

Adapting Routines to Improve Task Coordination

Michael Freed and Gregg Collins
The Institute for the Learning Sciences
Northwestern University
1890 Maple Avenue
Evanston, Illinois 60201
collins@ils.nwu.edu, freed@ils.nwu.edu

Abstract

Human agents typically evolve a set of standard routines for carrying out often-repeated tasks. These routines effectively compile knowledge about how to carry out sets of interacting tasks without causing harmful interference. By modifying its routines in response to observed failures, an agent can refine and enlarge its planning repertoire over time. We are constructing a software agent, *RAPTER*, that applies this incremental improvement strategy to the management of planning routines for use in the *Truckworld* simulator. Currently, we are building in the capacity to make a variety of repairs to the method used in a routine to coordinate constituent tasks; in the future, *RAPTER* will be extended to include methods for coordinating tasks from different routines.

1 Introduction

In highly automated industrial and workplace environments, computer programs must be able to coordinate the execution of diverse sets of tasks. Harmful interactions such as one task undoing progress from a previous task, or two tasks deadlocking over resources must be avoided. However, predicting such interactions between tasks imposes seemingly overwhelming demands on computational resources (Chapman, 1987). To keep the problem of avoiding task interactions manageable, human planners employ a two-part strategy that programs would do well to emulate: First, they develop and follow routine plans designed to avoid harmful interactions rather than generate an original plan each time a goal becomes active. For example, most people fill up their vehicles' gas tanks even when they don't have any particular driving tasks in mind (Wilensky, 1983). As a result, fuel is usually available; tasks which might otherwise come into conflict because of limited fuel (or the time it would take to acquire fuel) are less likely to do so. Second, whenever task interactions cannot be economically predicted, deliberation is deferred until after an interaction has occurred; at the cost of making some mistakes, the greater cost of continuous deliberation is avoided. Together, these strategies offer a powerful mechanism for coordinating tasks: whenever a harmful interaction occurs

between tasks of an existing routine, the system can determine why the interaction occurred and then adapt the routine to prevent recurrence.

For the control of automated work environments, the ability to automatically adapt routines in response to performance failure would be of enormous benefit for two reasons. First, as the number of automated tasks in an environment grows, it becomes increasingly unrealistic to expect programmers to anticipate all of the ways that tasks might interact. Instead, one should expect the system to produce some malign task interactions and then adapt the task coordination abilities of the controlling system to compensate. Second, typical task environments gradually change, causing old coordination strategies to become obsolete. In a manufacturing plant, for example, machines malfunction, undergo repairs, and are sometimes replaced with newer equipment; at times the plant is short-staffed; demand for goods from the plant fluctuate; raw materials may be in short supply. Each of these events can cause interactions within a routine composed of usually compatible tasks. For example, consider a manufacturing process in which parts processed by one machine are conveyed to a second, even as the first machine processes new parts; quality control inspectors watch the output of each machine to insure that each is working correctly. What happens if the plant becomes understaffed and there is only one inspector to observe both machines? The old routine will no longer suffice because both tasks will require the inspector's full attention at the same time. To fix this problem, the part processing routine might be modified so that, rather than performing the two tasks in parallel, a batch of parts is processed at the first machine and then conveyed *en masse* to the second while the first machine idles. Coordinated sequentially, the tasks carried out by the two machines will no longer come into conflict since the single inspector need concentrate on only one machine at a time.

Unexpected interactions within a usually reliable routine occur when tasks are combined in new ways or old combinations are performed under novel circumstances. In our view, this will occur in all but the simplest and most controlled task environments. The ability to automatically adapt programs for controlling highly

automated environments is thus a practical problem of machine learning. Our approach has focused on two problems in particular. First, we have developed a general framework for learning from failure in which the learning procedure uses knowledge of the system's task coordination and execution mechanisms to reason about how these mechanisms might have caused a failure (Birnbaum et al, 1990). Second, we have begun to represent the abstract planning knowledge agents need to reason about in order to cope with undesirable task interactions (Freed & Collins 1994, Freed & Collins 1993, Freed et al 1992; see also Jones 1991 and Owens 1991). Such knowledge can be incrementally specified and integrated into routine execution mechanisms as the system learns what kinds of interactions it must be prepared for.

We are testing our approach to learning about task coordination using the RAP task-execution system (Firby, 1989) and Truckworld simulator (Firby, 1987). The RAP system has a number of important properties for our purpose. First, it reflects the need for minimal deliberation while executing a task by sharply limiting the amount and kinds of inference allowed at that time. Second, it allows the system to assess its current knowledge and explicitly attempt to acquire more at any time in the process of carrying out a task. This ability is crucial for controlling task interactions since methods for coordinating tasks will often depend on information that only becomes available during task execution. 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. In a later section, we analyze an example in this domain in which tasks of a delivery truck routine interact and thereby cause a goal failure. We then consider how the RAP system controlling the truck could be modified to avoid such incidents in the future. In the section immediately following, we discuss our approach to the problem of learning to coordinate tasks in greater detail.

2 Learning from Failure

Routines are, ideally, reliable, reusable and computationally cheap to execute. Their success in these regards depends on the stability of certain features of the task environment. For example, shoppers at the grocery store can usually expect to get home to unpack before frozen items have a chance to thaw. Consequently, their routines for shopping may assume no more than a certain amount of traffic on the drive home, that the weather will not be excessively hot, the car won't malfunction, etc.. If a shopper executes his normal routine while any of these conditions are not met, he may wind up unpacking melted ice cream. Such failures are endemic to the use of

routines since most of the conditions that determine whether the routine will work are simply assumed to hold and not checked. At the cost of occasional failure, assuming the validity of a plan's most reliable preconditions saves an enormous amount of deliberation.

Several strategies exist for minimizing the failures of routines due to faulty underlying assumptions. One strategy involves enforcing *stability* in the task environment (Hammond, 1990). For example, keeping one's car in good repair increases one's chances of arriving at home from the grocery store in time. Another involves the use of heuristics to signal the need for extra deliberation (Pryor & Collins, 1992)—for example, by observing the traffic on the way to the grocery store in order to determine whether delays can be expected on the drive home.

The approach we emphasize is to adapt routines when they fail. This approach 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, Ram & Cox, 1991). In particular, when an agent expects a routine plan to achieve its goal, and this expectation is violated, one response is to attempt to determine what aspect of the routine's representation was responsible for the faulty behavior, and how that aspect can be modified to avoid the recurrence of such failures.

To carry out this diagnostic process automatically, a system must be able to reason about the assumptions underlying its failed expectation. In particular, it must be able to retrieve (or generate) potentially faulty assumptions in response to an observed expectation failure. In our model, the connections between underlying beliefs and consequent expectations are represented in terms of explicit justification structures (deKleer, 1987, Simmons, 1988, Birnbaum et al, 1990). Diagnosis, then, consists of searching through the assumptions organized by an expectation's justification structure in order to determine where the fault lies. A central goal of our research is to learn how to construct justification structures to enable the diagnosis of a wide variety of failures. In particular, we are studying what kinds of assumptions must be organized and represented in an expectation's justification.

2.1 Levels of compatibility

When an agent constructs a routine, it must make some assumptions about the degree of compatibility between tasks in the routine. When the routine is executed in unusual circumstances, such assumptions may fail, possibly causing the failure of the whole routine. For the purposes of adapting a routine to better coordinate such

tasks, it is thus important to represent the assumed degree of compatibility between tasks in a routine, so that diagnostic processes can identify compatibility assumptions that fail. The routine can then be modified to coordinate interacting tasks based on an assumption of reduced compatibility. In our model, there are five qualitatively distinct levels of compatibility between pairs of tasks, each of which suggests a different process for coordination:

- Mutually exclusive
- Order sensitive
- Order insensitive
- Specification sensitive
- Specification insensitive

Mutually exclusive tasks are those for which executing one task precludes ever successfully executing the other. For example, if one's house is burning down, one might have time to save the jewelry in the bedroom or the photo albums in the den, but not both. In general, this degree of incompatibility arises only when tasks have incompatible deadlines or when they use up some resource that is unlikely to be recovered such as a large inheritance or a favor owed by a powerful person. To coordinate such tasks, a routine should include a step for selecting between alternatives.

Order sensitive tasks can be executed successfully in one order but not in others. In general, this occurs when one task is likely to violate a pre-condition or post-condition of another. Coordinating such tasks means selecting execution order. Order insensitive tasks must be executed at different times, but are otherwise fully compatible. For example, one can only tie the shoelaces on one shoe at a time but the order doesn't typically matter. A pair of order insensitive tasks each require access to some common resource for which resource recovery is not an issue, such as a wrench or one's hands. Coordinating order insensitive tasks involves only preventing concurrent execution.

Specification sensitive tasks can be executed concurrently as long as one properly chooses the method of execution for one or both tasks. For example, one may be transporting armfuls of groceries while also trying to unlock a door. While one's usual procedure for unlocking the door may involve visually locating the keyhole, bags of groceries obstructing one's vision may require unlocking the door using only tactile feedback. Coordinating specification sensitive tasks entails selecting methods for carrying out a task or other task parameters so as to eliminate incompatible resource requirements. Specification insensitive tasks, such as chewing gum and

walking, can be executed concurrently without any explicit coordination.¹

3 Example

As we have discussed, insuring that a pair of tasks do not interact involves coordinating their execution using methods appropriate to their actual level of compatibility. When a routine relies on overly optimistic assumptions about the compatibility of its component tasks, failures are likely to result. We are developing our approach to learning from such failures using the RAP task-execution system (Firby, 1989). RAPs are representations of routine behaviors containing a set of tasks and information on how the tasks should be coordinated.² We are constructing a system called RAPTER³ that will diagnose the cause of a RAP execution failure and appropriately modify the faulty RAP. Below, we present an example from the Truckworld domain in which a RAP execution failure occurs; we then discuss how the faulty RAP could be identified and then modified to prevent recurrence of similar failures.

3.1 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 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 checks the oil, which takes a few seconds. Then it inspects each arm, taking over an hour to disassemble and visually inspect various arm components. Finally the truck leaves for the gas station and arrives to find that the station has been closed for twenty minutes.

¹Another way to coordinate tasks is to interleave them. This requires a relatively detailed analysis of the compatibility between the tasks; in particular, one must consider the degree of compatibility between constituent subtasks. For example, one can successfully interleave execution of a reading task with a train riding task by assuming that the subtasks read-one-page and notice-when-train-stops may be executed concurrently (specification-insensitive) but that check-station-sign-out-window must take precedence over read-one-page (order sensitive) when it becomes active.

²Currently three levels of coordination between pairs of tasks are possible, corresponding to assumptions of mutual-exclusivity, order-sensitivity and order-insensitivity; by default, each pair of tasks is order-insensitive. A newer version of the system allowing task concurrency is under development (firby, 1993).

³short for RAP Adapter.

```

(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: RAP for preparing to go on a long journey

3.2 Learning from the example

What should be done in response to this failure? If the circumstances of the failure could never be repeated—for example, if the truck would never again need to refuel—it would make sense to simply recover from the failure and not expend any effort to learn. In the example scenario however, the failure arose from the execution of a routine, and therefore often repeated, behavior. Failure to learn in this case would subject the system to potentially frequent repetitions of the failure. In general, the high degree of reuse of memory structures underlying a routine behavior—RAPs in our case—suggests that adaptation in response to failure is crucial.

circumstances required executing the refueling task first. 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.

To discover this fix for itself, the learning system must analyze the assumptions underlying the routine's current structure. These assumptions can be represented in terms of a justification structure supporting the belief that the refueling task should succeed (figure 2 shows part of this structure). In our example, when the refueling 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 behavior can be found. Diagnosis consists of finding a faulty assumption. To repair the failure, the agent's routine 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.

3.3 Transformation Rules

After locating a faulty assumption that caused a failure, and determining how the system should behave in the future to prevent the failure from recurring, the learning process must incorporate the proposed new behavior into the system by modifying appropriate task execution components—RAPs in our case. Knowledge about how to incorporate behavioral changes can be embodied in *transformation rules* (Hammond, 1989, Collins, 1989). The left hand side of a transformation rule indicates when the rule should be applied. A transformation rule's right hand side specifies a procedure for modifying a given RAP that will produce the designated adaptation. In effect, each transformation rule encodes a standard fix for a particular class of plan failures. The output of the diagnosis process is used to determine the class into which the current failure falls.

In the gas station example, the justification traversal process faults an assumption corresponding to the left hand side of the transformation rule called *add-conditional-ordering-constraint:fixed-deadline*. The effect of this rule is to cause two tasks in a RAP to be executed in a particular order whenever the current time exceeds some fixed deadline.⁴ The rule is applied to the

⁴See (Freed and Collins, 1994) for an explanation of why this rule is especially appropriate.

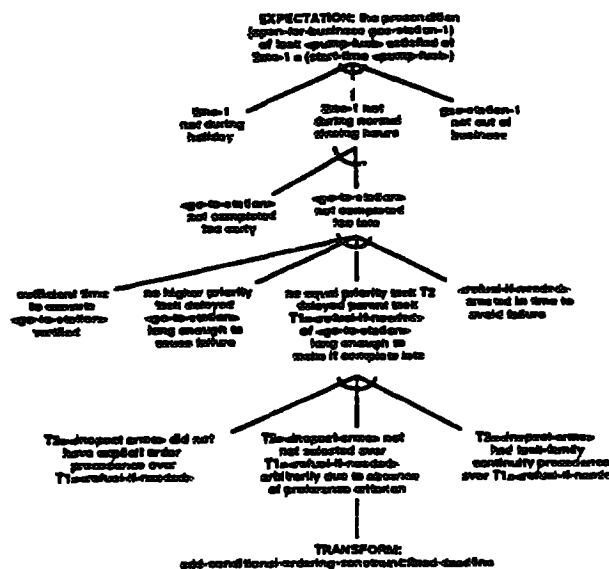


Figure 2: Justification structure in gas station example

In the example scenario, the truck failed to arrive in time at the gas station because its routine for preparing for a long journey failed to properly coordinate two of its component tasks. In particular, it relied on the assumption that refueling and performing an arm inspection were order-insensitive when, in fact,

PREP-JOURNEY RAP using modification parameters specified during the justification traversal process (Birnbaum et al, 1990, Krulwich, 1991).

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 3). 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 (pm 7 30))))
      (TASK-NET
        (t1 (add-oil-if-needed))
        (t2 (inspect-arms))
        (t3 (refuel-if-needed 15))))
      (METHOD method-2
        (CONTEXT
          (and (current-time ?time) (>= ?time (pm 7 30))))
        TASK-NET
          (t1 (add-oil-if-needed))
          (t2 (inspect-arms))
          (t3 (refuel-if-needed 15) (for t2))))))

```

Figure 3: PREP-JOURNEY after applying add-conditional-ordering-constraint:fixed-deadline

4 Conclusion

In order to avoid harmful plan interactions, human agents typically evolve a set of standard routines for carrying out often-repeated tasks. By modifying these routines in response to observed failures, the agent can refine its behavior over time. We are constructing a software agent, RAPTER, that applies this incremental improvement strategy to the management of planning routines for use in the Truckworld simulator. Currently, we are building in the capacity to make a variety of repairs to the method used in a routine to coordinate constituent tasks; in the future, RAPTER will be extended to include methods for coordinating tasks from different routines.

References

Agre, Philip (1988). *The Dynamic Structure of Everyday Life*. Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science.

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

Chapman, D. (1987) Planning for conjunctive goals. *Artificial Intelligence*, 32(3):333–374, July 1987.

Collins, G. (1987) *Plan Creation: Using Strategies as Blueprints*. Ph.D. thesis, Yale University, 1987. (Research Report #599).

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

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

Firby, R. (1993). Interfacing the rap system to real-time control. Technical report. (Available via FTP at cs.uchicago.edu:pub/users/firby/newraps.hqx)

Firby, R. and Hanks, S. (1987) The simulator manual. Technical Report YaleU/CSD/RR Number 563, Yale University, 1987.

Firby, R. (1989). Adaptive Execution in Complex Dynamic Worlds. Ph.D. thesis, Yale University. (Available as Report RR-672).

Freed, M. and Collins, G. (1994). Learning to cope with task interactions. Proceedings of the 1994 AAAI Spring Symposium on Goal-driven Learning.

Freed, M. and Collins, G. (1993). 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.

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

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

Jones, E. (1991) The flexible use of abstract knowledge in planning. Ph.D. thesis, Yale University.

Krulwich, B. (1991). 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.

Owens, C. (1991). A functional taxonomy of abstract plan failures. *Proceedings of the 14th Annual Conference of The Cognitive Science Society*, pages 167–172, Chicago, IL.

Pryor, L. and Collins, G. (1992). Reference features as guides to reasoning about opportunities. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, Bloomington, IN.

Ram, A. and Cox, M. (1991). Using introspective reasoning to select learning strategies. In *Proceedings of the First International Workshop on Multistrategy Learning*.

Schank, R. (1982). *Dynamic Memory*. Cambridge University Press, Cambridge, England.

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

Sussman, G. (1975). *A Computer Model of Skill Acquisition*. American Elsevier, New York.

Wilensky, R. (1983). *Planning and Understanding: A Computational Approach to Human Reasoning*. Addison-Wesley Publishing Company, Reading, Ma.