Learning Strategic Concepts from Experience: A Seven-Stage Process

Michael Freed
Northwestern University
The Institute for the Learning Sciences
Evanston, Illinois 60201

Abstract

One way novices improve their skill is by learning not to repeat mistakes. Often this requires learning entirely new concepts which must be operationalized for use in plans. We model this learning process in seven stages, starting with the generation of expectations which, when proven faulty, invoke mechanisms to modify decision-making mechanisms in order to prevent the failure from occurring again. This process is demonstrated in the context of our testbed system which learns new rules for detecting threats, formulating counterplans and other cognitive tasks. It is shown how this process may be used to learn the concept of *immobility* as it occurs in the domain of chess.

Novice Acquisition of Strategic Concepts

Some of the concepts needed to successfully practice a skill will be unknown to a novice when beginning to learn the skill. A novice basketball player, for instance, will be made aware of the concept of *scoring a basket* in the course of learning the rules of the game. However, the concept of a *fake*, in which a player pretends to go one way but actually goes another, may not be evident unless the player is told about it or experiences it first hand[1]. Novices advance their level of skill by discovering such concepts and using them in plans.

One way human novices improve their abilities is by learning not to repeat mistakes. A novice basketball player with the job of preventing an opponent with the ball from moving past him might believe that his opponent's leftward glance indicates imminent travel in that direction. In anticipation, the novice player may go left to block his opponent's motion. However, the opponent may look left and then go right, employing a fake to get past. The novice's mistake stems from an otherwise plausible assumption: namely that the direction of an opponent's gaze is predictive of the opponent's direction of motion. This assumption makes the novice vulnerable to a fake.

[1] Concepts may also be imported from other domains (see e.g. Collins and Birnbaum, 1988). The fake, for instance, may be learned from experience with almost any sport.

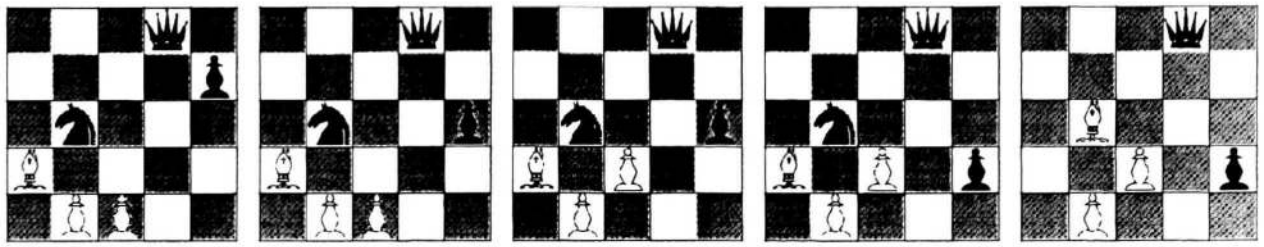
An approach to learning from failures

We have been exploring an approach to these problems in a system that learns to improve its chess playing abilities (see, e.g., Collins, Birnbaum and Krulwich, 1989; Birnbaum, Collins, Freed and Krulwich, 1990). The system learns when expectations about events in a chess game prove faulty. For example, the system expects to be able to block all capture-threats to pieces. If the opponent captures a piece, the expectation fails. Expectations are justified by a set of assumptions about what can and cannot happen in the domain and about what the system will do to insure that expectations succeed. Often, an expectation will fail because an opponent employs some tactic which the system was unaware of and which was therefore not accounted for by the assumptions underlying the expectation.

An expectation's justification makes assumptions not only about the 'physics' of the domain but also about the decision-making process employed by the planner (Birnbaum, Collins, Freed, Krulwich, 1990). For instance, the expectation that all threats against pieces will be blocked is justified in part by the assumption that all threats will be detected. Decision-making in our system is performed by a set of components, each designed to perform some domain-independent task such as detecting threats or formulating counterplans. Components are implemented as sets of rules which provide different methods for achieving the component's purpose. Learning to improve performance involves adding to or altering the set of rules in one or more of these components.

Case Study: The Pin

In the course of learning chess, novice players are almost invariably victimized by *the pin* which is a configuration of pieces achieved by positioning one piece



t_0 t_1 t_2 t_3 t_4

Figure 1: The pin: computer (black) to move at time t_0

on a line of attack that includes two opposing pieces so that the closer of the two pieces cannot be moved without exposing the other piece to the threat of imminent capture. In figure 1 at t_0, white's bishop is pinning black's knight; moving the knight would allow black's queen to be captured. In effect, the knight is immobilized — not because the rules of the game prevent movement, but because the consequent loss of the queen is undesirable. The opponent can exploit this situation in several ways; the simplest is to attack the immobilized knight. The player is then left in the position of choosing between leaving the pinned knight in place and allowing it to be captured, or moving it away, thereby permitting capture of the queen.

Not knowing about the pin tactic or how a pin might be exploited by the opponent, the computer might decide to advance its pawn at t_1 reasoning that there are no particularly dangerous threats or appealing opportunites and that advancing the pawn might pay off in the future. In coming to this conclusion the system looks at several moves including using the knight to capture the rightmost pawn. This move is rejected because it would expose the queen to attack from the opponent's bishop. In rejecting the pawn capture, the computer has noted the role of its knight in protecting the queen and has effectively committed itself to keeping the knight immobile. However, it does not yet construe this situation as a threat or have even implicit knowledge of the tactical concept of a pin. The opponent is in a position to exploit this situation by advancing its pawn to attack the knight. It does so after t_1, leaving the system in a *double bind* — i.e. a situation in which all plans to prevent one loss are mutually exclusive with plans to prevent the other loss. Since the queen is more valuable the computer decides to sacrifice the knight.

Learning a New Strategic Concept

In our model of the learning process, a strategic concept is discovered and operationalized following the failure of an expectation (see, e.g. Birnbaum, Collins, Freed, Krulwich, 1990). This process can be decomposed into a series of steps as summarized in figure 2, each of which will be described following an abbreviated account of how this process is used to learn about pins.

Before encountering the scenario of figure 1, the system prepares to learn by deciding what to expect and how often to check whether its expectations are true. In particular, it decides to have the expecation that all capture threats against its pieces will be blocked and to monitor this expectation by checking it after every opponent move. When the opponent takes the computer's knight after t_3, the computer's expectation monitor signals a failure immediately afterward at t_4. The signalled failure invokes diagnostic machinery which examines the computer's last few moves in search of an alternative move which would have avoided the loss of the knight (or anything worse). It discovers that advancing its pawn at t_0 was a mistake; if it had moved its queen out of the bishops's path at t_0, no pieces would have been captured.

Having identified a better action, the system uses its explicit model of the decision-making process to determine which component must be modified to allow the

1. Generate expectation
2. Note an expectation-failure
3. Determine what action or inaction caused the failure
4. Determine what bad underlying decision(s) led to the faulty action
5. Generate one or more candidate modifications for preventing recurrence of the failure
6. Select the best modification
7. Inductively merge with similar modifications

Figure 2: Seven-stage failure-driven learning process

planner to make the better move in similar circumstances in the future. Put another way, the diagnostic machinery determines which component of the planner is to blame for making the wrong move. The underlying cause is found to be the threat-detection component's failure to construe a certain configuration of pieces (the pin) as a threat. To prevent this failure in the future, a rule which identifies such a configuration as a threat is generated and then added to the threat-detection component[2].

Generating expectations

Expectations are explicitly represented beliefs about what will or will not occur. When an expectation fails, i.e. when the belief is demonstrably in error, the system invokes learning machinery to prevent recurrence of the failure[3]. For novices, it is worth asking how the initial set of expectations is arrived at. One way to learn which expectations to monitor is to be told. Since all initial expectations are entered by hand, our system learns what to monitor in essentially this way. Another way is to generate them automatically (Doyle, Atkinson and Doshi, 1986). Automatic generation of expectations should accompany the generation of new goals[4] For example, in learning to detect and avoid pins, the system should also learn to *expect* to avoid being pinned. The new expectation provides a basis for future learning.

Notice an expectation failure

Having generated a set of expectations, the system must devote resources to monitoring these expectations and detecting when they fail. If the system has a lot of expectations and checks them frequently, looking for failures can be very costly. Therefore decisions must be made about which expectations to monitor and when.

Many factors constrain the set of expectations which can profitably be monitored at any particular time. Clearly irrelevant expectations such as those used exclusively in one domain — say checkers — can not be profitably monitored while engaged in another domain such as chess. Even within a domain, the best time to monitor an expecation will vary. For instance, the expectation in our chess-playing planner that all capture-threats will be blocked is monitored by checking it after each opponent move; an expectation that a plan will prove workable should be monitored during the plan evaluation process.

Determining what action or inaction caused the failure

When an expectation-failure is detected, the learning machinery is invoked to diagnose the problem and repair the system. The first step in diagnosis is determining which executed action led to the failure. Locating this action is a matter of searching backward from the time of the failure for a time when the system had an alternative which, if taken, would have prevented the failure and would not have led to an equally bad or worse failure (cf. Minton, 1985).

Let us consider the pin example once again. At t_4 the expectation failure is noticed and the search for a faulty action begins. The search proceeds by examining the system's opportunites to act in reverse temporal order, starting with the time of the failure. The most recent opportunity to act before t_4 occurred at t_2, when the planner advanced its pawn. At this point, the system could have saved the knight and prevented the failure by moving it away from the attacking bishop and pawn, but this move would have been a poor choice since it would have exposed the queen and led to the even worse failure. At t_2, the computer was in a *double bind* — nothing could have prevented the loss of a piece.

The search continues until an opportunity to prevent the failure is found in which no double bind would negate the value of preventive action. At t_0, the system could have prevented the failure by moving its queen 2 spaces to the left, thereby defusing the pin and maintaining the guard on its knight. This would have prevented *any* expectation failure from occurring; therefore the action taken at this time is blamed for the failure.

This method of locating failure-producing actions is adequate for the simple turn-taking games we are studying. Similar methods have been proposed by Minton (1985) for learning about the *fork* and by Utgoff (1983) for constraint back-propogation.

Determining which component caused the failure

Every action in our system is the product of decisions made by components of the system's planner. One component is responsible for detecting threats and opportunities to achieve the system's goals. Another is responsible for formulating plans, another for comparing candidate plans to determine which is best. Our current formulation of the planning process includes 22 classes of decision and therefore 22 distinct planning components (Freed, 1991). When a faulty action is made, one or more of these components is responsible for the failure and should be modified to prevent

[2] Actually, many components besides the threat-detection component should be modified as a result of learning about pins. For example the counterplanning component will be modified so that detected pin-threats can be avoided. Other modifications cause pin-avoidance plans to be ranked against other plans, pinning plans to be employed tactically against opponents and expectations that the computer will not get pinned to be generated. The need to separate planning knowledge among components in this way is discussed in (Collins, Birnbaum, Krulwich, Freed, 1991)

[3] Alternately, the system may learn that an expectation is unrealistic in some respect and modify it (Krulwich, Birnbaum, Collins, 1988).

[4] Expectations are generated by inferentially determining which observable events indicate the success or failure of achievable goals.

has-legal-path(p_1 loc(p_1) loc(p_2) $path_A$)
$\wedge$ has-legal-path(p_1 loc(p_1) loc(p_3) $path_B$)
$\wedge$ has-legal-path(p_4 loc(p_4) sq $path_C$)
$\wedge$ has-legal-path(p_4 sq loc(p_2) $path_D$)
$\wedge$ blocks-path(p_2 $path_B$)
$\wedge \neg$ blocked($path_A$)
$\wedge \neg$ blocked($path_C$)
$\wedge \neg$ blocked($path_D$)
$\wedge$ I-own(p_2)
$\wedge$ I-own(p_3)
$\wedge \neg$ I-own(p_1)
$\wedge \neg$ I-own(p_4)

$\longrightarrow$ isa-threat(<...>)

Figure 3: Rule for detecting pin-threats against p_2

recurrence of the the faulty behavior. See (Birnbaum, Collins, Freed, Krulwich, 1990) for a description of the diagnostic process used by our system to determine which component is responsible for a faulty action.

In some cases, no single component can be held solely accountable for the failure. Diagnosing the failure in these cases means locating a (preferably minimal) set of components which may be modified to prevent recurrence of the failure. As the system was unaware of the pin tactic, many components require modification in order to defend against it and employ it against other agents. The component responsible for threat detection should be modified to detect the pin; the component for counerplanning must be able to generate a response to a pin-threat if one is has been detected. Learning and operationalizing knowledge of the pin entails modifying each component. For simplicity we will only consider modifications to the threat-detection component.

Generating candidate modifications

Once the cause of the failure has been traced to a faulty component, an explanation-based learning algorithm (cf. Mitchell, Keller and Kedar-Cabelli, 1986; DeJong and Mooney, 1986) is used to construct a modification to the component which will prevent recurrence of the failure. Modifying the threat-detection component to detect pins entails constructing a rule to signal a threat when the opponent is in a position to pin one of the computer's pieces.

Figure 3 shows a threat-detection rule which, if added to the rule set employed by the threat-detection component, will prevent recurrence of the piece-loss seen in the example. It should be noted that the rule detects a more specific class of threats than would be expected given a full understanding of the danger of being pinned; only pins which the opponent can exploit by attacking the pinned piece are detected. The process used by our system to produce component modifications is described by Krulwich (1991).

As it stands, the system learns to detect an overly specific class of threats because its single experience being pinned could not illustrate other ways being pinned could be costly. We will discuss how the system might learn the more general rule below.

Choosing the best modification

The learning element will frequently be able to generate several modifications that vary in effectiveness and cost. To see why this would be the case, consider the following informal example: a person breaks a traffic law — say by making a U-turn in an intersection — and consequently collides with another car. Attempting to prevent future collisions, the driver could decide: to be more careful when breaking that law in the future, not to break that law at all, not to break traffic laws in general or simply not to drive because it is too dangerous. In deciding which lesson to learn, the driver must reason about the tradeoff between effectiveness (at reducing the likelihood of future accidents) and cost (inconvenience).

In deciding which rule to learn, the system should consider such factors as generality and computational expense. The more general rules will typically be more expensive to use. For example, the threat-detection rule of figure 3 is more general and more expensive than a similar rule which only considers pins arising from bishops. Such a rule would prevent a recurrence of the exact failure seen in the example (in which the pinning piece was a bishop), but fail to detect pins made by rooks and queens. The more specific rule is cheaper to use because it only has to be checked on bishops.

Extensive knowledge of the domain may be used to predict what level of generality optimizes the tradeoff between effectiveness and computational expense. For the novice, such knowledge will normally be unavailable; however, conservative guesses about this level can be adjusted in light of new experience by inductively generalizing similar modifications (cf. Pazzani 1989).

Merging similar modifications

The rule for detecting pins illustrated in figure 3 detects too narrow a range of threats. In effect, it assumes that the only pins worth responding to are those which the opponent can exploit by attacking the pinned piece. However, the mobility-limit produced by the pin can have other costly consequences. For example, the computer will be unable to move a pinned piece to capture an opponent's piece without exposing itself to attack. Essentially any chess goal which can be served by moving a piece is compromised if the piece is pinned.

Although the necessary mechanisms are not yet implemented in our system, I will discuss how the more general rule may be produced after a second experience

of being pinned. Briefly, this second experience might take the following form:

One of the system's general expectations is that it will always be able to capture unguarded pieces. If the attack piece is pinned, the attack would have to be aborted, causing the expectation to fail. A learning episode, similar to the initial pin example, would then occur. The new threat-detection rule would be very similar to the initial pin rule, and could then be combined with it to yield a new rule, detecting all pins.

The crucial elements of this step are methods for identifying similar rules and for merging them (cf. Minton, 1988). For each of these steps we assume a concept called *reasons-to-move* for which *avoid-capture* and *capture-opponent-piece* are specializations. Associated with the latter concepts are abstract descriptions of the chessboard positions which enable them, represented in the same language as the rules employed by the detection component.

To notice the similarity of the two detection rules, the system must first label a subset of the propositions in the first rule (figure 3) as equivalent to the enabling conditions of *avoid-capture* and similarly label a portion of the second rule as equivalent to the enabling conditions for *capture-opponent-piece*. Treating these labels and the unlabelled propostions in each rule as features, a simple inductive generalization algorithm (e.g. Winston 1970) can be used to produce a new rule which includes only the common features of the first two. The new rule detects all pins.

Conclusion

We have presented a seven-stage process for discovering strategic concepts from expectation failures and constructing methods for utilizing this knowledge. We have demonstrated this within the context of our testbed system which learns new rules for detecting threats, counterplanning and other cognitive tasks. Future work will expand the more speculative stages of this process and elaborate the functional decomposition of planning tasks employed by the system.

Acknolwedgements

The ideas presented here were developed in collaboration with Gregg Collins, Larry Birnbaum and Bruce Krulwich. Additional thanks go to Matt Brand, Robin Burke, Eric Domeshek, Bill Ferguson and Louise Pryor for comments on this paper and discussions on the research presented. 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 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 throught the acquisition of abstract strategies. *Proceedings of the 1988 Case-Based Reasoning Workshop*

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.

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

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.

Collins, G., Birnbaum, L., Krulwich, B., and Freed, M. 1991. Plan debugging in an intentional system. Accepted for publication in the *Proceedings of Thirteenth IJCAI.*

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

Doyle, R., Atkinson, D., and Doshi, R. 1986. Generating perception requests and expectations to verify the execution of plans. *Proceedings of the Fifth National Conference on Artificial Intelligence,* pp. 81-88.

Freed, M. 1991. A Taxonomy of Planning and Learning Decisions. Unpublished manuscript.

Krulwich, B. 1991. Determining appropriate EBL target concepts in a multi-component planning system. Accepted for publication in the *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society.*

Minton, S. 1985. *A Game-Playing Program that Learns by Analyzing Examples* Technical report CMU-CS-85-130, Carnegie Mellon University School of Computer Science, Pittsburgh, PA.

Minton, S. 1988. *Learning effective search control knowledge: an explanation-based approach.* Technical Report CMU-CS-88-133, Carnegie Mellon University School of Computer Science, Pittsburgh, PA.

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

Utgoff, P. 1983. Adjusting bias in concept learning. *Proceedings International Machine Learning Workshop.*

Winston, P.H. 1970. Learning structural descriptions form examples. Reps No. TR-231, AI Laboratory, Massachusetts Institute of Technology. (Reprinted in P.II. Winston (Ed.). 1975. *The psychology of computer vision.* New York: McGraw-Hill, 157-209)