-carton` and there is no milk," "trying to `get-milk` and there is no milk," "trying to `prepare-cereal` and there is no milk," and "trying to `have-cereal` and there is no milk." The "toast and jelly" case will be indexed in all of these ways. The figure also shows this configuration.

The hierarchy can also be exploited during search. If a routine action fails and there is no case indexed under it, traversing up the goal/subgoal hierarchy will quickly find goals under which a case might be found. Thus, if Bob remembers he has no milk while he is getting cereal from the pantry, he can find the "toast and jelly" case by traversing up the hierarchy from `grasp-cereal-box` to `get-cereal` to `prepare-cereal`, which serves as an index along with "no milk" to the "toast and jelly" case.

A goal hierarchy preserves relationships between general and specific goals which can be used to select indices pertinent to a problem solving situation at varying levels of generality. When an agent is executing steps of a routine, it is not concerned with the details of all the goals it is achieving. It is only immediately privy to information about the step it is executing. However, it can follow subgoal links up the hierarchy and generalize its knowledge of the current situation by inspecting the goals it finds at the higher levels. When it needs to search for or to index a way to handle an exception, it has simple, directed access to indices which are both generally applicable yet pertinent to the current context.

In the following sections, we discuss our implementation of this scheme in EXPEDITER and the control issues that must be addressed to make it work.

Implementing the Hierarchical Memory. EXPEDITER's memory has nodes corresponding to goals and subgoals that are connected to each other according to goal/subgoal decomposition relationships. The agent knows, for example, that `get-ready-for-work` can be decomposed into `eat-breakfast` and `get-dressed`. Goal/subgoal decomposition links are bidirectional. They can be used both to find subgoal decompositions and to find goals a subgoal is instrumental to. These connections are illustrated in Figure 1 in a small portion of EXPEDITER's memory.

The lowest level goals have action sequences associated with them that, when carried out, achieve the goal. `Get-cereal`, for example, is carried out by the sequence `goto-pantry` followed by `grasp-cereal-box`. Action nodes are connected to each other by sequencing links. `Goto-pantry` is followed by `grasp-cereal-box` which is followed by `goto-cabinet`, etc. Subgoals are connected to each other in context and so are actions. Thus, because `get-cereal` and `get-bowls` are both subgoals of `prepare-cereal` and in that context are carried out in a specific order, the sequencing of the actions of each one is specified in that context. We assume that these sequences come from cases in which the same sequence of actions is carried out over and over again.

Exceptions to routines are indexed from these memory structures. In essence, each node in memory can act as a "branch point," providing access to both normal and specialized ways of carrying out the action or achieving the subgoal. Figure 1 shows how the "toast and jelly" case is indexed from these memory structures. A pointer to it is placed at each appropriate goal node in the memory structure, and the pointer is labeled with the circumstances under which that branch should be taken. For example, when attempting to `grasp-carton` and there is no milk, rather than going to the table (the normal routine), the `have-toast` branch is followed (its sequence of events is not specified in the illustration, but can be easily imagined). This same branch is attached to the `get-milk`, `prepare-cereal`, and `have-cereal` nodes.

Traversal of the Hierarchy under Normal Circumstances. During normal activity, goals arise through an agent's inter-

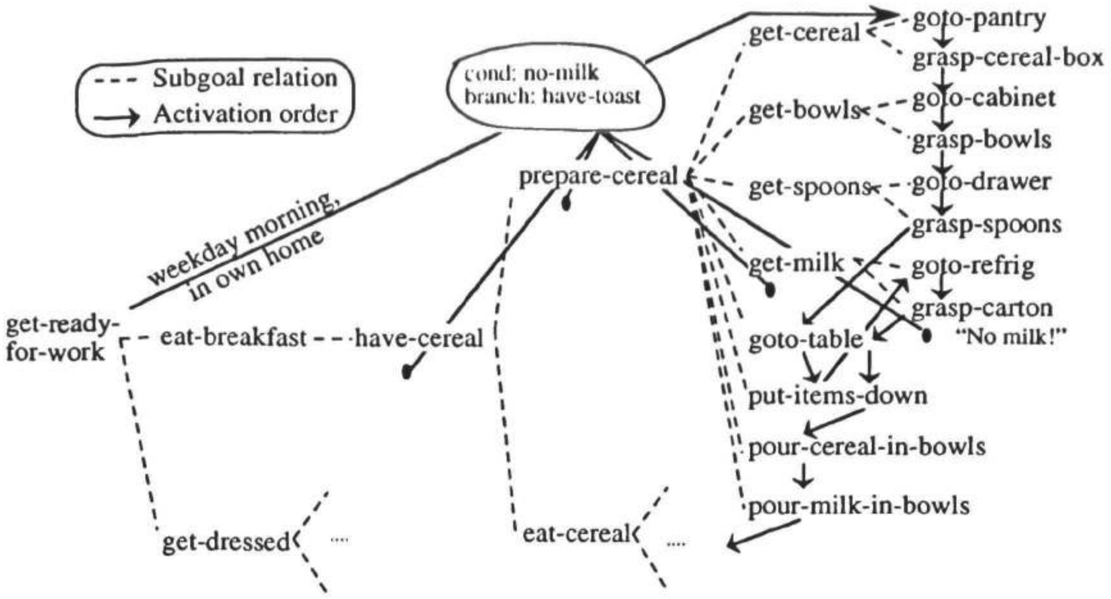

Figure 1: Simplified goal structure and routine with annotations for substituting toast.

actions with the environment. Routine methods for achieving those goals can be found by using features of the goals and the current context as indices to memory. In the normal case of getting up in the morning, for example, the actor's goal of `get-ready-for-work` and the fact that he is in his own home get him to the normal morning routine (beginning with `goto-pantry`). He follows the routine he finds there. When he gets to the `grasp-carton` step, if he has no milk, he follows the `have-toast` branch, otherwise he follows the normal sequencing and continues with `goto-table`.

Indexing a "New" Case. New cases must be placed in memory when a failure is encountered or when an alternate way of doing something is found. Indices to the new case must be general enough so that case can be found in a variety of appropriate circumstances. As discussed above, this is accomplished in EXPEDITER by placing indices on appropriate goals in the hierarchy designating that an alternate branch should be taken. Indices either describe circumstances under which the failure can be expected, describe the failure itself (e.g., no milk), or describe an enabling condition that allows the agent to take advantage of an opportunity to improve (e.g., "if a goal to do wash is active, then branch to ..."). They point to the case that provides the set of steps that has previously repaired the failure (or grabbed the opportunity).

Of course, the process must stop at some point in the hierarchy. One way to choose a stopping point is to only go as far up as the goal that was substituted when repairing a failure. In Figure 1 that point is at `have-cereal`. Going farther up is not warranted because there is no indication that goals above `have-cereal` are affected since milk is neccessary for cereal, not for breakfast in general. We are considering other methods for deciding where to stop, especially for cases where a failure should be anticipated and prevented.

Searching for a Previous Repair. When EXPEDITER encounters a failure, it attempts to find ways in which it has previously repaired similar failures. It begins by looking for indices associated with the failed action that might point to repairs. If no appropriate one is found, it looks next at the parent goal's indices, first for a direct match to the failure, and then for other branches. If `grasp-carton` fails again, there will be a direct match for the failure annotated at `grasp-carton` so no further search is needed. If EXPEDITER "recalls" that it has no milk at `grasp-cereal-box`, it finds a direct match at `prepare-cereal`. If `grasp-cereal-box` fails, EXPEDITER can find the "have toast" branch at `prepare-cereal` because, although "no milk" is not a direct match, `prepare-cereal` is a failing goal shared with a previous failure.

Searching for a "New" Repair. If no particular repair is found, the goal hierarchy can still provide clues that can help in finding a repair somewhere else in memory. Active goals, along with their constraints, reasons, and needed resources can be used to create a more abstract description of the situation needing repair. This more abstract description can be used to search more broadly in memory. For example, if `grasp-carton` fails and EXPEDITER can find no repair, descriptive features of `get-ready-for-work`, `eat-breakfast`, `have-cereal`, etc., can be used to describe the situation. A description including the features "limited time," "easy to fix," and "breakfast food," would be created. In the example scenario, "have toast" was chosen because it was suggested by the presence of bread in the refrigerator, although others may be equally usable. How these features can be selected is an issue we are studying.

In all of this, the goal hierarchy is providing a way to generalize a situation. It allows the choice of indices of varying degrees of generality, starting with very concrete, which are likely to be effective in both the current and future contexts. The indices are available at little cost since the only work is following subgoal links and gathering features from the goals.

Allowing and Avoiding Repeated Failure

When cost/utility tradeoffs are considered, not all exceptions merit equal treatment. For example, in the first scenario in the first section, the failure is deemed to be a minor problem that could be allowed to happen again. However, in the second scenario where Bob's children get upset, the failure is too "costly" to allow it to recur.

Our discussions so far have addressed the first scenario. The agent decides that there is no need to anticipate a failure (it is willing to let the failure recur, or at least not to take action to try to prevent it at the current time), and effort is put only into recording the ways in which it was repaired. The agent will want to recall how the current failure was handled if it is encountered again, but only needs to be reminded if the failure actually recurs.

However, when a failure is deemed too costly to allow it to happen again, it is not enough to index only on the failed goal and its immediate parents since those annotations will be encountered only if the failure happens again. Indices must include goals that will be active and other context features that will be present when failures and opportunities need to be anticipated. While this is a harder problem, we believe that the hierarchical goal structures can again be used to advantage. The extra step of blame assignment is required when a more costly failure occurs or an important opportunity arises. Blame or credit must be assigned to steps taken earlier that set up the failure or to steps which could have been taken to prevent it. We are not investigating the difficult credit assignment problem at this time; however, once an assignment is made, a process similar to that described in the section on exploiting goal hierarchies can be followed. Limited space does not allow us to add details.

How EXPEDITER Relates to Other Approaches

Hammond's study of *agency* (Hammond *et al.*, 1990; Hammond, 1989b) is also based on a dynamic memory representation and the acquisition of "well-tuned, default plans" for particular conjuncts of goals. Like EXPEDITER, the RUNNER system is designed to act in a changing environment, to learn from failure and to recognize opportunities. Our approach differs in at least two important aspects: it provides a method that allows it to choose effective indices rapidly and it recognizes the variable importance of analyzing and processing exceptions. RUNNER's spreading activation is good for recognition and recall but it is not clear how its indexing and search would be affected by considering cost/utility tradeoffs. Also, in Hammond's systems, action is always taken to attempt to prevent failure from happening again and every blocked goal results in marking memory to set up a possible opportunity to satisfy the goal. It is not clear that it is always desirable to do this since the effect on current activity may be deleterious. Our approach has a way to choose useful indices from knowledge of its goals quickly and can adjust the amount of effort it expends on indexing repairs to reflect the usefulness of doing so.

Alterman's work on adaptive planning and "ad-hoc" learning (Alterman, 1988; Zito-Wolf & Alterman, 1990) is similar to our work as well. PLEXUS adapts situated plans it already knows to similar situations in order to avoid a more complex planning problem. The FLOABN/SCAVENGER system also adapts situated plans but in addition learns discrimination points and ad-hoc categories which allow it to apply the adaptations it develops when it encounters the same situations again. SCAVENGER addresses choosing indices from features salient to the system's current knowledge and expending effort on reorganizing the plans used the most. SCAVENGER's method of generating indices seems effective but EXPEDITER goes farther by providing an efficient way to choose effective indices along a range of generality. In addition, EXPEDITER addresses the placement of discrimination points to support anticipation of exceptions and reusing repairs for related failures.

Conclusions

The computational load required for indexing and retrieving cases placed on a memory-based system in a complex domain can be high. Reducing this load becomes more important when the additional constraints placed on a system by dynamic environments are taken into account. However, when everyday, cyclic activity is considered, the hierarchical goal structures used in planning and routine activity may be exploited to lessen the high load.

Our approach addresses difficult problems for indexing cases in dynamic, complex environments. It allows choosing from a broad spectrum of concrete to more general indices, directed by goal knowledge pertinent to current context. It allows good choices to be made when time is limited and better choices to be made as more time is allowed. It is limited to known goal knowledge and may be too closely tied to goal hierarchies but we are working to better understand these issues.

References

Agre, P. E. 1988. The Dynamic Structure of Everyday Life. Technical Report TR 1085, AI Laborabory, MIT, Cambridge, Massachusetts.

Alterman, R. 1986. An Adaptive Planner. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 65–69, Los Altos, California. Morgan Kaufmann Publishers, Inc.

Alterman, R. 1988. Adaptive Planning. *Cognitive Science*, 12(3):393–421.

Hammond, K.; Converse, T.; and Martin, C. 1990. Integrating Planning and Acting in a Case-Based Framework. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 292–297, Boston, MA.

Hammond, K. J. 1989a. *Case-Based Planning: Viewing Planning as a Memory Task.* Academic Press, Inc. (Harcourt Brace Jovanovich, Publishers), New York.

Hammond, K. J. 1989b. Opportunistic Memory. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 504–510, Detroit, Michigan.

Schank, R. 1982. *Dynamic Memory.* Cambridge University Press, New York.

Zito-Wolf, R. and Alterman, R. 1990. Ad-Hoc, Fail-Safe Plan Learning. In *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society*, pages 908–914, Cambridge, MA.