Learning Plans for Competitive Domains

Susan L. Epstein*
Department of Computer Science
Hunter College of the City University of New York
New York NY 10021

Abstract

Many machine learning systems learn from their problem solving experience in single-agent domains. This paper discusses an algorithm to learn complex, relevant, cost effective plans for a broad class of competitive, multi-agent domains. Such a plan, called a fork, is extracted from the explanation of a single failure, and represents a set of mutually overlapping simple plans to achieve a goal. Each plan is non-linear, provides alternatives for contingencies, is applicable both offensively and defensively, and has a clear upper bound for its execution time. This paper describes the representation and implementation of forks in HOYLE, a system to learn to play any two-person, perfect information game well. Together with a weak theory for its general domain, HOYLE has used forks to learn to play a broad variety of games perfectly without extensive forward search in the game tree. In more challenging games, however, the selection of a relevant plan and its binding to the current game state is unacceptably costly. This paper details the significantly improved performance directly attributable to learning about appropriate forks, and the heuristics that guard against unacceptable degradation of performance as new knowledge is acquired.

1. Introduction

From their problem solving experiences, many machine learning programs extract and reformulate information that can improve their performance in single-agent domains. Many of these programs (e.g., Fikes, Hart, & Nilsson, 1972; Korf, 1985; Laird, Rosenbloom, & Newell, 1986) have learned macro-operators for planning. Others (e.g., Langley, 1985; Minton, 1988; Mitchell, Utgoff, & Banerji, 1983) have learned heuristic rules to improve their search performance in planning domains. Recent research in explanation-based learning (e.g., Minton, 1988; Mooney, 1988) has demonstrated how one problem-solving example can provide relevant, cost effective knowledge within a broad domain. In a domain with more than one agent, however, credit and blame assignment is more difficult, the next choice point for each agent is not absolutely predictable, and the absolute merit of goal states and paths to them is difficult to assess. There has been little work on learning useful plans for broad domains with more than one agent. Minton (1984) described an algorithm that learned recognition rules for plans in a two-agent domain. These rules, however, were limited to chess and slowed the system "dramatically."

This paper describes an algorithm to learn a class of extremely powerful plans in a domain where two or more adversarial agents perform some sequence of unretractable, possibly interfering actions in a race to achieve some goal. The algorithm has been implemented within HOYLE, a machine learning program for two-person, perfect information games (Epstein, 1989a). The plans are called forks; they are game-independent, partially ordered, and applicable both offensively and defensively. Each plan provides alternatives for contingencies and has a known upper bound on its execution time. Together with a weak theory for its general domain, HOYLE has previously used such preselected forks to learn to play a broad variety of games perfectly without extensive forward search in the game tree. In more challenging games, however, the selection of a relevant plan and its binding to the current game state is unacceptably costly. The algorithm discussed here enables HOYLE to learn a relevant plan as an explanation of a single losing experience, and to improve its performance significantly. Heuristics guard against unacceptable degradation of performance as new knowledge is acquired, and support the correct offensive and defensive application of these plans.

2. An Overview of HOYLE

HOYLE is a system that learns to play a broad class of two-person, perfect information games extremely well.

*This work was supported in part by NSF DCR-8514395 and PSC-CUNY 668287 and 666397.

A *two-person, perfect information game* is one in which all information about the game is disclosed and equally available to its two participants, e.g., there are no closed hands and no uncertain outcomes as with dice. Any such game can 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 board and playing pieces). 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. During a *contest* (a path in a game graph that represents one complete experience of the game), Player tries to arrive at a terminal node categorized as a win and Opponent tries to arrive at a terminal node categorized as 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 result, while minimizing the non-mover's chances to arrive at her own desired result. *True expertise* at a game is perfectly backed up knowledge from its entire game graph. Most interesting games have extremely large game graphs; for them, true expertise through exhaustive search of the game graph is computationally intractable. Instead, computer game playing programs usually approximate true expertise by heuristic search in the game graph. From a given node, such a program estimates the unseen finishes of deeper intermediate nodes and backs up this approximate knowledge through the game graph to the given node to select the best possible legal move. Human experts can foil such programs by executing plans whose threat lies at a deeper level than the programs search.

HOYLE accepts as input a representation of some game: its material, interface, and rules. The interface provides the graphics for the game and interactive communication with another participant through the keyboard. The rules describe the legal moves, when a contest at the game terminates, and how to determine the outcome of that contest. From this input HOYLE is able to play any such game correctly, i.e., according to the rules; HOYLE's task is to learn to play it perfectly. Without perfect knowledge, the performance of a game-playing program must be evaluated empirically, by playing contests. HOYLE is evaluated against two criteria: how well it learns to play in a *tournament* (a sequence of contests where the participants alternate roles) against a human expert and how quickly it achieves that skill. Measures of skill include ability to win, ability to draw, and the duration of contests played, measured in average number of moves.

HOYLE is a limitedly rational program that learns from its experience. Compared to traditional one-game programs (e.g., Anantharaman, Campbell, & Hsu, 1988; Berliner, 1980; Berliner & Ebeling, 1989; Lee & Mahajan, 1988; Rosenbloom, 1982; Samuel, 1967; Schaeffer, 1988), HOYLE does relatively little forward search into the game tree, has a limited memory, and has no heuristics for symmetry. Instead, HOYLE makes decisions based on a variety of advice. HOYLE's thesis is that, in learning to play any new game, a person actually has relevant prior knowledge about it, a weak theory for the general domain of two-person, perfect information games. This weak theory encapsulates the person's experience from contests at other games, and enables her to learn to play any such game well. HOYLE's weak theory includes a set of valid, but deliberately narrow, viewpoints (HOYLE's *Advisors*) that are modified by HOYLE's experience, i.e., learn. When it is HOYLE's turn to move, each Advisor generates one or more comments about the current state. To select a move, HOYLE weighs the comments carefully against each other (Epstein, 1989b) according to the game, the contest, and the nature of the participants. Among HOYLE's Advisors is Pitchfork, a procedure that exploits certain very powerful plans, called forks. The way this Advisor learns the particular forks effective in a given game is the subject of this paper.

3. An Example of a Fork in Execution

Assume a domain where a finite set of actions is available, two or more agents alternately take a single action, and no action may be taken by more than one agent. A *simple plan* in such a domain is an unordered set of unretractable actions that achieve a goal. Informally, a fork is two or more overlapping simple plans belonging to the same agent, directed toward different goals, and having one or more common actions. (Although the implementation described here is for games and two agents, the algorithm is limited only by the definitions of simple plan and fork.) Figure 1 shows an example of a fork used in tic-tac-toe where Player's most recent move (action), in position 3, threatens a win

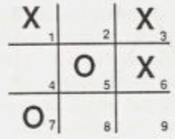

Figure 1: A Fork

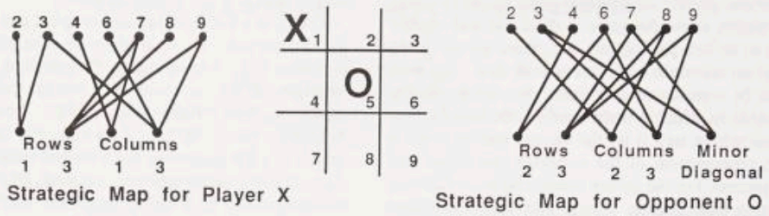

Figure 2: A Game State and Its Strategic Maps

in both the first row and the third column (the goals). Opponent is defenseless before this onslaught.

Banerji (1980) devised a graph representation for the mover's situation in a game focused upon certain patterns called "kernels." An early implementation of Banerji's ideas (Koffman, 1968; King, 1970) used a few simple kernels to suggest moves in several games, like tic-tac-toe. HOYLE has continued this research: it extends Banerji's original terminology, defines and explains the significance of a fork's depth, explores the generation and application of forks, and integrates forks with other techniques to support both *offensive* (goal-forwarding) and *defensive* (goal-inhibiting) play (Epstein, 1990).

4. Definition and Representation of Forks

In a task with only unretractable actions (actions from states that do not lie on a cycle in the search space), a strategic map for any state can be constructed for each agent. A *strategic map* for a state and agent is a labeled, undirected bipartite graph in which one set of nodes (the *top vertices*) is labeled with the actions available to that agent from that state, the second set (the *bottom vertices*) is labeled with the agent's simple plans to achieve her goal, and an edge joins any action available to an agent with each simple plan it forwards. Thus the strategic map for an agent at a state represents all the goal-forwarding options open to her. A strategic map may be denoted as <T,B,E>, where T is a set of top vertices, B a set of bottom vertices, and E a set of undirected edges from vertices in T to vertices in B. In Figure 2, 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 2 shows the strategic maps for Player and Opponent for the given board position.

Certain connected subgraphs of a strategic map, induced by a subset of its bottom vertices, represent the overlapping of more than one of the agent's simple plans. These subgraphs are denoted here as F-dn, where F stands for "fork," d is its depth, to be defined shortly, and n provides a distinct label. For example, in a subgraph of the form F-31 in Figure 3(a), either action represented by a degree-two vertex at the top furthers both the simple plans represented by the bottom vertices adjacent to it. Such an action, on the left for example, transforms F-31 into Figure 3(b), two connected graphs, of the forms F-1 on the left and F-21 on the right. (Immediately before her most recent move into position 3 in Figure 1, 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 taking the action represented by the second degree-two position transforms Figure 3(b) into Figure 3(c), three copies of F-1. Thus there are two actions represented in F-31 that, taken in any order, forward among them three simple

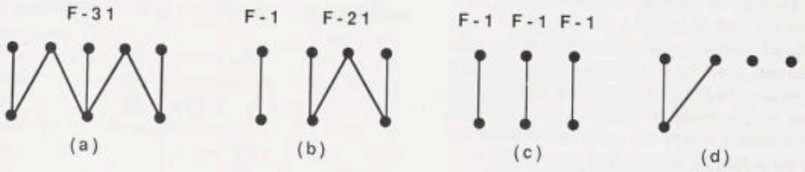

Figure 3: Applications of the Fork F-31

plans.

Formally, a *fork* is defined to be either F-1 = <{t},{b},{(t,b)}> with distinguished vertex (*key*) t, or an undirected bipartite graph containing at least one distinguished top vertex whose removal, along with its edges, disconnects the graph into two or more connected components, each of which is also a fork. The set of all forks is completely and correctly defined recursively as:

- F-1 is a fork.
- The following construction produces a fork: Copy any n forks. (They are the *parents* of the new fork under construction.) Add a new top vertex v. (This is the *join* of the new fork under construction.) Add an edge from v to at least one bottom vertex in each parent.
- Nothing else is a fork.

Observe that removal of the join and its associated edges transforms the graph into n connected graphs, its parents, each of which is itself a fork. F-21, for example, is the result of joining two copies of F-1. The removal of a key and its associated edges is called an *execution* of the fork.

The fork F-1 has depth 1, and for d >1, a *fork of depth d* is a fork such that removal of any one of its distinguished vertices produces at least one connected component that is itself a fork of depth d-1, and d is maximal. By this definition, for example, F-21 is a fork of depth two and F-31 is a fork of depth three. When some subset of the bottom vertices of an agent's strategic map induces a copy of a fork, the depth of that fork may be interpreted as an upper bound on the number of actions until the completion of one of those goals by the agent. For example, if F-31 is such an induced subgraph, the agent is certain to achieve one of her goals after three actions. (Reaching a goal state sooner is possible, of course, but against optimal defense, success will require three actions.) The keys and depth of a fork are non-trivial computations, however (Epstein, 1990). It is not the case, for example, that every fork built by combining two smaller forks takes longer to execute than its components. How quickly a fork can achieve its objective, quantified here as its depth, must be computed as the minimum of all fork depths arising in the map when the first agent takes any top vertex and then the other agent takes another.

In summary, a fork of depth d represents a game-independent set of mutually overlapping simple plans for a single agent to achieve a goal after at most d actions. The existence of at least one key in every fork imposes a partial ordering on the plan, and provides alternative plans: the parent forks that appear as vertices are removed. The power of a fork lies in the ability of one action to forward more than one plan.

5. The Application of Forks in Two-agent Domains

Because each agent in every state has a strategic map, forks are applicable as both offensive and defensive plans. In many search spaces, plans that guarantee the achievement of a goal state with d actions can be delayed, or even destroyed, by defensive interference from another agent. The best defense against any opposing fork is to take the action in one of its keys yourself, permanently eliminating many of the other agent's simple plans. For example, if one agent has a strategic map containing F-31 from Figure 3(a), the other agent can destroy the fork by taking the action in either of the degree-two keys (say, the right one), immediately reducing that strategic map to Figure 3(d). In what remains of the first agent's strategic map, the only remaining simple plan is readily prevented on the other agent's next turn. For an extended example of how this technique results in highly skilled play see (Epstein, 1990).

Strategic maps must be thoughtfully applied in the play of two-person, perfect information games. If the mover has more than one fork in her strategic map, the best offensive move (action) is always a key in the shallowest fork, that is, pursuit of her shortest certain plan to a win. (If F-1 is detected, this reduces to the common sense theory "If you see a winning move, take it.") If one participant's strategic map contains a fork F of depth d, a win in d turns is guaranteed, even when met by optimal defense, *but only when the second participant forwards no offensive plans of her own and it is the first participant's turn.* If the second pursues a shallower fork of her own, the first will be forced repeatedly to defend against it, delaying or perhaps even losing the ability to pursue F if the set of moves is small. If the second participant is the mover and plays a key in F, the first will have to search elsewhere for a fork.

The application of forks to two-person, perfect information games requires extensive computation, both before and during play. For a machine to exploit forks to their full advantage, it would have to identify correctly every fork, its depth, and its keys in both strategic maps. One way to recognize forks is to maintain a library of them, and search for subsets of bottom vertices in each strategic map that induce a subgraph isomorphic to an element of the library. Implementation of the recursive constructive definition of a fork to build such a library is non-trivial, however. The definition generates infinitely many forks at any given depth, can generate isomorphs of the same fork, and provides only an upper bound on the depth of the new fork. HOYLE's initial foray into

very quickly, averaging 7.6 moves out of a possible 64, a lower bound on performance. In another tournament (#2), the human expert repeatedly played the depth four fork of Figure 4 against the full original implementation of HOYLE with a library of forks only through depth three. During #2, Pitchfork struggled to identify forks within its time constraints and regularly signaled that it was not allocated sufficient resources. By the point in the contest that Pitchfork, with its limited resources, detected an offensive fork, it was too late to defend against it, and HOYLE conceded, still losing every contest. The average contest in #2 was 19.4 moves, however; HOYLE offered much stronger competition than mere random play. The handicaps of limited rationality (both as a time limit and as a fork depth limit) and lack of knowledge of symmetry were too great for this version of HOYLE to make any visible progress at learning Qubic. Against a set of strong game players who were not Qubic experts, however, this version of HOYLE always won.

At this point HOYLE was revised as follows. Pitchfork was modified to learn forks, as described here, and instructed to consider first any recently-learned forks applicable to the current game. Pitchfork was allocated five minutes of computation time. (Each Advisor is otherwise allocated one minute and typically uses only a few seconds.) Pitchfork was told to recommend any good moves found, even those based on partial search of the strategic maps.

During initial forays at Qubic, this modified program learned several different forks of depth greater than three. At least one of these was more elaborate than the pattern the human expert had in mind. The extended time allocation slowed some of the early and non-forced moves, but still permitted real time play; no contest ran more than an hour.

The revised version of HOYLE, without this initial experience, played a tournament of 10 contests (#3). In the first contest of #3, HOYLE was defeated by the successful fork from Figure 4. After the contest, however, Pitchfork analyzed the history and learned the fork, placing it first on Pitchfork's search list. In subsequent contests, each time the human expert attempted that fork, HOYLE identified and blocked it successfully. Play was at an extremely high level throughout #3; contests now averaged 39.8 moves and competition was intense. In the fourth contest, for example, both participants had the fork from Figure 4 in their strategic maps (HOYLE's was a version using central spaces rather than the corners it had learned the fork on). Neither participant was able to execute the fork because of interference from the other, and the contest was a draw. In the seventh contest, the human expert in one state had two different isomorphs of the fork from Figure 4. HOYLE was able to block these, execute successfully a shallower fork of its own, and go on to win the contest. HOYLE never lost a contest in #3 after the first one, and it regularly caught the human expert off guard, winning six times.

HOYLE was also tested on a 3-by-3-by-3 version of tic-tac-toe, a somewhat simpler game that Player should always win. The original program learned to play perfectly after several contests. The fork-learning version learned to play with true expertise just as quickly, but it also learned the depth-four fork produced by the opening move in the post mortem for the first contest. (This fork is different from the one learned in Qubic.) In the remainder of the tournament, as Opponent HOYLE resigned after the correct opening; as Player HOYLE declared victory before it made a single move, but still went on to play perfectly and win each contest.

8. Results and Future Work

Previously, a few sets of mutually overlapping simple plans with very limited applicability were identified as naive offensive strategies in two-agent domains. HOYLE's game-independent, graph representation for forks includes these plans and provides a plan generator. Each fork implicitly contains both conjunctive and disjunctive descriptions. Proper application of the seven smallest forks has been shown empirically to produce high-quality solutions and to focus attention on strategic possibilities not otherwise likely to be found in some large search spaces.

For relatively easy games, simple heuristics to construct, recall, and search for these seven forks in a contest will support learning to play perfectly. For more difficult games, however, high match cost and utility (Minton, 1988) become issues. Empirical work with HOYLE thus far indicates that, perhaps by the topology of their boards and the nature of their rules, most games intrinsically have a limited number of applicable forks. Each larger fork relevant to a particular game is readily identified from an explanation of the trace of a defeat at that game. Searching first for these learned forks in subsequent contests of the same game suffices to simulate expert play.

The learned plans are generalizations potentially applicable to any game, with a well-defined metric (depth) that supports their error-free application both offensively and defensively without forward search into the game graph, and an implicit game-independent execution plan that provides for contingencies. Because heuristics are used in the implementation, performance may be imperfect. In its current game testbed, however, HOYLE's ability to learn plans from observing its defeat at the hands of an expert rapidly enables it to learn to win. Finally, although the abstractions representing forks were designed for a particular class of games, they are clearly extendible to any multi-agent competitive planning domain.

Future research includes the extension of this work to contests where the winner deliberately deflects attention from a fork being executed, to games where moves are retractable, and to games where the goal is disabling playing pieces rather than achieving a specific configuration.

These results, while promising, raise an interesting issue: at the current time, HOYLE is not an aggressive player, merely an opportunistic one. On defense, HOYLE detects and blocks the forks it knows, within its heuristic constraints. On offense, however, HOYLE does not actively plan to construct a state in which it will have a fork; it only executes forks that happen to lie in a state. This opportunistic behavior is not aggressive play, or even aggressive planning. Current research includes *near forks*, aggressive plans that give rise to executable forks. Until HOYLE takes the construction of a fork, rather than its execution, as a goal, the program will continue to play excellent defense but only serendipitous offense.

References

Anantharaman, T., Campbell, M. and Hsu, F. 1988. Singular Extensions: Adding Selectivity to Brute Force Searching. In *Proceedings on Computer Game Playing*, 8-13. *AAAI 1988 Spring Symposium Series.*

Banerji, R. B. 1980. *Artificial Intelligence: A Theoretical Approach.* New York: North-Holland.

Berliner, H. J. 1980. Backgammon Computer Program Beats World Champion. *Artificial Intelligence* 14 (2): 205-220.

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

Epstein, S. L. 1989a. The Intelligent Novice - Learning to Play Better. In *Heuristic Programming in Artificial Intelligence - The First Computer Olympiad*, ed. D. Levy and D. Beal. New York: John Wiley.

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

Epstein, S. L. 1990. Deep Forks in Strategic Maps. To appear. Also available as Technical Report, TRS-89-04, Hunter College of the City University of New York.

Fikes, R. E. and Nilsson, N. J. 1972. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 2: 189-208.

King, P. F. 1970. A Computer Program for Playing Positional Games. M. S. diss., Case Western Reserve University.

Koffman, E. B. 1968. Learning through Pattern Recognition Applied to a Class of Games. *IEEE Trans. Sys. Sci Cybernetics*, SSC-4 .

Korf, R. E. 1985. Macro-Operators: A Weak Method for Learning. *Artificial Intelligence* 26 (1): 35-77.

Laird, J., Rosenbloom, P. and Newell, A. 1986. Chunking in SOAR: An Anatomy of a General Learning Mechanism. *Machine Learning* 1 (1): 11-46.

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

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

Minton, S. 1984. Constraint-Based Generalization - Learning Game-Playing Plans from Single Examples. In *Proceedings of the Fourth National Conference on Artificial Intelligence*, 251-254. Los Altos: William Kaufmann.

Minton, S. 1988. *Learning Search Control Knowledge - An Explanation-Based Approach.* Boston: Kluwer Academic.

Mitchell, T. M., Utgoff, P. E. and Banerji, R. 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. Palo Alto: Tioga Publishing.

Mooney, R. J. 1988. Generalizing the Order of Operators in Macro-Operators. In *Proceedings of the Fifth International Conference on Machine Learning*, 270-283. Morgan Kaufmann.

Paul, J. L. 1978. Tic-Tac-Toe in n-Dimensions. *Mathematics Magazine* 51: 45-49.

Rosenbloom, P. S. 1982. A World-Championship Level Othello Program. *Artificial Intelligence* 19 (3): 279-320.

Samuel, A. L. 1967. Some Studies in Machine Learning Using the Game of Checkers. II - Recent Progress. *IBM Journal of Research and Development* 11 (6): 601-617.

Schaeffer, J. 1988. Learning from (Other's) Experience. In Proceedings on Computer Game Playing, 51-53. *AAAI 1988 Spring Symposium Series.*