Learning to Prevent Task Interactions

Michael Freed and Gregg Collins
The Institute for the Learning Sciences
Northwestern University
Email: freed,collins@ils.nwu.edu

1 Introduction

Increasingly automated home, workplace, and industrial environments require programs capable of carrying out an ever wider assortment of tasks. As this trend continues, it will become increasingly difficult for computer programmers to anticipate all the ways in which these tasks may interact with one another. One solution to this problem is to attempt to automate the recognition of novel interactions between tasks.

Novel interactions typically occur when tasks are combined in new ways or old combinations are tried in new situations. Detecting such interactions involves two major challenges. First, the type and quantity of resources demanded by a task are often unknown until execution time. For example, an agent charged with picking up and transporting two objects cannot know whether it can pursue the tasks concurrently until it tries to pick the objects up, thereby determining whether they are too heavy to be carried together. Second, general knowledge about coping with interactions will necessarily be abstract and thus difficult to quickly specify into a situation-specific, executable form. On the assumption that this execution-time reasoning will often prove impossible, we see the application of general knowledge to specific problem situations as something that must be learned over time.

Our approach to learning has focussed on two problems in particular. First, we have developed a general framework for learning from failure in which the learning element uses knowledge about its own execution mechanisms to reason about how its decisions might have caused a failure and how they might be modified to prevent recurrence [Birnbaum *et al.*, 1990, Collins *et al.*, 1991, Freed and Collins, 1993]. Second, we have begun to enumerate and represent the kinds of abstract knowledge that agents need to reason about in order to cope with undesirable task interactions. Such knowledge can be integrated into execution mechanisms as the agent learns what kinds of interactions actually occur in a task domain.

We are testing our approach to learning about task interactions using the RAP task-execution system [Firby, 1989] and Truckworld simulator [Firby and Hanks, 1987]. The RAP system has a number of important properties for our purpose. First, it reflects the need to minimize deliberation while executing a task by sharply limiting the amount and kinds of inference allowed at that time. Second, it does not assume that the planner knows the state of the simulated world; instead, the agent can explicitly assess its knowledge and create knowledge acquisition tasks as needed. Many coping strategies depend on the ability to schedule such tasks.

Truckworld is a simulated environment in which a robot delivery truck faces problems such as obtaining fuel, navigating to destinations over possibly obstructed roads and acquiring items needed for delivery. The following section describes an example in which interacting delivery truck tasks cause a plan

```
(DEFINE-RAP
   (INDEX (prep-journey))
   (METHOD method-1
      (TASK-NET
        (t1 (add-oil-if-needed))
        (t2 (inspect-arms))
        (t3 (refuel-if-needed 15)))))
```

Figure 1: Delivery truck RAP for preparing to go on a long journey

failure. We then discuss how the RAP system controlling the truck could be modified to avoid such incidents in the future.

2 Example

Scenario: arriving late at the gas station

Before setting out on a long journey, the delivery truck always checks its oil, inspects its manipulator arms and refuels. Since, in most cases, it does not matter in what order these tasks are executed, the RAP[1] that encodes the journey preparation procedure does not impose any order (see figure 1). On one particular day, the truck readies itself for a journey, arbitrarily deciding to run the refuel task last. First, it takes a few seconds to check the oil. It then inspects each arm, taking as long as an hour to disassemble and visually inspect various arm components. Finally, the truck leaves for the gas station and arrives to find that it has been closed for twenty minutes.

[1]RAPs (reactive action packages) are essentially planning operators designed for use in the time-pressured and uncertain task situations that arise at plan execution time. For a fuller discussion see [Firby, 1989].

Learning from the example

To have avoided this failure, the truck should have executed its refuel task first and then inspected its manipulator arms. One way to prevent a recurrence of the failure is to modify the journey preparation RAP so that whenever the time needed to travel to the gas station plus one hour (to inspect the arms) would bring the truck to the gas station after closing, the refuel task will be executed first. There are several ways the PREP-JOURNEY RAP could be modified to achieve the new behavior:

- by making REFUEL-IF-NEEDED *always* execute before INSPECT-ARMS
- by making REFUEL-IF-NEEDED execute before INSPECT-ARMS whenever both tasks come up for execution past some fixed deadline [e.g. 7:30 pm]
- by making REFUEL-IF-NEEDED execute before INSPECT-ARMS whenever both tasks come up for execution past a dynamically recomputed deadline

The advantages and disadvantages of each modification may enumerated and used to decide what to learn [Krulwich *et al.*, 1992]. The first modification option prevents recurrences of the failure, but reduces the truck's flexibility in potentially harmful ways. For example, if the truck prepares for a journey early in the morning, the first modification prevents the truck from using its time productively — by executing INSPECT-ARMS — while it waits for the station to open. The second modification allows more flexibility in that it only imposes an ordering constraint at a late hour when the risk of arriving at the station after closing is high. There is some loss of scheduling flexibility since the fixed deadline will be too late in some cases, as when the travel distance to the station is unusually long, and too early in others. The third method causes the deadline to be recomputed every time the RAP comes up for execution; this maximizes flexibility and minimizes recurrence

of the failure but raises the computational cost of executing the RAP. Our model of learning uses learning goals to bias selection between alternative execution mechanism adaptations in favor of minimal added computational cost, maximum flexibility and maximum reduction of performance error (i.e. maximum transfer), in that order. Thus, the second modification would be selected.

To incorporate selected modifications, the system uses a set of RAP transformation rules. These are used to carry out modifications selected by the learning element[Hammond, 1989, Collins, 1989]. In this example, the transform called **add-conditional-ordering-constraint:fixed-deadline** is applied to the PREP-JOURNEY RAP. This transform takes as input:

- A RAP and RAP-method to modify:

  ?rap = PREP-JOURNEY

  ?method = method-1

- An absolute time after which the ordering constraint should apply:

  ?deadline = (time-of-day 7 30 pm)

- The task that should be constrained to come earlier:

  ?task1 = REFUEL-IF-NEEDED

- The task that should be constrained to come later:

  ?task2 = INSPECT-ARMS

As output, the transform produces a modified version of the original PREP-JOURNEY RAP containing an extra method in which the specified tasks are ordered (see Figure 2). Ordering constraints are denoted by *for* statements following the task call in a method's TASK-NET. Applicability restrictions, expressed inside a CONTEXT clause, are added to the old and new methods to cause the appropriate one to be selected when the tasks are considered for execution.

```
(DEFINE-RAP
   (INDEX (prep-journey)
   (METHOD method-1
      (CONTEXT
        (and (current-time ?time)
             (< ?time (time-of-day 7 30 pm))))
      (TASK-NET
        (t1 (add-oil-if-needed))
        (t2 (inspect-arms))
        (t3 (refuel-if-needed 15))))
   (METHOD method-2
      (CONTEXT
        (and (current-time ?time)
             (>= ?time (time-of-day 7 30 pm))))
      (TASK-NET
        (t1 (add-oil-if-needed))
        (t2 (inspect-arms))
        (t3 (refuel-if-needed 15) (for t2)))))
```

Figure 2: PREP-JOURNEY after applying add-conditional-ordering-constraint:fixed-deadline

3 Learning from Failure

Our approach to adapting execution machinery is based on the paradigm of *failure-driven learning*, in which the agent relies on the observed failure of specific, monitored expectations to signal an opportunity to learn[Sussman, 1975, Schank, 1982, Hammond, 1989, Birnbaum *et al.*, 1990]. In particular, when an agent expects a plan to achieve its goal, and this expectation is violated, one response is to attempt to determine what aspect of the agent's plan execution machinery was responsible for the faulty behavior, and how that aspect can be modified to avoid the recurrence of such failures.

Monitoring expectations

This first step in this learning process is to generate *expectations*. Since verifying an expectation may entail substantial computational and physical effort, an agent should actively monitor only a subset of the assumptions

underlying its decisions [Doyle *et al.*, 1986, Hunter, 1989, Freed, 1991]. For example, it is worth confirming that a gas station is open before trying to use the gasoline pump, but probably not worth verifying that the pump produces gasoline and not some other substance.

Our current approach to expectation monitoring is very conservative. The vast majority of the agent's assumptions are never checked at all, except possibly during the analysis of a failure. Most others are checked only when confirming or disconfirming information results from incidental sensing actions initiated for reasons unrelated to checking the expectation. In a few cases, verification tasks are incorporated explicitly into execution methods. For instance, an agent might learn to call ahead to confirm that a store is open for business.[2]

Searching for a modification

When performance expectations fail, a learning task is initiated with the goal of modifying execution mechanisms — i.e. RAPs — so as to prevent similar failures from occurring again. This process can be characterized as a search through the space of possible transformation sequences [Gratch and DeJong, 1992]. Four kinds of knowledge are used to constrain the search, including:

- Domain knowledge
- Self knowledge
- General planning knowledge
- Transformation knowledge

The role of domain knowledge is to isolate the agent actions (or inactions) that led to the failure. In the preceding example, the agent can use its knowledge of the fact that gas stations open and close at predetermined times to characterize its performance failure as *arriving too late* at the gas station. This reduces the goal of preventing future failures to the more specific goal of avoiding future attempts to obtain gas from a closed station.

Self-knowledge includes a model of how an agent's decision mechanisms employ RAPs and other memory structures to specify actions and interpret events [Freed and Collins, 1993, Cox, 1993]. Learning processes use this knowledge to localize the cause of a performance failure to some portion of the agent's decision-making mechanism. The principle is identical to the use of models in diagnostic systems [Davis *et al.*, 1982] for finding underlying faults in devices such as electronic circuits, except that it is the agent's own decision processes that are being evaluated as possible causes of failure [Birnbaum *et al.*, 1990, Ram and Cox, 1994]. For example, knowing that a PREP-JOURNEY task can initiate a REFUEL-IF-NEEDED task allows the learning element to hypothesize that a performance failure in the latter task could be caused by an error of the former.

General planning knowledge, especially including knowledge about the way plans fail, can be used to direct the learning process towards useful modifications [Leake, 1990, Owens, 1990, Jones, 1991, Freed *et al.*, 1992]. For example, some of this knowledge describes how a task can fail to meet its deadline and, implicitly, how it may be prevented from doing so in the future. In the gas station scenario, one subtask missed a deadline due to arbitrary selection between subtask execution orders. General planning knowledge can be used to identify this type of failure and direct the learning process to find modifications that will cause the tasks to be ordered appropriately. General planning knowledge is also needed to facilitate the transfer of learned lessons across domains [Krulwich *et al.*, 1990] and to anticipate undesirable side-effects of system modifications [Freed *et al.*, 1992].

[2] In the RAP system, the success or failure of primitive actions is verified automatically. For example, the GRASP primitive will always return whether or not the grasp succeeded.

The role of transformational knowledge [Collins, 1989, Hammond, 1989] is to specify particular methods for modifying execution machinery (RAPs). This knowledge comes into play when learning mechanisms have a well specified learning goal for modifying execution behavior that is represented in general planning terms. Transformation rules translate such a learning goal into a specific system modification, represented as appropriate for system execution machinery. For example, the top-level learning goal of preventing late arrival at a gas station can be specified and redescribed as the goal of sequencing two subtasks in PREP-JOURNEY; this goal could in turn be specified, using a transform, as the goal of appending *(for t2)* to a particular part of the execution system's representation of PREP-JOURNEY. Transforms are the primitive operators of the learning process. [3]

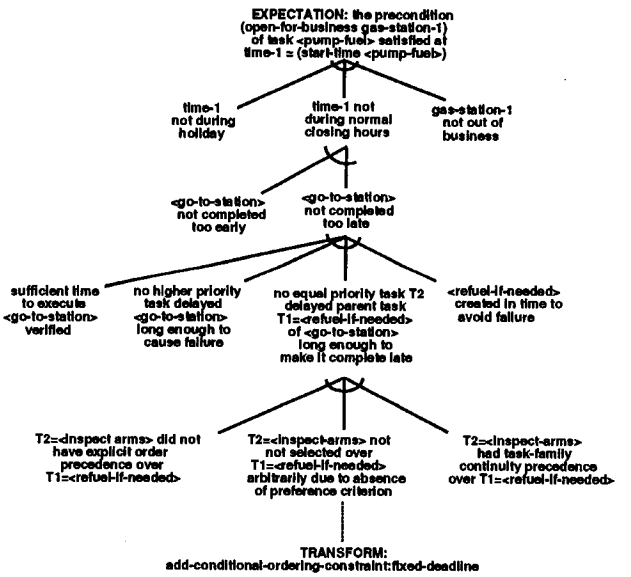

Figure 3: Justification structure used in gas station example

Justification structures

The process of learning from failure has traditionally been seen as taking place in two stages: diagnosis and repair. To be useful for learning, however, the process of diagnosing a failure must somehow be biased to produce explanations that are useful in formulating a repair. If the requirements of the repair process are not taken into account, valid but useless explanations for the failure are likely to result [Leake, 1990, Owens, 1990]. For example, the failure to arrive at the gas station in time can accurately be explained as a consequence of the gas station manager deciding on a particular schedule, or the fact that a closed valve prevents gas from flowing. As neither of these explanations points to a condition under the truck's control, they do not indicate a way to prevent recurrences of the failure.

Diagnosis and repair must be integrated so that useful knowledge influences the learning process as it is needed. The key to this integration is the ability to retrieve and explore a representation of the assumptions underlying any given performance expectation.[4] These representations, *justification structures* [deKleer and Williams, 1987, Simmons, 1988a, Birnbaum *et al.*, 1990], can be designed to combine each kind of relevant knowledge in a way that tightly constrains the search for a useful failure-explanation and apparatus-modification (see Figure 3).

When an expectation fails, its justification structure is retrieved and assumptions immediately underlying the expectation are checked. Figure 3 shows part of the justification structure used to select and specify the transformation rule add-conditional-ordering-constraint:fixed-deadline in the gas station example. In our example, when the

[3]For some purposes, it is important to view learning, not only as a planning process, but as part of the same planning process that controls the agent's sensors and effectors. Learning from failure may require actions to gather diagnostic information from the task environment and must therefore be able to command cognitive and physical resources just like any other task [Hunter, 1990]. Moreover, learning tasks can compete for the same resources as other tasks and should therefore be scheduled right along with them.

[4]People are sometimes unaware of factors that contribute to the success of their actions. This suggests that to diagnose a failure, an agent will sometimes have to generate the (implicit) justification for a failed action. [Agre, 1988, Collins, 1987, Kass, 1990]

refuelling task fails, assumptions immediately underlying the expectation that the gas station would be open are examined. One of the supporting assumptions — that the truck's refuel attempt would not occur during the station's normal closing hours — proves faulty. The system then attempts to explain the failure of this assumption in terms of its immediate supporters. The process continues to follow the path of faulty assumptions through the justification structure until a basis for repairing the system's failure can be found. Diagnosis consists of finding a faulty assumption. To repair the failure, the system's plan execution mechanisms must be modified to no longer depend on the assumption. Thus, the diagnosis process ends when a faulty assumption on which to base a repair has been located.

Learning thus involves "backing up" through the justification structure, recursively explaining the failure of an expectation as the result of failure of one or more of its immediately antecedent assumptions [Smith *et al.*, 1985, Simmons, 1988b, Birnbaum *et al.*, 1990]. At each step in this process, more is learned about what kind of modifications could be used to prevent the failure from recurring. For example, the knowledge that the truck arrived at the gas station during closing hours allows the learner to focus on strategies that help the truck meet fixed deadlines. Thus, progress at producing a failure diagnosis is made concurrently with progress at selecting a repair.

To enable diagnosing a failure to focus the search for a useful repair, parts of the justification structure used for a diagnosis must be associated with appropriate transformation rules (cf. [Minton, 1988, Krulwich, 1991]). For example, the (partial) justification structure in figure 3 includes an assumption that execution mechanisms will not accidentally cause a fatal delay as a result of *arbitrarily* choosing between the tasks <inspect-arms> and <refuel-if-needed>. This is justified by the underlying assumption that such a delay, if it were otherwise possible for it to occur, would have been prevented by previous application of the transform add-conditional-order-constraint:fixed-deadline to the PREP-JOURNEY RAP.

When it is discovered that, lacking any ordering criteria, execution mechanisms did in fact select between the tasks arbitrarily, the justification traversal mechanism finds the faulty underlying claim that add:conditional-ordering-constraint:fixed-deadline will have been applied to REFUEL-IF-NEEDED and INSPECT-ARMS in method-1 of PREP-JOURNEY. Because this faulty claim contains a fully specified transform,[5] it is recognized as providing a way to prevent its parent assumption from failing in the future. Moreover, since the assumption is on a critical path of events leading to the failed expectation, enforcing the assumption also enforces the original expectation, satisfying the overarching goal of the learning process. After applying the transform, the learning process terminates.

4 Conclusion

Agents will often lack the time and/or knowledge to predict and prevent novel interactions between executing tasks. When such interactions occur and result in performance failures, learning processes should be invoked to prevent similar failures from recurring. To support the learning process, a set of monitorable expectations should be generated; when an expectation failure is observed, a justification structure should be used to guide the failure explanation process towards a useful repair. Justification structures combine many kinds of knowledge including self-knowledge of the agent's decision-making apparatus and representations of strategic knowledge and transformation knowledge. Assumptions represented by these structures can index transformation rules to effect repairs when the associated assumption causes a failure. By incorporating knowledge needed for diagnosis and repair into a single structure, justification structures

[5] Variable bindings for the transform are generated while checking intermediate assumptions on the path from expectation to transform.

enable learning process that integrate diagnosis with repair.

We have presented an example of how a system could learn to cope with a novel interaction. We are currently exploring more complicated examples in which an agent employs sophisticated coping strategies such as learning to stabilize the task environment before interrupting a task. An example of this kind is inserting a bookmark before interrupting a reading task. Other kinds of strategies include setting alarms to indicate when an interrupted task should be resumed and speeding progress at a task as soon as it seems likely that an important future task will be delayed.

We are developing our failure-driven approach to learning using the RAP execution system and Truckworld simulator. Truckworld simulates a dynamic domain in which unpredictable events can occur and is thus especially suitable for exploring problems arising from task interactions. The RAP execution system enforces minimal deliberation in the process of carrying out tasks and allows an agent to operate effectively without complete world knowledge. These features allow us to test our assumptions about the relationship between plan execution and learning and allow us to implement the kinds of interaction coping strategies we feel are required of an agent operating in the real world.

References

[Agre, 1988] Philip Agre. *The Dynamic Structure of Everyday Life.* PhD thesis, MIT Department of Electrical Engineering and Computer Science, 1988.

[Birnbaum *et al.*, 1990] L. Birnbaum, G. Collins, M. Freed, and B. Krulwich. Model-based diagnosis of planning failures. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 318–323, Boston, MA, 1990.

[Collins *et al.*, 1991] G. Collins, L. Birnbaum, B. Krulwich, and M. Freed. Plan debugging in an intentional system. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 353–358, Sydney, Australia, 1991.

[Collins, 1987] Gregg Collins. *Plan Creation: Using Strategies as Blueprints.* PhD thesis, Yale University, 1987. Research Report #599.

[Collins, 1989] G. Collins. Plan adaptation: A transformational approach. In K. Hammond, editor, *Proceedings of a Workshop on Case-Based Reasoning*, Palo Alto, 1989. Defense Advanced Research Projects Agency, Morgan Kaufmann, Inc.

[Cox, 1993] M.T. Cox. Introspective multistrategy learning. Technical Report Cognitive Science Technical Report No. 2, Georgia Institute of Technology, 1993.

[Davis *et al.*, 1982] R. Davis, H. Shrobe, W. Hamscher, K. Wieckert, M. Shirley, and S. Polit. Diagnosis based on description of structure and function. In *Proc. AAAI-82*, pages 137–142, Pittsburgh, Pa., August 1982.

[deKleer and Williams, 1987]
J. deKleer and B.C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–129, April 1987.

[Doyle *et al.*, 1986] R.J. Doyle, D.J. Atkinson, and R.S. Doshi. Generating perception requests and expectations to verify theexecution of plans. In *Proc. AAAI-86*, pages 81–87, Philadelphia, PA, August 1986. AAAI.

[Firby and Hanks, 1987] R. James Firby and Steve Hanks. The simulator manual. Technical Report YaleU/CSD/RR Number 563, Yale University, 1987.

[Firby, 1989] R. James Firby. *Adaptive Execution in Complex Dynamic Worlds.* PhD thesis, Yale University, 1989. Available as Report RR-672.

[Freed and Collins, 1993] M. Freed and G. Collins. A model-based approach to learning from attention-focussing failures. In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*, pages 434–439, Boulder, CO, 1993.

[Freed *et al.*, 1992] M. Freed, B. Krulwich, L. Birnbaum, and G. Collins. Reasoning about performance intentions. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, pages 242–247, Bloomington, IN, 1992.

[Freed, 1991] M. Freed. Learning strategic concepts from experience: A seven-stage process. In *Proceedings of the 14th Annual Conference of The Cognitive Science Society*, pages 132–136, Chicago, IL, 1991.

[Gratch and DeJong, 1992] J. Gratch and G. DeJong. An analysis of learning to plan as a search problem. In *Proceeding of Machine Learning 1992*, pages 179–188, Aberdeen, Scotland, 1992.

[Hammond, 1989] K. Hammond. *Case-based planning: Viewing planning as a memory task.* Academic Press, San Diego, CA, 1989.

[Hunter, 1989] L. Hunter. *Knowledge-acquisition planning: Gaining expertise through experience.* PhD thesis, Yale University, 1989.

[Hunter, 1990] L. Hunter. Planning to learn. In *Proceedings of the twelfth annual conference of the cognitive science society*, pages 261–268, Cambridge, Mass., August 1990.

[Jones, 1991] E. Jones. *The flexible use of abstract knowledge in planning.* PhD thesis, Yale University, 1991.

[Kass, 1990] Alex Kass. *Developing Creative Hypotheses by Adapating Explanations.* PhD thesis, Yale University, 1990. In Preparation.

[Krulwich *et al.*, 1990] Bruce Krulwich, Gregg Collins, and Lawrence Birnbaum. Cross-domain transfer in planning strategies: Alternativeapproaches. Technical report, The Institute for the Learning Sciences, 1990.

[Krulwich *et al.*, 1992] B. Krulwich, L. Birnbaum, and G. Collins. Learning several lessons from one experience. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, pages 242–247, Bloomington, IN, 1992.

[Krulwich, 1991] B. Krulwich. Determining what to learn in a multi-component planning system. In *Proceedings of the 14th Annual Conference of The Cognitive Science Society*, pages 102–107, Chicago, IL, 1991.

[Leake, 1990] D.B. Leake. *Evaluating Explanations.* PhD thesis, Yale University, 1990. In Preparation.

[Minton, 1988] S. Minton. *Learning Effective Search Control Knowledge: An Explanation-Based Approach.* PhD thesis, Carnegie Mellon University, 1988.

[Owens, 1990] C. Owens. Representing abstract plan failures. In *cogsci90*, pages 277–284, Cambridge, MA, July 1990. The Cognitive Science Society.

[Ram and Cox, 1994] A. Ram and M. T. Cox. *Introspective reasoning using meta-explanations for multi-strategy learning.* Morgan Kaufmann, 1994.

[Schank, 1982] R.C. Schank. *Dynamic Memory.* Cambridge University Press, Cambridge, England, 1982.

[Simmons, 1988a] R. Simmons. *Combining associational and causal reasoning to solve interpretation and planning problems.* PhD thesis, MIT AI Lab, 1988.

[Simmons, 1988b] R.G. Simmons. A theory of debugging plans and interpretations. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, St. Paul, MN., 1988.

[Smith *et al.*, 1985] Reid G. Smith, H. Winston, T. Mitchell, and B.G. Buchanan. Representation and use of explicit justifications for knowledge base refinement. In *Proc. of IJCAI-85*, pages 673–680, Los Angeles, CA, August 1985. IJCAI, Inc.

[Sussman, 1975] G.J. Sussman. *A Computer Model of Skill Acquisition.* American Elsevier, New York, 1975. based on Phd thesis, MIT, Cambridge, MA, 1973.