Learning How to Satisfice

Susan L. Epstein

Department of Computer Science
Hunter College and The Graduate School
The City University of New York
New York, NY 10021
epstein@roz.hunter.cuny.edu

Jack Gelfand

Department of Psychology
Princeton University
Princeton, NJ 08544
jjg@princeton.edu

Esther Lock

Department of Computer Science
The Graduate School of
The City University of New York
New York, NY 10021
elock@csp.gc.cuny.edu

Abstract

If a system is to both satisfice and compute with a high degree of reliability, a balance must be struck. This paper describes an architecture that effectively integrates correct reasoning with satisficing heuristics. As a program inductively learns new heuristics, the architecture underlying it robustly incorporates them. Many of these heuristics represent significant information previously inexpressible in the program's representation and, in some cases, not readily deducible from a description of the problem class. After training, the program exhibits both an emphasis on its new heuristics and the ability to respond correctly to novel situations with them. This significantly improves the program's performance. The domain of investigation is board games, but the methods outlined here are applicable to an autonomous learning program in any spatial domain.

1. Issues in Satisficing

To *satisfice* is to make a good enough decision (Simon 1981). An agent that satisfices is sometimes termed limitedly rational because, despite the availability of a perfectly correct process, it uses a more expedient *heuristic,* a reasonably reliable rule of thumb. The faster choice permits the agent to conserve its resources, and so is often the wiser one. Human experts, for example, do not search the full game tree for most board games, even though they understand that the best possible move is deduced that way.

Although heuristics may speed computation, sole reliance on them risks obvious oversights. A heuristic is liable to overlook an otherwise evident correct choice, or make an egregious error. Heuristics fail when they are overly broad generalizations. Consider, for example, the quite reputable heuristic *H* "always play the center" for tic-tac-toe. In Figure 1 it is X's turn, and *H* clearly makes the wrong choice.

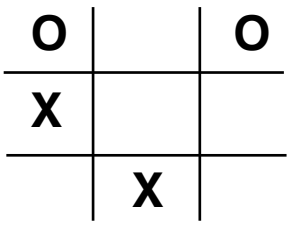

Figure 1: A tic-tac-toe state with X to move.

Overly broad generalizations are of particular concern for programs that learn inductively, because such heuristics make them brittle. For example, imagine an inductive learning program that watched two contestants, who always made the correct move based on minimax from the full game tree, compete against each other at tic-tac-toe. (Figure 1 would never appear in such a tournament.) The program would observe that one or the other always played the center quite early in the contest. As a result, the program might learn *H*. Although that would stand it in good stead against such contestants, in the broader contest of the full game tree, *H* is not completely reliable. Applying *H* to Figure 1 overlooks an essential defensive move. A more accurate heuristic has a caveat attached, something on the order of "play the center unless…."

Learning heuristics with caveats, however, subverts the fundamental rationale for satisficing: efficiency. Because they strive to be more accurate, such heuristics are less general, and therefore there are many more of them. The caveats become lengthy as the program has more experience, and they may be somewhat repetitive. For example, on one run of 800 contests a program learned 45 tic-tac-toe rules with 52 caveats (Fawcett & Utgoff 1991).

An inductively learned heuristic cannot be guaranteed trustworthy unless the program has the domain knowledge or the experience to test it on every state in the space. Nonetheless, in very large or dynamic environments learning heuristics is often essential, and induction is itself a satisficing technique. Possible inaccuracy is not adequate reason to abandon induction, only a warning that learned heuristics must be treated even more gingerly than those anticipated by the programmer.

This paper describes *FORR* (FOr the Right Reasons), a satisficing architecture that addresses these issues. FORR includes enough correct computation at manageable cost to prevent egregious errors and make obvious decisions easily. FORR's procedural hierarchy integrates economical, correct routines with heuristic ones. FORR also supports the incorporation of inductively learned heuristics in a way that preserves the integrity of the system while it expands

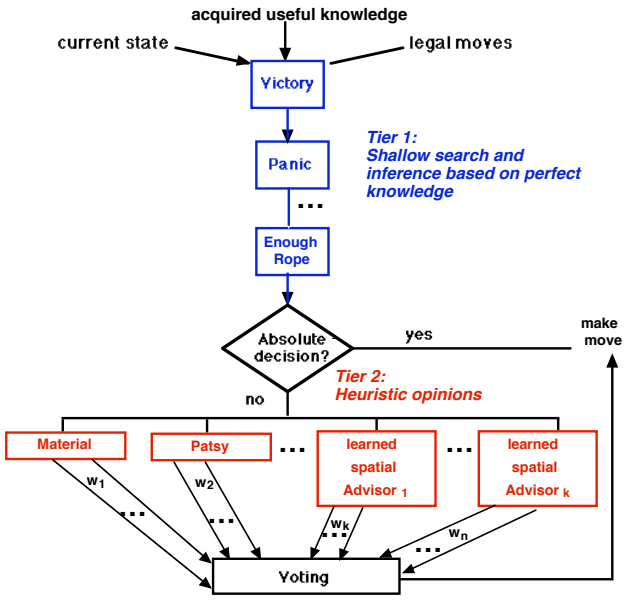

Figure 2: A schematic of decision making in Hoyle.

its capabilities. As a result, a FORR-based program is both robust and adaptive.

The next section of this paper describes the architecture and a FORR-based program for game playing. Subsequent sections detail one way new heuristics can be learned inductively and integrated with care. Finally we discuss experimental results, and related and future work.

2. The FORR Architecture

FORR is a general architecture for learning and problem solving in a broad domain of related problem classes (Epstein 1994a; Epstein 1998). FORR is an implemented system, available in Common Lisp for a variety of platforms. *Hoyle*, the example used throughout this paper, is a FORR-based program for the domain of two-person, perfect information, finite-board games (Epstein 1992).

In FORR, a procedural implementation of a particular decision-making rationale is called an *Advisor*. The input to any Advisor at any decision point is the current state of the world, a list of the legal actions possible, and a collection of *useful knowledge,* information that is potentially applicable and probably correct for a specific problem class. In Hoyle, the current state is the game board and whose turn it is to move, and the legal actions are the rule-abiding moves. Useful knowledge is data learned from experience and expected to enhance performance. It may or may not be correct, depending upon the algorithm used to acquire it. Good openings are an example of useful knowledge for game playing. The output of an Advisor is any number of *comments*, each of the form

<Advisor, action, strength>

Strength is an integer in [0, 10] that indicates the attitude of the Advisor to the action, from firm opposition (0) to strong support (10). Advisors need not reference useful knowledge and they do not learn; they simply generate comments.

For the most part, FORR Advisors are expected to be domain dependent but problem class independent. In Hoyle, for example, where the domain is game playing and a problem class is a particular game, such as tic-tac-toe, an Advisor should be applicable (although not necessarily valuable) to board games in general, and not intended for some particular game. Hoyle also has, however, some number (determined during execution) of learned, game-specific heuristic Advisors (*learned spatial Advisors*) that proceduralize spatial concepts based on visual perception, as described in the next section.

FORR accepts both correct and heuristic reasoning procedures for a domain, but requires the programmer to distinguish in advance between them. FORR organizes a program's Advisors into a hierarchy; Hoyle's is shown in Figure 2. The correct Advisors appear in tier 1, with a sequence prespecified by the programmer. The heuristic ones are relegated to tier 2. Table 1 lists Hoyle's full complement of Advisors: 7 in tier 1 in their prespecified sequence, and 15 in tier 2 alphabetically, with the learned ones grouped as the final entry. Note that in each tier there are some general game-playing decision makers (indicated by *'s) that reference any learned useful knowledge, and others that do not.

Tier-1 Advisors must be perfectly correct. For example, Hoyle's tier-1 Advisors do valid inference on data that is true in the context of the full game tree. The only limitation on a tier-1 Advisor is the extent of its computation. For example, Hoyle's Victory recommends a move that wins a contest immediately, and its Panic forbids a move that loses a contest after the opponent's response. At most, however, Hoyle's tier-1 Advisors do two-ply search. If some tier-1 Advisor mandates a move, or if as a group they find only one move acceptable, that move is made and the heuristic Advisors are never consulted.

Otherwise, the tier-2 Advisors collectively make their less reliable comments from their individual, heuristic viewpoints. For example, Hoyle's Material embodies the heuristic "maximize the number of your playing pieces and minimize the number of your opponent's." In tier 2 a move may be supported or opposed by many Advisors. Therefore each tier-2 Advisor has a weight (e.g., w_1 in Figure 2) that reflects its relevance to and relative significance in a particular problem class. When a decision must be made in tier 2, FORR selects the move with maximal support, summing the product of the strength from each comment about the move with the weight of the commenting Advisor.

Empirical experience with Hoyle indicates that appropriate Advisor weights are game-specific and should therefore be learned. After every contest Hoyle plays against an expert, *AWL* (Algorithm for Weight Learning) considers, one

at a time, only those states in which it was the expert's turn to move. For each such state, AWL distinguishes among support and opposition for the expert's recorded move and for other moves. AWL cumulatively adjusts the weights of tier-2 Advisors at the end of each contest, and uses those weights to make decisions throughout the subsequent con-

Table 1: Hoyle's Advisors for game playing. Tier 1 is in its prespecified order. Advisors with a * apply useful knowledge.

Name	Description
Tier 1	
Victory	Makes winning move from current state if there is one.
Wiser*	Makes correct move if current state is remembered as certain win.
Sadder*	Resigns if current state is remembered as certain loss.
Panic*	Blocks winning move non-mover would have if it were its turn now.
Don't Lose*	Eliminates any move that results in immediate loss.
Shortsight*	Advises for or against moves based on two-ply lookahead.
Enough Rope*	Avoids blocking losing move non-mover would have if it were its turn.
Tier 2	
Anthropomorph*	Moves as winning or drawing non-Hoyle expert did.
Candide	Formulates and advances naive offensive plans.
Challenge	Moves to maximize its number of winning lines or minimize non-mover's.
Coverage	Maximizes mover's influence on predrawn board lines or minimizes non-mover's.
Cyber*	Moves as winning or drawing Hoyle did.
Freedom	Moves to maximize number of its immediate next moves or minimize non-mover's.
Greedy	Moves to advance more than one winning line.
Leery*	Avoids moves to state from which loss occurred, but where limited search proved no certain failure.
Material	Moves to increase number of its pieces or decrease those of non-mover.
Not Again*	Avoids moving as losing Hoyle did.
Open*	Recommends previously-observed expert openings.
Patsy*	Supports or opposes moves based on their patterns' associated outcomes
Pitchfork *	Advances offensive forks or destroys defensive ones.
Vulnerable	Reduces non-mover's capture moves on two-ply lookahead.
Worried	Observes and destroys naive offensive plans of non-mover.
Learned spatial Advisors	Supports or opposes moves based on their creation or destruction of a single pattern.

test. Essentially, Hoyle learns to what extent each of its tier-2 Advisors simulates expertise, as exemplified by the expert's moves. Eventually, irrelevant and self-contradictory Advisors in a particular game should have weight 0, and more trustworthy Advisors should have higher weights than less trustworthy ones. AWL was based upon Winnow, a fast, perceptron-like learning algorithm (Littlestone 1988).

3. Learning and Applying New Heuristics

One current research effort is to have Hoyle learn new heuristics from patterns. For Hoyle, a *pattern* is a visually-perceived regularity, represented as a small geometric arrangements of playing pieces (e.g., black or X) and *blanks* (unoccupied positions) in a particular geographic location. A move can create a pattern by providing some missing piece or blank. In games where pieces are not permanently placed, a move can also destroy a pattern by relocating a playing piece. When it first learns a new game, Hoyle constructs a set of board-dependent *templates* as a filter for its perceived patterns: straight lines, L's, triangles, squares, and diagonals of a limited size composed of legal piece positions. When a template is instantiated with some combination of pieces, blanks, and don't care (#) symbols, it becomes a pattern. Further details on this process are available in (Epstein, Gelfand, & Lesniak 1996).

The *associative pattern store* (a pattern queue, a pattern cache, and generated spatial concepts) is a heuristically-organized database that links patterns with *contest outcome* (win, loss, or draw). Figure 3 is an overview of the development of Hoyle's spatial orientation from the game-specific associative pattern store. After each contest, the patterns newly created by each move are extracted with the templates. Next, patterns are associated with winning, losing, or drawing and stored in a pattern queue. Patterns that persist on the pattern queue over time and are identified with a single consistent outcome enter the pattern cache.

Proceduralization is the transformation of expert knowledge into expert behavior. Patterns in the cache are proceduralized with a tier-2 Advisor called *Patsy*, as described below. Periodic sweeps through the pattern cache also attempt to generalize sets of patterns into spatial concepts. Each concept is proceduralized as an individual, game-specific, learned spatial Advisor, a heuristic that is then validated during subsequent learning. Because pattern knowledge is extensive and contradictory, each segment of the

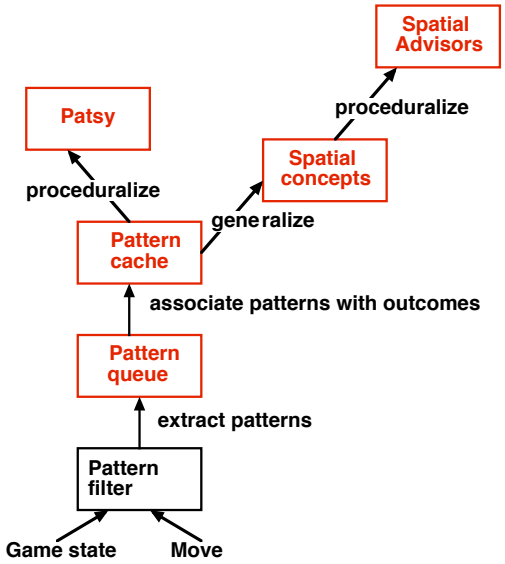

Figure 3: Hoyle's model for spatial learning.

associative pattern store relates differently to decision making. Queued patterns have no impact at all, cached patterns serve as input to Patsy, and pattern-based concepts become game-specific, learned spatial Advisors. Both patterns and learned Advisors are efficiently represented. A hierarchy from less to more specific speeds pattern matching and supports subsumption testing. In addition, Hoyle normalizes with respect to the symmetries of the plane, thereby reducing matching costs.

Patsy considers the set of possible next states resulting from the current legal moves. Each next state is compared with the patterns in the appropriate, game-specific cache. (No new patterns are cached during this process.) Each pattern is assigned a value computed from the total number of won, lost and drawn contests since the pattern was first seen. The strength of Patsy's comment on each legal next move is a function of the values of the patterns created and destroyed by the move in the state to which it leads. Thus Patsy encourages moves that lead to states introducing patterns associated with a win or a draw for the mover, while

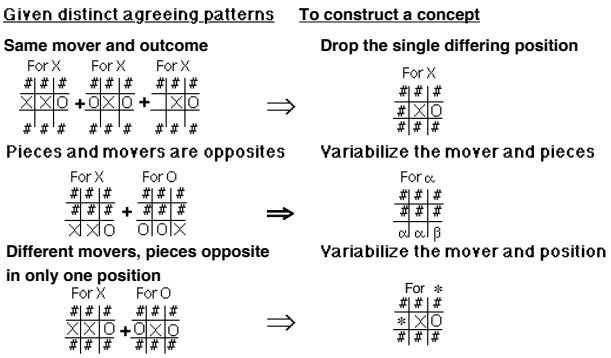

Figure 4: Generalizing patterns into spatial concepts.

it discourages moves that lead to states introducing patterns associated with a loss.

Generalization summarizes a set of detailed experiences into a more useful and efficient representation. After every 10 learning contests, three generalization processes sweep the pattern cache to form spatial concepts. Patterns *agree* when they originate from the same template. As shown in the examples of Figure 4, one generalization drops a position, a second variabilizes the mover and all pieces, and a third variabilizes the mover and a single piece. During generalization, only patterns with compatible associations may be combined. For example, a pattern must be associated with both a win for X and with a win for O to be generalized as a win for the mover α. In addition, if a new concept subsumes one that already exists, the more specific is eliminated. Each concept is proceduralized as a tier-2, game-specific, learned spatial Advisor.

In the experiments that follow, we shall see that, despite the care taken in Figure 3, learned spatial Advisors based upon play experience do not always provide correct guidance. As Hoyle's skill develops further and the learned spatial Advisors are introduced into tier 2, some of them prove irrelevant, self-contradictory, or untrustworthy, despite prior empirical evidence of their validity. To support their smooth integration into tier 2, the weights of learned spatial Advisors are initially discounted by an additional multiplier. This factor begins at 0.1 and reaches 1.0 after the learned spatial Advisor comments appropriately 10 times.

4. Results

Hoyle now learns pattern associations and game-specific spatial Advisors while it plays tic-tac-toe and *lose tic-tac-toe* (played like tic-tac-toe but whoever gets three in a row, column, or diagonal first loses). Because they both have the same board, they begin with the same templates. Tic-tac-toe is extremely easy for Hoyle to learn well, and we expected no improvement; it was present only to demonstrate that weights, patterns, and learned spatial Advisors were game-board independent. Lose tic-tac-toe is a far more difficult game to learn to play well, both for humans and for machines (Ratterman & Epstein 1995). It has been solved mathematically (Cohen 1972) and the correct strategies for the two contestants are different. Thus it forces the program to distinguish between patterns and concepts good for one contestant and those good for both. Both are *draw games,* that is, play between two contestants that make no errors must, by the nature of the game graph, end in a draw.

A *run* here is learning followed by testing. On each run, Hoyle alternately moved first in one contest and second in the next. During learning, the other contestant was a hand-coded *perfect player* for the game being learned, one that always moved to secure the best possible outcome despite subsequent error-free play by the opposition, and chose from among equally good moves at random. After learning,

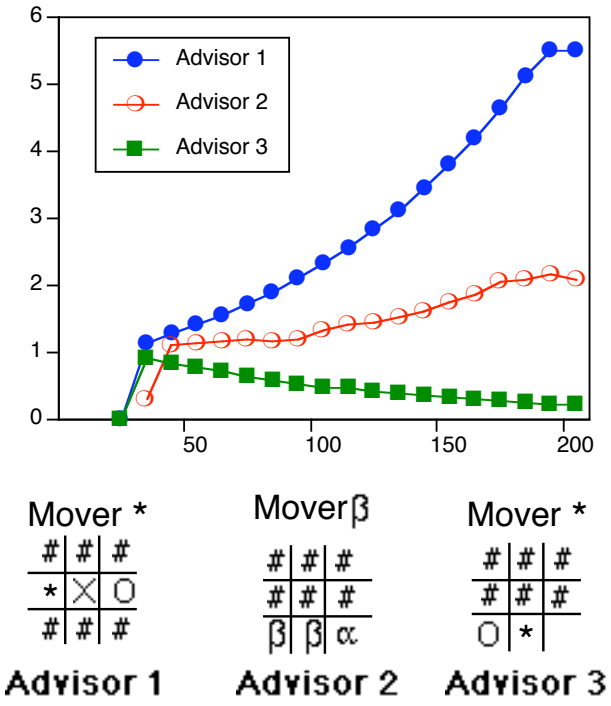

Figure 5: Three learned spatial Advisors for lose tic-tac-toe, and their weights during 200 consecutive contests. The mover for each Advisor is in the current state; the pattern is matched for in the subsequent state In an α–β Advisor, either α = X and β = O or α = O and β = X. In an Advisor with mover *, * is either X or O consistently.

Hoyle was tested against four *challengers*: the same perfect player, a 10% random and 90% perfect player, a 70% random and 30% perfect player, and a 100% random player. Because Hoyle is non-deterministic (it breaks ties in tier-2 voting by selecting a top-ranked move at random), data is averaged over 10 runs to form a single *experiment*. Each experiment was run once with a version of Hoyle that included Patsy but did not learn new spatial Advisors as heuristics, and a second time where it did learn heuristics. Figure 3. Improvements cited here are statistically significant at the 95% confidence level. We also, for reasons described in the next section, ran one additional experiment for each game.

Hoyle learns to value *pattern-oriented* play (i.e., Patsy and the learned spatial Advisors) highly. After learning in 200 tic-tac-toe contests, 32.6% of all weight is assigned to Advisors that are pattern-oriented, and Patsy and the best learned spatial Advisor always have the top two weights. In lose tic-tac-toe, 29.3% of all weight is assigned to Advisors that are pattern-oriented, and Patsy ranks second on all but one run, where it ranks third. On 80% of the runs, Hoyle learned at least one spatial Advisor for lose tic-tac-toe with weight at least one, and that Advisor ranked fifth on average. Learning new spatially-oriented heuristics also reduces the number of contests Hoyle requires to develop consistent expert performance. With patterns and the learned spatial Advisors, the program never lost at lose tic-tac-toe during learning after contest 29.0, versus contest 56.0 without patterns and the learned spatial Advisors.

Because learned spatial Advisors are produced by induction, not all of them are correct. This is especially true during early learning experience, when Hoyle is not yet playing well enough to exploit good pattern knowledge. Figure 5 shows how AWL adjusted the weights of three learned spatial Advisors for lose tic-tac-toe, based on their performance in one run of 200 contests. Advisor 1 is the horizontal and (through symmetry) vertical portion of the heuristic "reflect through the center," proved optimal play for most situations in the game (Cohen, 1972). The weight of Advisor 1 increases rapidly after its creation. Advisor 2 advocates playing in a row that does not go through the center, where each contestant already has one piece and the non-mover has a corner. Advisor 2 recommends a correct but infrequently applicable action, and its weight increases moderately. Advisor 3 recommends a move into a side row between an O and a blank. The weight of Advisor 3 initially increases but then falls off rapidly as Hoyle finds it misleading and discounts it on the basis of further experience.

Table 2: Average performance of versions with and without new learned heuristics against four challengers. Estimated optima are in italics. Boldface is represents an improvement over Hoyle without new heuristics at the 95% confidence level.

	Perfect Player		**90% Perfect**		**30% Perfect**		**Random Player**	
Decision maker	**Wins+Draws**	**Wins**	**Wins+Draws**	**Wins**	**Wins+Draws**	**Wins**	**Wins+Draws**	**Wins**
Tic-tac-toe	*100.0*	—	*100.0*	*16.4*	*100.0*	*80.7*	*100.0*	*93.6*
Without new heuristics	100.0	—	98.0	15.0	98.0	75.0	100.0	97.0
With new heuristics	100.0	—	97.0	13.0	94.0	77.0	98.0	93.0
Broader context and weight ≥ 1 only	100.0	—	**100.0**	**22.0**	**99.0**	**85.0**	98.0	94.0
Lose tic-tac-toe	*100.0*		*100.0*	*18.5*	*100.0*	*66.4*	*100.0*	*74.3*
Without new heuristics	100.0	—	97.0	19.0	87.0	56.0	77.0	55.0
With new heuristics	100.0	—	98.0	18.0	**92.0**	49.0	**91.0**	**66.0**
Weight ≥ 2 only	100.0	—	**99.0**	18.0	**96.0**	**68.0**	**93.0**	**68.0**

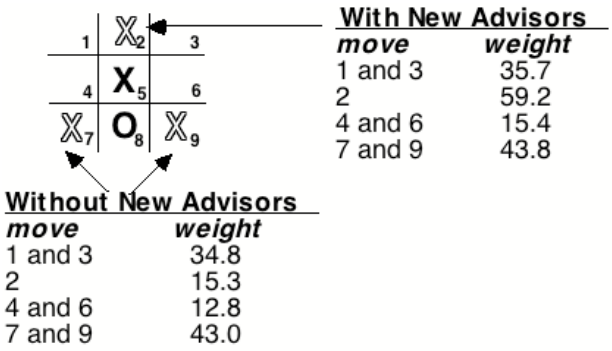

Figure 6: A learned spatial Advisor affects decision making in lose tic-tac-toe.

Table 2 shows learning to satisfice can change decision making, where improved performance over Hoyle without new heuristics is in boldface. When Hoyle learns with patterns and spatial Advisors, it improves its performance in lose tic-tac-toe against the less perfect challengers, even though it is forced to play from board states unseen during learning. In tic-tac-toe, its performance actually degrades somewhat, for reasons discussed in the next section.

5. Discussion

Two items of particular interest in these experiments are Hoyle's performance against different opponents and its response in novel situations. Recall that Hoyle trains against a perfect player. This results in stereotyped movement through the game tree (Epstein 1994b). During testing, the challengers with varying degrees of randomness in their move selection lead to states provably never experienced by Hoyle during learning.

For lose tic-tac-toe, Table 2 demonstrates dramatic improvement when Hoyle learns new heuristics (its learned spatial Advisors). The secret here, we believe, is to carefully screen potential heuristics (as in Figure 3) and then phase them in gradually (as AWL does). Figure 6 demonstrates how spatially-oriented play can improve behavior in novel situations. The position shown from lose tic-tac-toe never occurs during learning against the perfect player when Hoyle is X, but arises often in testing against the other three challengers. Without the learned spatial Advisors, Hoyle typically votes to move into a corner adjacent to O's move. Advisor 1 of Figure 5, however, though it has never experienced the specific instance of reflection through the center required here, swings the vote to do so because of the generalization used in its concept formation algorithm. Thus Hoyle can make correct decisions against the imperfect challengers in situations it has never seen during training. It parlays inductive spatial generalizations into game playing expertise.

When new heuristics are learned inductively, there are several dangers. First, they can slow the system. Indeed, Table 3 shows the decision making time during testing lose tic-tac-toe for Hoyle without patterns and Hoyle with patterns and new learned spatial Advisors. The price of greater accuracy appears to be speed. The introduction of new spatial heuristics increases computation time. We therefore also ran a *high-weight* lose tic-tac-toe experiment, where Hoyle learned new spatial Advisors as before, but during testing only Advisors with weight at least 2 were permitted to comment. This not only reduced computation time but also improved performance against the 30% perfect challenger, as indicated in Table 2.

The second danger is that during introduction of new heuristics, performance will temporarily degrade while the system adjusts to their presence. There was no evidence of this in these runs. Hoyle did not lose more often during learning at the times new spatial Advisors were introduced. We attribute this smooth integration to AWL's gradual phase in for newly learned Advisors.

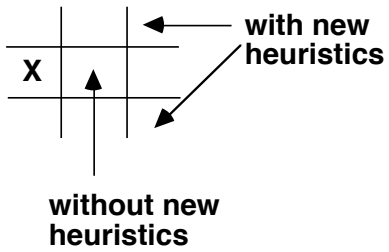

Figure 7: Voting for O's first tic-tac-toe move, Patsy and a learned spatial Advisor incorrectly select two corners.

The third danger in learning heuristics, as Table 2 shows for tic-tac-toe, is that performance might actually degrade. Inspection revealed that Hoyle overgeneralized on tic-tac-toe. On every run Hoyle learned only a single new spatial Advisor that ever commented: "move into the corner of an empty row or column." Let us call this Advisor *Corner*. On every run Patsy had the highest weight, and Corner had the second highest. Corner actually degraded Hoyle's reliability against the imperfect challengers. Inspection revealed that Hoyle's new errors all stemmed from Corner's incorrect response playing O to a side-square opening by X shown in Figure 7. Because the perfect player always opens in the center or a corner, Hoyle had never encountered this opening during learning. Without Patsy and Corner, Hoyle correctly plays the center. Patsy,

Table 3: Decision time in seconds per contest for 3 versions of Hoyle against 4 challengers in lose tic-tac-toe.

	Challenger plays perfectly			
Decision maker	**100%**	**90%**	**30%**	**0%**
Without new heuristics	.84	.83	.70	.71
With new heuristics	1.33	1.33	1.23	1.19
Weight ≥ 2 only	1.10	1.06	.95	.94

however, learned a specific pattern in the context of competition against a perfect player, a pattern that does not always apply when the competition is not perfectly expert. Corner suffers from a similar lack of context. Together, every time an imperfect challenger opens in a side square, Patsy and Corner swing the vote in tier 2 to a corner not adjacent to X, a fatal error. (X will play the corner between its first move and O. A sequence of forced moves then guarantees a win for X.)

The use of context should correct such problems with learned heuristics. In this case, the strength of a pattern is in large part determined by the environment in which it is experienced. Hoyle's pattern induction did detect an important tic-tac-toe pattern, *but only for play against a perfect player*. In other words, the induction process overgeneralized to say "corners in empty rows or columns are good," whereas the more accurate statement would be "if someone already holds the center, corners in empty rows or columns are good." Thus heuristic learning is demonstrably context-dependent. When the context changes, the automatic response to a heuristic (in this case playing the corner of an open column) may be wrong.

We believe that the benefits of learning heuristics justify the cost of overlooking an occasional context. Even Corner gives generally good advice. Rather than keep a list of caveats, our preference is to broaden the context in which the pattern is learned, that is, to learn against all the challengers. The extra experiment we ran for tic-tac-toe had Hoyle train equally against each of the challengers and was high-weight, with weights at least 1. (The minimum weight was 1 instead of 2, as it was for the lose tic-tac-toe run, because there were fewer opportunities to adjust the weights. Recall, AWL models only a perfect player.) This time the error in Figure 7 never occurred, and Hoyle's performance actually improved against the 30% perfect challenger. Hoyle simply needed additional knowledge to prevent the misuse of the new spatial Advisor; knowledge it acquired with broader training.

We have begun to test this approach on five men's morris to explore scaling up. This game's board has 16 positions, contests average about 50 moves, and there are approximately 9 million states in its game graph. Results appear similar. Despite the substantial increase in scale, pattern-oriented play improves Hoyle's ability to win and draw against the challengers, and the learned spatial Advisors are often salient subgoals. For example, five men's morris is lost by a contestant who has too few pieces or cannot slide. Hoyle regularly learns and weights highly two spatial Advisors that enable capture, and a third that traps opposition pieces so they cannot slide, both keys to winning the game.

6. Related Work

Any architecture that autonomously acquires skill in a specific problem class should be able to perform tasks there while learning. For a system to learn through experience, it must be able to perform at some low level of competence that supports the kind of experience required to achieve a higher level of performance through practice. Many cognitive models (e.g., SOAR) have such ability, and can perform tasks in a serviceable fashion while mechanisms such as chunking operate to improve performance with experience (Newell 1990). These systems, however, typically have a complete set of knowledge for the particular problem class, as well as a knowledge representation initially optimized for it. In contrast, the architecture described here retains its capacity for general baseline performance, while it introduces an additional representation that specializes its behavior to a particular problem class, in this case an individual game.

We do not claim that Hoyle is a full cognitive model, but it does contain many appropriate elements which enhance its performance as an autonomous learning program. For example, the templates that filter perceived patterns and curtail a potential combinatoric explosion are not an ad hoc device, but are inherent in the human perceptual system (Goldstein 1989). FORR's approach is supported by evidence that people integrate a variety of strategies in order to accomplish problem solving (Biswas, Goldman, Fisher, Bhuva, & Glewwe 1995; Crowley & Siegler 1993; Ratterman, & Epstein 1995). Hoyle's use of these patterns was inspired by repeated laboratory experiences with people who relied upon familiar, sometimes symmetrically transposed patterns while learning (Ratterman, & Epstein 1995).

7. Conclusion

The experiments described here directly address the issues raised in the introduction. Three aspects in this work constitute satisficing as resource-bounded reasoning:

- Design to curtail search in tier 1, and subsequent calculation with heuristics in tier 2 as necessary.
- Reliance in tier 2 on knowledge not guaranteed to be correct, typically because it is learned by induction.
- Elimination of low-weight Advisors (during testing).

FORR's hierarchy integrates correct and satisficing rationales in a constructive and natural way, so that obvious decisions are quick, and egregious errors are avoided. Search is minimized by the rationales' design. AWL supports the gradual introduction of learned heuristics so that the system remains robust and yet is able to improve. Finally, context notwithstanding, the cost of additional procedural knowledge can be controlled by allocating resources only to those satisficing heuristics that prove reliable. We conclude that the role of inductive learning is to present well-founded new heuristics and then force them to prove their worth in an agent's ongoing problem solving experience.

Acknowledgments

We acknowledge helpful discussions with Phil Johnson-Laird, Ron Kinchla, and Anne Treisman. This work was supported in part by NSF grant #9423085, PSC-CUNY #666318, the NSF Center for Digital Video and Media, and the New Jersey Center for Multimedia Research.

References

Biswas, G., Goldman, S., Fisher, D., Bhuva, B. and Glewwe, G. (1995). Assessing Design Activity in Complex CMOS Circuit Design. In P. Nichols, S. Chipman, & R. Brennan (Ed.), *Cognitively Diagnostic Assessment* Hillsdale, NJ: Lawrence Erlbaum.

Cohen, D. I. A. 1972. The Solution of a Simple Game. *Mathematics Magazine*, 45 (4): 213-216.

Crowley, K. and Siegler, R. S. 1993. Flexible Strategy Use in Young Children's Tic-Tac-Toe. *Cognitive Science*, 17 (4): 531-561.

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

Epstein, S. L. 1994a. For the Right Reasons: The FORR Architecture for Learning in a Skill Domain. *Cognitive Science*, 18 (3): 479-511.

Epstein, S. L. 1994b. Toward an Ideal Trainer. *Machine Learning*, 15 (3): 251-277.

Epstein, S. L. 1998. Pragmatic Navigation: Reactivity, Heuristics, and Search. *Artificial Intelligence*, to appear :

Epstein, S. L., Gelfand, J. and Lesniak, J. 1996. Pattern-Based Learning and Spatially-Oriented Concept Formation with a Multi-Agent, Decision-Making Expert. *Computational Intelligence*, 12 (1): 199-221.

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

Goldstein, E. 1989. *Sensation and Perception* (third ed.). Belmont, CA: Wadsworth.

Littlestone, N. 1988. Learning Quickly when Irrelevant Attributes Abound: A New Linear-threshold Algorithm. *Machine Learning*, 2 : 285-318.

Newell, A. 1990. *Unified Theories of Cognition* . Cambridge, MA: Harvard University Press.

Ratterman, M. J. and Epstein, S. L. 1995. Skilled like a Person: A Comparison of Human and Computer Game Playing. In *Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society*, 709-714. Pittsburgh: Lawrence Erlbaum Associates.

Simon, H. A. 1981. *The Sciences of the Artificial* (second ed.). Cambridge, MA: MIT Press.

Tversky, B. 1989. Parts, Partonomies, and Taxonomies. *Developmental Psychology*, 25 (6): 983-995.