MODEL-BASED DIAGNOSIS OF PLANNING FAILURES

Lawrence Birnbaum, Gregg Collins, Michael Freed, and Bruce Krulwich

Northwestern University
The Institute for the Learning Sciences and
Department of Electrical Engineering and Computer Science
Evanston, Illinois

Abstract

We propose that a planner should be provided with an explicit model of its own planning mechanism, and show that linking a planner's expectations about the performance of its plans to such a model, by means of explicit justification structures, enables the planner to determine which aspects of its planning are responsible for observed performance failures. We have implemented the ideas presented in this paper in a computer model. Applied to the game of chess, the model is capable of diagnosing planning failures due to incomplete knowledge of the rules, improper or overly optimistic focus of attention, faulty projection, and insufficient lead time for warning about threats, and is therefore able to learn such concepts as *discovered attack* and the *fork*.

1 Introduction

Learning by debugging, or *failure-driven learning,* has historically been the most prominent approach to learning how to plan in AI (see, e.g., Sussman, 1975; Schank, 1982; Hayes-Roth, 1983; Kolodner, 1987; Chien, 1989; Hammond, 1989). In much of this work, particularly in its early stages, no real distinction was made between the diagnosis methods used to uncover planning errors, and the repair strategies used to correct them. The hope was that relatively superficial descriptions of planning errors could be used to index directly to repair strategies that were likely to be useful in eliminating those errors. More recently, efforts have been made to tease these apart so that relatively general *model-based* methods for diagnosis can be applied to the task of finding bugs in faulty plans (see, e.g., Simmons, 1988). Such methods, derived originally from work in circuit debugging (see, e.g., Stallman and Sussman, 1977; Davis, 1984; DeKleer and Williams, 1987), depend upon having an explicit representation of how a plan is intended to function given certain assumptions. The task of the diagnosis component is then to determine which of these assumptions are faulty when a plan fails to operate as expected. We have been developing such an approach within the context of learning to plan in competitive situations (see, e.g., Birnbaum and Collins, 1988; Collins, Birnbaum, and Krulwich, 1989).

Many assumptions made during planning, obviously, concern the state of the world in which the plan will be executed. In addition, however, plans often depend upon assumptions about the capacities or properties of the agent executing them. Because this agent is often the planner itself, a planner needs some degree of self-knowledge in order to plan correctly. For example, if a robot planner were unaware of its lifting power, it might build plans requiring it to lift objects that were too heavy; if it were unaware of its width, it might build plans requiring it to squeeze through openings that were too narrow; and so on.

Perhaps less obviously, plans do not draw only on the physical capabilities of a planner: They also depend on its *cognitive* capabilities. Thus, a planner will require some knowledge of its mental and perceptual processes in addition to knowledge about its physical properties (Collins and Birnbaum, 1990). For example, chemists often put a stopper in a test tube in which they are boiling something, so that the "Pop!" that occurs when rising pressure forces the stopper out will alert them to the fact that it is time to remove the test tube from the heat.[1] This is a simple instance of a powerful and general strategy often employed by human planners, namely, setting an external alarm to alert the planner when some task needs his attention, thus freeing him to attend to other

[1]Thanks to Ken Forbus for this example.

matters in the interim. The success of such a plan depends upon two things: First, the planner must actually notice the alarm, and second, he must recall the task which is supposed to be resumed when the alarm goes off. In order for a robot planner to successfully apply this strategy, therefore, it will need to know something about the kinds of events that attract its attention—and which can therefore function as an alarm—and something about the reasoning processes that occur when its attention is drawn to such an event. The planner's theory of its cognitive machinery might, for example, specify that flashing lights, loud noises, and quick movements attract its attention; that once its attention is so attracted, the planner will attempt to explain the cause of the event; and that if the cause is due to the planner itself, the planner will recall its purpose in setting up the event. Armed with this theory, the planner can decide whether to attribute to itself the ability to be alerted by a particular type of event, for instance the "Pop!" of a stopper being disgorged from a test tube.

When faulty assumptions in a planner's model of itself cause planning errors, failure-driven learning can be invoked to enable the planner to identify and alter these assumptions, and thus improve its model of itself just as it would its model of anything else. However, in dealing with assumptions about its own capabilities as a planner there is another possibility: For in many cases, these assumptions about the planner are represented in terms of parameters that are *under the planner's own control*. Thus, *rather than changing the model to reflect the way the planner functions, the planner may be able to change the way it functions to conform with the model.*

2 Modelling the planner

Developing a model-directed approach to the diagnosis of planning failures, as discussed above, requires developing explicit models of the planning and plan-execution mechanisms employed, and of how the agent's plans depend upon those mechanisms. In this section, we describe models of two portions of a simple planner, dealing with threat detection and with execution scheduling.

2.1 A model of threat detection

Every planning domain presents a planner with a variety of threats against its goals. In coping with any particular class of threats in some domain, the planner has a choice of either attempting to permanently insulate itself from all instances in that class—i.e., of removing some necessary precondition for all instances of that sort of threat—or of attempting to deal with threats individually as they arise. We refer to these two approaches as the "Great Wall of China" paradigm, and the "Firefighting" paradigm, respectively. It is in the second of these two general strategies that the necessity for threat detection arises.

We model threat detection as a simple rule-based process: For each known type of threat, there is a rule that is able to detect instances of that threat if it is evaluated at some point in time prior to the realization of the threat. The planner must take account of a variety of constraints in formulating and deploying such rules. For example, in order to fulfill the function of threat detection within the Firefighting paradigm, each rule must be capable of detecting its threat in time to permit the planner to carry out an effective counterplan. Moreover, the planner must formulate a control regimen for checking its threat detection rules that ensures that whenever a threat arises, the appropriate threat detection rule will in fact be checked in time. We have previously argued (Collins, Birnbaum and Krulwich, 1989) that, all else being equal, the planner's task is simplified by attempting to detect threats as late as possible. In chess, that point is one move before the threat is carried out. A novice chess planner, then, starts out with a model in which it checks its threat rules at each turn, looking for threats that may be executed on the next turn.

Having decided upon this control regimen, the planner is faced with another choice: It can recompute the set of outstanding threats each time it checks its rules, or it can incrementally compute this set by looking only for changes that have occurred since the previous check. The latter approach is generally more efficient when the number of new threats arising per detection cycle is small compared to the total number of threats outstanding, provided that an effective method for focussing on threats that result from changes can be devised. In turn-taking games such as chess, incremental detection of threats means computing the threats added and removed by each move. Given the rule-based framework for threat-detection described above, focussing on threats resulting from changes can be implemented as a set of restrictions on the domain of application of the threat detection rules. In our model these restrictions are themselves implemented as a set of *focus rules* that specify the domain over which the threat detection rules will be applied. A portion of the model is shown in Figure 1.

$\forall x \, \exists t \;\; t \leq \text{ert}(x) \;\&\; \text{detect}(x, t) \rightarrow \text{added-to}(x, T, t)$

$\forall x, t_1, t_2 \;\; \text{added-to}(x, T, t_1) \;\&\; t_2 \leq \text{ert}(x) \;\&$
$\sim\exists\, t_r \;\; (t_1 \leq t_r \leq t_2 \;\&\; \text{removed-from}(x, T, t_r))$
$\rightarrow \text{member-of}(x, T, t_2)$

$\forall x \;\; \exists\, r, f, \theta, t \;\;\; t \leq \text{ert}(x) \;\&\; \text{detects}(r, x) \;\&\; \text{active}(x, t) \;\&$
$(\text{evaluate-rule}(r, \theta, t) \rightarrow \text{detect}(x, t)) \;\&$
$\theta \in \text{focus-results}(f, t)$

$\forall r, f, t, \theta \;\;\; \theta \in \text{focus-results}(f, t) \rightarrow \text{evaluate-rule}(r, \theta, t)$

$\forall x, t \;\;\; \text{remove-from}(x, T, t) \leftrightarrow \text{member-of}(x, T, t) \;\&$
$\sim\text{active}(x, t)$

Figure 1: Partial model of threat-detection[2]

2.2 A model of execution scheduling

Since a planner has limited resources, the formulation of a viable plan in service of an active goal is not, by itself, enough to guarantee that the plan will be carried out. Thus, another important aspect of any planner is a priority-based scheduling mechanism for determining which plans should be executed when. While such a mechanism could, in principle, be arbitrarily complex, we have chosen to model execution scheduling using a simple priority queue, assuming discrete time (which is sufficient for the turn-taking games we are currently investigating). Given such an approach, the basic model of execution scheduling is as follows: Once a goal is formed, a plan is chosen for that goal, assigned a deadline and a priority, and placed on the priority queue. At each time increment, the queue is checked, and the highest priority plan on the queue is chosen for execution. Thus a plan will be successfully executed if and only if there is a time before its deadline when it is the highest priority plan on the queue. A portion of the model appears in Figure 2.

[2]The predicate "ert" stands for earliest realization time, i.e., the earliest time at which the threatened action can be expected to occur.

$\forall p, t, g \;\; \text{execute}(p, t) \;\&\; t \leq \text{deadline}(p) \;\&\; \text{plan-for}(p, g)$
$\rightarrow \text{achieve}(g)$

$\forall p, t \;\; \text{execute}(p, t) \leftrightarrow \text{member-of}(p, Q, t) \;\&$
$\sim\exists\, p_a \, (\text{member-of}(p_a, Q, t) \;\&$
$\text{priority}(p) < \text{priority}(p_a))$

$\forall p, t_s, t_e \;\; \text{added-to}(p, Q, t_s) \;\&\; t_s \leq t_e \;\&$
$\sim\exists\, t_r \, (t_s \leq t_r \leq t_e \;\&\; \text{removed-from}(p, Q, t_r))$
$\rightarrow \text{member-of}(p, Q, t_e)$

$\forall g, t \;\; \text{goal}(g, t) \;\&\; \text{call-planner}(g, t) \;\&\; \text{planner}(g, t) = p$
$\rightarrow \text{added-to}(p, Q, t)$

Figure 2: Partial model of execution scheduling

3 Case study: The fork

The models described above were developed as part of an account of learning in competitive situations—in particular, chess. We will now consider their application to the acquisition of a classic chess tactic, the *fork*. In our account, the novice chess player initially notices the fork when it fails to block a threat against one of its pieces. Such a failure leads the planner to search for a fault in the planning and execution mechanisms involved in its ability to block threats—either, that is, in its mechanisms for detecting the threat, for formulating counterplans to the threat, or for executing such counterplans. This search employs the models described in the last section, along with certain other assumptions that specify how the mechanisms they describe are being deployed in order to block threats in chess. For example, one of these assumptions states that for each threat against materiel that is detected, a goal to block that threat will be formulated. In general, these assumptions can be derived by regressing (see, e.g., Manna and Waldinger, 1979) a specification of the desired behavior of the planner through the rules that specify its planning mechanisms.

Consider a situation in which white's rook and knight are under attack from a black pawn. Though it is white's turn to play, there is no move that would protect both pieces. Rooks are more valuable than knights, so white will save the rook, and black will capture the knight. If our planner is placed in this situation, the capture of the knight will cause a failure of its expectation that all threats to its pieces will be blocked. We now give an informal account of how fault diagnosis must proceed in this case. (A more detailed description of the process is contained in the next two sections.)

By the first two axioms of execution scheduling (see Figure 2), the planner's ability to block a threat depends upon there being a time such that a counterplan for the threat is on the queue, it is before the deadline, and there is no higher priority plan on the queue. Thus, for blocking to have failed, it must have been the case that there was no such time, i.e., some necessary condition for execution must have been missing at each time when the plan might, in principle, have been executed. Thus we can group all of the times into equivalence classes depending upon which necessary condition failed to hold.

The diagnostic process has now reached a point at which the planner is presented with a choice of ways to fix the problem, since it can choose the time for which it wants to reinstate the missing condition. For example, execution could occur at an earlier time if the planner could find a way to schedule the plan earlier; execution could occur at a later time if the planner could find a way to postpone the carrying out of the threat, say, by putting the opponent in check; and execution could occur at a time in between if the planner could make the plan for saving the knight a higher priority than the plan for saving the rook. In the particular instance that we are considering, the only possible approach is to try to execute the blocking plan earlier. However, by the axioms describing the execution scheduling and threat detection mechanisms, this requires adding the blocking plan to the priority queue earlier, which in turn requires formulating a goal to block the threat earlier, which in turn requires detecting the threat earlier. Our approach, then, makes it possible for a planner to determine for *itself* that the way to cope with forks is to detect them sooner, rather than needing this piece of advice to be built in.

4 Diagnosing faults in plan justification structures

The goal of fault diagnosis is to explain the failure of an expectation as a consequence of the failure of some set of underlying assumptions. In our model the connections between underlying beliefs and consequent expectations are represented in terms of explicit justification structures (see, e.g., DeKleer *et al.*, 1977; Doyle, 1979). These justification structures, as alluded to above, record how the planner's expectations about the performance of its plans are inferred from the policies it has adopted, in conjunction with the axioms that constitute its model of the planning and execution mechanisms it employs. For instance, the justification structure underlying the expectation that the planner will block the threat to its knight in the fork example above is a conjunction of three antecedent expectations: That moving the knight will block the threat, that the move will be executed, and that such execution will take place before the opponent takes the knight. The expectation that the knight move will be executed is in turn justified by the conjunction of two prior expectations that the move appears on the plan queue and that it is the highest priority item on the queue; and so on. Diagnosing the fault thus involves "backing up" through these justification structures, recursively explaining the failure of an expectation as the result of a failure of one or more of its immediately antecedent expectations. This approach is similar to that proposed by Smith, Winston, Mitchell, and Buchanan (1986) and Simmons (1988).

More generally, a justification structure links an expectation with a set of supporting beliefs. The justification may be either *conjunctive*, meaning all the supporters must be true to justify the expectation, or *disjunctive*, meaning at least one of the supporters must be true to justify the expectation. When an expectation fails, the diagnosis algorithm attempts to determine which of its supporters should be faulted. If the support set is conjunctive, then at least one supporter must be faulted. If the support set is disjunctive, then all of the supporters must be faulted. The basic algorithm nondeterministically expands the set of faulted expectations according to these simple rules until a stopping criterion is met. However, when faulting conjunctive supports, the degree of arbitrary choice required can be reduced by checking whether a proposition in the support set was observed to be true during the execution of the plan, and if so removing it from consideration.

Since the goal of the diagnosis process is to provide enough information to allow a repair to be effected, the process should, in principle, continue until a repair is actually generated. Because of this, the diagnostic module in our system calls the repair module at each step, checking to see if the current set of faulted expectations provides enough information for a repair to be generated. This is particularly important for reducing the effort entailed in propagating a fault through a disjunctive justification. Although, in principle, every proposition in such a support set must be faulted when the supported proposition is faulted, it does not necessarily follow that the diagnostic procedure must explain the failure of each disjunct. Since the overarching goal of the process is to *fix* the problem, and since the repair of a single disjunct will suffice to restore the supported expectation, it may be enough to explain the failure of one disjunct. In other words, the attempt to fault a disjunctive justification results in the goal to *unfault* one of the disjuncts.

One additional point deserves special mention. There will be many instances in which the planner expects that the execution of a plan will involve some entity that meets a particular set of constraints, but does not know in advance the exact identity of this entity. This may be true, for example, of objects that will be found in the planning environment, including tools, raw materials, obstacles, and other agents. In our model, it is also true of execution times for plans, since the planner leaves the execution scheduler some latitude in determining when the plan should actually be carried out. The upshot is that some of the assumptions underlying the expectation that a plan will succeed are existentially quantified: They assert that a time, tool, or material meeting certain constraints will actually exist. Faulting such expectations causes special problems, stemming from the fact that the faulting of an existentially quantified proposition implies the faulting of an infinite set of disjuncts, one for each object over which the existential ranges. Of course, even in principle, the diagnostic engine cannot consider why the assertion failed for each instance of the variable. One solution is to partition the set of instances into classes in such a way that the elements of each class have all failed to meet exactly the same constraint or constraints, while meeting all others. This organizes the set of faulted instances in such a way that the unfaulter need only consider one instance of each type of failure, rather than repeatedly trying to unfault instances that have failed for the same reason about which it can do nothing. In the case of the fork, for example, the planner's expectation that there will be a time at which it will be able to carry out the plan to save the knight depends on there being a time before the deadline when the plan is on the schedule queue, and is of higher priority than anything else on the queue. Each of these three constraints defines a class of times, namely, those at which the expectation failed because that particular constraint was not met.[3]

5 Conclusions

We have proposed that a planner should be provided with an explicit model of its own planning mechanism, and have shown that linking a planner's expectations about the performance of its plans to such a model, by means of explicit justification structures, enables the planner to determine which aspects of its planning are responsible for observed performance failures. We have implemented the models of threat detection and execution scheduling described above in our test-bed system for failure-driven learning in planning domains (Collins, Birnbaum, and Krulwich, 1989). Applied to the game of chess, the model is capable of diagnosing planning failures due to incomplete knowledge of the rules, improper or overly optimistic focus of attention, faulty projection, and insufficient lead time for warning about threats, and is therefore able to learn such concepts as *discovered attack* and the *fork*. In other work (see, e.g., Collins, Birnbaum, and Krulwich, 1989), we have addressed the issue of how to use such diagnoses as a basis for repairing the planner's procedures in order to avoid such failures in the future.

We are currently developing explicit models of other aspects of planning, including plan formulation and optimization. In addition, we have identified a number of common sense planning concepts that could be acquired given richer models of threat detection and execution scheduling. For example, one possible reason for a planning system's failure to detect a threat in time might be that it does not evaluate its threat-detection rules often enough. In general, the frequency with which these rules should be evaluated must be a function of the

[3]In fact, there are several other equivalence classes, since multiple constraints may fail for a given time. However, since the goal of fault diagnosis given disjunctive justifications is to find something to *unfault*, we start with those classes corresponding to a single failed constraint on the grounds that these are likely to prove easier to fix.

speed with which threats develop in a particular environment. It is therefore possible that a planner could learn the need to alter this frequency in different domains by applying failure-driven methods. Thus, extending the model of threat detection presented in this paper will allow us to provide a computational account of the acquisition of the common-sense planning heuristic that when things are happening quickly, a planner should pay closer attention.

Acknowledgments: We thank Matt Brand, Kris Hammond, and Dick Osgood for many useful discussions. This work was supported in part by the Office of Naval Research under contract N00014-89-J-3217, and by the Defense Advanced Research Projects Agency, monitored by the Air Force Office of Scientific Research under contract F49620-88-C-0058. The Institute for the Learning Sciences was established in 1989 with the support of Andersen Consulting, part of The Arthur Andersen Worldwide Organization.

References

Birnbaum, L., and Collins, G. 1988. The transfer of experience across planning domains through the acquisition of abstract strategies. *Proceedings of the 1988 Workshop on Case-Based Reasoning,* Clearwater Beach, FL, pp. 61-79.

Birnbaum, L., Collins, G., and Krulwich, B. 1989. Issues in the justification-based diagnosis of planning failures. *Proceedings of the Sixth International Workshop on Machine Learning,* Ithaca, NY, pp. 194-196.

Chien, S. 1989. Using and refining simplifications: Explanation-based learning of plans in intractable domains. *Proceedings of the Eleventh IJCAI,* Detroit, MI, pp. 590-595.

Collins, G., and Birnbaum, L. 1990. Problem-solver state descriptions as abstract indices for case retrieval. *Working Notes of the 1990 AAAI Spring Symposium on Case-Based Reasoning,* Palo Alto, CA, pp. 32-35.

Collins, G., Birnbaum, L., and Krulwich, B. 1989. An adaptive model of decision-making in planning. *Proceedings of the Eleventh IJCAI,* Detroit, MI, pp. 511-516.

Davis, R. 1984. Diagnostic reasoning based on structure and behavior. *Artificial Intelligence,* vol. 24, pp. 347-410.

DeKleer, J., Doyle, J., Steele, G., and Sussman, G. 1977. AMORD: Explicit control of reasoning. *Proceedings of the ACM Symposium on Artificial Intelligence and Programming Languages,* Rochester, NY, pp. 116-125.

DeKleer, J., and Williams, B. 1987. Diagnosing multiple faults. *Artificial Intelligence,* vol. 32, pp. 97-130.

Doyle, J. 1979. A truth maintenance system. *Artificial Intelligence,* vol. 12, pp. 231-272.

Hammond, K. 1989. *Case-Based Planning: Viewing Planning as a Memory Task.* Academic Press, San Diego.

Hayes-Roth, F. 1983. Using proofs and refutations to learn from experience. In R. Michalski, J. Carbonell, and T. Mitchell, eds., *Machine Learning: An Artificial Intelligence Approach, Vol. 1,* Tioga, Palo Alto, CA, pp. 221-240.

Kolodner, J. 1987. Capitalizing on failure through case-based inference. *Proceedings of the Ninth Cognitive Science Conference,* Seattle, WA, pp. 715-726.

Manna, Z., and Waldinger, R. 1979. A deductive approach to program synthesis. *Proceedings of the Sixth IJCAI,* Tokyo, pp. 542-551.

Schank, R. 1982. *Dynamic Memory: A Theory of Reminding and Learning in Computers and People.* Cambridge University Press, Cambridge, England.

Simmons, R. 1988. A theory of debugging plans and interpretations. *Proceedings of the 1988 AAAI Conference,* St. Paul, MN, pp. 94-99.

Smith, R., Winston, H., Mitchell, T., and Buchanan, B. 1985. Representation and use of explicit justifications for knowledge base refinement. *Proceedings of the Ninth IJCAI,* Los Angeles, CA, pp. 673-680.

Stallman, R., and Sussman, G. 1977. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence,* vol. 9, pp. 135-196.

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