This paper appeared in 1992 in International Journal of Intelligent Systems, 7 : 547-586.

Prior Knowledge Strengthens Learning to Control Search in Weak Theory Domains

Susan L. Epstein
Department of Computer Science
Hunter College of the City University of New York

Abstract. Rather than search exhaustively in large problem spaces, people use heuristics that entail minimal search and yet support intelligent decisions. Human performance in such spaces is robust and improves with experience. Game playing is a good example. The traditional AI approach to game playing, however, has been to build a program that plays only a single game, and plays it very well. Because exhaustive search of the game graph and minimax of the resultant values guarantees perfect play, programmed champions rely on deep, fast, efficient search. Despite their more limited, fallible memories and arguably slower processors, human masters often defeat a program whose search is incomplete. People search less, remember less, and somehow make the right choices. This paper describes HOYLE, a program that learns to play specific games under the direction of a *weak theory*, clearly delineated prior knowledge about games in general. HOYLE differs from most game-playing programs in two ways: it is able to play any of a broad class of games according to the rules, and it improves its performance through a variety of learning paradigms. HOYLE shows that, for a class of related problems, it is possible to replace deep search with a weak theory based upon heuristically selective cache memories and a consensus about action among reliable rationales.

The thesis of the work described here is that human expert decisions are based on more than search, that they rely on a synergy between intelligent rationales for decision making and limited recollection of significant experience. The human expert has what we describe here as a *weak theory*, prior general knowledge about a broad domain that includes ways to represent and apply information about it, heuristics for preferring and avoiding alternatives in it, and reasons to remember certain kinds of information. The expert's weak theory, we believe, supports the acquisition and application of detailed information to control search for specific problems.

Although the acquisition of the weak theory itself is beyond the scope of this paper, some characterization of it is appropriate here. Human experts solve problems robustly in their domain, hence the emphasis on a general weak theory, one not restricted to a single problem. Human expertise also degrades gracefully in the face of problems that are too difficult; a program that learns under a weak theory should do the same.

To explore our thesis we have constructed HOYLE, a program to model learning under a weak theory in the domain of two-person, perfect information games. The only general technique guaranteed to play every game perfectly is exhaustive search with minimax. For very large game graphs, that is not a computationally viable alternative, yet the classic trade-off in AI between generality and power makes any other approach suspect. The best game-playing programs, after all, have software, and sometimes hardware, that is explicitly constructed for the game they play. These programs share common techniques, but the techniques are specialized for each game, and the burden of that specialization lies with the programmer.

HOYLE accepts the need for such specialization, but performs it itself, i.e., HOYLE *learns* how to apply its general, game-independent weak theory in a game-specific way. From experience at a specific game, the prespecified weak theory is gradually instantiated. Initially the program makes rule-abiding moves, and may make tactical errors. Ultimately, however, HOYLE learns to play extremely well, often perfectly. That learning requires relatively few playing experiences, and the storage of relatively little data, i.e., HOYLE quickly and efficiently identifies key information about the game graph that is adequate for expert move selection.

The results described here demonstrate that learning under a weak theory is an alternative to exhaustive search for the seven games described in Appendix A. We do not claim that learning under a weak theory will be appropriate to every domain, nor even that it will someday produce a champion Go program. Our results do suggest that, if one can formulate an appropriate weak theory, learning under it is an interesting alternative development method for expert systems that address a related class of problems.

Games are a toy domain, but they provide an inexpensive, readily replicable, highly varied, noise-free environment in which to explore questions of strategy, control, and learning. The particular games used here were selected because they are commonly played and have a broad anthropological frame of reference. The commonalities in such a varied set of games helped identify many of the basic principles required in a weak theory for the domain. HOYLE's complete ignorance of symmetry was deliberate; it enables us to monitor performance on large, but still well-understood, game graphs.

This paper is not only about learning to play games; it is also about the representation and development of expertise. HOYLE is an expert system without rules, a loosely connected collection of reasonable rationales that adjusts dynamically to its environment, degrades gracefully, and improves with time. HOYLE's rationales are in the spirit of Minsky's Society of Mind[25] and Brooks' mobots;[6] their narrow, low-level expertise combines to simulate high-level competence, like Minsky's agents and Brooks' circuits. HOYLE may be seen as an exploration of the degree to which intelligent behavior in an ambitious domain can be simulated by a collection of primitive ideas. Kirsh[19] argues that intelligence demands concepts and representation, and indeed HOYLE has a few primitive concepts: game, tournament, contest, move, win, loss, draw, comment, and fork. The last of these certainly entails a representational transformation, but the extent to which HOYLE reacts, rather than believes, models, plans, or reasons logically, lies far closer to Brooks' approach. Where HOYLE digresses from Brooks' theory is in its reliance on memory; the program would never challenge human experts if it did not remember.

This paper begins with a discussion of the research context that suggested learning under a weak theory. A careful semantics for HOYLE's domain appears in the second section, while Section 3 provides background on other game-playing programs. Section 4 describes HOYLE's prior knowledge of two-person games, its weak theory for the domain. Section 5 describes how HOYLE modifies that knowledge as it learns to improve its performance. Section 6 provides some results from an initial testbed of games, and Section 7 addresses issues raised there. Our research intends to progress to more difficult games; the scaling up process, the significance of the results, and plans for future work are addressed in Section 8. Appendix A defines the games HOYLE's testbed; Appendix B gives a brief overview of HOYLE's rationales and conflict resolution among them.

1. Extending One-Person Task Results to a Strategic Domain

One clear indication of intelligence is the ability to choose productively among alternatives. Given a set of tools, or algorithms, or production rules, intelligent behavior selects and applies those that advance its goal in an efficient, ideally in an optimal, manner. Expertise in a domain includes both the generation of problem solving alternatives and the heuristics to select appropriately from among these alternatives. In any challenging domain, such expertise is not inborn; it is learned.

SAGE.2[20] and LEX[26] have demonstrated that intelligent selection among alternatives can be learned both during an experience and during a subsequent analysis of the trace of that experience. Each of these programs solved problems in a search space where brute force consideration of the legitimate alternatives at each node would have created a combinatoric explosion. Using heuristics (Langley's *weak methods*) that exploited the nature of their domain, these programs generalized principles and modified their behavior, based both on their original solution attempt and their subsequent analysis of it. The programs' improved performance on identical and similar problems demonstrated that they had indeed learned. SAGE.2 was an adaptive production system that learned heuristics to reduce search alternatives in one-person puzzles. It constructed and evaluated rules to generate alternatives during problem solving. With experience, SAGE.2's learned rules actually generated fewer alternatives at each choice point, and thereby curtailed search. Rule construction and evaluation were based upon traces of problem solving experience; according to their performance in these traces, rules were modified and assigned credit and/or blame. “Better” solutions in SAGE.2 were those with fewer steps. LEX learned heuristics for the application of integral calculus formulae. Based on its experience solving self-generated problems, the program generalized the circumstances under which each integration rule was most likely to be useful. LEX critiqued its problem solving traces and often explored alternatives not considered during the original solution of the problem. “Better” solutions in LEX were more economical in their use of computing resources.

Both of these programs, however, made important, tacit assumptions about their domains. First, the research assumed that there was a single agent, the problem solver. Any success or failure was attributable to that agent’s behavior, any improvement in performance was due to some modification of that behavior, and the result of any proposed behavior modification was always known in advance. Second, a decision made at one node always placed the problem solver at an absolutely predictable second node. Third, the research assumed that the goal was, in fact, reachable. Although both programs could learn while doing, the bulk of their work was performed on traces of solutions that reached a goal state. Fourth, each domain had a well-defined and readily computable metric to evaluate the “goodness” of a solution. This metric was used to compare the quality of solutions and the worth of alternative generators. Finally, the research assumed that the knowledge to be learned could be represented as constraints on the generation of alternatives during search. Good alternatives were identified and rules learned to encourage their selection; bad alternatives were identified and rules learned to discourage them.

A program to learn two-person, perfect information games typically confronts a far more complex search space than does one for a one-person problem. The search space for a game is usually substantially larger than that of the problems given to SAGE.2. For example, the 5-disk version of the Tower of Hanoi, SAGE.2’s most difficult problem, has only 243 reachable nodes. Tic-tac-toe has more than 5000 reachable nodes, checkers about 10^{20}, and chess about 10^{40}. The search space for a game also tends to branch more widely than that of either program. For example, LEX had an average branching factor of 8, but many games average 15 or 20 alternative legal moves from each node, and Go averages 200. While SAGE.2 and LEX encountered many of the nodes in their problem's search space during an initial problem solving attempt, a single contest of a game visits only a small fraction of the nodes in the search space and the game player may consider only some of its alternatives.

A program to learn two-person, perfect information games cannot make the same tacit assumptions about its domain that SAGE.2 and LEX did. Unlike the one-person tasks, a game has two agents, and the program controls only one of them. In a game, the behavior of only one agent can be modified with certainty, and the response of the other is not known in advance. Success or failure at play may be attributable either to one's own prowess or to the other agent's error. Even if there were some way to be certain that one agent had made no errors, it is not always clear, even to experts, which one or more of the other agent's decisions is responsible for the outcome of the game.

Unlike a choice in a puzzle, a single game move does not arrive at a predictable next choice point, because the other agent makes an intervening decision. Although that agent's move may be predictable, it is rarely certain. Indeed, it is unlikely that an attempt by the program simply to reproduce a previous experience at a game will result in an identical path through the search space, since the other agent may learn and modify her behavior, even while the program does.

A game-playing "solution" is a path through the game graph to a state where the program wins. There may, however, be no traces to such a state from which to learn. The program may not win initially or there may even be no way to win at a particular game, regardless of one's prowess. Thus the goal states may be ill-defined, particularly when the game is first encountered. In games with a score, some goals may be clearly better than others. Unlike the one-person tasks, however, an optimal solution in a two-person game must minimize the risk of loss while it maximizes the chance of ultimate success. A correct metric to compare two solutions in a substantial game graph would require an intractable minimax search. Thus it is often difficult to be certain whether one solution is truly "better" than another. Once again, even if the outcome of one experience in playing a game is deemed better, there is no guarantee that an attempt by one agent to replicate that behavior will result in a similar subsequent experience.

How then might a program learn to play games well? It must generate behavior, assign credit, and modify behavior.[20] Behavior generation is quite easy: the program simply plays the game, against another participant or against itself. For credit assignment and behavior modification, the key issues are really what such a program knows about a game and what it can learn. LEX and SAGE.2 know and learn search control rules; their weak methods provide a simple theory of their domain. (For example, "loops are bad," or "cheaper paths are better paths.") One can argue that LEX and SAGE.2 possess prior knowledge about one-person problems before they solve any, i.e., they have a primitive problem-solving expertise. This suggests that a game-playing program might also begin with a weak theory of its domain, prior knowledge about game playing that it exploits to play games correctly and to learn to play them better.

2. A Semantics for Game-Playing Expertise

The semantics in this section highlights the commonalities among games, commonalities upon which HOYLE's weak theory is based. Informally, a game consists of some *material* (a board and playing pieces), a roster of one or more participants, and a list of rules. The rules describe how the participants take turns affecting the presence and location of the pieces on the board until some rule terminates the process. When the process terminates, the rules declare which participant has won or that the process was a draw. If all information about the game is disclosed and equally available to all the participants (e.g., no closed hands, no uncertain outcomes as with dice), it is said to be a *perfect information* game. For the games considered here, there are exactly two participants: *Player* (the one who moves first) and *Opponent.* Player and Opponent are the *roles* in a two-person game.

Any game may be represented as a finite, directed graph (the *game graph*) in which a node (*state*) both identifies the participant whose turn it is (the *mover*) and describes a possible arrangement of the material. The participant who is not the mover in a state is referred to here as the *non-mover.*

An edge in the graph that goes from one (*originating*) state to another (*resultant*) state represents a *move,* the changes that occur when the mover in the originating state behaves (*takes a turn*) in any manner permitted by the rules. Together, the material, participants, and game graph are called the *elements* of a game.

There are three kinds of nodes in the game graph: starts, finishes, and intermediate states. A *start* is an initial state of the material before any participant has taken a turn, with Player as mover. A *finish* is a node with no out edges, and is labeled either *win* (Player wins and Opponent loses), *loss* (Player loses and Opponent wins), or *draw* (no winner or loser). Each game has one or more starts and one or more finishes. A node that is neither a start nor a finish is called an *intermediate state.*

In the game graph, a *path* in a game is a sequence $n_1e_1n_2e_2....n_ke_kn_{k+1}$ of nodes $n_1,n_2,...,n_k,n_{k+1}$ and edges $e_1,e_2,\ldots,e_k$, such that e_i is an edge in the graph from n_i to n_{i+1}, for i=1,2, …, k. If there is a path $n_1e_1n_2e_2....n_ke_kn_{k+1}$ in a game graph, node n_{k+1} is said to be *at depth k* or *k-ply* from node n_1. The average number of out edges from each non-finish node is the *branching factor* of the game. Relatively brief paths beginning with a start are called *openings*; relatively brief paths ending with a finish are called *endgames.* A path from the end of an opening to the beginning of an endgame is called a *middlegame.* The *stage* of a node is the path type (opening, middlegame, or endgame) it is most likely to appear on. If there is any path in the graph from a node back to itself, the game is said to contain a *cycle*. A move from a given node is said to be *invertible* if and only if it is the first edge in some path to a (not necessarily distinct) node where the mover's position is identical to the current one; otherwise it is *uninvertible*. A path in a game from a start to a finish represents one complete experience of the game, and is called a *contest.* During a contest, Player tries to arrive at a win and Opponent tries to arrive at a loss. Since from any state there are usually many legal moves, the challenge in play is for the mover to select the best move, i.e., one that will maximize the mover's opportunity to arrive at a desired state, while minimizing the non-mover's opportunity to do so.

AI research has traditionally approached game playing as a behavior to be modeled and evaluated in competition against people. A *theory* for a game is one or more statements about the properties of and relations among its elements. For example, a possible theory statement for tic-tac-toe is "the center square is the key position." A *heuristic* for a game is an explicit directive to the mover for selecting a move. For example, "if the center square is vacant, move there." A heuristic is an operationalized version of a theory statement; there may be more than one way to construct such an operationalization. Finally, a *strategy* is a set of heuristics for a specific game with a control method for choosing among them. A strategy is thus a procedure that, given any state, returns a move for the mover. To *learn to play a game well* is to produce and refine a strategy for it, constructing, testing, and refining theories.

A participant with a good strategy for a game is an *expert* at it. An expert should be able to take the role of either Player or Opponent and consistently maximize that role within the game graph. Consistency is important; the expert must perform well over a long sequence of contests (a *tournament*). This does not mean that the expert should always win, or even never lose; many popular games (e.g., tic-tac-toe) result in a draw when played by experts. Rather, an expert should consistently win when possible, failing that draw, and lose as rarely as the nature of the game permits. In addition, in games where there is a final score for each participant, an expert should win by a large margin, and lose by a small one. This definition of expertise deliberately ignores the skill level of any other participant, and instead defines the strength of a strategy with respect to the game itself. *Absolute expertise* is perfectly backed up knowledge from the entire game graph.

3. Related Work

The traditional AI approach to game playing has thus far been to build a program that plays only a single game, and plays it very well. Most of these programs play checkers,[34,35,36] Othello,[15,22,23,33] backgammon,[4,40] chess,[5,13,16] or Go.[18,31] The Chess 4.5 paradigm,[39] on which many of them are based, advocates extensive forward search from the node in the game graph representing the current state to a set of intermediate nodes, called *tip nodes*. These tip nodes are then heuristically evaluated and their values are backed up to estimate the best move from the current state.[3,24,27,28,32] Such search may be supported by very large historical knowledge bases about openings, endgames, and other play experience.[13,37,38] Despite the machines' superior speed, accuracy, and retention during search, the outstanding human participants at most of these games are still able to defeat the best of these one-game programs. For the most part, game-playing programs have little ability to improve without programmer intervention. For example, Hitech's chess rating continues to climb because its software is continually improved. As they pore over its most recent experiences, it is the researchers who are learning from Hitech's experience, not the program.

A notable exception to the one-game approach is the Smart Game Board, a "game workbench" environment for the study of two-person, perfect information games.[18] It accepts a game definition, records and annotates contests, executes specified search algorithms, and remembers contests and openings as directed. A game-playing module in the Smart Game Board is defined by the ability to recognize, execute, and retract legal moves, display the game states as the contest proceeds, and record contest traces. Move selection relies on the Chess 4.5 paradigm. The operator tells the Smart Game Board what to remember and how to proceed; the programmer defines game-dependent search control functions that compare states and allocate resources. In particular, the evaluation function for tip nodes relies on game-dependent features, and can be extremely complex. Competitive programs for Go[7] and Othello[17] have been constructed within this framework. By pitting two modules for the same game with different control structures against each other, it is possible to test different strategies for playing a specific game, but all the initiative lies with the programmer and the operator. The Smart Game Board is a flexible and well-designed software testing environment, one that relies on human guidance and does not learn.

Few of the major game-playing artifacts have *learned* to play very well. From treatises on how to succeed at checkers, Samuel's checker program preselected and hand-coded a list of features, measurements defined on a checkers state to evaluate its merit. Instead of a fixed evaluation function, his program repeatedly calculated weights for a linear sum of these features, and used that formula to distinguish among tip nodes during heuristic search. Over a series of contests, the program learned to play extremely well. The program could determine gradually, by letting a weight move toward zero, that some feature was irrelevant. It could not, however, identify new features that might be significant, generalize its knowledge, or play any game other than checkers.

Several chess programs have also incorporated some learning with respect to their evaluation functions. In a tournament, Bebe[37] can avoid the repetition of previous mistakes by the retention of tip node values on each state in the earlier contest paths. This longer term storage of transposition table data effectively extends the program's search depth. Deep Thought[16] has a separate tuning, as opposed to playing, mode where it refines input weights for the chess features in its evaluation function. The tuning mode attempts to simulate grandmaster-level move selection on 65,000 positions carefully selected by its creators.

Recent research has attempted to add learning components to some game-playing machines with limited success. Learning a preference between two states[41] has not yet improved performance, but connectionist techniques[40] to construct an evaluation function for backgammon have simulated expert play. Programs like ParaPhoenix[13] and GINA[38] maintain extensive historical knowledge-bases and supplement them with their own experience, learning by rote.

4. HOYLE's Weak Theory

HOYLE begins with explicit, but general, prior knowledge about the domain of two-person perfect information games. Like an experienced game player, HOYLE has predefined, fixed expectations about what a game is and how one is played. HOYLE's game frame, its game-playing algorithm, and the uninstantiated versions of its Advisors and game library form a weak theory for its domain.

4.1 The Game Frame

The *game frame* represents all HOYLE's known declarative knowledge about a specific game. Each game is defined for HOYLE by an instantiation of the game frame, i.e., HOYLE learns about a game initially by being told, and the instantiation for a particular game never changes. The slots in the game frame are prespecified; they are the same for every game. A game is defined to HOYLE by user-provided values for these slots. A game frame slot value is a constant or a LISP function.

As an example, the instantiated game frame for tic-tac-toe appears in Table 1. Values provide the name of the game, the tokens for the participants, and the initial state of the board. Functions offer optional directions, query non-HOYLE participants for moves and validate them as legal, display a state on the screen, effect a move upon a state (i.e., move to the next state), generate all legal moves from any state, define “end of contest,” define “win,” and define “loss.”

Name: *tic-tac-toe*
Token for Player: *X*
Token for Opponent: *O*
Initial-board: *(NIL NIL NIL NIL NIL NIL NIL NIL NIL)*

Directions for user: *directions-tic-tac-toe*
Move input reader: *reader-tic-tac-toe*
Move filter: *legalp-tic-tac-toe*

Display function for current state: *display-tic-tac-toe*
Move effector: *effector-tic-tac-toe*
Legal move generator: *generator-tic-tac-toe*

Predicate to detect end of contest: *endp-tic-tac-toe*
Predicate to calculate winner: *winp-tic-tac-toe*
Predicate to calculate loser: *lossp-tic-tac-toe*

Table 1. The instantiated game frame for tic-tac-toe. Four variables and eight functions completely define the game.

Unless the display graphics are elaborate, the functions referenced in the game frame typically require only about 70 lines of code. A game definition manages to be so brief for a variety of reasons. The graphics used to represent the game state are not complex or mouse sensitive, and Symbolics Extended Common LISP supplies a concise loop macro and many primitive graphic functions. The code is object-oriented; it has no game-specific features or evaluation function. The move generator has no filter on it, i.e., it is exhaustive. Directions are extremely simple, much like those found printed on the inside of the top of the box of a commercial game. HOYLE’s game-dependent prior knowledge is kept to a minimum.

4.2 The Game-Playing Algorithm

The *game-playing algorithm* is a uniform procedure that takes any game frame as input and is able to play, and moderate, a contest in that game. A pseudocode version of the game-playing algorithm appears in Table 2. The game-playing algorithm asks who goes first, sets up the initial state of the board, provides a running commentary on whose turn it is, accepts and verifies the legality of all moves, and announces the end of the contest and its results. The game-playing algorithm is the same for every contest in every game; it represents all HOYLE's procedural knowledge about how games in its domain are played. With valid data in the pertinent game frame, HOYLE plays any game according to the rules.

```
Select a game
Initialize the game frame
For some number of contests                          *tournament length or leave unspecified*
  Determine the identities of Player and Opponent
  Initialize the board
  Do while the contest is not over
    If it is the keyboard's turn                      *non-HOYLE mover*
      then  do until a legal move is input
              request a move
      else  generate all legal moves                  *HOYLE to move*
            case:
              there are no legal moves: resign
              there is exactly one legal move: make it
              there is more than one legal move: consult the Advisors to select one
    Record the most recent move in the contest's history
  Announce the result of the contest                  *winner or draw?*
  Conduct a post mortem on the contest
```

Table 2. A pseudocode version of HOYLE's game-playing algorithm. This algorithm is used to play any game.

4.3 The Advisors

Another segment of HOYLE's prior knowledge is the uninstantiated version of its Advisors. HOYLE has a panel of Advisors that serve as a weak theory to direct search. Each Advisor epitomizes a different perspective on game playing, and takes a fairly narrow, but rational, view of the move selection problem, one found generally applicable by experienced game players. At the moment there are 15 Advisors in use, descriptively named Victory, Panic, Sadder, Wiser, Don't Lose, Enough Rope, Pitchfork, Open, Candide, Worried, Shortsight, Greedy, Anthropomorph, Not Again, and Cyber. A brief description of each of them appears in Appendix B. Although, as we shall see in Section 5, the Advisors may reference selected memories of deeper search conducted between contests, no Advisor ever looks more than two-ply ahead in the game graph.

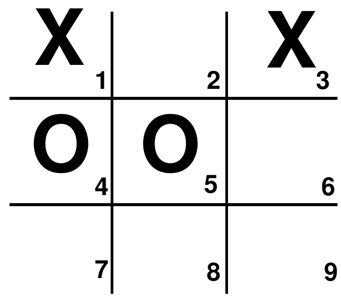


Figure 1 . A state in a tic-tac-toe trial. Player (X's) is mover. The Advisors are called upon for move recommendations on this state.

When it is HOYLE's turn to move in a contest, a procedure from the relevant game frame computes all legal moves from the current state and offers them to its Advisors. The Advisors comment on these alternatives, recommending them or advising against them. These comments are posted on a blackboard, which can be made visible during play. For any node where HOYLE is mover, an Advisor may make any number of comments, each with a *strength* from 0 (adamant opposition) to 10 (insistent support). Consider, for example, the node in Figure 1 from a tic-tac-toe contest where HOYLE as Player (X's) is the mover. The legal moves from this state are the squares 2, 6, 7, 8, and 9. Each Advisor now evaluates these alternatives from its own narrow perspective. The Advisor Panic looks to see whether the non-mover has a sure win on her next move. In Figure 1, Panic would insist that a move in square 6 is the only way to prevent a win by Opponent. This comment is based upon the threat of the O's in squares 4 and 5, and would have maximum strength, since a win by a reasonably observant Opponent across the second row on the next turn is virtually certain. Another Advisor, Worried, looks for longer-range wins for the non-mover. In Figure 1, Worried would suggest moves in both square 2 and square 8, since Opponent could win in the second column after two turns. Worried also suggests moves in squares 7, 8, and 9, to prevent Opponent's eventual win in the third row. The comments about the third row would have a lesser strength than those about the second column because they project a longer contest.

Victory is another Advisor; it looks for a certain win for itself on the current move. In Figure 1, Victory would insist that a move in square 2 is the best choice, because Player will win immediately.

4.4 The Game Library

As in the example of Figure 1, the Advisors rarely agree. The *game library* describes HOYLE's theories about how to win at a particular game, including how to mediate among disagreeing Advisors.[10] The game library focuses attention on recognized strategic questions in game playing, such as "Can either participant win at this game with perfect play?" Initially the game library is uninstantiated, and HOYLE determines its next move by applying the default control information found there to the current contest and the Advisors' recommendations. The program recognizes that all Advisors are not created equal, i.e., that in every game some Advisors are always more important than others. When it is HOYLE's turn to move, the program consults its Advisors in tiers, and continues on to the next tier only when unable to make a clearly-substantiated decision from the information at hand. This hierarchical consultation system, sketched in Appendix B, is an important part of HOYLE's weak theory; it encapsulates general knowledge about game playing that does not have to be relearned for each game. The tiers capture such commonsense rules as "If you can move to a win immediately, do it" and "If you cannot win now and are about to lose at the next turn, block the move that threatens you." Thus HOYLE's prior knowledge of two-person, perfect information games encompasses a variety of general information: procedural knowledge of game playing, declarative knowledge about specific games, narrow perspectives whose expertise severely restrict search, and some commonsense control strategy for move selection.

4.5 Additional Implementation Details

The version of HOYLE described here was implemented in Extended Common LISP on a Symbolics. Of its approximately 3750 lines of code, 60% was devoted to the environment for playing, 18% to learning, and 22% to the Advisors. Function definition for individual games required roughly an additional 70 lines of code for each of the five two-dimensional games, and 150 for the three-dimensional games with their more elaborate graphics.

5. Learning to Play Better

In the course of its experience in contests of a particular game, HOYLE seeks and caches information to aid search control. Although it is able to learn while doing, HOYLE, like SAGE.2 and LEX, learns primarily from traces. HOYLE learns by rote, by generalization, and by credit assignment.

5.1 Selective Rote Learning

HOYLE's weak theory for games includes certain instantiable structures. These structures, and the procedures to fill them, focus attention on significant aspects of particular games. Slot filling is a form of rote learning;[9] knowledge can often be retrieved more quickly than it can be recalculated, thereby speeding search. A frequently-played game with a large search graph, however, is likely to provide more experience than it is practical to store. Thus HOYLE caches information selectively.

After each contest, HOYLE conducts a *post mortem* to analyze the trace. During the post mortem, information may be cached in the caches constructed for each game. There is a cache for openings, a cache for expert moves, and a cache for previous contests. HOYLE has heuristics to determine what information is sufficiently interesting to save in each of these. After HOYLE completes a contest, it may extract and cache a play-by-play description of the contest itself, the opening (the first 10% of the moves), and/or expert moves.

An opening is cached if Player is presumed expert, the opening has never been encountered before, and Player does not lose with that opening. One of HOYLE's Advisors, Open, recommends openings from this cache. In addition, if a non-HOYLE expert wins any contest against the program, the expert's moves are stored in the expertise cache. One HOYLE Advisor, Anthropomorph, regularly suggests what a winning non-HOYLE participant has done in the current state. A contest is cached if it is the origin of at least one significant state (described in Section 5.2), if it contributes a new opening, or if the *expected role winner* (the role in which a non-HOYLE participant wins most often) loses.

5.2 Learning by Generalization

HOYLE has two generalization devices: significant states and forks. The first of these provides HOYLE with a perfect evaluation of certain intermediate nodes . The second is an analytic tool for strategy representation .

Significant states are identified during the post mortem after a contest. Significant states are generalizations, not of board configurations (e.g., “corners”), but of the play experience that must follow from them. Regardless of the history preceding the state, the state is always presumed to have the same intrinsic merit. (This may occasionally produce inelegant play but it does not compromise skill.) The post mortem includes a controlled examination of hypothetical contests to determine if the loser could have played better. The examination is controlled in the sense that limited resources are allocated to it.

A state is labeled *significant* if it is certain to lead to a win or loss. This certainty summarizes the outcome of exhaustive forward search from the state in question to terminal states and previously-identified significant states. (Some states, of course, are certain only to result in a draw; these are not considered significant states, although an argument could be made that such knowledge is relevant, even useful.) If search from the state is inconclusive within the time allocated to it, the state is not identified as significant at that point. After additional contests of the game, however, as neighboring states become significant, the same state could eventually be identified as significant itself. A state cannot be labeled a significant win state at one time, and a significant loss state at another; if such a label is generated for a state, it is an error-free synopsis of part of the game graph.

Significant states are those that both appear in the trace and would be identified by retrograde analysis on backup from a single node, the last state in the contest just completed where the loser had a non-forced move. Consider the description in Table 3 of a contest that Player wins on the 25th move, from state S-25 to state S-26. The second-to-last state, S-25, is cached as “win in one” with the final move, M-25. If it ever encounters S-25 again, the Advisor Wiser will recall that state and insist upon the winning move M-25 without search. The third-to-last state, S-24, is cached as “lose in two” if and only if, under a limited resource allotment, exhaustive forward search from it is completed and fails to find a better alternative for the mover at that point. If the third-to-last state is so cached, then the fourth-to-last state, S-23, is cached as "win in 3" with the winning M-23 move. On the other hand, if Opponent is the winner in state S-26, HOYLE caches S-25 as "lose in one" with move M-25, and may, based on a search similar to the one described, cache S-24 as "win in two" and S-23 as "lose in 3." Then, if it ever encounters S-25 again, the Advisor Sadder will recall the state and strongly recommend against the losing move M-25. Theoretically, as long as nodes had clear win/loss outcomes, this cycle of caching and searching might ultimately reach back in the game graph as far as the start node. Such search is costly, however, and HOYLE devotes only a finite amount of resources to it.

Mover	***State before Move***	***Move***
Player	S-1	M-1
Opponent	S-2	M-2
.	.	.
.	.	.
.	.	.
Player	S-21	M-21
Opponent	S-22	M-22
Player	S-23	M-23
Opponent	S-24	M-24
Player	S-25	M-25
Opponent	S-26	none

Table 3. Calculating significant states. The game history is used to identify and test significant win states and significant loss states.

As HOYLE plays more contests in a given game, its knowledge about the game graph is gradually incremented. In the contest of Table 3, assume that allocated resources were exhausted after S-23 was identified as a significant win state and cached. If some subsequent contest reproduces the last four states of this one, HOYLE will apply all of its post-mortem resources to the analogs of states S-22 and S-21, and perhaps even to their predecessors. Thus, significant states identified in early contests may be used in later post mortems to identify significant states "further back" in the game graph. In HOYLE, recurrent experience triggers expanded search.

When HOYLE caches a significant win state it is effectively learning an alternative goal. In a scoreless game, for example, if S-24 is cached as a significant state from which one will win in two moves, arrival at S-24 is just as desirable as arrival at S-26 itself. (It may be argued that, if the path to S-26 were shorter, that S-24 was a less elegant conclusion, but HOYLE is not concerned with optimal paths, only with winning.) Rather than learn by rewarding and/or transforming the operator that chooses move M-23 from S-23 to S-24, the way SAGE.2 or LEX would do, HOYLE's Advisor Wiser is instantiated to recognize that any move resulting in S-24 is an excellent one, i.e., that S-24 is a new goal state.

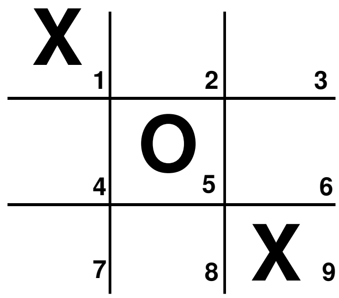


*Figure 2***.** A dangerous position. Player (X's) has an opportunity to fork in each corner. HOYLE is mover as Opponent (O's). The appropriate response is not immediately obvious.

When HOYLE caches a significant loss state, it is summarizing its experience about alternatives to be eliminated. For example, the tic-tac-toe state of Figure 2 offers an interesting challenge for HOYLE, the mover as Opponent (O's). Player has the prospect of two excellent moves; each corner produces a fork, in this case a threat to win in both the row and column it lies on. Opponent can block only one corner on this turn, with a move either to square 3 or square 7. It looks as if HOYLE will certainly lose. Surprisingly, there are four good defensive moves. A move to any of squares 2, 4, 6, or 8 in Figure 2 will force Player to defend against a win by Opponent. Player's defense will be offensive as well; her block will place two X's in a column or row whose third square is blank. In all four variations this cycle continues, with each participant threatening to win on the next move and defending against the other's threat, until the contest ends in a draw. HOYLE learns this defense quickly; if the program tries to defend by playing 3 or 7, it loses, analyzes the loss, and rules out that defense for the future. The states following Figure 2, with an O in 3 or in 7, become significant loss states; whenever the program is again faced with the state in Figure 2, the Advisor Sadder will forbid moves to 3 or 7. Rather than learn by blaming and/or modifying the operator that chooses a corner move from the state in Figure 2, HOYLE's Advisor Sadder is instantiated to recognize that a move to 3 or 7 is a poor one. Although Pitchfork, the Advisor on forks, continues to identify and recommend blocking the forks in Figure 2, Sadder, in a higher tier, has the ability to veto such ideas.

Figure 3 is a schematic representation of how HOYLE's learning effectively reformulates a game graph. (This data is implicit, not explicit, in the program.) The nodes shown represent those encountered during play. The circled nodes represent states that have been found significant; they replace entire subgraphs in the original game graph. By caching significant win states, HOYLE proliferates alternative goal states closer to the root; by caching significant loss states HOYLE directs its attention to other alternatives. Both generalizations (i.e., "good" and "bad") prune entire sections of the graph and speed search. This gradual transformation is a sort of suspended minimax that capitalizes on HOYLE's experience. It may be argued that the significant states are actually chunks whose value, or lack thereof, has been recognized and summarized. For example, once 3 and 7 are ruled out as "bad moves" from the state in Figure 2, HOYLE has effectively learned "do not move in the corners here."

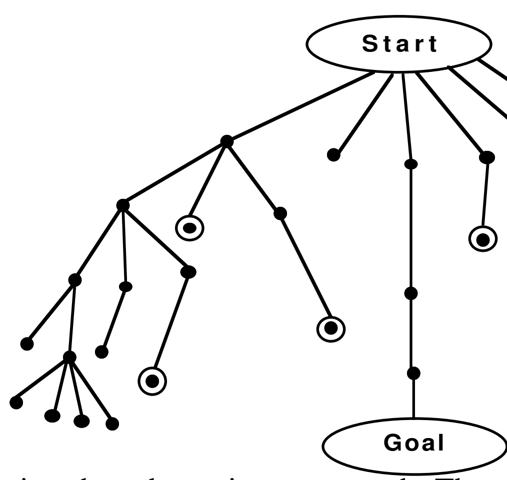


Figure 3. Transforming the schematic game graph. The graph emerges and contracts based on experience during play and analysis afterward.

HOYLE does not attempt to back up the entire graph to the root for several reasons. First, the graph is presumed to be extremely large and an exhaustive minimax search intractable. Second, in some games, portions of the graph may rarely become visible because they are intrinsically uninteresting. If human experts in either role never direct play that way, perhaps that portion of the graph can be safely ignored, i.e., there may be no need to learn about it. Finally, HOYLE is a limitedly rational program, one whose goal is not perfection but continual improvement. Thus the post mortem is allowed only limited resources. For HOYLE, experience is as important as analysis.

Significant states are carefully restricted subsets of permanent transposition tables from retrograde analysis. A chess transposition table[39] stores tip nodes and their heuristic evaluations; in contrast, HOYLE's significant states have only the certain values win or loss. Retrograde analysis finds the optimal play for all possible states in a specific endgame; it backs up a loss state, for example, to all possible immediate predecessor states, and labels them as wins. HOYLE's post mortem, on the other hand, only backs up from a loss state that occurs within the contest trace to the predecessor within the contest state. These differences deliberately reflect the limitedly rational approach of learning under a weak theory: reluctance to search intensively, restricted resources, and no evaluation function.

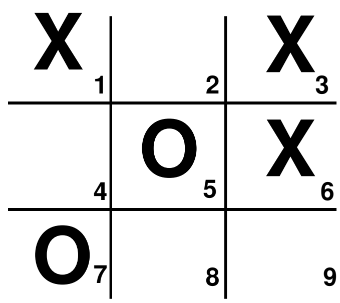


Figure 4. A fork in tic-tac-toe. X's most recent move, to position 3, threatens a win in both the first row and the third column.

HOYLE's second form of generalization is the fork. Let a *simple plan* be an unordered set of unretractable moves that reach a desired state. Informally, a *fork* is two or more overlapping simple plans belonging to the same participant, directed toward different desired states, and requiring one or more common moves. (When "desired state" is replaced with "piece capture," this definition is consistent with a fork in chess, where a piece threatens to capture two others.) Figure 4 contains an example of a fork in tic-tac-toe, where Player's most recent move, into position 3, threatens a win in both the first row and the third column. Opponent is defenseless before this onslaught.

In a game with only uninvertible moves, a strategic map for any state can be constructed for each participant. A *strategic map* for a state and participant is a labeled, undirected bipartite graph in which one set of nodes (the *top vertices*) is labeled with the moves available to that participant from that state, the second set (the *bottom vertices*) is labeled with the participant's simple plans to reach each winning state, and an edge joins any move available to a participant with each simple plan it forwards. In Figure 5, for example, positions 2, 3, 4, 6, 7, 8, and 9 are available to both participants; rows 1 and 3 and columns 1 and 3 are simple plans for Player; and rows 2 and 3, columns 2 and 3, and the minor (from upper right to lower left) diagonal are simple plans for Opponent. Figure 5 shows the strategic maps for Player and Opponent for the given game state. The numbers inside the circles for the bottom vertices are their respective degrees.

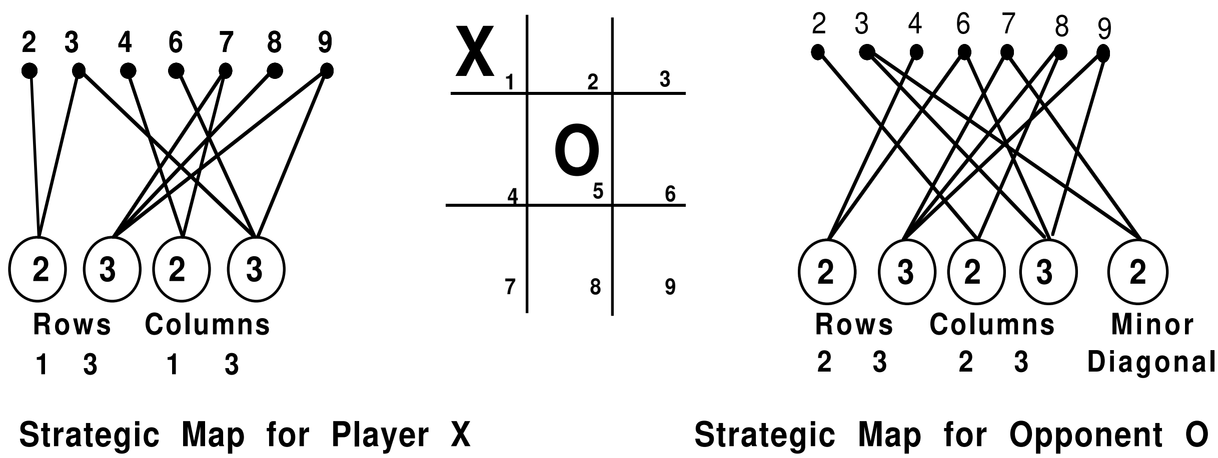


Figure 5. A tic-tac-toe game state and its strategic maps. Each state is represented as two strategic maps, one for each participant. Offensive strategy is planned in the mover's strategic map, defensive strategy in the non-mover's.

Certain connected subgraphs of a strategic map, induced by a subset of its bottom vertices, represent the overlapping of more than one of its participant's simple plans. For example, in the subgraph of Figure 6(a), either move represented by a degree-two vertex at the top furthers both the simple plans represented by the bottom vertices adjacent to it. Such a move, on the left for example, transforms Figure 6(a) into Figure 6(b), two connected graphs labelled F-1 on and F-21.

(Immediately before her most recent move into position 3 in Figure 4, Player's strategic map was isomorphic to F-21, where the top vertices represented 2, 3 and 9 and the bottom vertices represented Row 1 and Column 3.) Subsequently making the move represented by the second degree-two position transforms Figure 6(b) into Figure 6(c), three copies of F-1. Thus there are two moves represented in Figure 6(a) that, made in any order, forward among them three plans to win in one move. A fork is a generalization because it is contest-independent and game-independent. More formal definitions, notation,[11] and a description of how HOYLE learns new forks from its play experience[12] are available elsewhere.

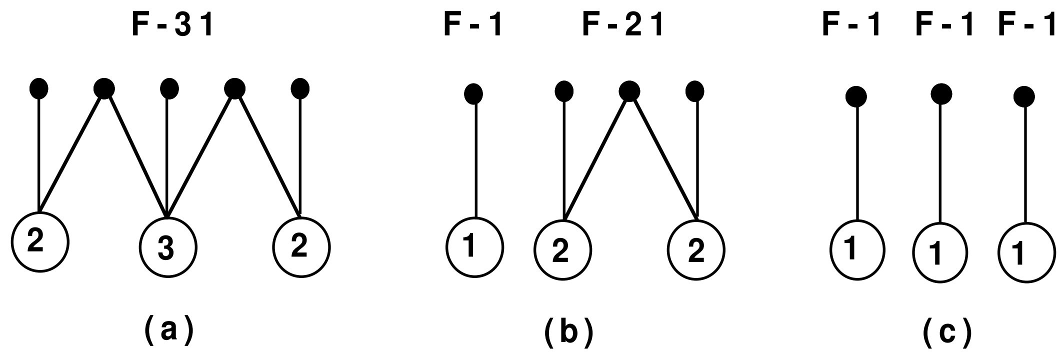


Figure 6. Execution of a fork. Each move in a top vertex forwards every plan it is adjacent to as a bottom vertex.

The power of a fork lies in the ability of one move to forward more than one plan. Thus a fork represents a strategic plan to advance multiple goals simultaneously. An offensive fork appears in the mover's strategic map; it advances the mover's opportunity to win and requires careful sequential execution. A defensive fork appears in the non-mover's strategic map; it advances the non-mover's opportunity to win and demands precise disruption.

How quickly a fork can achieve its objective is quantified as its *depth*. A fork of depth d represents a game-independent set of mutually overlapping simple plans for a participant to achieve a winning state after at most d moves. There are infinitely many forks of varying depths for two-player games. Their applicability is determined both by the topology of the board and the rules of the game. Even at depth three and four, most forks are difficult for people to recognize offensively or defensively during play.

Initially HOYLE learns about forks by generating the 7 least deep forks with a prespecified, recursive procedural definition. When it encounters a new game, HOYLE calculates which of these known, fairly elementary forks could ever appear, based on the topology of the board, and caches that information in the game frame. With play experience, the program may add additional learned forks to this list. Since fork identification requires the costly detection of a labeled bipartite subgraph isomorphism and since the Advisors have limited resources, HOYLE directs its offensive and defensive search only to forks it expects. During play, Pitchfork, HOYLE's Advisor on forks, guards against those forks. Thus for a given game HOYLE learns what forks to use and to guard against.

5.3 Credit Assignment

Once a problem solving session or a contest is over, an intelligent person usually attempts to identify the cause of her success or failure. This instinct is an attempt to distinguish, among the steps taken or the moves made, the responsible portions of the behavior, so that they may be credited with or blamed for the outcome. As discussed in Section 1, it is rarely clear in two-person games exactly which steps or moves are actually responsible.

Machine learning programs span a spectrum of credit assignment approaches from liberal to conservative. LEX takes the liberal approach: when it examines a search trace, LEX's Critic credits every individual step on the lowest cost solution path and blames every individual step that either leads away from that path, leads to a higher cost solution, or fails to find one. Such uniform reward and punishment learns only preconditions for individual operator application (rather than sequences of operators) and is highly dependent on the description language. It may also attend to the wrong operators, since it works only with the parts of the search space encountered during solution and post mortem. Genetic algorithms[14] lie at the other, more conservative, end of the credit assignment spectrum. Valuable operators must repeatedly prove themselves, in comparison with the entire population, as highly fit individuals and superior parents in order to survive.

The way credit and blame are allocated in these two examples appears closely related to the likelihood that the assignment will be correct. LEX is likely to encounter an optimal solution in its trace or under the guidance of its Critic; it can reasonably expect to learn from the perfect answer.

Genetic algorithms, however, are extremely unlikely ever to produce an optimal solution; their caution in credit and blame assignment is well-founded. HOYLE has some possibility, at least with simple games, of playing the game graph perfectly, although it may not recognize its behavior as such. Thus HOYLE's approach to credit assignment is, quite sensibly, in the middle of the spectrum.

Previous rationales for strategic learning do not transfer well to the domain of two-person games. OCCAM,[30] for example, is a program that successfully exploits spatial and temporal relationships to detect causality. This technique would not work very well for most games, however. In many contests at chess, for example, the credit-worthy move is neither immediately before the mate nor in the immediate geographic vicinity of the trapped king. Explanation-based learning requires a reasonably detailed, preexisting domain theory that may be unavailable here. Nor does HOYLE have operators, as LEX and SAGE.2 do, to credit or blame for its decisions. Instead, HOYLE has a weak domain theory about what may or may not be relevant during play, and it seeks to explain its experience in those terms.

Because the material in the game library controls the fundamental way HOYLE selects a move, changes to the game library are neither sweeping nor permanent. Recall, from Section 2, that a strategy is a procedural control method for choosing among a set of heuristics for a specific game. HOYLE begins with default information for a game in its original, uninstantiated game library, and periodically refines it based on playing experience. The program uses its play experience to develop inductively an instantiation for the game library of a specific game. Thus the instantiated game library represents a compendium of learned knowledge about the game. Eventually the game library should indicate whether or not it is an advantage to go first, what the best opening moves are, whether it is possible to win as Player and as Opponent, any territory that is most important to control, which Advisors are relevant to the game, which are usually most helpful, and how to balance advice from one Advisor against advice from another. As slots are assigned values, HOYLE's Advisors can reference them to produce more knowledgeable comments.

6. Results

HOYLE plays a broad spectrum of culturally diverse games correctly under the direction of its prior knowledge. The games in the current testbed are tic-tac-toe, lose tic-tac-toe, two versions of three-dimensional tic-tac-toe, tsoro yematatu, pong hau k'i, and achi.[2,42] They were chosen because they span a variety of cultures, and therefore probably capture some aspects of game playing that people find particularly intriguing. Their game graphs lack the complexity of chess or Go, but some of them offer the challenge of cycles and stage transitions, and at least one of the games has a game graph of billions of nodes.

HOYLE should be judged against two criteria: how well it learns to play and how quickly it achieves that skill. Appropriate measures of skill at the games in the testbed include ability to win, ability to draw (five of the testbed games should have no winner when played by two absolute experts), and the duration of the contest, measured in average number of moves. If two participants at a game lose regularly to an expert, the one able to prolong the contests is judged the better player. Similarly, if two participants regularly achieve an optimal outcome against a third, the one able to do so more quickly is judged the better player. Although, theoretically, one could learn while playing a novice, a skilled model provides a less noisy learning environment. HOYLE therefore plays against *experts*, knowledgeable humans or separate expert programs. Although they made occasional errors (typically failures to defend thoroughly), the human experts had the knowledge to play with absolute expertise at all the games except Qubic©. At Qubic they played imperfectly, but extremely well. HOYLE's skill in each tournament is measured both by the total number of wins and/or draws it achieves and by the average duration of those contests.

Learning in the tournament environment is measured by changes in performance. The game graph for each of the games in the testbed is well understood; it is therefore possible to predict how a participant with absolute expertise should fare in any contest. For each game, the *goal curve* plots the cumulative number of wins and draws for absolute expertise as a function of the number of contests played. In each tournament the *performance curve* plots the cumulative number of wins and draws for HOYLE as a function of the number of contests played. (Although statistics on the number of contests won are also available, calculations based on non-losses absorb human error and are more appropriate for games where perfect play results in a tie.) By definition, unless the human participant makes errors, the distance between any performance curve and the goal curve cannot decrease, i.e., HOYLE cannot play better than with absolute expertise. While a performance curve parallels the goal curve, the machine is playing with absolute expertise, i.e., as well as the expert. Thus HOYLE may be said to have learned to play with absolute expertise once its performance curve consistently parallels the goal curve. While the distance between a performance curve and the goal curve is increasing, HOYLE is not playing as well as the expert; while the rate of change in that distance is decreasing, HOYLE is learning. The segment of HOYLE's performance curve not parallel to the goal curve represents the contests during which it is learning, the length of that segment represents the time required to learn, and the slope of that segment represents the speed with which it learns. In addition, a program that loses repeatedly, but is able to prolong its contests as time passes is learning; a program that wins or draws repeatedly and is able to achieve that result more quickly as time passes is learning too.

Initial experiments with such tournaments indicated that HOYLE was indeed quickly learning to play very well. How wide, however, was the margin between absolute expertise and complete lack of knowledge? That is, was HOYLE accomplishing a difficult task? And were HOYLE's game libraries or its Advisors responsible for its performance? To answer these questions, for each of its games HOYLE played four tournaments of 20 contests each against an expert, with a resource limit of three minutes per Advisor. (Only two Advisors ever used a significant amount of their allotment, however, and that was only during early play in the most difficult games.) In the first tournament (#1) for each game, HOYLE always played at random, making legal moves without a post mortem or any Advisors. This provides a lower bound on performance; the gap between the curve for absolute expertise and #1 indicates the extent of HOYLE's task. In the second tournament (#2) for each game, HOYLE was permitted only the post mortem and reference to the game libraries it constructs there. This *experiential version* of HOYLE employed only the Advisors

Sadder, Wiser, Open, Anthropomorph, Not Again, and Cyber, as well as partial implementations of Don't Lose and Shortsight restricted to the cached data without the ability to reason forward from the current state. Tournament #2 measures learning directly attributable to experience. In the third tournament (#3) for each game, HOYLE played only with those Advisors omitted in the second tournament and without the post mortem. This measured learning attributable to a successful binding of prior knowledge of the domain to the specific game. Finally, in the fourth tournament (#4) for each game, HOYLE played with both the post mortem and all of its Advisors.

The outcome of these tournaments appears in Figures 7 through 13, one figure for each game. Each figure shows the goal curve and the four performance curves. Since some two-person, perfect information games are known to provide an advantage to a particular role, the participants in the tournaments alternated roles; HOYLE always began as Opponent. Cache memory was examined from time to time, and the blackboard with the Advisors' comments appeared on the screen during all contests; this transparency is reflected in the discussion that follows.

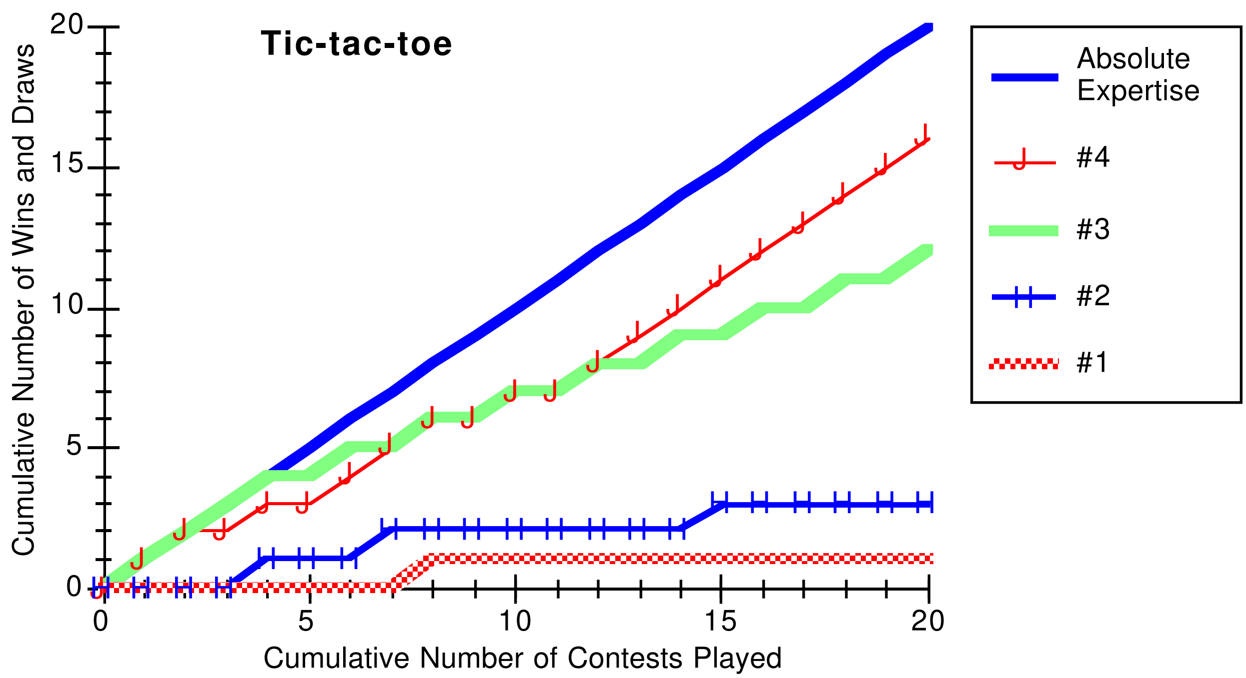


Figure 7. Cumulative non-losses in four tic-tac-toe tournaments of twenty contests each. HOYLE learns to play with absolute expertise after contest 11 in tournament #4.

In the first tic-tac-toe tournament (#1) in Figure 7, random play against an average branching factor of 4.5 provides roughly a one in four chance of making the best move. It is not surprising, therefore, that HOYLE loses all but one contest of tic-tac-toe in tournament #1. In the second tic-tac-toe tournament (#2), HOYLE relies upon its experience. Its Advisors immediately learn, and recommend, the successful openings of the human participant, but find play in the other stages of the game more difficult. In this tournament HOYLE very gradually learns to avoid losing moves in the tic-tac-toe endgame through the post mortem. Although, theoretically HOYLE could learn about all possible states by backing up such experience, Figure 7 shows that it is a slow process. In the third tic-tac-toe tournament (#3), HOYLE relies on its prior knowledge of the domain. After a few initial contests, however, the human tries the approach in Figure 2, and observes that HOYLE does not remember its mistakes from one contest to the next. Each time the human is Player that gambit succeeds. Confronted with the state in Figure 2, Pitchfork always tries to defend one corner, and the human takes the other, guaranteeing a win for the human on alternate contests. Thus the Advisors without memory play a reasonably intelligent tic-tac-toe contest, but perform unintelligently in a tournament situation. In the fourth tic-tac-toe tournament (#4), HOYLE plays as it was intended to, with both the post mortem and all of the Advisors. Figure 7 shows that in the eleven contests, HOYLE's combination of memory and skilled advice quickly learns to play tic-tac-toe with absolute expertise.

Lose tic-tac-toe is identical to tic-tac-toe, except that the objective is to avoid getting three markers in a row, column, or diagonal. Cohen[8] has solved this game mathematically, i.e., clearly delineated and proved a strategy for both roles. The human participant used this strategy consistently against HOYLE. There are several idiosyncrasies of the game graph for lose tic-tac-toe that clearly influence play. The game should, like ordinary tic-tac-toe, always end in a tie. For Player there is a single correct opening move: if Player makes that move, a draw is guaranteed in perfect play, and if Player does not, a loss is certain. The correct opening also places Player in an extremely vulnerable position: for each of Opponent's subsequent possible moves, there is only one correct response that guarantees a draw, and any other choice against a perfect Opponent guarantees a loss. Thus the role of Player demands considerable skill. For Opponent, all first and

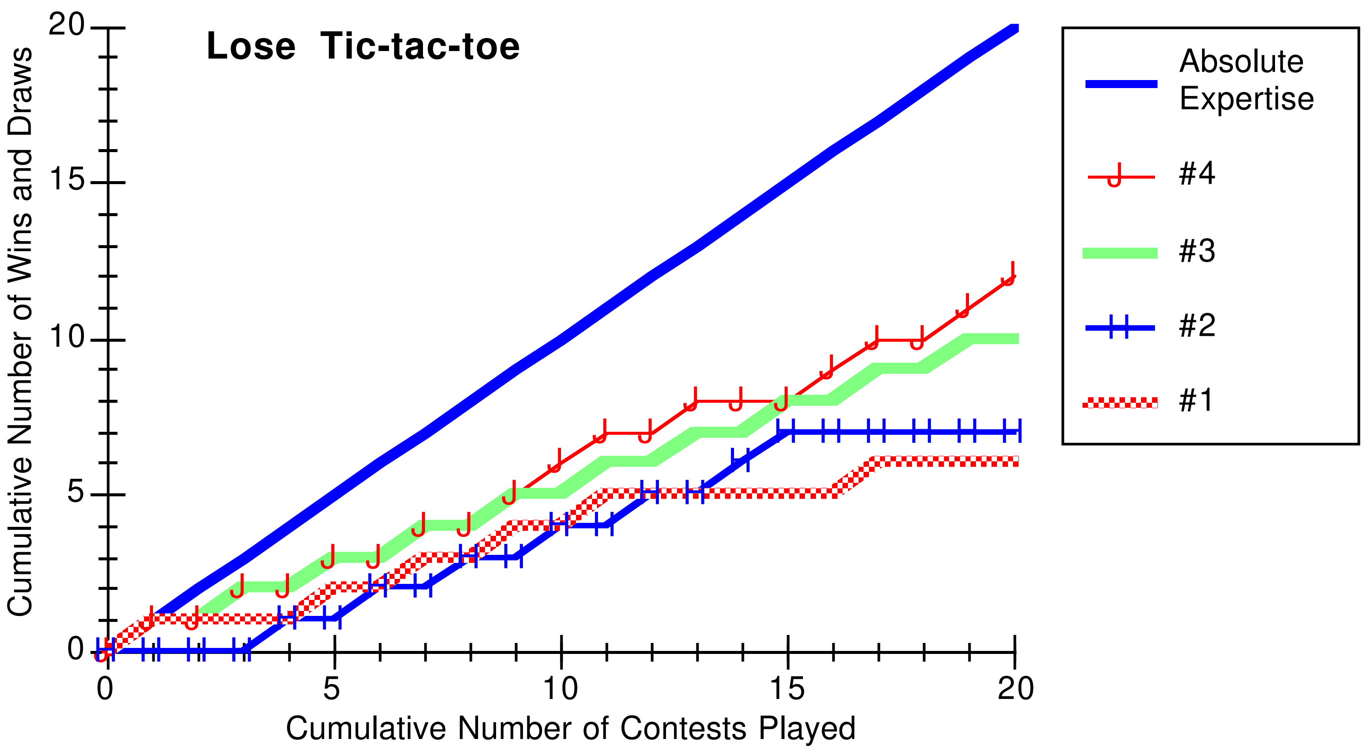


Figure 8. Cumulative non-losses in four lose tic-tac-toe tournaments of twenty contests each. HOYLE may need more experience to learn this game.

second moves are equally good; the selection of the third move from among four choices, however, requires consideration. As shown in Figure 8, in the first tournament, random play is a strong defense: contests average 7.5 out of a possible nine moves, and HOYLE manages six draws, all, not surprisingly, in the role of Opponent. In the second tournament, the experiential version, HOYLE still draws seven times, all as Opponent. The contests in #2 now average 8.4 moves out of an optimal nine, so HOYLE is putting up a somewhat stronger defense. As Player in #2, HOYLE learns the counterintuitive optimal opening (the center square) from the expert's behavior in the first contest and uses it consistently thereafter. HOYLE also remembers the correct human response to any Opponent's first move that it sees during play. In this game Player has only one correct second move out of seven possibilities, one correct third move out of five, and one correct fourth move out of three. Thus the program is likely to encounter an unremembered state, even after 20 contests. With no knowledge of symmetry or the Advisors' theories, HOYLE frequently chooses its later Player moves at random, and loses, in #2. As Opponent in #2, HOYLE must learn to avoid eight straight lines in many different states, only by recollection, without lookahead. Once again, HOYLE is forced to random choice in later moves, even in the final contests. Tournament #3 shows some playing improvement; HOYLE draws a total of eight times, always as Opponent. In #3 HOYLE's ability to look ahead saves it from blunders as Opponent, but without memory it does not learn the opening or the replies it needs as Player. Finally, in tournament #4, HOYLE is permitted its full faculties; it draws 12 of the 20 contests and all but one of the contests go the full nine moves. All but one of HOYLE's losses are as Player, but now it is able to hold its own four times in that role. HOYLE learns the Player role for lose tic-tac-toe primarily through Anthropomorph, i.e., by recollection of what an expert participant did in the identical situation. Because HOYLE has no representational transformations for symmetry, this learning is relatively slow. (See Section 7 for a discussion of a subsequent, longer version of tournament #4 for lose tic-tac-toe.) The difference between HOYLE's performance in this game and in ordinary tic-tac-toe is primarily due to Pitchfork's inapplicability; that Advisor is currently only relevant in games where one actively assembles a winning position, rather than avoids a losing one. Pitchfork is unable, for example, to prevent HOYLE from forking itself, the only way it ever lost in #4 as Opponent.

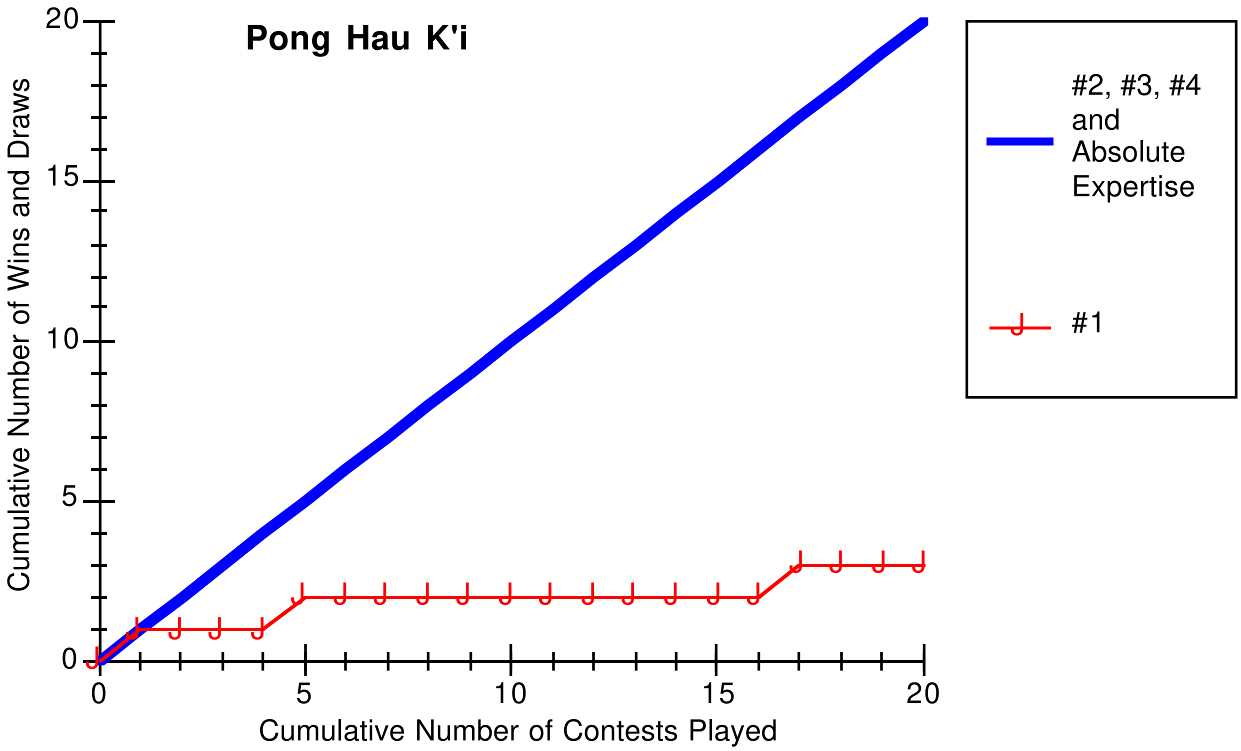


Figure 9. Cumulative non-losses in four pong hau k'i tournaments of twenty contests each. HOYLE plays with absolute expertise in tournaments #2, #3, and #4, where it has access to the post mortem and the Advisors with memory and/or one-ply lookahead.

Pong hau k'i is a Korean game with only 30 possible states, a small branching factor (about 1.5), and a heavily cyclic game graph. Like lose tic-tac-toe, pong hau k'i ends with a loss, when a participant is unable to move. Given two participants with perfect expertise, no contest should ever terminate; HOYLE was therefore instructed to declare a draw if a state repeated more than three times. Every contest in the last three tournaments is declared a tie in this manner. In random play, HOYLE fares better in pong hau k'i than in most other games, both because with a smaller branching factor there is a higher probability of selecting the correct move, and because the human opponent occasionally tires and makes an error. HOYLE's anthropomorphic play is remarkably successful at this game. With memory and generalization (tournaments #2 and #4), HOYLE plays with absolute expertise from the beginning. In the last two tournaments, the Advisor Shortsight avoids losses two ply away, enough to guarantee absolute expertise in this simple game. If HOYLE can be said to learn pong hau k'i, that learning comes immediately, from the application of memory and prior knowledge of the domain to the rules of the game.

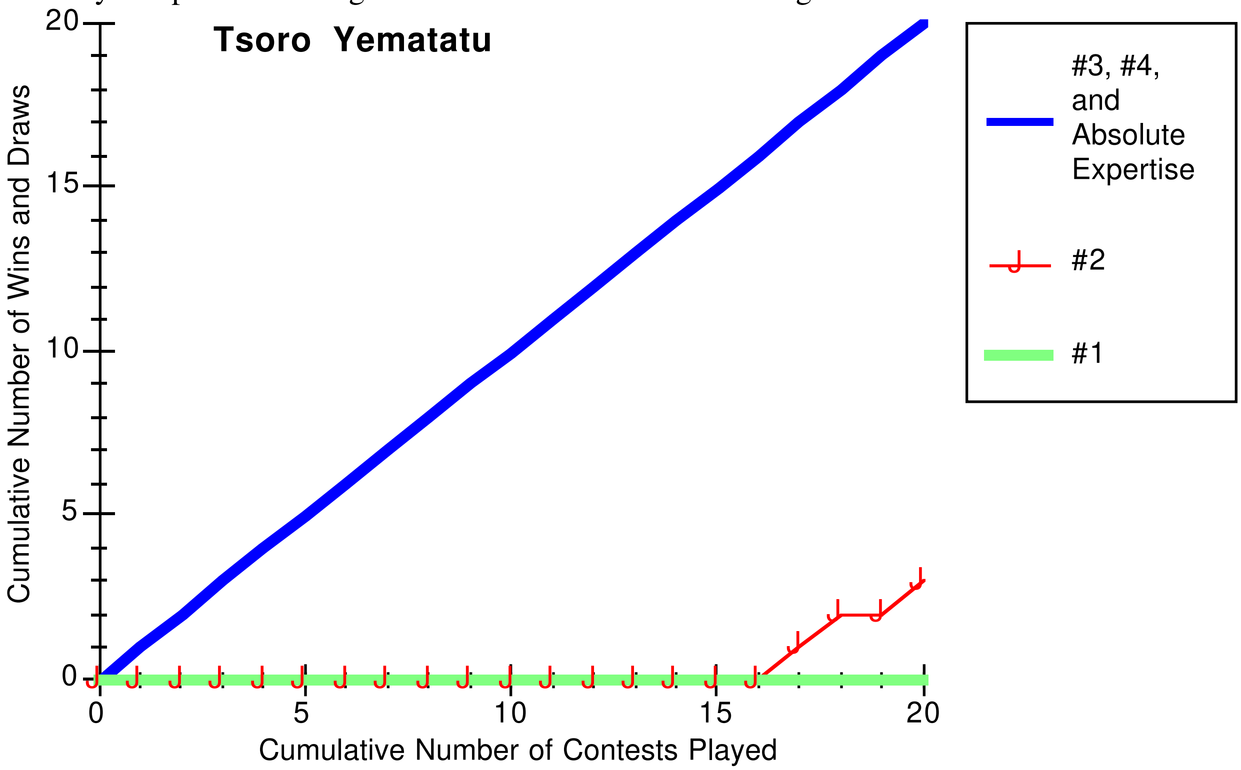


Figure 10. Cumulative non-losses in four tsoro yematatu tournaments of twenty contests each. HOYLE plays with absolute expertise when it has access to the Advisors, in tournaments #3 and #4. Without those Advisors, HOYLE's learning makes no impact until the seventeenth contest.

Tsoro yematatu is an African game with two stages: the first has a tree-like game graph of 715 states, the second cycles among 140 states from the first stage. Like pong hau k'i, expert play should terminate in repetitive cycles as a declared draw. In the first stage the branching factor averages 3.5; in the second, it is roughly 2. HOYLE never even manages to draw a randomly-played contest in tournament #1, as shown in Figure 10. For tsoro yematatu, Shortsight is extremely important; HOYLE immediately plays with absolute expertise when it has access to that Advisor, in tournaments #3 and #4. In the experiential version of #2, however, it takes 16 contests before HOYLE begins to play effectively enough to fend off an expert with regularity. Examination of the caches after tournament #2 indicates that HOYLE would eventually learn to play very well, but that it would be a slow process, particularly without symmetry. HOYLE can learn tsoro yematatu either gradually, through generalization, or after one contest, from the application of memory and its prior knowledge of the domain to the rules of the game.

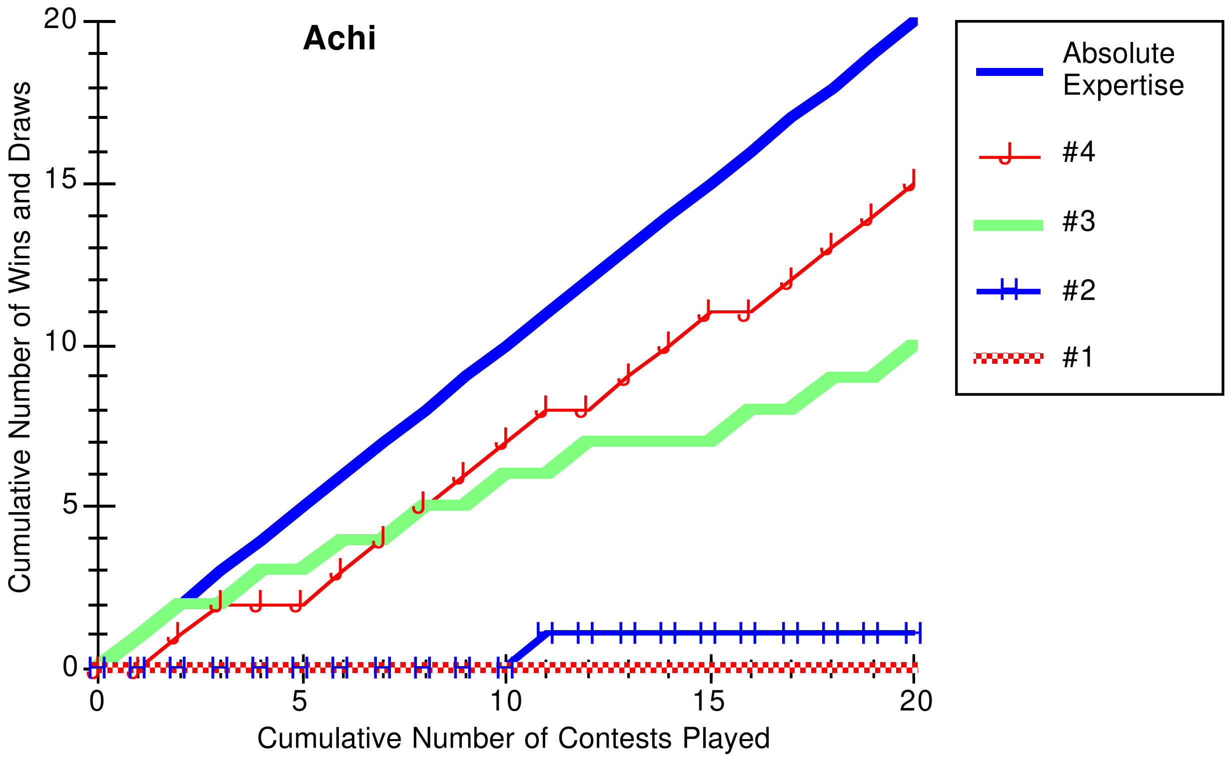


Figure 11. Cumulative non-losses in four achi tournaments of twenty contests each. As with tic-tac-toe, HOYLE's learning is strongest when generalization and all the Advisors are combined.

Achi, in Figure 11, is another African game with two stages: the tree-like game graph of the first stage is more than seven times larger than tsoro yematatu's, and the second stage cycles among four and a half times as many states as tsoro yematatu's does. In the first stage the branching factor averages 4.5; in the second, it is 3. As in pong hau k'i and tsoro yematatu, expert play should terminate in repetitive cycles as a declared draw. As expected, HOYLE never even manages to draw a randomly-played contest in tournament #1, and the contests end quickly, averaging 6.2 moves in length. Although the rules are different, achi's board is isomorphic to that for tic-tac-toe, and early in a contest HOYLE's behavior has a crucial weakness in a state isomorphic to the one in Figure 2. In the experiential version, HOYLE manages to prolong the contests in tournament #2 a bit (7.75 moves), but it continues to lose quite steadily. Inspection of the caches indicates that it will be many contests before mere memory and generalization in tournament #2 will enable HOYLE to play better. In tournament #3, Shortsight helps substantially when HOYLE is Player; the contests now average 24.3 moves, although the contest isomorphic to Figure 2 continues to defeat HOYLE. In tournament #4, however, HOYLE won contest 7 as Opponent and contest 8 as Player. There followed a series of draws, as the human experimented with a variety of techniques and then HOYLE began to win in either role! On subsequent examination, the strategy the program discovered is defeatable, but it was unanticipated and extremely effective. Without errors by the other participant, the behavior after contest 16 is representative of HOYLE's performance. In tournament #4 HOYLE certainly learns to play achi with absolute expertise.

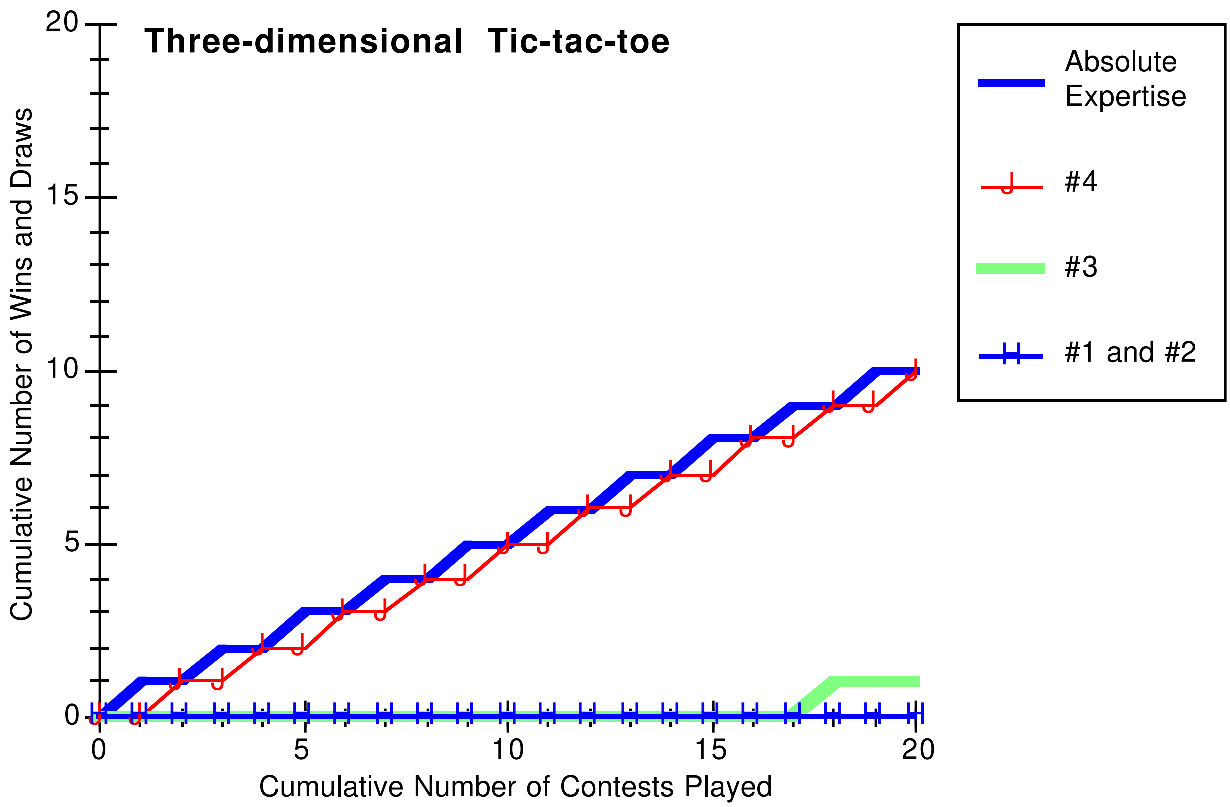


Figure 12. Cumulative non-losses in four three-dimensional tic-tac-toe tournaments of twenty contests each. HOYLE learns to play with absolute expertise when it combines memory and generalization with all the Advisors in tournament #4.

Three-dimensional tic-tac-toe is played on three parallel tic-tac-toe grids. Three in a straight line still wins, but this time the line may include points in more than one grid. There are 49 such winning lines, compared to eight in conventional tic-tac-toe. Unlike most of the other games in the testbed, three-dimensional tic-tac-toe gives a decided advantage to Player. (Note the difference in the curve for absolute expertise.) There are many isomorphs to a depth four fork in the initial state of the board, all of them requiring the same opening move. If Player can identify and properly execute any one of these, she is certain to win in four moves. Figure 12 shows the results of the tournaments for three-dimensional tic-tac-toe. Against an average branching factor of 13.5, random play is, as expected, hopeless. HOYLE loses quickly and consistently in tournament #1, averaging 5.5 moves per contest. In tournament #2, HOYLE's experience is virtually useless for so few contests without symmetry; indeed, the human won every contest playing the major diagonal on the second level. Because the branching factor is so large, there are 562 different states HOYLE could encounter on its third move against this simple strategy, and even then Sadder could only warn it away from its prior losing moves, with 21 losing moves to be learned for every state. Although the caches compiled important information, the contests averaged 5.6 moves, not a significant difference. In tournament #3, Pitchfork should be expected to make a significant impact. Recall, however, that HOYLE deliberately limits its Advisors' resources. Pitchfork does not have time to search for forks beyond depth three. Thus, in tournament #3 Pitchfork is rarely able to play assertively. It wins only one contest, as Player, and it loses all the others. HOYLE is able to prolong the contests somewhat, to 7.1 moves on average in tournament #3, but it does not exploit its role as Player because it does not make the key opening move. In tournament #4, Open learns the key opening during the first contest, and Pitchfork, when it has the resources on its subsequent moves, exploits that opening brilliantly. In a game where two less skilled participants could prolong the contest into a 27-move draw, tournament #4 averages 8.4 moves. HOYLE wins whenever it is Player, and defends valiantly, but necessarily loses, as Opponent. Despite a search space with millions of nodes, HOYLE learns to play three-dimensional tic-tac-toe with absolute expertise after a single contest in tournament #4.

Qubic is a three-dimensional version of tic-tac-toe played on four parallel four-by-four grids. A win is four in a straight line, and there are 76 such lines. In this game, with well over a billion possible states, a Player with absolute expertise should always win.[29] The forks that arise are initially at least depth five, well beyond the average human's notice, and outside Pitchfork's resource limit as well. The average branching factor in Qubic is 32, so random play loses very quickly, averaging 7.6 moves in tournament #1 in Figure 13. As with three-dimensional tic-tac-

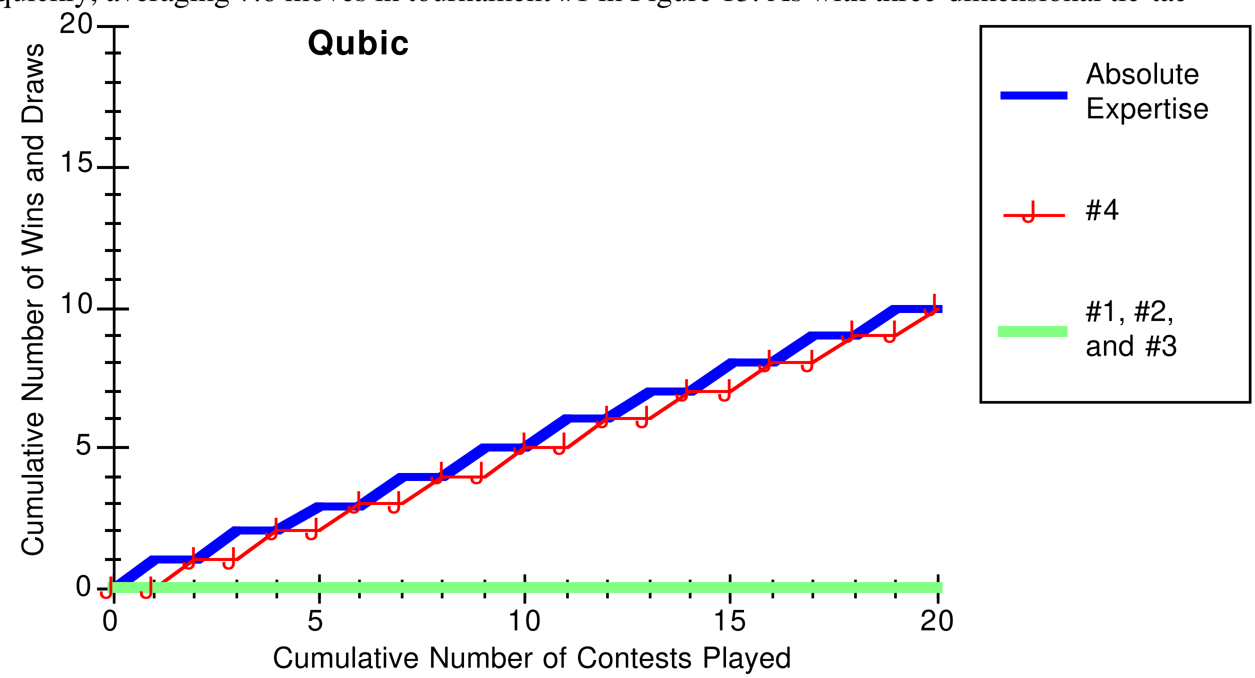


Figure 13. Cumulative non-losses in four Qubic tournaments of twenty contests each. HOYLE learns to play extremely well with memory, generalization and all its Advisors.

toe, memory and generalization are not enough; HOYLE continues to lose every contest in tournament #2, even though it is able to prolong the contests somewhat, to an average of 8.5 moves. In tournament #3, Pitchfork struggles to identify fork isomorphs within its time constraints and regularly signals that it was not allocated sufficient resources. Analogously, the human participant, faced with the increased complexity of this game, is more likely to overlook such plans, but the human competitor knows a three-dimensional analog to the fork in Figure 2. Since HOYLE is not caching its experience, the person can repeatedly exploit that fork, and does so, in both roles. HOYLE does not detect the threat immediately because the fork is so deep. By the time Pitchfork, with its limited resources, recognizes the challenge, it is too late to defend against it, and HOYLE concedes. The average contest in tournament #3 is 19.4 moves, however, indicating that these HOYLE Advisors offer much stronger competition than the experiential version. With an earlier version of the program, in tournament #4 the handicaps of limited rationality (both as a time limit and as fork depth limit) and lack of knowledge of symmetry were too great for HOYLE to make any visible progress at learning Qubic. There were simply too many forks, and too many fork isomorphisms, for Pitchfork to consider. Against the ordinary person, however, this early version of HOYLE repeatedly proved to be the better player, despite the theoretical nature of the search space. A series of strong game-playing volunteers, although admittedly not Qubic experts, challenged the program in its learning mode and always lost. The early version of HOYLE defended reasonably well and played a superb offensive game as measured against these participants.

The signal from Pitchfork that it lacked adequate resources to complete its task suggested modifications detailed elsewhere.[12] Briefly, HOYLE was modified to learn a fork during the post mortem as an explanation of its loss and provided with heuristics that focused its attention first on recently-learned forks applicable to the current game. Since such behavior encompasses both skill and memory, its effects should only be considered in tournament #4. The data for tournament #4 in Figure 13 is from the replay after modification. This time HOYLE is defeated by the usual deep fork in the first contest, but it analyzes the history and learns the fork, placing it first on Pitchfork's search list. The next time the human expert attempts the fork, HOYLE identifies and blocks it. HOYLE goes on to defend successfully; it never loses another contest, and is regularly able to catch the human expert off guard, winning all but three of the remaining contests.

This revised version of HOYLE was also tested on three-dimensional tic-tac-toe. Previously, Open had learned the one-move opening that began each contest so well that Pitchfork had time to make the subsequent decisions correctly. With the fork-learning version, HOYLE learned to play with absolute expertise just as quickly, but it also learned to concede as soon as Player made the single correct opening move.

7. Discussion

HOYLE is intended to learn to play any two-person perfect information game well. As such, the program has no game-specific representational transformations or strategies, only the ability to learn from its experience and some prior knowledge about its general domain. HOYLE has been tested and observed in a series of experimental tournaments carefully designed to measure and identify the origin of its prowess over a testbed of seven different games. As in any competition, the human participants deliberately attempted to outwit the machine and probe for its shortcomings. In tic-tac-toe, for example, human participants who had not been alerted in any way to the state in Figure 2 quickly found HOYLE's initial weakness there and exploited it as long as possible. Empirical results show that HOYLE learns to play all of its games extremely well in only 20 contests, most of them with absolute expertise.

Tournament #1 serves as a norm for unintelligent behavior. Random play is expected to perform quite badly, and that expectation is clearly realized in all the games, particularly those with large branching factors. Although human onlookers often impute intelligent motives to moves actually chosen at random in these contests, random play neither plans nor defends, as the results indicate.

The experiential version (#2) explores the power of memory and generalization, without symmetry. Using only the post mortem generalization and the aspects of the Advisors that access memory, HOYLE caches much information. For example, inspection of the caches from the second tic-tac-toe tournament revealed that, in those 20 contests, HOYLE acquired 36 pieces of knowledge during its post mortems: 17 moves to make in given states and 19 moves not to make in given states. Among these were two successful identifications of how to exploit offensive forks properly and two recognitions that the mover had been forked and should resign, *without* the Advisor for forks. In a game with well over 5000 reachable states, however, the experiential version would have to play hundreds, perhaps thousands, of contests to cache the expertise it would need to win in this mode. In larger games, such as Qubic, HOYLE's post mortem requires several minutes of real time after each contest. HOYLE's lack of a symmetry heuristic highlights how much information there is to be collected and how slowly expertise develops in most games when it is dependent only on experience. In a very large game graph, of course, mere memory, even with symmetry, would be overwhelmed for the middlegame and the endgame. For the opening, however, HOYLE, like many human players, memorizes the other participant's successful choices. In one game, pong hau k'i, simple aping of the other participant was enough to achieve absolute expertise. In another, lose tic-tac-toe, memory is far more helpful in one role than the other. For the rest of the games, although memory and generalization permit HOYLE to learn from its mistakes and speed the selection of moves, the experiential version of HOYLE learns to play better very slowly.

The Advisors without memory (#3) offer prior knowledge of the domain for play at any specific game; they play many of the games surprisingly well. When it is applicable, the Advisor for forks, the Advisor based on a simple two-ply lookahead, and the Advisor based on the null move heuristic[1] are particularly important. In tsoro yematatu and pong hau k'i, prior knowledge is sufficient to play with absolute expertise immediately. For the other games, at best this limited version of HOYLE plays well initially. Unless the human participant discovers a weakness, this version's lack of consistency, i.e., its inability to learn from its mistakes, proves devastating in a tournament situation.

In its full version (#4), with all the Advisors and the post mortem, HOYLE has the advantages of both memory and its prior knowledge of the domain. There is a clear synergy between the intelligently-applied memory of tournament #2 and the exacting weak methods of tournament #3, as evidenced by the machine's performance in tournament #4 for tic-tac-toe, lose tic-tac-toe, achi, and three-dimensional tic-tac-toe. HOYLE begins as a thoughtful participant and steadily learns to be a skillful one.

HOYLE quickly learns to play all but one of its games, lose tic-tac-toe, with absolute expertise in either role, i.e., after only 20 contests it consistently wins whenever possible and draws otherwise. Although it plays imperfectly after tournament #4, HOYLE is not a novice at lose tic-tac-toe; it defeats all but the most sophisticated human players. Inspection of the knowledge base suggested that HOYLE could learn lose tic-tac-toe with absolute expertise given a longer tournament and/or a representational transformation for symmetry.

To explore this premise, the full version of HOYLE played an additional lose tic-tac-toe tournament (#5) of 200 contests. A human expert as Opponent deliberately moved through all 48 possible ordered pairs of first two moves, repeating the pair in each successive contest until

HOYLE successfully defended against it. The intent was to test HOYLE's expertise thoroughly, but theoretically this could train HOYLE to play perfectly. As the tournament progressed, HOYLE was more often able to declare a clear win or loss based on its caches; 14 of the last 82 contests were called this way, one after only six moves. Despite the larger caches in such a long tournament, HOYLE's play was virtually instantaneous. As Opponent, HOYLE won or drew 96 of its 100 contests; these losses were always due to errors on the third move. As Player, HOYLE won or drew 54 of its 100 contests. HOYLE made many errors, but the tournament situation, where participants alternate roles, combined with the deliberate testing, supported learning in an unanticipated fashion. When HOYLE made a mistake in one role in a contest, the post mortem presumed that the human in the other role had demonstrated a good line of play. In the next contest, with their roles exchanged, Anthropomorph therefore encouraged HOYLE to recreate the previous contest. If HOYLE took Anthropomorph's advice, it had the opportunity to observe, and thereby learn, the correct human response to its previous error. Without symmetry or extensive lookahead, HOYLE appeared to have learned to defend against all 56 good two-move sequences after contest #190.

HOYLE's learned expertise at lose tic-tac-toe highlights three important issues:

• *Is behavioral evidence sufficient?* During the 200 contests, the blackboard indicated that sometimes HOYLE chose a correct move fortuitously, i.e., purely by chance. That move would be recommended in any subsequent replay of the contest, on Cyber's recommendation, but the initial impetus for the human trainer to move on to the next opening was based on superficial evidence that HOYLE had learned. Has a machine learned when it does the correct thing for the wrong reason, or for no reason at all? The training tournament was misleading in that some expert openings for Opponent have the same first two moves but a variation on the third move. During HOYLE's last five contests as Player, after it had supposedly mastered the game, the human participant deliberately challenged the program with some of these variations, the first two moves of which HOYLE defended against correctly, based on its previous success. The third move, however, challenged HOYLE in a way previously unexperienced, and HOYLE lost twice. After tournament #5 HOYLE will not make those same mistakes again, but there are probably a few openings to which it will be vulnerable. Behavioral criteria can be deceiving.

• *How does the display of skill differ from understanding?* The trainer as Player never challenged HOYLE with an imperfect opening. No Player with absolute expertise would ever open imperfectly in lose tic-tac-toe because it guarantees a loss against a strong Opponent. Thus, although HOYLE learns to defend against perfect expertise, it could be defeated by a novice who opened incorrectly but played skillfully thereafter. One could argue that HOYLE achieves perfect expertise at this game and yet lacks understanding. This suggests that the goal of game playing may require redefinition, to include competition against mediocre players as well.

• *How does real world training affect learning?* A system that is to learn in a real world environment must be robust to human error. In one contest, for example, the human Opponent accidentally forked himself, but HOYLE did not defend properly and lost. In the next contest, HOYLE as Opponent immediately tried to reproduce the previous gambit and forked itself but, against the correct defense, lost again. The post mortem then recognized the fork and cached a significant loss state to guard against its repetition. In another contest the human Player made a typing error on her second move, but HOYLE lost anyway, and therefore believed the error to be a good opening. Correction of this misconception required 30-odd contests while Not Again explored alternative later moves. Eventually the post mortem forbade HOYLE from the human's original mistake. Clearly, training against a mediocre human, although manageable, would be substantially slower.

8. Conclusions

This paper has described a program that learns to play seven games very well. Labeling HOYLE, however, is difficult. It seems to be an expert system without production rules, a learner without a learning algorithm, a game player without search.

HOYLE is certainly an expert program. Its initial expertise lies in its weak theory about games in general. It has substantial prior knowledge about its broad domain: how to function there and how to learn there. Under the direction of this weak theory, the program goes on to become an expert at certain problems (games) within its domain. The Advisors substitute for production rules; they have complex conditions and recommend multiple actions, leaving the final decision to a control strategy that seeks a consensus. Individual Advisors do not overpower each other with their presumptive or historic importance; they collaborate to make a choice.

HOYLE actually applies several well-known algorithms to learning. It learns patterns and caches them. Significant states are board patterns; forks are graph patterns; openings, expert moves, and the program's own successful moves are stimulus-response patterns. Significant states are acquired from restricted retrograde analysis, forks from explanation-based learning, and caches are constructed, under the direction of heuristics, by rote. HOYLE also learns answers to questions embodied by its weak theory, questions whose answers (e.g., "Is it an advantage to go first?") may be referenced by its Advisors in the construction of their comments. These answers rely on inductive inference. Each question is a slot in the game library, and each answer the result of a heuristic procedure.

For the experienced game-playing programmer, HOYLE's ostensible lack of search is puzzling. Expert game-playing programs have repeatedly demonstrated that deeper search is the way to win, yet we claim that HOYLE looks no more than two-ply ahead during play. HOYLE does search, in a limited manner, *after* a contest, when it examines the trace for significant states. HOYLE also has a few Advisors that search the game graph: Victory, Don't Lose, and Enough Rope search one move ahead, Shortsight and Panic search two. Pitchfork searches, not in the game graph, but in the strategic maps, for subgraphs isomorphic to forks. This last search could be costly if it were not severely resource-restricted.

HOYLE does learn, that is, its experience transforms its subsequent behavior. How it learns varies with the slot. What it learns is not always highly significant; it may, for example, be unduly impressed by an opening. What it learns is not always correct; it may, for example, infer from a skewed set of contests that a game gives the advantage to the wrong participant. Most startling, however, is that what it *retains* is very little relative to what it experiences and to what it could experience in the game graph. Tic-tac-toe, for example, has 5478 nodes in its game graph. On average, in a 20-contest tournament HOYLE sees 180 of these and, without any knowledge of symmetry, stores only 10, less than .02% of the game graph. Playing more contests does not substantially increase this storage, i.e., HOYLE seeks only enough knowledge to play well. Up to a point, more experience accesses more knowledge and better performance; after that HOYLE does not collect redundant or unnecessary information. Some of this perspicacity is attributable to HOYLE's expert opposition; it serves as a model of goal behavior and a guide through the most important parts of the game graph.

Our strategy has been to develop HOYLE incrementally. The program began learning two games; as it mastered those we introduced others and watched for mistakes. Analysis of these errors identified underlying principles and led to reformulation of the weak theory to avoid repetition of such mistakes, both in other contests and other games. We have, waiting in the wings, several Advisors that we expected to need, particularly one for mobility. We have not called upon them yet because the ones described here have thus far sufficed. We are now adding 16 somewhat more difficult games to HOYLE's testbed. As the program fails to develop adequate expertise we will refine the weak theory to improve learning and decision making. Once the program is proficient at the new games we will add yet more difficult ones. HOYLE is a long-term project; as such, it will be continually, but gradually, challenged and strengthened, as BACON[21] has been.

In its current form, HOYLE could play checkers or chess, or even Go correctly, but it would lose steadily. In more difficult games one would expect more sophisticated advice to be required. For example, Pitchfork is currently inapplicable to games where moves are invertible, like pong hau k'i, and in games where the objective is to avoid a configuration, like lose tic-tac-toe. HOYLE probably cannot continue to ignore symmetry in more difficult games either; its exclusion thus far has been an empirical device, with no justification in the world of game playing. HOYLE also lacks useful game-playing primitives, such as "intermediate goal" and "sacrifice," with which humans reason about more difficult games. These abstractions and representational transformations suppress and refocus search and memory. For sufficiently large game graphs, storage of even

.02% is infeasible, and new concepts may be necessary. Finally, in complex search spaces people traditionally plan. HOYLE's planning is now limited to the bottom vertices of its strategic maps, and may have to be extended to contend with larger search spaces.

If HOYLE learns to play checkers or chess or Go, it will not be for several years. Although scaling up is an important issue, this research is as much about economical, reactive learning in related problem classes as it is about playing games. The number of different behaviors that can be learned under a weak theory is itself an increase in scale. It is also important to note that a larger game graph does not necessarily present a more difficult learning problem. For example, tsoro yematatu's graph is about twice the size of lose tic-tac-toe's, and achi's is more than twice as large again, but HOYLE took substantially longer to learn to play lose tic-tac-toe expertly.

We plan to have HOYLE compare different learning and playing strategies for a specific game, to have it learn strategies for use against particular participants, to have it learn strategies that take advice differently in different stages of a contest (e.g., openings, endgames), to have it draw analogies between games, and to have it discover the control strategy that makes a decision based upon the Advisors' comments. Additional future work includes reorganization of the Advisors' caches as their stored experience grows unwieldy, analysis of contests in which the expected role winner did not win, the nature of optimal (as opposed to merely winning) play, a natural language for the initial game frame instantiation, and representational transformations of the game board, including symmetry.

The experiments described here demonstrate that there is some synergy between HOYLE's game-independent expertise and its memory, and that an instantiable weak theory for the entire domain of two-person, perfect information games can effectively learn appropriate search control heuristics for seven specific games. As the games in its knowledge base become more difficult, HOYLE will continue to be evaluated by the prowess it displays and the speed with which it learns. Meanwhile, HOYLE provides an alternative to traditional, search-driven game-playing programs. More broadly, HOYLE suggests that many smart, narrow perspectives may be broadly relevant, may be generated quickly, and may be incrementally evaluated to learn a class of related problems.

Acknowledgements

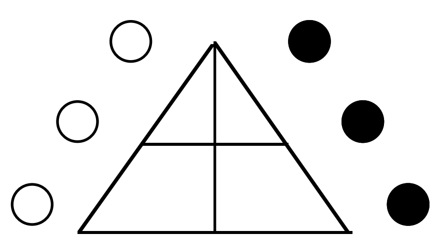
This work was supported in part by NSF 9001936 and PSC-CUNY 666397. The author thanks Hans Berliner, Doug Fisher, Virginia Teller, and an anonymous referee for their thoughtful critiques and support in the development and presentation of this material. Michael Georgopoulos and Kouros Esfahany provided expert level programming support.

Appendix A. The Games

• **Tic-tac-toe** is played on a 3 × 3 grid. Player has five X's, and Opponent four O's. Initially the board is empty. A turn consists of placing one of your markers in any empty square. The first one to place three of the same markers in a row, vertically, horizontally, or diagonally, wins. There are eight such winning lines. Play ends in a draw when there are no more empty squares. The 5478 nodes in the game graph form a tree.

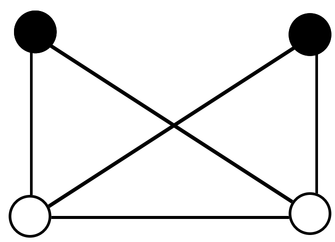
• **Lose tic-tac-toe** is played on a 3 × 3 grid. Player has five X's, and Opponent four O's. Initially the board is empty. A turn consists of placing one of your markers in any empty square. The first one to place three of the same markers in a row, vertically, horizontally, or diagonally, loses. There are eight such losing lines. Play ends in a draw when there are no more empty squares. The 5478 nodes in the game graph form a tree.

• **Tsoro yematatu** is played on a board like this:



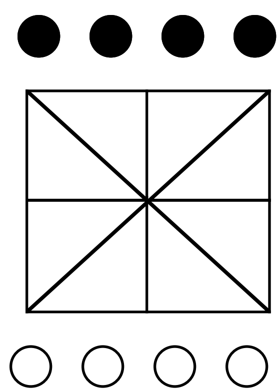
Player has 3 black markers, and Opponent 3 white ones. Initially the board is empty, and a turn consists of placing one of your markers on the intersection of two or more lines. There are seven such positions. Once all 3 of your markers are on the board, a turn consists of moving one of your markers to the single empty position. The first one to place three of the same markers in a row on any of the five lines wins. Play ends in a draw when it cycles through the same state often enough. The game graph consists of 855 nodes as a tree for the first stage; each leaf from the first stage then has a cyclic graph through 140 nodes with branching factor four from every node.

• **Pong hau k'i** is played on a board like this:



Player has two black markers, and Opponent two white ones. The initial state of the board is shown. A marker may lie on any intersection of two or more lines. There are five such positions. A move slides one of your markers to an adjacent position along a line without going through a third position or over another marker. The first one unable to move loses. Play ends in a draw when it cycles through the same state often enough. There are 30 nodes in the game graph.

• **Achi** is played on a board like this:



Player has four black markers, and Opponent four white ones. Initially the board is empty, and a turn consists of placing one of your markers on the intersection of two or more lines. There are nine such positions. Once all four of your markers are on the board, a turn consists of moving one of your markers to the single empty position. The first one to place three of the same markers in a row, vertically, horizontally, or diagonally, wins. There are eight such winning lines. Play ends in a draw when it cycles through the same state often enough. The game graph consists of 5240 nodes as a tree for the first stage; each leaf from the first stage then has a cyclic graph through 630 nodes with branching factor four from every node.

• **Three dimensional tic-tac-toe** is played on a $3 \times 3 \times 3$ grid. Player has 14 X's, and Opponent 13 O's. Initially the board is empty. A turn consists of placing one of your markers in any empty square. The first one to place three of the same markers in a row, vertically, horizontally, or diagonally, wins. There are 49 such winning lines. Play ends in a draw when there are no more empty squares. The millions of nodes in the game graph form a tree.

• **Qubic** is played on a $4 \times 4 \times 4$ grid. Player has 32 X's, and Opponent 32 O's. Initially the board is empty. A turn consists of placing one of your markers in any empty square. The first one to place four of the same markers in a row, vertically, horizontally, or diagonally, wins. There are 76 such winning lines. Play ends in a draw when there are no more empty squares. The billions of nodes in the game graph form a tree.

Appendix B. The Advisors and Move Selection

HOYLE's Advisors are an important part of its weak theory for the domain of two-person perfect information games. They provide advice on move selection when it is HOYLE's turn, and their implementation is game-independent. The Advisors are grouped into tiers. If any tier produces a single move recommendation, then none of the subsequent tiers is consulted. The tiers, and their internal conflict resolution technique are summarized below. Because HOYLE plays a broad variety of games, certain Advisors may not be relevant to a specific game.

Tier 1:

Within this tier Advisors are consulted in the indicated order. If any of the first three Advisors makes one or more comments, the remainder of the Advisors are not consulted and control returns to the game-playing algorithm.

• *Wiser:* If this is a significant win state, make the correct move. This Advisor is completely dependent upon cache memory.

• *Sadder:* If this is a significant loss state, resign. This Advisor is completely dependent upon cache memory.

• *Victory:* If you see a winning move now, take it.

• *Don't Lose:* Do not make a move that will result in a loss. This Advisor has a cache memory component and may reduce the set of legal moves.

Tier 2:

Within this tier Advisors are consulted in the indicated order. If the first two Advisors make one or more comments, the remainder of the Advisors are not consulted and control returns to the game-playing algorithm.

• *Panic:* If it were the non-mover's turn now and she would have a winning move, block it. If all such winning moves for the opposition are not blockable by a single move, resign.

• *Shortsight:* Advise for and against moves based on a two-ply lookahead. This Advisor has a cache memory component and may reduce the set of legal moves.

• *Enough Rope:* If it were the non-mover's turn now, and she would have a losing move, avoid blocking it, i.e., leave her the opportunity to err. This Advisor may reduce the set of legal moves.

Tier 3:

Within this tier all Advisors are consulted before control returns to the game-playing algorithm. Conflict resolution is based on the frequency and weight of the comments for and against each move.

• *Pitchfork:* Advance offensive forks and destroy defensive ones. This Advisor has a cache memory component.

• *Candide:* Formulate and advance naive offensive plans, i.e., ones that do not anticipate any opposition.

• *Worried:* Observe and destroy naive offensive plans for the other participant, i.e., ones that do not anticipate any opposition.

• *Open:* Recommend previously-observed non-HOYLE openings. This Advisor is completely dependent upon cache memory.

• *Anthropomorph:* Move as a winning or drawing non-HOYLE expert once did. This Advisor is completely dependent upon cache memory.

• *Not Again:* Do not move as a losing HOYLE once did. This Advisor is completely dependent upon cache memory.

• *Cyber:* Move as a winning or drawing HOYLE once did. This Advisor is completely dependent upon cache memory.

•*Greedy:* Propose moves that advance more than one plan to win.

References

1. Beal, D.F. 1990. A Generalized Quiescence Search Algorithm. *Artificial Intelligence* **43**(1): 85-98.

2. Bell, R.C. 1969. *Board and Table Games from Many Civilizations*. London: Oxford University Press.

3. Berliner, H.J. 1979. The B*Tree Search Algorithm: A Best-First Proof Procedure. *Artificial Intelligence* **12**: 23-40.

4. Berliner, H.J. 1980. Backgammon Computer Program Beats World Champion. *Artificial Intelligence* **14**: 205-220.

5. Berliner, H. and C. Ebeling. 1989. Pattern Knowledge and Search: The SUPREM Architecture. *Artificial Intelligence* **38** (2): 161-198.

6. Brooks, R.A. 1991. Intelligence without Representation. *Artificial Intelligence*, **47** (1-3): 139-160.

7. Chen, K. 1989. Group Identification in Computer Go. In *Heuristic Programming in Artificial Intelligence - The First Computer Olympiad*, ed. D. Levy and D. Beal. New York: John Wiley.

8. Cohen, D.I.A. 1972. The Solution of a Simple Game. *Mathematics Magazine* **45**: 213-216.

9. Dietterich, T.G. 1982. Learning and Inductive Inference. In *The Handbook of Artificial Intelligence*, Volume 3, ed. P.R. Cohen and E.A. Feigenbaum. Los Altos, CA: William Kaufmann, Inc.

10. Epstein, S.L. 1989. Mediation among Advisors. In *Proceedings on AI and Limited Rationality*, 35-39. AAAI 1989 Spring Symposium Series.

11. Epstein, S.L. 1991. Deep Forks in Strategic Maps - Playing to Win. In *Heuristic Programming in Artificial Intelligence 2 - The Second Computer Olympiad,* 189-203. Ed. D.N.L. Levy and D.F. Beal . Chichester: Ellis Horwood Limited.

12. Epstein, S. L. 1990. Learning Plans for Competitive Domains. In *Proceedings of the Seventh International Conference on Machine Learning*, 190-197. Ed. B.W. Porter and R.J. Mooney. Morgan Kaufmann.

13. George and Schaeffer. Chunking for Experience. In *Advances in Computer Chess VI,* ed. D.F. Beal. To appear.

14. Goldberg, D. 1989. *Genetic Algorithms.* Reading, MA: Addison Wesley.

15. Gupton, G. 1989. Genetic Learning Algorithm Applied to the Game of Othello. In *Heuristic Programming in Artificial Intelligence - The First Computer Olympiad*, ed. D. Levy and D. Beal. New York: John Wiley.

16. Hsu, F-h., Anantharaman, T.S., Campbell, M.S., and Nowatzyk, A.. 1990. Deep Thought. In *Computers, Chess and Cognition*, ed. T.A. Marsland and J. Schaeffer, 55-78. New York: Springer-Verlag.

17. Kierulf, A. 1990. New Concepts in Computer Othello: Corner Value, Edge Avoidance, Access, and Parity. In *Heuristic Programming in Artificial Intelligence - The First Computer Olympiad*, ed. D. N. L. Levy and D. F. Beal, 225-240. Chichester, England: Ellis Horwood.

18. Kierulf, A., Chen, K., and Nievergelt, J. 1990. Smart Game Board and Go Explorer: A Study in Software and Knowledge Engineering. *Communications of the ACM*, **33**(2): 152-166.

19. Kirsh, D. 1991. Today, the Earwig, Tomorrow Man? *Artificial Intelligence*, **47** (1-3): 161-184.

20. Langley, P. 1985. Learning to Search: From Weak Methods to Domain-Specific Heuristics. *Cognitive Science* **9**: 217-260.

21. Langley, P., H.A. Simon, G.L. Bradshaw, and J.M. Zytkow. 1987. *Scientific Discovery:*

Computational Explorations of the Creative Processes. Cambridge: MIT Press.

22. Lee, K.F. and S. Mahajan. 1988. A Pattern Classification Approach to Evaluation Function Learning. *Artificial Intelligence* **36**: 1-26.

23. Lee, K.F. and S. Mahajan. 1990. The Development of a World Class Othello Program. *Artificial Intelligence*, **43**(1), 21-36.

24. McAllester, D.A. 1988. Conspiracy Numbers for Min-Max Search. *Artificial Intelligence* **35**: 287-310.

25. Minsky, M. 1985. *The Society of Mind* . New York: Simon & Schuster.

26. Mitchell, T.M., P.E. Utgoff, and R. Banerji. 1983. Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics. In *Machine Learning: An Artificial Intelligence Approach*, ed. R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, 163-190. Palo Alto, California: Tioga Publishing.

27. Nilsson, N.J. 1980. *Principles of Artificial Intelligence*. Palo Alto: Tioga Publishing.

28. Palay, A.J. 1982. The B* Tree Search Algorithm - New Results. *Artificial Intelligence* **19**: 145-164.

29. Patashnik, O. 1980. Qubic: 4 x 4 x 4 Tic-Tac-Toe. *Mathematics Magazine*. **53**: 202-216.

30. Pazzani, M.J. 1987. Inducing Causal and Social Theories: A Prerequisite for Explanation-based Learning. In *Proceedings of the Fourth International Workshop on Machine Learning*, 230-241. Ed. P. Langley. Los Altos, CA: Morgan Kaufmann.

31. Pell, B. 1991. Exploratory Learning in the Game of GO: Initial Results. In *Heuristic Programming in Artificial Intelligence 2 - The Second Computer Olympiad*, ed. D.N.L. Levy and D.F. Beal, 147-152. Chichester, England: Ellis Horwood.

32. Rivest, R.L. 1987. Game Tree Searching by Min/Max Approximation. *Artificial Intelligence* **34**: 77-96.

33. Rosenbloom, P.S. 1982. A World-Championship Level Othello Program. *Artificial Intelligence* **19**: 279-320.

34. 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, 71-105. New York: McGraw-Hill.

35. Samuel, A.L. 1967. Some Studies in Machine Learning Using the Game of Checkers. II - Recent Progress. *IBM Journal of Research and Development* : 601-617.

36. Schaeffer, J., J. Culberson, N. Treloar, B. Knight, P. Lu, and D. Szafron. 1991. Reviving the Game of Checkers. In *Heuristic Programming in Artificial Intelligence - The Second Computer Olympiad*, ed. D.N.L. Levy and D. Beal. Chichester: Ellis Horwood. 119-136.

37. Scherzer, T., L. Scherzer, and D. Tjaden. 1990. Learning in Bebe. In *Computers, Chess, and Cognition,* ed. T.A. Marsland and J. Schaeffer, 197-216. New York: Springer-Verlag.

38. Schultz, A.C., and K.A. De De. 1988. Using Experience-Based Learning in Game Playing. *Proceedings of the Fifth International Conference on Machine Learning*, 284-290. Ann Arbor, Michigan.

39. Slate, D.J., and L.R. Atkin. 1977. Chess 4.5 - The Northwestern University Chess Program. In *Chess Skill in Man and Machine*, ed. P.W. Frey, 82-118. New York: Springer-Verlag.

40. Tesauro, G. and T.J.Sejnowski. 1989. A Parallel Network that Learns to Play Backgammon. *Artificial Intelligence,* **39** (3), 357 - 390.

41. Utgoff, P.E., and P.S. Heitman. 1988. Learning and Generalizing Move Selection Preferences. In *Proceedings of the AAAI 1988 Spring Symposium Series: Computer Game Playing*, 36-40. Stanford, California.

42. Zaslavsky, C. 1982. *Tic Tac Toe and Other Three-in-a-Row Games, from Ancient Egypt to the Modern Computer*. New York: Crowell.