Thinking and Seeing in Game Playing: Integrating Pattern Recognition and Symbolic Learning

Susan L. Epstein
Pascal Abadie
Joanna Lesniak
Department of Computer Science
Hunter College and The Graduate School
The City University of New York
sehhc@cunyvm.cuny.edu

Jack Gelfand
Department of Psychology
Frank Midgley
Department of Computer Science
Princeton University
Princeton, NJ
jjg@phoenix.princeton.edu

Abstract

Although people rely heavily on visual cues during problem solving, it is non-trivial to integrate them into machine learning. This paper reports on three general methods that smoothly and naturally incorporate visual cues into a hierarchical decision algorithm for game playing: two that interpret predrawn straight lines on the board, and a third that uses an associative, hierarchical pattern database for pattern recognition. They have been integrated into Hoyle, a game learning program that makes decisions with a hierarchy of modules representing individual rational and heuristic agents.

Key words: machine learning, game playing, hierarchical decision algorithms, visual cues, pattern recognition

1. Introduction

Since the early work of Chase and Simon, researchers have noted that expert chess players retain thousands of patterns (Holding, 1985).

There has been substantial additional work on having a program learn specific patterns for chess (Berliner, 1992; Campbell, 1988; Flann, 1992; Levinson and Snyder, 1991). There is conflicting evidence as to whether or not expert game players learn to play solely by associating appropriate moves with key patterns detected on the board, but it is believed that pattern recognition is an important part of a number of different strategies exercised in expert play (Holding, 1985). In AI, visual cues have previously demonstrated their power as explicit search control directives and as hand-selected terms in an evaluation function (Gelernter, 1963; Samuel, 1963). Learned visual cues have also been derived from goal states with a predicate calculus representation (Fawcett and Utgoff, 1991; Yee, et al., 1990).

This paper integrates the pattern recognition and the explanatory heuristics that experts use into a program called Hoyle that learns to play two-person, perfect information, finite board games against an external expert. As in the schematic of Figure 1, whenever it is Hoyle's turn to move, a hierarchy of resource-limited procedures called *Advisors* is provided with the current game state, the legal moves, and any useful knowledge (described below) already acquired about the game. There are 22 heuristic Advisors in two tiers. The first tier sequentially attempts to compute a decision based upon correct knowledge, shallow search, and simple inference, such as Victory's "make a move that wins the contest immediately." If no single decision is forthcoming, then the second tier collectively makes many less reliable recommendations based upon narrow viewpoints, like Material's "maximize the number of your markers and minimize the number of your opponent's." Based on the Advisors' responses, a simple arithmetic vote selects a move that is forwarded to the game-playing algorithm for execution.

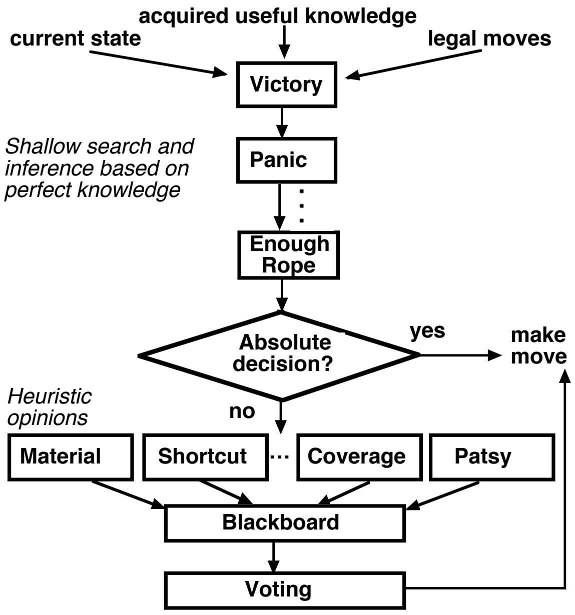

Figure 1: How Hoyle makes decisions.

The program learns from its experience to make better decisions based on acquired useful knowledge. *Useful knowledge* is expected to be relevant to future play and may be correct in the full context of the game tree. Examples of useful knowledge include recommended openings and states from which a win is always achievable. Each item of useful knowledge is associated with at least one learning algorithm. The learning methods for useful knowledge vary, and include explanation-based learning, induction, and deduction. The learning algorithms are highly selective about what they retain, may generalize, and may choose to discard previously acquired knowledge. When individual Advisors apply current useful knowledge to construct their recommendations, they integrate these learning strategies. Full details on Hoyle are available in (Epstein, 1992).

Visual cues are integrated into Hoyle's decision-making process as new Advisors in the second tier. These Advisors react to lines and clusters of markers without reasoning. This is prompted by our observation that people guide their play with frequently-observed patterns of pieces before they understand their significance. The distinction drawn here between thinking and seeing in game playing is an important one. By "thinking" we mean the manipulation of symbolic data, such as "often-used opening gambit;" by "seeing" we mean inference-free, explanation-free reaction to visual stimuli. The three Advisors described here are directed toward the construction of a system that both uses and learns visual cues. They provide powerful performance gains and promise a natural integration with learning. This paper indicates how Hoyle, already a multistrategy learning program, can integrate knowledge about visual cues, and methods to learn them.

2. Using Predrawn Lines

Morris games have been played for centuries throughout the world on boards similar to

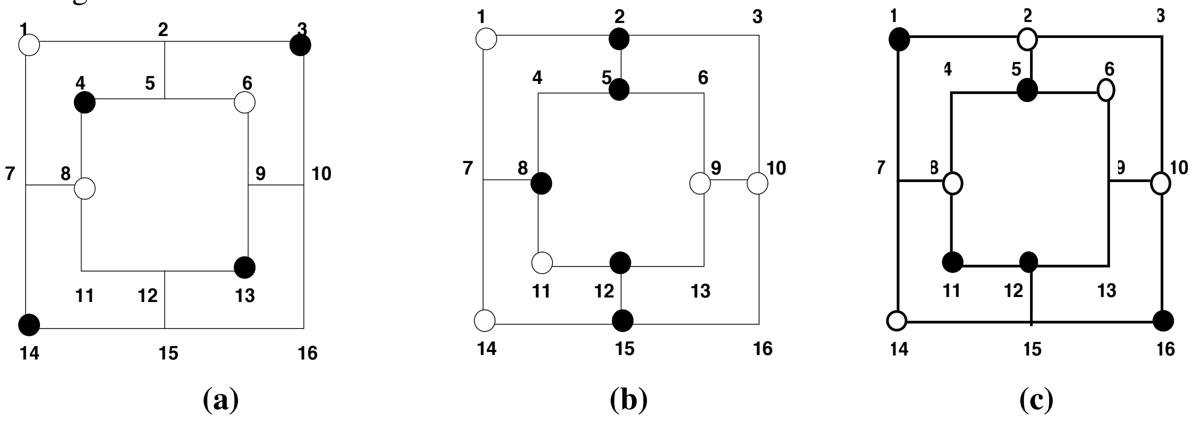

Figure 2: Some five men's morris states with white to move: (a) in the placing or the sliding stage, (b) and (c) in the sliding stage.

those in Figure 2. For clarity, we distinguish carefully here between a *game* (a board, markers, and a set of rules) and a *contest* (one complete experience at a game, from an initially empty board to some state where the rules terminate play). We refer to the predrawn straight lines visible in Figure 2 simply as *lines*. The intersection of two or more lines is a *position*. A position without a marker on it is said to be *empty*. Although the program draws pictures like those in Figure 2 for output, the internal, computational representation of any game board is a linear list of position values (e.g., black or white or blank) along with the identity of the mover and whether the contest is in the placing or sliding stage. The program also makes obvious representational transformations to and from a two-dimensional array to normalize computations for symmetry, but the array has no meaningful role in move selection. The game definition includes a list of predrawn lines and the positions on them.

A morris game has two contestants, black and white, each with an equal number of markers. A morris contest has two stages: a *placing stage*, where initially the board is empty, and the contestants alternate placing one of their markers on any empty position, and a *sliding stage,* where a turn consists of sliding one's marker along any line drawn on the game board to an immediately adjacent empty position. A marker may not jump over another marker or be lifted from the board during a slide. Three markers of the same color on immediately adjacent positions on a line form a *mill*. Each time a contestant constructs a mill, she *captures* (removes) one of the other contestant's markers that is not in a mill. Only if the other contestant's markers are all in mills, does she capture one from a mill. (There are local variations that permit capture only during the sliding stage, permit hopping rather than sliding when a contestant is reduced to three near a contest's end, and so on.) The first contestant reduced to two markers, or unable to move, loses.

2.1 The Coverage algorithm

When a marker is placed on any position on a line, it is said to *affect* all the positions on that line, including its own. The *coverage* of a position is the multiset of all distinct positions that it affects. A marker positioned where two lines meet, induces two copies of its position. Thus the coverage of 3 in Figure 2(a) is {1, 2, 2·3, 10, 16}. A set of markers belonging to a single contestant P produces a *cover,* a multiset denoted $C_P = \{c_1 \cdot v_1, c_2 \cdot v_2, \ldots, c_n \cdot v_n\}$ that lists the affected positions $v_1, v_2, \ldots, v_n$ and the number of lines c_i on which v_i lies that are affected by one of P's markers. In Figure 2(a), the white cover is $C_W = \{2 \cdot 1, 2, 3, 2 \cdot 4, 5, 2 \cdot 6, 2 \cdot 7, 2 \cdot 8, 9, 11, 13, 14\}$. The *cover difference* C~D for $C = \{c_1 \cdot v_1, c_2 \cdot v_2, \ldots, c_n \cdot v_n\}$ and $D = \{d_1 \cdot w_1, d_2 \cdot w_2, \ldots, d_m \cdot w_m\}$, is defined to be the multiset $C \sim D = \{x \cdot y \mid y = v_i$ for some $i = 1, 2, \ldots, n;\ x \cdot y \in C;\ y \neq w_j$ for any $j = 1, 2, \ldots, m\}$. In Figure 2(a), $C_B \sim C_W = \{10, 12, 15, 2 \cdot 16\}$ and $C_W \sim C_B = \varnothing$. We take the standard definitions from graph theory for adjacency, path, and path length.

A marker offensively offers the potential to group others along lines it lies on (*juxtaposition*) and to facilitate movement there (*mobility*), while it defensively obstructs the opposition's ability to do the same. The Coverage algorithm attempts to spread its markers over as many lines as possible, particularly lines already covered by the other contestant, and tries to do so on positions with maximal coverage. Assume, without loss of generality, that it is white's turn to move. In the placing stage, Coverage recommends a move to every empty position $c_i \cdot v_i \in C_B \sim C_W$ where $c_i > 1$. If there are no such positions, it recommends a move to every position in $C_B \sim C_W$ with maximal coverage. If there are no such positions of either kind, it recommends a move to every empty position with maximal coverage. In Figure 2(a) with White to move in the placing stage, $C_B \sim C_W = \{10, 12, 15, 2 \cdot 16\}$ so Coverage recommends a move to 16.

In the sliding stage, Coverage recommends each legal move that increases $|v_i|$, the number of the mover's distinct covered positions. Let (p,q) denote a sliding move from position p to position q. In Figure 2(b) the legal moves (1,7), (9,6), (9,13), (10,3), (10,16), (14,7) change $|v_i|$ by -1, +2, 0, 0, 0, -1, respectively, so Coverage recommends (9,6). In the sliding stage, however, one's cover can also decrease. Therefore, Coverage also recommends each legal slide to a position $c_i \cdot v_i \in C_B$ where $c_i > 1$ but for which $c_i \leq 1$ in C_W. In Figure 2(c), where $C_B = \{2 \cdot 1, 2 \cdot 2, 2 \cdot 3, 2 \cdot 4, 2 \cdot 5, 6, 7, 8, 10, 3 \cdot 11, 3 \cdot 12, 2 \cdot 13, 2 \cdot 14, 2 \cdot 15, 2 \cdot 16\}$, $C_W = \{2 \cdot 1, 2 \cdot 2, 2 \cdot 3, 2 \cdot 4, 2 \cdot 5, 2 \cdot 6, 2 \cdot 7, 2 \cdot 8, 2 \cdot 9, 2 \cdot 10, 11, 13, 2 \cdot 14, 15, 2 \cdot 16\}$, and the legal moves are (2,3), (6,9), (8,4), (8,7), (10,3), (10,9), (14,7), (14,15), those vertices are 11, 12, 13, 15, so Coverage can only recommend (14,15).

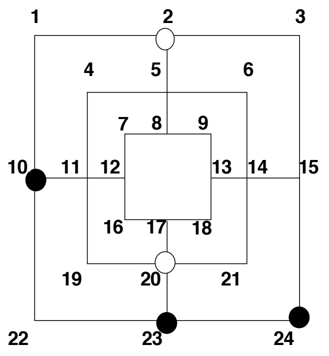

Figure 3: A placing state in nine men's morris, white to move.

2.2 The Shortcut algorithm

The Shortcut algorithm addresses long-range ability to move, and does so without forward search into the game graph. The algorithm for Shortcut begins by calculating the non-zero path lengths between pairs of same-color markers, including that from a marker to itself. For example, in Figure 3 the shortest paths between the white markers on 2 and 20 are [2, 5, 6, 14, 21, 20], [2, 3, 15, 14, 21, 20], and [2, 5, 4, 11, 19, 20]. Next, the algorithm selects those pairs for which the shortest non-zero length path between them is a minimum. It then retains only those shortest paths that meet the following criteria: every empty position lies on some line without a marker of the opposite color, and at least one position on the path lies at the intersection of two such lines. All three paths identified for Figure 3 are retained because of positions 5, 14, and 5, respectively. Shortcut recommends a placing or sliding move to the middlemost point(s) of each such path. In Figure 3, Shortcut therefore recommends moves to the midpoints 6 and 14, 15 and 14, and 4 and 11. Computation for this algorithm, styled as spreading activation, is very fast.

2.3 Results with Coverage and Shortcut

Prior to Coverage, Hoyle never played five men's morris very well. There are approximately 9 million possible board positions in five men's morris, with an average branch factor of about 6. After 500 learning contests Hoyle was still losing roughly 85% of the time. Once Coverage was added, however, Hoyle's decisions improved markedly. (Shortcut was not part of this experiment; data averages results across five runs.) With Coverage, Hoyle played better faster; after 32.75 contests it had learned well enough to draw 10 in a row. The contests averaged 33 moves, so that the program was exposed during learning to at most 1070.5 different states, about .012% of the search space. From that experience, the program was judged to simulate expert play while explicitly retaining data on only about .006% of the states in the game graph.

In post-learning testing, Hoyle proved to be a reliable, if imperfect, expert at five men's morris. When the program played 20 additional contests against the model with learning turned off, it lost 2.25 of them. Thus Hoyle after learning is 88.75% reliable at five men's morris, still a strong performance after such limited experience and with such limited retention in so large a search space. Additional testing displayed increasing prowess against decreasingly skilled opposition, an argument that expertise is indeed being simulated.

With a search space about 16,000 times larger than that of five men's, nine men's morris is a more strenuous test of Hoyle's ability to learn to play well. Because there is no definition of expert outcome for this game, we chose simply to let the program play 50 contests against the model. Without Coverage and Shortcut, Hoyle lost every contest. With them both, however, there was a dramatic improvement. Inspection showed that the program played as well as a human expert in the placing stage of the last 10 contests. During those 50 contests, which averaged 60 moves each, it lost 24 times, drew 17 times, and *won* nine times. (Some minor corrections to the model are now underway.) The first of those wins was on the 27th contest, and four of them were in the last six contests, suggesting that Hoyle was *learning* to play better. With the addition of less than 200 lines of game-independent code for the two new visually-cued Advisors, Hoyle was able to learn to outperform expert system code that was more than 11 times its length and restricted to a single game. The morris family includes versions for 6, 9, 11, and 12 men, with different predrawn lines. At this writing, Hoyle is learning them all quickly.

It should be noted that neither of these Advisors applies useful knowledge; instead, they direct the learning program's experience to the parts of the game graph where the key information lies, highly-selective knowledge that distinguishes an expert from a novice
(

Ericsson and Smith, 1991). If this knowledge is concisely located, as it appears to be in the morris games, and the learner can harness it, as Hoyle's learning algorithms do, the program learns to play quickly and well. As detailed here, this general improvement comes at a mere fraction of the development time for a traditional game-specific expert system.

3. Learning Patterns

Hoyle is a limitedly rational system that deliberately avoids exhaustive search and complete storage of its experience. Consistent with this approach, the work described here retains only a small number of the patterns encountered during play, ones with strong empirical evidence of their significance. The program uses a heuristically-organized database to associate small geometrical arrangements of markers on the board with winning and losing. The associative, hierarchical pattern database is a new item of useful knowledge. The first level of the database contains states; the second level contains patterns.

The pattern database is constructed by the *pattern classifier*, an associated learning algorithm, as follows. At the end of each contest, every state that occurred during the contest is cached in a fixed-size hash table, noting the sequence number of the most recent contest in which it appeared and whether Hoyle won, lost, or drew there. Each new state in the pattern database is now matched against nine templates for a 3×3 grid, adjusted for symmetry and shown in Figure 4. A "?" in a template represents an X, an O, or an empty space; "#" is the don't care symbol. A *subpattern* is an instantiation of a template, e.g., X's in the corners of a diagonal. (Preliminary empirical tests showed this to be the smallest set of effective templates.)

The second level of the pattern database consists of those subpatterns which appear in at least two states of the first level. Most states match several ways and therefore make multiple contributions to counting on the second level. Each subpattern also records the number of contests in which it participated in a win, a loss, and a draw. Thus a subpattern is a generalization over a class of states: those that have recently occurred with some frequency and contain simple configurations of pieces. Each subpattern is categorized as winning, drawing or losing based upon which kind of contest it appeared in most frequently.

```
? ? #    ? # #    # ? #    ? # ?    ? # #
# # #    # ? #    # ? #    # # #    # # #
# # #    # # #    # # #    # # #    # # ?

     # ? #    ? ? ?    ? # #    # ? #
     # # #    # # #    # ? #    # ? #
     # ? #    # # #    # # ?    # ? #
```

Figure 4. The set of templates used by the pattern classifier.

It is important to forget in the pattern database, primarily to discount novice-like play during the early learning of a game. There will be winning contests, and patterns associated with them, that were due to the learner's early errors. We have therefore implemented two ways to forget in the pattern database. First, when a hash table for either states or patterns is full, and a new entry should be made, the least recently used entry is eliminated, based on its most recent contest number. Second, at the end of every contest, the number of times each state was encountered is multiplied by 0.9.

Patsy is an Advisor that ranks legal next moves based on their fit with the pattern database. Patsy looks at the set of possible next states resulting from the current legal moves. Each next state is compared with the subpattern level of the database. A matched winning subpattern awards the state a +2, a matched drawing subpattern a +1, and a matched losing subpattern a -2. A state's score is the total of its subpattern values divided by the number of subpatterns in the cache. Patsy recommends the move whose next state has the highest such score. Ties are broken by random selection among the best moves.

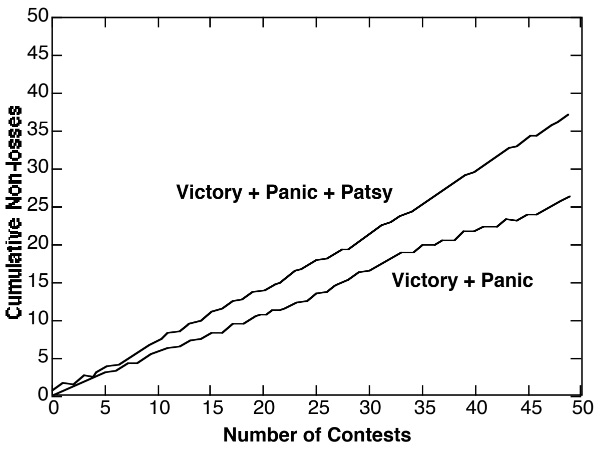

Figure 5. The performance of Hoylite with and without Patsy.

Patsy was tested within a severely pared-down version of Hoyle, called Hoylite here. Hoylite has only two of Hoyle's original Advisors, plus Patsy. The pattern classifier forms categories based on observed game states and associates responses to the observed states by learning during play. The hash table sizes were limited to 50 game states and 30 subpatterns. Three tournaments between Hoylite and a perfect tic-tac-toe player were run to assess the performance of Hoylite. The perfect player was a look-up table of correct moves. Each tournament was continued for 50 contests. The average cumulative number of Hoylite's wins and draws is plotted against contest number in Figure 5. The graph compares Hoylite's average performance against the perfect contestant with and without Patsy. Clearly Hoylite performs consistently better with Patsy.

There are many games that are played on a 3×3 grid. At this writing we are testing whether the same pattern templates in Figure 4 apply to several other games. We are also gradually adding Hoyle's Advisors to Hoylite, to see what conflicts, if any, arise. Finally, we are experimenting with more sophisticated pattern classifiers, ones that model the response of the human eye to arrangements such as lines of pieces and lines of open spaces.

4. Discussion

Predrawn game board lines are shown here to be important, readily accessible regularities that support better playing decisions. Historical data on patterns attractive to the human eye are demonstrably helpful in distinguishing good middlegame positions from mediocre ones. The brevity of the code required to capitalize on these visual cues for a variety of problems argues for the limitedly rational perspective of the architecture. The improvement the new Advisors have on play argues for the significance of visual representations as an integral part of decision making. When predrawn board lines are taken as visual cues for juxtaposition and mobility, Hoyle learns to play challenging games faster and better. Coverage and Shortcut in no way diminish the program's ability to learn and play the broad variety of games at which it had previously excelled (Epstein, 1992).

Our preliminary examination of the impact of a recognition-association competitive learning pattern classifier on several other expert knowledge sources and learning methods via a blackboard architecture is promising. The game played was a simple one, and only two of the 22 preexisting Advisors were included. A simple game was chosen to facilitate debugging the pattern classifier and measuring performance against an absolute standard. More than two Advisors would have obscured the contribution of the pattern-associative component. Hoylite's pattern classifier is quite simple and does not learn new templates; it only learns which game states are important for the given set of templates. It can be seen from these preliminary results that a pattern recognition component can be smoothly integrated into a game playing system that involves reasoning and limited search.

Heuristic Advisors are needed most in the middlegame, where the large number of possible moves precludes search. It has been our experience with more complex games, where one would have many Advisors, that openings are typically memorized, and that the endgame can be well-played with Advisors that reason about known losing and winning positions. Inspection reveals that Shortcut and Coverage contribute to decisions only in the middlegame, while Patsy works on the opening and middlegame. In the full version of Hoyle, other Advisors cover the opening, and an experience-driven partial retrograde analysis learns enough useful knowledge to tune the endgame. In Hoylite, the other two Advisors, Victory and Panic, address the endgame, leaving Patsy to consider patterns important at the earlier stages. All three new Advisors prove to filter the middlegame alternatives to a few likely moves, ones that might then benefit from limited search.

Acknowledgments

J. G. was partially supported by a grant from the James S. McDonnell Foundation to the Human Information Processing Group at Princeton University. S. L. E. was partially supported by NSF Grant 9001936.

References

Berliner, H. 1992. Pattern Recognition Interacting with Search. Tech. Rpt., CS-92-211, Carnegie Mellon University.

Campbell, M. S. 1988. Chunking as an Abstraction Mechanism. Ph.D. diss., CMU.

Epstein, S. L. 1992. Prior Knowledge Strengthens Learning to Control Search in Weak Theory Domains. *International Journal of Intelligent Systems* 7: 547-586.

Ericsson, K. A. and Smith, J. 1991. Prospects and Limits of the Empirical Study of Expertise. In *Toward a General Theory of Expertise,* ed. K. A. Ericsson and J. Smith. Cambridge: Cambridge University Press. 1-38.

Fawcett, T. E. and Utgoff, P. E. 1991. A Hybrid Method for Feature Generation. In *Proc. 8th Int'l Workshop on Machine Learning,* 137-141. Morgan Kaufmann.

Flann, N. S. 1992. Correct Abstraction in Counter-Planning: A Knowledge Compilation Approach. Ph.D. diss., Oregon State.

Gelernter, H. 1963. Realization of a Geometry-Theorem Proving Machine. In *Computers and Thought,* ed. E. A. Feigenbaum and J. Feldman. NY: McGraw-Hill. 134-152.

Holding, D. 1985. *The Psychology of Chess Skill.* Hillsdale, NJ: Lawrence Earlbaum.

Levinson, R. and Snyder, R. 1991. Adaptive Pattern-Oriented Chess. In *Proc. 8th Int'l Machine Learning Workshop,* 85-89.

Samuel, A. L. 1963. Some Studies in Machine Learning Using the Game of Checkers. In *Computers and Thought,* ed. E. A. Feigenbaum and J. Feldman. NY: McGraw-Hill. 71-105.

Yee, R. C., Saxena, S., Utgoff, P. E. and Barto, A. G. 1990. Explaining Temporal

Differences to Create Useful Concepts for Evaluating States. In *Proc. 8th Nat'l Conference on AI,* 882-888. AAAI Press.