This paper appeared in the 1994 AAAI Fall Symposium on Relevance.

Identifying the Right Reasons: Learning to Filter Decision Makers

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

Abstract

Given a domain of related problem classes and a set of general decision-making procedures applicable to them, this paper describes AWL, an algorithm to filter out those procedures that prove irrelevant, self-contradictory, or untrustworthy for a particular class. With an external model of expertise as its performance criterion, the algorithm uses a perceptron-like model to learn problem-class-specific weights for its decision-making procedures. Learning improves both the efficiency of the decision-making process and the performance of the system.

1. The Reasoning Framework

The problem-solving and learning architecture called FORR (FOrr the Right Reasons) relies upon a set of *Advisors*, general purpose, heuristic rationales that make decisions across a set of related problem classes (Epstein 1994). This paper describes an algorithm that learns the relevance of each Advisor to any particular problem class. In this context, *relevance* is the usefulness of a general-purpose heuristic for decision making in a specific problem class.

The thesis behind FORR is that a "general expert" in a domain can become a "specific expert" for some problem class in that domain by learning problem-class-specific data (*useful knowledge)* that is potentially applicable and probably correct. As it combines powerful heuristics, FORR simulates a synergy among good reasons as a mental model for decision making.

A background theory for a general domain is defined in FORR by a *problem frame* that delineates the nature of a problem class, a *behavioral script* that represents how an expert proceeds in the domain, a *useful knowledge frame* that identifies what can be learned about a problem class, and a set of "right reasons," heuristic procedures (Advisors) that represent reasonable arguments for decision making throughout the domain. One or more learning methods is attached to each useful knowledge slot and triggered by the behavioral script. The same domain-specific Advisors, learning methods, and behavioral script are used on every problem class; problem-specific information is represented only in the data that initialize the problem frame and the data learned to instantiate the useful knowledge frame.

Although several FORR-based programs are in development, the most accomplished is Hoyle, for the domain of two-person, perfect information, finite board games (Epstein 1992). In Hoyle, a problem class is a game, such as chess or tic-tac-toe. Each game is defined by a different instantiation of the problem frame that describes the board, the playing pieces, and the rules. The behavioral script, for all games, describes how contestants take turns, make legal moves, and so on. The useful knowledge frame holds data worth learning, such as average contest length or good openings. Hoyle has 23 game-independent Advisors. These include one that concerns mobility, another that considers material (number of pieces on the board), and another that monitors openings.

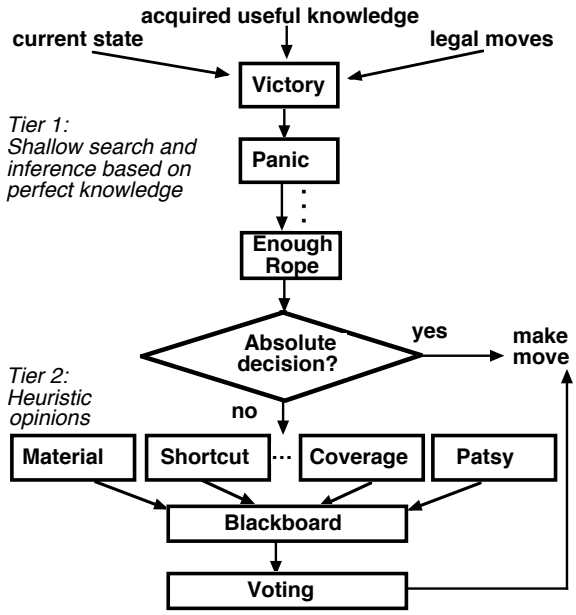

Figure 1: How Hoyle makes decisions.

When the behavioral script indicates that a decision is to be made, Advisors receive as input the current state of the world, the legal actions from that state, and whatever useful knowledge is available about the current problem class. (For Hoyle, a decision must be made when it is the program's turn. Input is the current game state, the legal moves from it, and the useful knowledge frame for the game.) Each Advisor then outputs any number of

comments that support or discourage an action. A comment lists the Advisor's name, the action commented upon, and a *strength*, an integer from 0 to 10 that measures the intensity and direction of the Advisor's opinion.

As the schematic of Figure 1 indicates, all Advisors are not created equal. Some are always perfectly correct, take precedence over others, and may have the authority to make a decision unilaterally or to eliminate a legal action from any further consideration. Initially, Hoyle always attempts decisions in its *first tier* of seven such Advisors. These include one that moves to win immediately, and another that refuses to make a losing move. Only when the first tier fails to make a decision does control default to the 17 Advisors in Hoyle's *second tier*.

Second-tier Advisors are not necessarily independent, or even correct in the full context of the state space. Each of them epitomizes a heuristic, specialized view of reality that can make a valid argument for or against one or more actions. All of them have a resource-limited opportunity to comment before any decision is made. It is here that relevance, internal consistency, and trustworthiness become issues, and that the *AWL* (Advisor Weight Learning)
algorithm is based.

A FORR-based program improves its performance within a specific problem class as it learns useful knowledge and then applies it as input to the Advisors. Advisors do not learn, but they do make better decisions when their input improves. Since they epitomize many different ways to reason in a broad domain, however, the Advisors may not always provide constructive information within a specific problem class. What FORR did not do, prior to this paper, is learn which Advisors to consult when.

As we have challenged Hoyle with increasingly more difficult games, we have incorporated several new Advisors. Although these Advisors were essential to Hoyle's ability to learn to play some of these new games well, they were also irrelevant to some of the early games Hoyle had already learned. Tests indicated that, while these new Advisors did not degrade Hoyle's ability to learn to play extremely well, they did slow its decision-making time when they were given the opportunity to comment. In the newer and more difficult games, more complex issues arose. Some Advisors regularly contradicted themselves, and others were actually a hindrance to learning, i.e., Hoyle could learn to play expertly only if they were not present. Hoyle needed to learn about relevance.

2. Learning about Relevance

Prior to AWL, a decision in the second tier was made by tallying the Advisors' comments on each legal action with their strengths. The action made was the one with the strongest support:

$$\max_i \left(\sum_A \text{comment strength from Advisor A on action i} \right)$$

For example, given comments:

< Advisor-1, action-1, 6 >
< Advisor-1, action-2, 8 >
< Advisor-2, action-1, 7 >
< Advisor-3, action-3, 8 >

action-2 and action-3 each only have support 8, while action-1 has support 13 and will be executed. This is because the comments have strengths (6, 8, 7, 8, respectively) but the Advisors do not have *weights*, i.e., an 8 from one Advisor is just like an 8 from another. In the preceding example, however, if Advisor-3 had weight 2 and the others had weight 1, then action-3 would have had support 16 and have been executed instead. AWL, the algorithm that learns weights for second-tier Advisors, was devised to exploit empirical evidence that the accuracy of each second-tier Advisor indeed varied with the game.

For each game, AWL is expected, based upon experience, gradually to reduce the weights of irrelevant and self-contradictory Advisors to 0, and to increase the weights of the more reliable Advisors beyond those of the less reliable ones. AWL executes after every contest Hoyle plays against an external (human or computer) expert. The algorithm considers, one at a time, only those states in which it was the expert's turn to move and Hoyle's first tier would not have made a decision. For each such state, AWL tallies the first valid statement among the following *f values*:

f_1: the Advisor supports the expert's recorded move

f_2: the Advisor does not support the expert's move and supports another move

f_3: the Advisor opposes the expert's move

f_4: the Advisor does not oppose the expert's move and opposes another move

Essentially, AWL fits Hoyle to the expert, learning to what extent each of its Advisors simulates expertise as exemplified by the expert's moves. The learned weights are a modification of Littlestone's perceptron-like algorithm (Littlestone 1988). The weight for Advisor i is

$$\alpha_i^{c_{ij}} \beta_i^{c'_{ij}} \text{ where } \alpha_i = \frac{1}{\beta_i}, \; c_{ij} = f_1, \text{ and } c'_{ij} = f_3$$

If weights were updated during the play of the contest, j would be the state. Because Hoyle plays in real time and updating the weights during play would slow it down considerably, we chose instead to "massively update" the weights during the postmortem at the end of each contest where much of the other learning occurs.

3. Applying the Learned Weights

The Advisors' weights serve as the criterion for adapting the decision-making process in three different ways.

• *Weights identify irrelevant Advisors*. Any Advisor whose f values all remain at 0 has made no comments at

all and, after some time, can be eliminated. For example, the Advisor that calculates strength based upon how many pieces each contestant has on the board is irrelevant in tic-tac-toe.

• *Weights identify self-contradictory Advisors*. In some games, the same moves consistently and simultaneously look both good and bad to an Advisor. For example, in tic-tac-toe one Advisor will support every legal move because it reduces the number of options for the other contestant on its next turn, and oppose the same move with the same strength because it reduces Hoyle's own options on its next turn. (In other games this Advisor behaves quite differently and is extremely constructive.) When $f_1 = f_3$, an Advisor is self-contradictory and, after some time, can be eliminated.

• *Weights identify untrustworthy Advisors*. Although some weights will "look better" than others, AWL uses an absolute standard rather than a relative one. For this purpose, we introduced a dummy advisor called *Anything* that makes comments with random strengths. Because some Advisors make more comments than others and Anything is intended as a general benchmark, Anything makes $i > 0$ comments for each state with probability 2^{-i}. The Advisor Anything serves as benchmark for random opinion; it is not consulted when Hoyle plays, only when the algorithm learns weights after the contest is over. An Advisor that consistently underperforms Anything is untrustworthy and, after some time, can be eliminated.

4. Experimental Design

AWL has been tested with Hoyle on tic-tac-toe, lose tic-tac-toe (played exactly like tic-tac-toe except that the first contestant to achieve three of the same playing piece along a row, column, or diagonal *loses*), and some morris games. A morris game has two contestants, black and white, each with an equal number of playing pieces. The morris boards vary from one game to the next; they are characterized by concentric squares connected with intersecting lines. Playing pieces can be located at the intersection of any pair of lines. In five men's morris, for example, each contestant has five playing pieces and there are 16 possible locations for the playing pieces. Each morris contest has two stages: a *placing stage*, where initially the board is empty, and the contestants alternate placing one of their playing pieces on any empty position, and a *sliding stage,* where a turn consists of sliding one's playing piece along any line drawn on the game board to an immediately adjacent empty position. Morris games offer substantial challenges: five men's morris has about nine million states in its search space, nine men's about 143 billion.

For the AWL runs described here, Hoyle learned to play a specific game against an external expert until it achieved 10 consecutive draws. At that point learning was turned off, and Hoyle was tested in a tournament of 20 contests against each of four *challengers:* a perfect player, an expert, a novice, and a random player at the same game. The same α value was used for all Advisors. Because the external expert chooses against equally valid moves at random, and so does Hoyle's second tier, no two learning experiences are identical. Therefore results are monitored over a set of runs.

5. Results

Several values were tried for α on tic-tac-toe, lose tic-tac-toe, and five men's morris: 1.2, 1.1, 1.05, 1.02, and finally 1.01. The larger values narrowed Hoyle's learning experience too fast; during testing the program then performed less well against the weaker challengers than without weight learning. This is because, with larger α's, the weights were fitted so quickly that Hoyle had no time to acquire other items of useful knowledge that required more playing experience but would stand it in good stead against weaker competition during testing.

AWL offered the following advantages in learning:

• *Hoyle's reaction time improved* when irrelevant Advisors were eliminated, without any increase in the number of contests required to learn or any degradation of the program's ability to play. Hoyle with AWL learns to play tic-tac-toe in just as few contests and learns to play just as well as without AWL. With AWL, however, the program also eliminates many Advisors it would otherwise consult, so that Hoyle with AWL now plays perfect tic-tac-toe more quickly.

• *Hoyle learned some games faster and more reliably,* losing less frequently to the less skilled challengers. Lose tic-tac-toe has a fairly small state space (5478) but is non-trivial for Hoyle to learn. With AWL, Hoyle not only learns to play the game faster, it also learns to play it more reliably, losing less frequently to the less skilled challengers.

• *Hoyle could identify and eliminate Advisors that made learning impossible* (by offering powerful but incorrect advice) without forcing them to be discarded for other games. Although an Advisor called Shortcut is necessary to learn nine men's morris, Shortcut makes learning a somewhat easier game, five men's morris, impossible, i.e., the runs do not achieve 10 consecutive draws even after several hundred contests. With AWL, however, after 10 contests Hoyle is able to identify Shortcut as worse than Anything, eliminate it, and go on to learn to play the game extremely well. In nine men's morris, Hoyle with AWL does not eliminate Shortcut and goes on to learn to play that quite well too.

6. Discussion

The phrase "after some time [an Advisor] can be eliminated" runs throughout Section 3. For now, Hoyle pauses every 10 contests to report its weights and to ask permission to eliminate Advisors because they are self-contradictory or untrustworthy. This is an anomaly in a

previously-autonomous discovery program, but when to take dramatic action based upon the weights themselves is a real concern. Usually Hoyle's judgments are accurate, but occasionally an Advisor is prematurely labeled "untrustworthy" and, after more experience, Hoyle would not have attempted to eliminate it. (One example is Hoyle's occasional overeagerness in lose tic-tac-toe to discard the Advisor that considers openings because of what appears to be initially poor performance. The real problem is that the opening Advisor needs more useful knowledge than has yet been acquired; once that knowledge is in place it performs quite well.) For now, we have selectively refused permission to discard Advisors, and have observed that Hoyle eventually chooses to retain the right ones. If we increase the parameter 10, however, many irrelevant Advisors will continue to slow the real time devoted to play, and some incorrect Advisors may continue to delay learning. We are now exploring a variety of methods to establish the correct balance between efficiency and overeagerness.

For Hoyle, AWL actually uses two new useful knowledge slots for each game: one for weights in the placing stage and one for the weights in the sliding stage. In this domain the change in the rules makes the stage boundary obvious. In another domain, or even in more complex games, there might be more than two kinds of problem solving (here, the placing and sliding stages) that merit individually tabulated sets of weights. The weights identify how the Advisors' performance differs between stages, but as yet we have no code to detect such stages, and believe that to be a non-trivial task.

In more difficult domains or problem classes, the external expert will not perform perfectly. (There is, for example, no perfect-playing expert program for nine men's morris.) Weight-learning there would attempt to fit Hoyle's decision making to that of an imperfect, although strong, expert, with unpredictable results. And how would one learn weights in a domain without an expert model? If Hoyle were to learn weights while playing against itself, for example, the early contests would be entirely untrustworthy. (When one poor player beats another, there is likely to be little worth imitating.) Perhaps α should be treated like the temperature in simulated annealing, but increase over time or with improved performance, to reflect the algorithm's increasing confidence in its own judgment.

A recent application of AWL is for the validation of pattern-based Advisors in a new, game-dependent third tier (Epstein, Gelfand, & Lesniak submitted). These Advisors are generalizations of persistent patterns with a single, consistent association. Although they are based upon experience in play, they are inductive guesses whose reliability and relevance merit close examination. AWL has proved an able filter for them. It strengthens the most important, eliminates the irrelevant, and disregards the incorrect ones.

We continue to refine AWL. The f_2 and f_4 values should be put to use. The weights also provide us with a clear metric on Advisor performance that could be helpful in the design and testing of new Advisors. Finally, AWL is under study with FORR-based programs in additional domains.

7. Conclusion

Advisors in FORR are broad, general theories about how to perform well in a domain of related problem classes. In any specific problem class, some will prove more relevant than others. Advisor weights are only one item of useful knowledge, however. It is important to allow the program enough time to learn other items that serve as input to the Advisors and allow them to make a strong contribution. With a good choice for α, the AWL algorithm speeds and improves the quality of FORR-based learning. Given a set of good reasons for making a decision in a broad domain, AWL quickly detects those that are irrelevant, self-contradictory, or untrustworthy in a segment of the domain. When the decision makers are filtered this way, a FORR-based program can learn the relevance of its heuristics and learn to perform more efficiently and more accurately.

References

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

Epstein, S. L., Gelfand, J. & Lesniak, J. (Submitted). The Integration of Pattern-Based Learning and Spatially-Oriented Reasoning with a Multi-Agent, Decision-Making Expert.

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