The Integration of Visual Cues into a Multiple-Advisor Game-Learning Program

Susan L. Epstein
Hunter College and
The Graduate School of
The City University of New York
695 Park Avenue
New York, NY 10021 USA
sehhc@cunyvm.cuny.edu

Jack Gelfand
Department of Psychology
Green Hall
Princeton University
Princeton, NJ 08544 USA
jjg@phoenix.princeton.edu

Joanna Lesniak
Pascal Abadie
The Graduate School of
The City University of New York
33 West 42nd Street
New York, NY 10036 USA
leshc@cunyvm.cuny.edu
abahc@cunyvm.cuny.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 learning: two that interpret predrawn straight lines on the board, and one that uses an associative 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. One method relies upon a bounded pattern language for visual features, called BPL. As a direct result, the program now learns to play more difficult games faster and better.

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 & 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). Learned visual cues have also been derived from goal states with a predicate calculus representation (Fawcett & Utgoff, 1991; Yee, Saxena, Utgoff & Barto, 1990).

This work integrates both 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. Hoyle is based on a learning and problem-solving architecture for skills called FORR, predicated upon multiple rationales for decision making (Epstein, 1992a). Hoyle learns to play in competition against a hand-crafted, external expert program for each specific new game. 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 23 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. Further details on Hoyle are available in (Epstein, 1992b).

A FORR-based 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 is probably correct in the full context of the game tree. Each item of useful knowledge is associated with at least one learning algorithm whose learning strategy (or strategies, like explanation-based learning or induction) vary with the item. The learning algorithms are highly selective

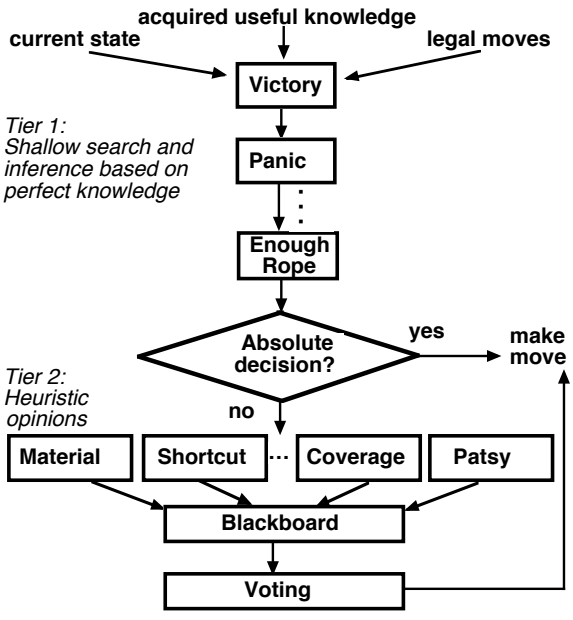


Figure 1: How Hoyle makes decisions.

Advisor	Tier	Relevant Information	Useful Knowledge	Learning Strategy
Victory	1	Next move is a win	Significant states	Deduction
Panic	1	Null move heuristic detects a loss	Significant states	Deduction
Leery	2	Questionable move	Play failure and proof failure	Abduction
Pitchfork	2	Applicable fork	Fork knowledge base	Explanation-based learning
Open	2	Reliable opening	Opening knowledge base	Induction
Patsy	2	Pattern associated with win or loss	Pattern knowledge base	Associative pattern classifier

Table 1: Examples of Hoyle's Advisors. These procedures are general rationales that reference one or more items of useful knowledge, each supported by its own learning strategy.

about what they retain, may generalize, and may choose to discard previously acquired knowledge. Individual Advisors apply current useful knowledge to construct their recommendations. Examples of Hoyle's Advisors, along with their useful knowledge and its associated learning strategy, appear in Table 1.

There is an important distinction drawn in this paper between thinking and seeing in game playing. 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. This acquired "sight" is compiled expert knowledge. Hoyle "sees" through visual cues integrated into its decision-making process with three new Advisors in the second tier. Consistent with Hoyle's limited rationality, these Advisors react to lines and clusters of markers without any human interpretation of their significance and without reasoning. The three are a step 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. The first two new Advisors rely on predrawn lines on the game board; other notices patterns of markers. Although the descriptions of these Advisors are presented in the context of a single family of games, Section 5 extends the results to Hoyle's entire domain.

2. Using Predrawn Lines

Morris games have been played for centuries throughout the world on boards similar to those in Figure 2. We use them as examples here because their substantial search spaces (ranging from 9 million to 776 billion states) provide interesting challenges. 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*. Any location where a marker may legitimately rest is called a *legal position* or simply a *position*. (In morris games, 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,

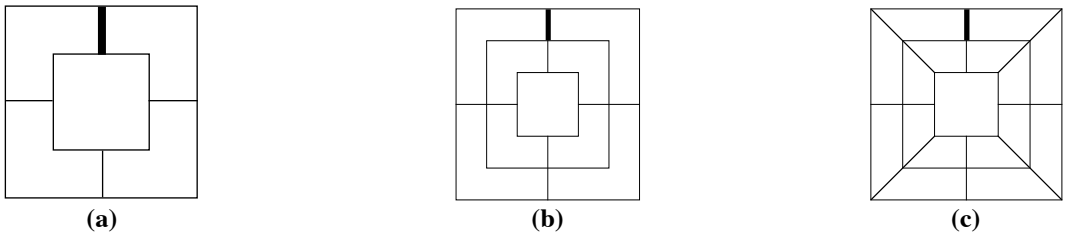


Figure 2: Some morris boards with (a) 16 positions for five or six men's morris, and 24 positions for (b) nine men's morris and for (c) 11 men's morris. The darkened line segments represent the metric unit discussed in Section 4.

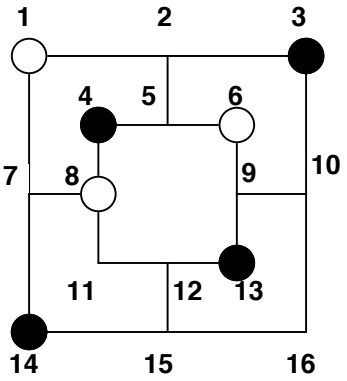


Figure 3: A five men's morris state with white to move in the placing or the sliding stage.

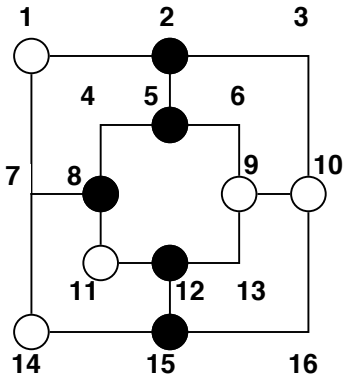


Figure 4: A five men's morris state with white to move in the sliding stage.

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 Intuitively, 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. Intuitively:

- the coverage of a marker represents the positions over which it has potential influence to cluster and move
- the cover of a contestant represents the combined influence of her markers
- the cover difference between two contestants represents those positions over which the first has potential influence and the second does not.

More formally, 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 *multiset* is a set in which each element may appear more than once. An element that appears more than once is preceded by a count of the number of its occurrences. Thus $\{a, 2 \cdot b\}$ denotes a set of one a and 2 b's. A multiset can represent *repeated* as well as singular influence, a helpful way to describe the relative potential of positions.

A marker positioned where two lines meet induces two copies of its position in its coverage. Thus the coverage of the marker on 3 in Figure 3, for example, is $\{1, 2, 2 \cdot 3, 10, 16\}$. 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 positions $v_1, v_2, \ldots, v_n$ that P's markers affect and the number of times c_i which each v_i is so affected. C_P is the union of the coverages of the positions where P has markers. In Figure 3, for example, 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 of

$$C = \{c_1 \cdot v_1, c_2 \cdot v_2, \ldots, c_n \cdot v_n\}$$

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 $j = 1, 2, \ldots, m\}$.

Cover difference is not commutative. In Figure 3, for example, $C_B \sim C_W = \{10, 12, 15, 2 \cdot 16\}$ while $C_W \sim C_B = \varnothing$.

The Coverage algorithm attempts to spread its markers over as many positions as possible, particularly positions already covered by the other contestant, and tries to do so on positions that are multiply covered (i.e., $c_i > 1$) by the other contestant. 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 $C_B \sim C_W$ where $c_i = 1$. If there are no such positions of either kind, it recommends a move to every empty position where $c_i > 1$. For example, in Figure 3 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 4 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

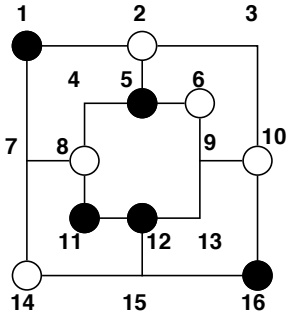


Figure 5: Another five men's morris state with white to move in the sliding stage.

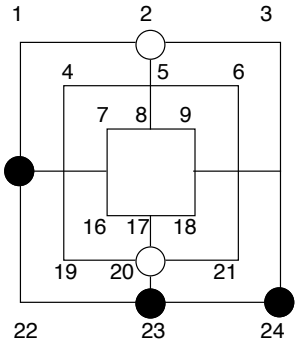


Figure 6: A placing state in nine men's morris, white to move.

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 5, for example, where the legal moves are (2,3), (6,9), (8,4), (8,7), (10,3), (10,9), (14,7), and (14,15), and

$$C_B = \{2 \cdot 1, 2 \cdot 2, 2 \cdot 3, 2 \cdot 4, 2 \cdot 5, 6, 7, 8, 10, 3 \cdot 11, 3 \cdot 12, 2 \cdot 13, 2 \cdot 14, 2 \cdot 15, 2 \cdot 16\}$$

$$C_W = \{2 \cdot 1, 2 \cdot 2, 2 \cdot 3, 2 \cdot 4, 2 \cdot 5, 2 \cdot 6, 2 \cdot 7, 2 \cdot 8, 2 \cdot 9, 2 \cdot 10, 11, 13, 2 \cdot 14, 15, 2 \cdot 16\}$$

those positions are 11, 12, 13, 15, so Coverage can only recommend (14,15).

2.2 The Shortcut algorithm The Shortcut algorithm addresses long-range ability to move, and does so without forward search into the game graph. We take the standard definitions from graph theory for adjacency, path, and path length. 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 6 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 6 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 6, Shortcut therefore recommends moves to the midpoints 6 and 14, 15 and 14, and 4 and 11. This algorithm, styled as spreading activation, is very fast.

2.3 Results with Coverage and Shortcut When predrawn board lines are taken as visual cues for juxtaposition and mobility, Hoyle learns to play challenging games faster and better. 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.

There are approximately 20 million states in akidada (referred to here as "six men's morris"). Although it has a substantially larger search space than five men's morris, akidada is played on the same board. Given the crowding that results, the game is actually somewhat easier to learn. With Coverage and Shortcut Hoyle learns to play expert-level akidada in 14 contests on average.

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,

Figure 7. A template set used by the original version of the pattern classifier for a 3 × 3 grid.

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 rapidly.

It should be noted that neither Coverage nor Shortcut 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 (Epstein, 1993; Ericsson & 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

One of the authors is an expert game-player. Initially he was asked to learn one new game every week. so that we might develop, for each new game, a perfect player algorithm to serve as Hoyle's opposition during learning. The games we gave him were progressively more difficult. It is a habit in our laboratory to sketch the game board on a piece of paper and then use coins to represent the markers. One week, when he arrived to report his progress, the game "board" was so well used that the coins had worn translucent paths on the paper. A protocol follows.

> I played this game for a very long time. At first I played it with a friend, but then, after a few hours, he tired of it, so I played it by myself. I played it a long, long time, for many hours. After a while I began to notice that patterns appeared. [When queried, he described these as small clusters of same-colored markers forming a V-like or L-like shape.] After an even longer time, I began to notice that, once these patterns appeared, something happened. I would win, maybe, or lose. Maybe not right away, but after a few moves. Then I figured out *why* those patterns made this happen, and here is the algorithm.

It was not the (correct) algorithm we were now interested in, but this remarkable description of learning. An accomplished game player was confronted with a task in which he could not bring his usual expertise to bear. As he persisted, some mental process collected visual cues for him, a process he had not consciously initiated. But once that process had results, and he noticed them, he could use those visual cues to play well and even to calculate why those visual cues were correct. If it worked for our expert, it could work for a machine.

Of course, game learning with pattern recognition is not new. De Groot proposed, and Chase and Simon refined, a recognition-association model to explain chess skill (Chase & Simon, 1973; de Groot, 1965). Despite a thoughtful refutation of their recognition-association theory, the idea of patterns as chunks in experts' memories has persisted (Holding, 1985). MACH integrated chunks identified by human master chess players from grandmaster games, into the evaluation function of Phoenix (George & Schaeffer, 1991). With this addition, Phoenix made better moves but no longer played in real time. Levinson modified chunks to include empty squares and threat-defense relations (Levinson, et al., 1991). His chess-learning program, Morph, learns and stores about 5000 patterns that it uses to play chess.

The novelty of the approach described in this section is that it integrates a real-time, low-level pattern learner into a high-level reasoning framework. Our premise is that visual patterns are not a primary reasoning device (an argument Holding supports with substantial empirical evidence) but that they *are* an important fallback device, just as they were in the protocol.

Hoyle, as a limitedly rational program, deliberately avoids exhaustive search and complete storage of its experience. Therefore when Hoyle learns patterns, it retains only a small number of those encountered during play, ones with strong empirical evidence of their significance. The program uses a heuristically-organized, fixed-size database to associate small geometrical arrangements of markers on the board with winning and losing. The associative pattern database is a new item of useful knowledge.

The pattern database is constructed from templates by the *pattern classifier*, an associated learning algorithm. A *template* is a partial description of the location of markers on the board. A "?" in a template represents an X, an O, or an empty space; "#" is the don't care symbol. A sample template set for a 3 × 3 grid is shown in Figure 7. The middlemost template, for example, could be instantiated as "X's on the endpoints of some diagonal." The templates in Figure 7 were chosen from experience; more general methods for their construction appear in the next section.

At the end of each contest, the pattern classifier matches every state against a set of templates, adjusting for all the symmetries of the two-dimensional plane. A *pattern* is an instantiation of a template, e.g., X's in the corners of a diagonal. The pattern database consists of those patterns which have appeared at least twice during play. Most states match one or more templates several ways and therefore make multiple contributions to the pattern database. Each pattern also records *pattern strengths:* the number of contests in which it participated in a win, a loss, and a draw.

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 the database is full and a new entry should be made, the least recently used entry is eliminated. Second, at the end of every contest, the pattern strengths are multiplied by 0.9. Thus a pattern is a generalization over a class of states: those that have recently occurred with some frequency and contain simple configurations of markers. The pattern classifier forms categories (winning, drawing or losing) based on observed game states and associates responses to the observed states by learning during play.

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 pattern level of the database. A matched winning pattern awards the state a +2, a matched drawing pattern a +1, and a matched losing pattern a -2. A state's score is the total of its pattern values divided by the number of patterns in the cache. Patsy recommends the move whose next state has the highest such score. Ties were broken by random selection.

To show Patsy's contribution to game learning, the Advisor was tested first with a severely pared-down version of Hoyle that had only two of the original Advisors, Victory and Panic, plus Patsy. The pattern store was limited to 30.

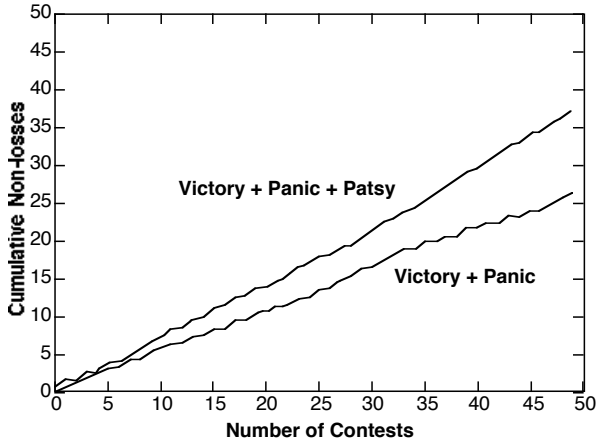


Figure 8. The performance of a pared-down version of Hoyle, with and without Patsy.

Three 50-contest tournaments between a perfect tic-tac-toe player and this program were run to assess its performance. The perfect player was a look-up table of correct moves. The average cumulative number of wins and draws for the learning program is plotted against contest number in Figure 8. The graph compares the pared-down version's average performance, with and without Patsy, against the perfect contestant's. Clearly the pared-down version performed consistently better with Patsy.

This experiment showed that a pattern recognition component could be a smoothly integrated, contributing element of a game playing system. 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.

Patsy was then tested, again with the templates of Figure 7, in the full version of Hoyle on nine different games defined on a 3 × 3 grid (e.g., lose tic-tac-toe, les pendus, marelle). Once again the pattern database was limited to 30. There was no degradation of prior performance, i.e., the games that Hoyle had been able to learn to play expertly without Patsy it could also learn to play just as quickly and well with Patsy. In addition, statistics indicated that Hoyle frequently judged Patsy among the most *significant* of its Advisors for learning such a game, i.e., an Advisor whose recommendations agreed with expert play more often than those of most Advisors. The extension of Patsy to more difficult games, where it is expected to make an important contribution, appears in the next section.

4. Identifying Board-Specific Templates

Although Patsy does eventually contribute to good move choices, its original templates were for a specific game board (a 3 × 3 grid) and predetermined by people. Our new version of Patsy has a template generator that is applied to deduce templates from the topology of the board.

Hoyle now calculates the templates for Patsy when it encounters a new game. These templates are detected as instantiations of a feature language with a visual bias called *BPL* (Bounded Pattern Language). Let the version of the game board drawn for output, as in Figure 2, be called the *picture*. The program constructs a *metric graph* whose nodes are the legal positions for markers, e.g., in morris games the intersection points of predrawn lines. There is an edge between a pair of nodes in the metric graph if and only if it is sometimes legal to move a marker in one turn from one of the corresponding positions to the other, e.g., in morris games the metric graph's edges correspond to predrawn line segments. Each edge is labeled with the Euclidean distance between its two positions that is measurable from the picture. The metric graph for five men's morris appears in Figure 9, with letters to identify the

nodes. Now Hoyle computes the *metric unit* for the game board, the smallest label in the metric graph. (Each game

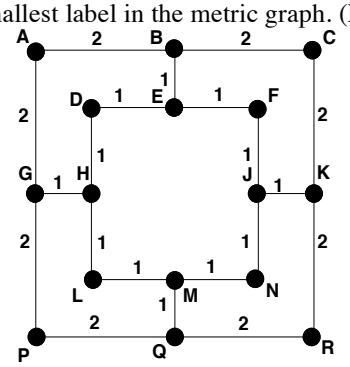


Figure 9: The metric graph for five men's morris.

board in Figure 2 has a single bold line that represents an instance of its metric unit.) Next the program finds instantiations of the BPL expressions. A BPL expression is a class of shapes with ?'s in its required positions and #'s in its irrelevant ("don't care") ones.

There are five valid expressions in BPL: straight lines, squares, diagonals, L's and triangles (the equivalent of the V's in the protocol). Each is defined below; examples list the vertices in Figure 9 that are identified with ?'s to form the appropriate shape. BPL takes a single *field of view parameter* that determines the area over which a template may extend. Field of view has been set to 4 in the following definitions. The valid BPL expressions are

• A BPL *straight line* is a predrawn straight line segment in the picture whose endpoints are no more than 4 metric units apart. A BPL straight line may include more than two positions. The endpoints, and no other points, are marked by a ?.

• A BPL *square*, with any orientation, is a square whose vertices are positions and whose edges are predrawn straight line segments in the picture. A BPL square labels its 4 vertices, and no other points, with ?'s. Every pair of vertices of a BPL square are no more than 4 metric units apart.

• Given a square whose vertices are positions and whose edges may or may not be predrawn in the picture, a BPL *diagonal* is any contiguous segment on a diagonal of such a square, such that the endpoints of the segment are no more than 4 metric units apart. A BPL diagonal may include more than two positions. The endpoints, and no other points, are marked by a ?.

• A BPL *L* is a pair of perpendicular line segments that form a single right angle in the picture. The three endpoints of the line segments must be positions, and every pair of such positions must be no more than 4 metric units apart. The two "legs" of a BPL L may be different lengths. A BPL L may include more than three positions. The three endpoints of the line segments, and no other points, are marked by a ?.

• A BPL *triangle* is a triangle whose vertices are positions, all three of whose edges are predrawn in the picture, and along which every pair of positions is no more than 4 metric units apart. The sides of a BPL triangle may be different lengths. The three vertices of a BPL triangle, and no other points, are marked by a ?.

A *board-specific template* is an instantiation of a BPL expression. The concern is, of course, that there not be too many board-specific templates, nor too few. Too many templates could overwhelm a learner, particularly one with a small pattern cache. Too few templates could overlook important patterns. Patsy must have enough time to test for patterns with all its templates in the few seconds of computing time allotted to it. The parameterized field of view is a heuristic that must balance power with efficiency. For now we set field of view to 4, because it seems to strike that balance.

For the five men's morris board with field of view 4, Hoyle generates only 13 templates when it applies BPL to the board in Figure 9: straight lines AB, AC, DE, BE, and DF; the square DFNL; the diagonal DN; and the L's BED, EDH, ABE, BAG, EDL, and FDL. (Recall that templates are unique up to the symmetries of the two-dimensional plane, so that, for example, DN and FL are considered the same template.) We have prototyped the instantiation of BPL for several game boards and found that the number of templates continues to be quite reasonable. (For nine men's morris, it is 18.)

It is important to understand that board-specific templates are calculated only once, when Hoyle first encounters the game board. Once the templates are identified, then the patterns that are instantiations of them are learned at the same time as any other useful knowledge. A pattern is an instantiation of a template in one of four ways: all blanks, all markers of the first contestant, all markers of the second contestant, or all occupied by a specific configuration of markers. The *response* associated with a pattern's presence is tabulated as a win for the first contestant, a win for the second contestant, or a draw.

The new version of Patsy uses the board-specific templates much the way the old version used the templates in Figure 7, but with several important modifications:

• Ties are represented by recommending every move to a highest-scoring state.

• Only two states per contest are used to generate the patterns, the state in which each contestant made its last non-forced move.

• Only patterns whose predictions are perfectly reliable are retained. If a pattern enters the pattern database and its response changes, it is discarded. This insistence on flawless correlation expects that contestants eventually play very well, else Hoyle will repeatedly discard a pattern that should be learned but appears in a "misplayed" contest.

• Pattern matching is *hierarchical,* i.e., if a pattern is matched to the current state, no attempt is made to match any of its subpatterns. For example, in Figure 9, DFNL is a BPL square. If an instantiation of DFNL were found with all white markers, then the white L's of the form FDL would be ignored, since they are subpatterns of the square. Hierarchical pattern matching was essential to Hoyle's ability to learn the game; recommendations based on subpatterns as well as

patterns overemphasized the importance of a single large pattern, and adversely affected decision making.

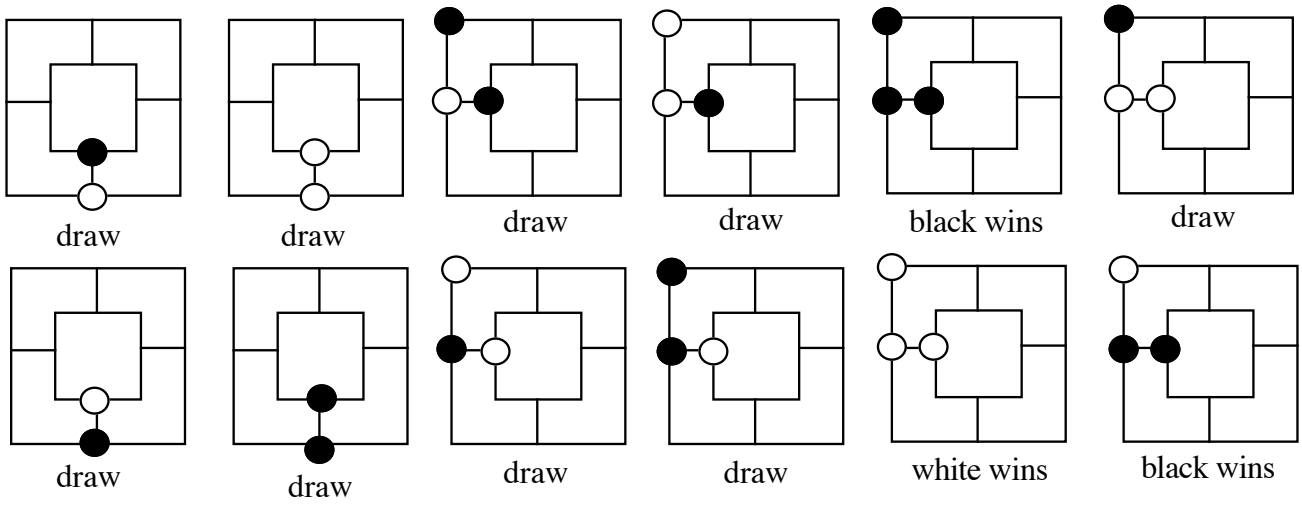


Figure 10: Hoyle's learned patterns for five men's morris, with their responses.

The new version of Patsy is a more sophisticated Advisor, and replaces the old one. At this writing, the new version of Patsy has been tested for five men's morris. Hoyle computed the 13 templates and applied them to learn 12 patterns, shown with their responses in Figure 10. The pairing of states in the first and second lines of Figure 10 is deliberate. Except for the rightmost pair, each pattern in the second row may be derived from the pattern directly above it by interchanging the colors of the markers and the result. The advantage that black seems to derive in the last pair may be related to the fact that black moves first in this game.

Patsy is no less fallible than any other second tier Advisor. Although the patterns of Figure 10 have arisen in every run, they are not perfect. One can, for example, construct a game state including a "winning" pattern from which the "winner" must inevitably lose the contest.

We have run preliminary tests on the full version of Hoyle with and without Patsy. The program played five men's morris against the external expert program until Hoyle could draw 10 consecutive contests. Then we turned learning off and tested the program against the expert program and three other programs of gradually decreasing ability. (See (Epstein, 1993) for a full description of this process.) Hoyle with Patsy learned to play well slightly faster, and performed more reliably after training than Hoyle without Patsy. Further testing is required to demonstrate the statistical significance.

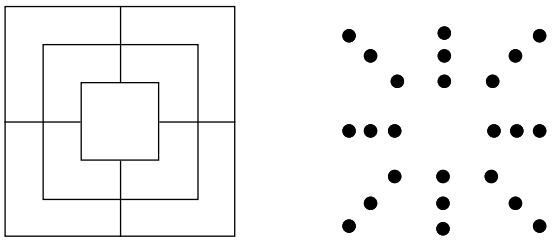
5. Discussion

All three new second-tier Advisors arose in the context of the class called morris games, which includes versions for up to 12 men on boards of various designs, like those in Figure 2. To confirm that the new Advisors are game-independent., we have tested Coverage and Shortcut on all the games Hoyle had previously learned, we have tested the earlier version of Patsy on those to which the Figure 7 templates are applicable, and we have tested the new version of Patsy on three games played on different boards. The three new Advisors in no way diminish the program's ability to learn and play the broad variety of games at which it had previously excelled (Epstein, 1992b). In many of the games all three were applicable and even gave excellent advice, although there was no dramatic improvement in learning time. (Recall that Hoyle already was able to learn these games quite efficiently, usually in less than 100 contests.) Heuristic Advisors are needed most in the middlegame, however, 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. All three new Advisors prove to filter the middlegame alternatives to a few likely moves, ones that might then benefit from limited search.

Current work includes extensions to other games. We are now in the expert development cycle for other members of the morris family: trique, nerenchi, murabaraba, and 11 men's morris. We build an expert model, Hoyle learns to defeat it, we observe and correct the model's flaws, and then we challenge Hoyle with the new version.

Unlike Morph's patterns, Hoyle's are purely visual, without an underlying rationale (Levinson, et al., 1991). We plan next to test a variety of off-the-shelf learning strategies (e.g., decision trees, neural nets) to learn pattern strengths for use with the new version of Patsy. Future work also includes fining the field of view value automatically, and augmenting BPL so that it includes bounded regions in its language.



(a) **(b)**

Figure 11: A pair of game boards for nine men's morris.

6. Conclusions

Hoyle is an ongoing exploration of how general domain expertise, like knowledge about how to play games, can be applied to develop expertise rapidly in a specific subdomain, like a particular game. The program is progressing through a sequence of games ordered by human-estimated degree of difficulty. At watersheds where knowledge must be added to continue Hoyle's progress through the list, it is important to assess what was missing, and why it was relevant. Until the morris games, there was little need for vision, except for economy afforded by symmetry.

Predrawn lines on a game board are important, readily accessible regularities that support better playing decisions. People find it far simpler, for example, to learn to play on a board with lines, like the one in Figure 11(a), than on a board that only has small dots to mark legal positions, like the one in Figure 11(b). The lines, we believe, serve as heuristics to formulate plans and guide search.

Historical pattern data is demonstrably helpful in distinguishing good middlegame positions from mediocre ones. The work on Patsy differs from other's work in that it does not address the strategic rationale behind the patterns, only their presence (Levinson, et al., 1991).

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 FORR architecture. FORR facilitates the transparent introduction of new, good reasons for making decisions. Testing a new Advisor is a simple task.

Finally, the improvement Shortcut, Coverage, and Patsy have on skill development argues for the significance of visual representations as an integral part of decision making. Each of them demonstrably improves Hoyle's ability to learn to play well.

Acknowledgments

This presentation was improved by conversations with Ron Kinchla and Stan Matwin, and by comments from several anonymous referees. 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

Berlin er, H. (1992). *Pattern Recognition Interacting with Search* (CMU-CS-92-211). Carnegie Mellon University.

Campbell, M. S. (1988). *Chunking as an Abstraction Mechanism*, Ph.D. thesis. Carnegie Mellon.

Chase, W. G. & Simon, H. A. (1973). The Mind's Eye in Chess. In W. G. Chase (Ed.), *Visual Information Processing* (pp. 215-281). New York: Academic Press.

de Groot, A. (1965). *Thought and Choice in Chess*. The Hague: Mouton.

Epstein, S. L. (1992a). Capitalizing on Conflict: The FORR Architecture. In *Proceedings of the Workshop on Computational Architectures for Supporting Machine Learning and Knowledge Acquisition, Ninth International Machine Learning Conference* Aberdeen, Scotland.

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

Epstein, S. L. (1993). Toward a Theory of Well-Guided Search. In *Proceedings of the AAAI Fall Symposium on Games: Planning and Learning* Raleigh.

Ericsson, K. A. & Smith, J. (1991). Prospects and Limits of the Empirical Study of Expertise: An Introduction. In K. A. Ericsson, & J. Smith (Ed.), *Toward a General Theory of Expertise - Prospects and Limits* (pp. 1-38). Cambridge: Cambridge University Press.

Fawcett, T. E. & Utgoff, P. E. (1991). A Hybrid Method for Feature Generation. In *Proceedings of the Eighth International Workshop on Machine Learning* (pp. 137-141). Evanston: Morgan Kaufmann.

Flann, N. S. (1992). *Correct Abstraction in Counter-Planning: A Knowledge Compilation Approach*, Ph.D. thesis. Oregon State University.

George & Schaeffer. (1991). Chunking for Experience. In D. F. Beal (Ed.), *Advances in Computer Chess VI* (pp. 133-147). London: Ellis Horwood.

Holding, D. (1985). *The Psychology of Chess Skill*. Hillsdale, NJ: Lawrence Erlbaum.

Levinson, R. & Snyder, R. (1991). Adaptive Pattern-Oriented Chess. In *Proceedings of the Eighth International Machine Learning Workshop* (pp. 85-89). Morgan Kaufmann.

Yee, R. C., Saxena, S., Utgoff, P. E. & Barto, A. G. (1990). Explaining Temporal Differences to Create Useful Concepts for Evaluating States. In *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 882-888). Boston, MA: AAAI Press.