An Adaptive Model of Decision-Making in Planning

Gregg Collins[1]
University of Illinois
Dept of Computer Science
Urbana, Illinois
61801
Collins@p.cs.uiuc.edu

Lawrence Birnbaunv
Yale University
Dept. of Computer Science
New Haven, Connecticut
06520-2158
Birnbaum@yale.arpa

Bruce Krulwiclv
Yale University
Dept. of Computer Science
New Haven, Connecticut
06520-2158
Krulwich@yale.arpa

Abstract

Learning how to make decisions in a domain is a critical aspect of intelligent planning behavior. The ability of a planner to adapt its decision-making to a domain depends in part upon its ability to optimize the tradeoff between the sophistication of its decision procedures and their cost. Since it is difficult to optimize this tradeoff on *a priori* grounds alone, we propose that a planner start with a relatively simple set of decision procedures, and add complexity in response to experience gained in the application of its decision-making to real-world problems. Our model of this adaptation process is based on the explanation of failures, in that it is the analysis of bad decisions that drives the improvement of the decision procedures. We have developed a test-bed system for the implementation of planning models employing such an approach, and have demonstrated the ability of such a model to improve its procedure for projecting the effects of its moves in chess.

1 Introduction

The ability to plan effectively involves many different forms of reasoning: projecting the effects of actions in a current or hypothetical situation, deciding which goal to pursue from among the many that might be pursued at any given time, constructing sequences of actions that can achieve a given goal, determining whether to execute such a sequence, and so on. By and large, most research on planning in AI has concentrated on those aspects of planning that involve reasoning about actions, and more specifically, on how sequences of actions can be constructed to achieve particular and well-defined goals. Considerably less attention has been devoted to those aspects of planning that we might term *decision-making:* Deciding whether (or when) to pursue a given goal, or whether to use a given plan for the goal, or for that matter whether even to consider the goal in the first place.

To make such decisions correctly, a planner must know a great deal about the particular environment in which they arise. For example, a person's decision about whether to turn around and look if he hears footsteps behind him depends crucially on the situation in which he finds himself: His decision is likely to be quite different on a moderately populated suburban street in the middle of the day than in an empty city street late at night. The relevant characteristic of the environment, obviously, is the rate at which threats can be expected to arise, and in particular the threat of being victimized by a criminal.

One of the major issues confronting any planner is determining how much effort to expend on evaluating a given decision. Decision-making in general is subject to a tradeoff between the sophistication of the procedure employed to make the decision, and the time and effort required to perform the analysis. A more sophisticated decision-making procedure will, generally speaking, either consider more alternatives, consider each alternative in more depth, or do both. This requires more reasoning, and is correspondingly more expensive. Whether a more sophisticated procedure is worth using depends on whether the gain from its superior performance outweighs this extra cost. Consider, for example, the tradeoffs involved in deciding which goal to pursue: The planner must first determine which of a myriad possible goals—including goals to acquire resources, protect desired states, or improve the current situation, among others—are reasonable candidates, and then compare the expected benefits from each of these candidates. Obviously, the more goals the planner considers as reasonable candidates, the more likely it will be to find a better goal to pursue. On the other hand, the more goals

[1]This research was supported in part by the Defense Advanced Research Projects Agency, monitored by the Air Force Office of Scientific Research, under contract number F49620-88-C-0058, and in part by the Office of Naval Research under contract number N0014-85-K-010.

the planner must compare in making its decision, the more this procedure will cost.

The key point about such tradeoffs in the sophistication of the decision-making process is that, like the decisions themselves, they are highly dependent upon the particular planning environment. For example, in a highly time-limited situation (e.g., basketball or some other fast-paced game), a planner simply cannot afford to consider very many candidate goals when trying to determine what it should do next. Thus, when confronted with a novel planning environment, a planner must be prepared to *adapt* its decision-making processes to that environment, balancing the tradeoff between their sophistication and their cost.

It seems difficult to imagine how an agent might find the roughly optimal level of decision-making sophistication on the basis of *a priori* reasoning alone. This suggests that planners should be prepared to alter their decision-making processes on the basis of experience within a new planning environment. The approach we have taken, therefore, is to start with a relatively simple model of decision-making—one that ignores many potential alternatives, and many of the factors that might bear on deciding among them—and to progressively make it more sophisticated in response to poor decisions, i.e., decisions to pursue goals and plans that ultimately fail. In other words, our model of how a planner adapts its decision-making to new planning domains is *failure-driven* (see, e.g., Sussman, 1975; Schank, 1982; Hayes-Roth, 1983; Minton, 1984; Hammond, 1986). In such an approach, when a plan fails, the goal of preventing a similar failure in the future leads to an attempt to explain the current failure.[1] The explanation of the failure, in turn, suggests ways in which similar failures might subsequently be avoided.

We have elsewhere presented our model of failure-driven learning in planning domains (Birnbaum and Collins 1988;Collinsand Birnbaum, 1988a and 1988b). In our approach, the planner encodes a description of the intended functioning of its plans in explicit *justification structures.* When the expectation that a plan will succeed is violated, the planner can trace back through the justification structure to identify the culprit among the initial assumptions (see, e.g., deKleer, *et ah,* 1977; Doyle, 1979). By identifying which assumptions have failed, the planner is able to arrive at a characterization of how the particular si tuation brought about this failure, which can then serve as the basis for a rule that suggests what to do when similar situations arise subsequently.

[1]**Thus the approach can also be viewed as a form of explanation-based learning (see, e.g., Dejong and Mooney, 1986; Mitchell, Keller, and Kedar-Cabelli, 1986).**

This paper describes the application of our model to the progressive adaptation of decision-making procedures to new planning environments. Such an application requires spelling out the assumptions involved in the expectation that simple decision procedures will suffice, how those assumptions can fail, and how the procedures can be improved upon the diagnosis of such failures. We first discuss how these issues arise in a case study of the chess fork. We next briefly describe a program which implements some of our ideas, and then discuss its application to a second case study, concerning the development of a planner's ability to predict its opponent's move in deciding its own move in chess.

2 Case study: The fork

As our first case study in improving decision-making in planning, we will consider how a novice chess planner can learn to deal with a fork—a situation in which the planner's opponent confronts it with an attack on two pieces simultaneously. Unless a move can be made that will defend both pieces at once—or alternatively one piece can be moved or defended in a way that at the same time poses a strong counter-threat to the opponent—one of the pieces in question will surely be captured. The fork is something that all chess players eventually learn about, usually as a result of being victimized by it. The question is, what does a novice learn about the fork from this experience?

A planner victimized by an opponent's tactic should learn two things: how to avoid being victimized in the future, and how to use the same tactic to victimize others. We will concentrate here on the first of these, the question of how a planner learns to avoid the fork. Since the fork takes advantage of the planner's plan for defending its pieces, we must begin by considering what the planner knows about this plan, and in particular, what he knows about the assumptions upon which its efficacy depends.

The plan that almost every novice chess player seems to have for defending his pieces is this: Before each turn, check to see if any of the opponent's pieces is in position to move to the location of one of your pieces. If this is the case, then you can prevent the execution of the threat by moving the threatened piece, or by guarding it with another piece. This plan makes the assumption that a one-move warning of a threat to a piece is sufficient to to allow it to be defended, and this is generally the case. It is this assumption, however, that the fork takes advantage of: When two imminent threats are detected on the same turn, there is not time to block both of them.

Given that a novice planner understands that its plan depends upon this assumption, a plausible explanation

for how it comes to understand the fork can be constructed. The fork results in the failure of the plan to protect materiel, since it results in the capture of a piece. This failure can be traced to the violation of an underlying assumption of the plan, namely that a one-move warning of any threat would be sufficient.[1] The violation of this assumption was, in turn, brought about by the opponent's positioning of a piece so that it attacked two pieces simultaneously. The key point here is this: Such an analysis not only explains the fork, it suggests the way to guard against it. The failure of the assumption that a one-move warning is sufficient implies the need to detect at least one of the two threats earlier. Moreover, the fact that the failure was brought about by the positioning of a piece to attack two pieces at once means that efforts at earlier detection can be limited to situations where the opponent can, with a single move, produce a threat to two pieces. Thus, understanding how the fork violates its assumptions enables a planner to determine how it can avoid being victimized by the fork in the future: It must modify its plan for scanning the board to include a check for opponent's pieces that can be moved into position to attack two unguarded pieces of its own.

This explanation of how a novice planner learns from experiencing the fork leaves us with a question, however: Why would the planner have assumed that a one-move warning was enough in the first place? The most obvious answer is that the planner was simply unable to imagine a circumstance in which this would not be the case. Unfortunately, this explanation does not hold up upon further reflection. The idea that multiple threats pose a problem for defenses that involve detecting and blocking individual threats is well known to all human planners: This is the fundamental reason, for example, why "ganging up" allows a superior force to be defeated by a collection of inferior forces in any competitive situation. It seems likely that any human planner would have experienced this phenomenon, and grasped its significance, long before learning to play chess. But given the knowledge that detection-based plans for blocking threats are vulnerable to an overload of threats, plus the lack of any particular reason to believe that this problem would not arise in chess, it seems unlikely that the planner would nevertheless conclude that a one-move warning would always be enough. To argue this would be to conclude that vulnerability to the fork is the result of a mistake—a mistake made consistently by almost every person who ever learns the game of chess. It would be difficult to explain why such an error in reasoning should be made by nearly everyone.

In fact, the answer lies not in the logic of the situation, but in its economics. The reason for not detecting threats two moves in advance is that it would cost too much to do so: The planner would be swamped by the need to respond to a massive number of threats, since in two moves most of the opponent's pieces would be able to move to at least one square currently occupied by one of the planner's pieces. Most of these "threats" would never develop, and the planner would at best waste a great deal of time that could better be used plotting strategy. There is, in other words, a tradeoff: By detecting threats early, the planner provides more time to deal with multiple threats, but at the cost of being forced to consider an enormous number of possible—but unlikely—threats. By detecting threats later, the planner runs the risk of not having enough time to deal with multiple threats, but avoids being swamped with too many low probability reports. In this case, the tradeoff clearly is on the side of later detection, and this is likely to be true in many domains.

A key characteristic of the appropriate defense against the fork is that it avoids a blow-up in the number of detected threats by looking only for cases in which one piece can be moved so as to attack two pieces at once. In other words, by looking for forks in particular, rather than detecting all threats two moves in advance, the planner focusses its attention narrowly enough to make the extra effort worthwhile. So this is what the planner really learns from the fork: Not that one move's warning might not be enough—it already knew that—but a characterization of the circumstances in which it is not that is precise enough to allow them to be detected efficiently.

We can now provide a coherent account of how the fork is learned. The planner begins with the understanding that threats to pieces will arise periodically in chess. The standard strategy for dealing with threats in any domain is this: First, construct methods for detecting individual instances of threats in advance. Second, construct methods for blocking threats that can be executed when a threat is spotted. And third, ensure that all threats will be detected in time for a blocking routine to be carried out.

In chess, first, the obvious method for detecting threats is to determine which of the opponent's pieces could move to locations occupied by the planner's pieces in some particular number of moves. Second, the obvious method for blocking these threats is to move the threatened piece out of harm's way; a slightly more sophisticated plan is to guard the piece with another. And third, since any piece can generally either be moved or guarded in a single turn, and since—all other things being equal—it is best to detect threats as late as possible, a planner can conclude that one move's warning will generally be enough. However, the planner cannot *prove* that one move will always be enough, and so is forced to make the assumption that this will generally be the case. By

[1] See Collins and Birnbaum, 1988b, for a discussion of how this is accomplished.

making this assumption explicit, and by ensuring that it is monitored whenever a threat arises, the planner assures itself of being alerted to cases in which the assumption fails, as it does in the fork. So, while the planner is forced to accept the cost of a few failures as the price of an efficient threat-detection strategy, by monitoring the assumptions it is forced to make, and analyzing the failures that occur, it is able to gradually improve its performance by dealing with problematic cases one at a time.

In a sense, this explanation argues that a planner is deliberately courting failure as a shortcut to a better plan. Viewed this way, the alternative would have been for the planner to reason out, *a priori,* the circumstances in which multiple threats could occur, rather than waiting for those circumstances to arise in practice. In effect, this would mean inventing the fork before actually playing the game. While this is possible in principle, in practice the computational cost is likely to be extremely high—high enough that it is almost certainly cheaper to wait for the fork to happen. We believe that similar explanations underlie the acquisition of a wide range of decision-making strategies employed by planners.

3 An implementation and a second case study

In order to explore more concretely our failure-driven approach to the adaptation of decision-making processes in planning, we have constructed a test-bed—comprising mechanisms for inference and rule application, justification maintenance, expectation handlers, failureexplanation, and rule patching—and implemented within it a simple model of decision-making and planning for turn-taking games. This model, in turn, has been used for exploring rudimentary planning and learning in tic-tac-toe and in chess.

```
(def-brule proj-factor-2
   (world (move move) (player player) result (weight integer)
          (opp-move move) (time time)
          (better-move move) (better-goal goal)
          (better-value integer) (orig-value integer) )
 (project-factor proj-factor-1.5 world move player result weight)
 (and (= world (world-at-time time))
      (decide (player-opponent player)
      (possible-moves-at-time (player-opponent player) time)
      world simple-dec-factors opp-move)
      (= result (world-after-move opp-move (world-after-move
          move world)))
      (= weight 1)
   )
 [Justification here!)
```

Figure 1: Initial Projection Method

The planning model is implemented in 14 fairly general rules, concerning the detection of threats and opportunities, making choices, forming and checking expectations, and the like; some typical rules are shown later in this section. In addition, there are 22 chess-specific reasoning rules. All of these rules, general as well as specific, have associated justification structures, as do all particular facts and expectations contained in the system's data base. These justification structures are constructed and utilized by the rule applier, the failure explanation algorithm, and the rule patching mechanism. We devote the rest of this section to describing the program's application to a second case study, concerning the development of a planner's ability to predict its opponent's move in deciding its own move in chess.

The decision-making process within our planning model is composed of three interleaved components: a *decider,* which determines what move to make, a *projector,* which projects the results of a possible move in a given situation, and an *evaluator,* which evaluates a situation from a given player's perspective. Our planner starts with extremely primitive methods for accomplishing each of these decision-making tasks. First, to decide what move to make it simply projects the results of each move available to it, evaluates each resulting situation, and chooses the move that yields the best result. To project the results of making a given move, the planner simply assumes that its opponent will behave as it would in any given situation; the projection method that implements this is shown in figure 1. Finally, to evaluate a given situation, the planner computes the difference between the total values of the two player's pieces remaining on the board.

Consider now how a planner endowed with these primitive decision-making methods would behave in the situation shown in figure 3. To such a planner, moving the rook to take the knight looks like the best move. This decision is based on the erroneous expectation that its opponent will take the planner's knight. It forms this expectation because the projection method shown in figure 1 simply evaluates the opponent's options in the *original* situation, i.e., without taking into account the effects of the planner's own move. The use of such an unsophisticated procedure considerably simplifies the projection problem, because it obviates the need to recompute the opponent's response individually for each move contemplated by the planner. However, its validity, and hence the validi ty of theexpectationsthe planner generates concerning the opponent's moves, depends on an assumption that nothing will occur that will enable the opponent to make a better move than those currently available to it. This assumption, is an instance of what is sometimes referred to as an *inertia assumption,* namely an assumption that things will stay as they are as much as possible. In our planner, this assumption is itself justified by a conjunction that says roughly "Nothing will happen to give him a better move because he can't do anything to give himself one, I won't do anything to give

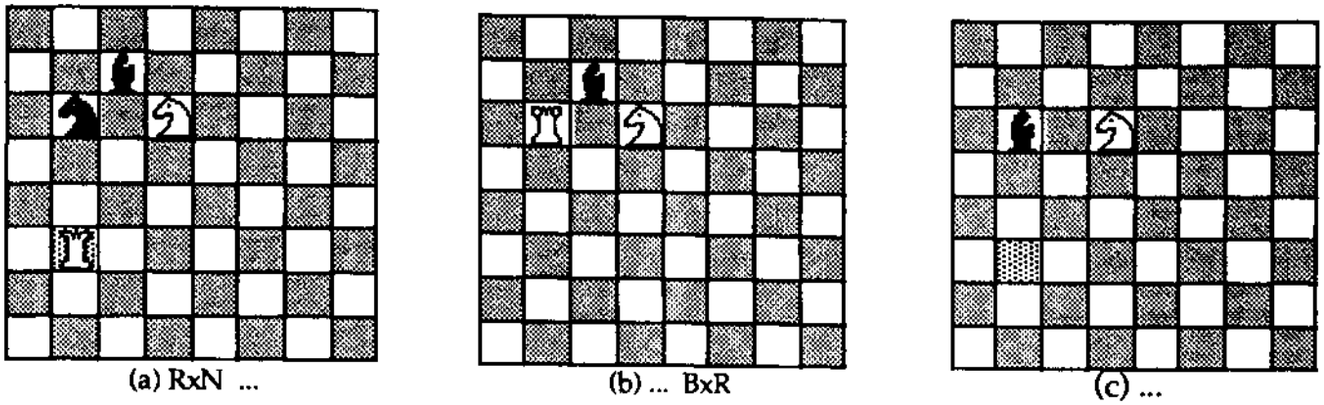

Figure 3: Game Board Sequence

him one, and no outside forces will give him one." Because the planner's decision to take the knight is justified by its projection that its opponent will take a knight in response, it monitors the status of this expectation. The expectation, in turn, is justified by the assumptions underlying the projection method used to produce it.

```
Applying rules in NEW-EXPECTATION-FAILURE-RULES
—Expectation failure:
  item=(move-to-make (move opponent move-take pawn
                                (ro>loc 2 3) (rc->loc 3 4) knight)
                     opponent (goal-unknown) 2)
Traversing...
Checking fact
#(fact 1785: (project (move computer move-take rook
                      (move opponent move-take   pawn
                            (rc->loc 2 3} (rc->loc 3 4) knight)

Checking b-rule #{b-rule proj-factor-2)
—> Valid:
(not (exists (c event)
        (move-to-make ?e.216 opponent ?better-goal.209 1)))
—> Valid:
(not (exists (e event)
        (and (extraneous-event ?e.2l7)
         (event-enables-event ?e.217 ?better-move.208))))
—> Found a bug:
(no (and (move-enables-move ?move.210 ?better-move.208)
        (move-possible ?better-move.208 ?time.207)
        (move-legal ?better-move.208)
        (evaluate (world-after-move ?better-move.208
                     (world-after-move ?move.210 ?world.211)
                  eval-factors
                  (player-opponent ?player.212)
                  ?better-value.213)
        (evaluate (world-after-move ?opp-move.214
                     (world-after-move ?move.210 ?world.21D)
                  eval-factors
                  (player-opponent ?player.212)
                  ?orig-value.215)
        (> ?better-valuc.213 ?orig-value.215)))
  *** Returning a BUG ***
```

Figure 4: Traversing an Expectation's Justification

Unfortunately, as we can see, the opponent is not going to make the move that the computer expected, and the move that he *will* make—namely taking the planner's rook—is a better move than the one the planner expected. When this happens, the planner detects that its expectation that the opponent will take the knight has failed. This then causes the planner to traverse the justification structure for that expectation, checkingthe assumptions upon which it depends.

In particular, the planner will discover that its assumption that the opponent will not have a better move has been violated. A partial transcript of this process is shown in figure 4, but it should be clear that the assumption in the justification that will fail is the inertia assumption that no event will occur that will give the opponent a better move than we expect him to have, and in particular the assumption that no action of the program's will give the opponent a better move.

As can be seen in figure 4, the system knows more than just the identity of the assumption that failed: It also knows *how* that assumption failed. The program's domain knowledge allows it to determine that the opponent's move was in fact a legal one, and a comparision of the justification for this belief with the program's prior expectation reveals that the event that enabled the opponent to make a better move than expected was in fact the computer's own move. Once the program discovers this, it must modify its reliance on the inertia assumption to take the problem into account, using the information provided in its justification structures and its analysis of the failure. The program uses standard goal regression techniques (see, e.g., Dejong and Mooney, 1986; Mitchell *etal.,* 1986) todetermine just those aspectsof the situation that caused the expectation to fail.

The resulting generalized explanation for the failure is used to patch the method that predicts the opponent's response, as utilized by the projection component, so that it checks for opportunities presented to the opponent by the planner's own move, thus ensuring that this mistake will be avoided in the future (see figure5). The new rule says roughly the following: 'To see what the world will be like after making move M, see what you would do in the opponent's situation at the current time, and assume

that he will make that move, as long as your move M doesn't enable a better move for him." When the same example is run with the new rule antecedant, the computer does not make the same mistake.

```
(def-brule proj-factor-3
  (world (move move) (player player) result (weight integer)
         (opp-move move) (time time)
         (better-move move) (better-goal goal)
         (better-value integer) (orig-value integer) )
         (project-factor proj-factor-1.5-mod world move player
              result weight)
 (and (= world (world-at-time time))
      (decide (player-opponent player)
        (possible-moves-at-time (player-opponent player) time)
      world simple-dec-factors opp-move)
      (= result (world-after-move opp-move
                   (world-after-move move world)))
      (= weight 1)
      ; here's the new stuff:
      (no (and (move-enables-move move better-move)
               (move-possible better-move (current-time))
               (move-legal better-move)
               (evaluate result eval-factors
                  (player-opponent player) orig-value)
               (evaluate (world-after-move better-move
                            (world-after-move move world))
                  eval-factors (player-opponent player)
                  better-value)
               (> better-value orig-value) ))))
```

Figure 5: The Modified Projection Method

4 Conclusion

Learning how to make decisions in a domain is a critical aspect of intelligent planning behavior. The ability of a planner to adapt its decision-making to a domain depends in part upon its ability to optimize the tradeoff between the sophistication of its decision procedures and their cost. Since it is difficult to optimize this tradeoff on *a priori* grounds alone, we propose that a planner start with a relatively simple set of decision procedures, and add complexity in response to experience gained in the application of its decision-making to real-world problems. Our model of this adaptation process is based on the explanation of failures, in that it is the analysis of bad decisions that drives the improvement of the decision procedures. We have developed a test-bed system for the implementation of planning models employing such an approach, and have demonstrated the ability of such a model to improve its procedure for projecting the effects of its moves in chess.

References

Birnbaum, L. and Collins, G. 1988. The transfer of experience across planning domains through the acquisition of abstract strategies. In J. Kolodner, ed., *Proceedings of a Workshop on Case-Based Reasoning,* Morgan Kaufmann, San Mateo, CA, pp. 61-79.

Collins, G. and Birnbaum, L. 1988a. An explanation-based approach to the transfer of planning knowledge across domains. *Proceedings of the 1988 AAAI Spring Symposium Series: Explanation-Based Learning,* Stanford, CA, pp. 107-111

Collins, G. and Birnbaum, L. 1988b. Learning strategic concepts in competitive planning: An explanation-based approach to the transfer of knowledge across domains. Research report no. UIUCDCS-R-88-1443, University of Illinois, Dept. of Computer Science, Urbana, IL, 1988.

Dejong, G. and Mooney, R. 1986. Explanation-based learning: An alternative view. *Machine Learning,* vol. 1, pp. 145-176.

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

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

Hammond, K. 1986. Case-based planning: An integrated theory of planning, learning, and memory. Research report no. 488, Yale University, Dept. of Computer Science, New Haven, CT.

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.

Minton,S. 1984. Constraint-based generalization: Learning game-playing plans from single examples. *Proceedings of the 1984 AAAI Conference,* Austin, TX, pp. 251-254.

Mitchell,T., Keller, R. and Kedar-Cabelli,S.1986. Explanation-based generalization: A unifying view. *Machine Learning,* vol. 1, pp. 47-80.

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

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