Learning and Applying Competitive Strategies

Esther Lock and Susan L. Epstein

Hunter College and The Graduate Center of The City University of New York
Department of Computer Science
New York, New York 10021

Abstract

Learning reusable sequences can support the development of expertise in many domains, either by improving decision-making quality or decreasing execution speed. This paper introduces and evaluates a method to learn action sequences for generalized states from prior problem experience. From experienced sequences, the method induces the context that underlies a sequence of actions. Empirical results indicate that the sequences and contexts learned for a class of problems are actually those deemed important by experts for that particular class, and can be used to select appropriate action sequences when solving problems there.

Repeated problem solving can provide salient, reusable data to a learner. This paper focuses on programs that acquire expertise in a particular domain. The thesis of our work is that previously experienced sequences are an important knowledge source for such programs, and that appropriate organization of those sequences can support reuse of that knowledge, in novel as well as familiar situations (Lock 2003). The primary contributions of this paper are a learning method for the acquisition of *action sequences* (contiguous subsequences of decisions) and an empirical demonstration of their efficacy and salience as *competitive strategies* in game playing. We claim not that sequences are *all* there is to learn, but that they are a powerful part of expert behavior, and well worth learning. In the work reported here, our method learns many useful action sequences and highlights the ones human experts consider significant.

The *context* of an action sequence is the set of world states where it is relevant. Rather than deduce all possible contexts from the operators available to an agent, or from inspection of the search space, our method exploits the knowledge inherent in experience to learn context. It associates a sequence of actions with the set of *origin states* where the sequence's execution began. The underlying assumption is that some commonality among origin states drives the same sequence of moves. Discovering context is thus reduced to generalization over the origin states to extract it. A learned context is paired with its sequence, and subsequently recommended as a course of action when the context matches the current state of the world.

Macro operators to speed problem solving and planning have been widely studied. MACROPS learned to compose a sequence of operators together, but it was limited to domains with a set of operators with well-defined preconditions and post-conditions, ones where the goal state was a conjunction of well-defined subgoals (Fikes, Hart et al. 1972). Other programs learn to acquire macros that achieve subgoals without undoing previously solved subgoals (Korf 1985; Tadapelli and Natarajan 1996). A game, however, may lack accessible subgoals, and actions in games can have subtle, far-reaching effects that are not clearly delineated. MACLEARN and MICRO-HILLARY use a domain's heuristic function to selectively learn macros from problem solving experience (Iba 1989; Finkelstein and Markovitch 1998). Our method assumes neither the existence of subgoals nor of a heuristic function. Planning for Markov decision processes learns macro actions to achieve subgoals (Moore, Baird et al. 1998; Sutton, Precup et al. 1999; Precup 2000). Most closely related to our method is reinforcement learning work that discovers useful sequences online from successful experience (McGovern 2002). Neither of McGovern's methods generalize context, however.

Our approach uses problem-solving experience to learn the context for a sequence, and uses that knowledge to make decisions. The next section describes how this method learns action sequences and their contexts, and applies them. Subsequent sections describe a program that applies learned competitive strategies, details our experimental design, and discuss results and related work.

1. Learning action sequences with contexts

We explore sequence learning first within the domain of two-player, perfect-information, finite board games, using the following definitions. (Broader applicability is addressed in Section 3.) A problem-solving experience is a *contest*, the agents *contestants*, and a legal move an *action.* A *state* is uniquely described by the game board and whose turn it is to move. A place on a game board where a contestant's piece may be placed is a *location.* All locations on a game board are numbered in row order, as in Figure 1. A *blank* is an empty location. A *pattern* is a set of locations and their contents. Our method learns action sequences only from successful experience; in game playing, those are move sequences from contests in which a particular contestant won or drew at a *draw game*, where the value of the root of the game tree is a draw under full

minimax.

We consider two games here: lose-tic-tac-toe and five men's morris. *Lose-tic-tac-toe,* where the first to place three pieces in a row in Figure 1(a) loses, is far more difficult for people to learn than ordinary tic-tac-toe. The object is to avoid a certain pattern rather than to achieve it, and a non-optimal move is a fatal error. Flawless lose tic-tac-toe play involves different strategies for each contestant, some of which are non-intuitive and go undiscovered by many people (Cohen 1972). In *five men's morris*, the contestants (black and white) each have five pieces, which they take turns placing on the board in Figure 1(b). Once the pieces are all placed, a move slides a piece from one location on the board to an adjacent, empty location. A contestant who achieves a *mill* (three owned pieces on a line drawn on the board) removes any one opposition piece from the board. A contestant reduced to two pieces or unable to slide, loses. This is a challenging game, with a significantly larger game graph than lose tic-tac-toe.

1.1 Overview of the learning method

The program *SeqLearner* observes computer programs of various skill levels in competition at a game. After each contest, SeqLearner examines the contest's actions for sequences of prespecified lengths. It collects action sequences that are *duets* (include both agents' actions) and *solos* (include one agent's actions.) In some two-agent domains, an agent determinedly executes a sequence of actions, so long as they are legal, no matter what the other agent does. If one gathers only sequences that include both agents' actions, the other agent's varied responses to a solo could cast instances of the same solo as different sequences. Therefore, SeqLearner also gathers solos. In the trace of a contest of length n, the number of duets of any length is $O(n^2)$, as is the number of solos.

To associate these sequences with states, SeqLearner organizes the collected data into *SeqTable*, a sequence hash table. A key for SeqTable is a sequence; a value stored there is a list of states. For each extracted sequence, SeqLearner records in SeqTable the sequence's origin state, where the sequence's execution began. For repeatedly encountered sequences, SeqLearner extracts a context from the set of origins associated with it. Periodically, SeqLearner sweeps through SeqTable, examining one sequence at a time. For game playing, SeqLearner identifies the maximal common pattern that exists in all the stored states as the context for the sequence. SeqLearner takes

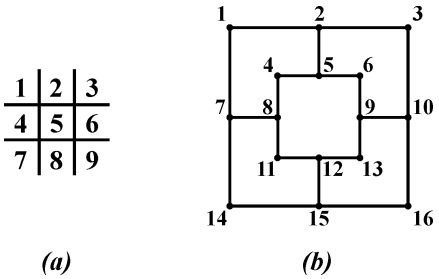


Figure 1: Two game boards and their numbered location for (a) lose tic-tac-toe and (b) five men's morris.

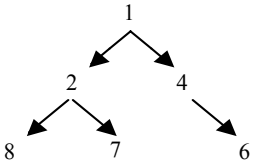


Figure 2: A sequence tree merging three sequences that have the same context.

each location of the board as a feature, and examines its value in each of those states, retaining only features with the same value on all the boards. Any location whose value is not retained is labeled # (don't care). The resultant *maximal common pattern* is associated with its sequence in a *context pair*. Both contexts and sequences are normalized for horizontal, vertical, and diagonal reflection, plus rotations of 90°, 180°, and 270°. As a result, a sequence is retrievable whenever its context or a symmetric equivalent arises. With additional experience, the candidate sets for each sequence are likely to grow, engendering new, more general context pairs. If more than one sequence leads to the formation of the same context, those sequences are merged into a *sequence tree* (a tree of alternative, overlapping sequences) to form a single context pair. For example if the three sequences <X to 1, O to 2, X to 8>, <X to 1, O to 2, X to 7>, and <X to 1, O to 4, X to 6> have the identical context, they are merged into the competitive strategy in Figure 2.

Given their inductive origin, these context pairs are not applied immediately in problem solving; the incorrect context pairs must first be carefully filtered out. A variant of *PWL* (Probabilistic Weight Learning) is used to weight them (Epstein 1994). The algorithm determines how well each context pair simulates expert behavior. After every experience, it takes the states where it was the expert's turn to act, checks whether a context pair's advice supports or opposes the expert's decision, and revises the pair's weight in [0,1] accordingly. A context pair's weight rises consistently when it is reliable and frequently applicable.

1.2 An example

Figure 3, for example, shows a repeatedly detected lose tic-tac-toe sequence, <X to 1, O to 3, X to 7>, that began at six different states. The sequence is the key to the

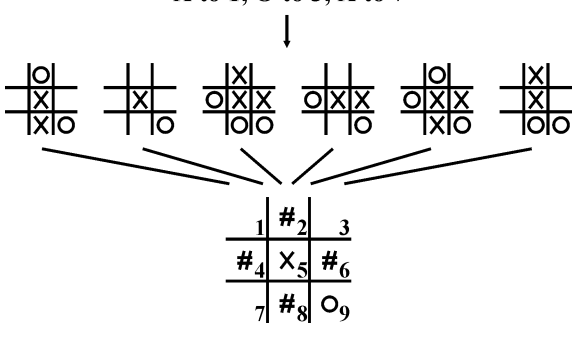


Figure 3: The knowledge stored in the SEQTABLE and its associated extracted context.

SeqTable, and the list of states is its stored value. The state descriptions are the data from which context is induced.

Figure 3 also shows the maximal common pattern SeqLearner induces as a context for that sequence. The locations consistently present in the origin states are 1, 3, 5, 7, and 9, containing blank, blank, X, blank, and O, respectively. Rather than associate the sequence with all six states, SeqLearner associates the sequence with the pattern at the bottom of Figure 3. If this pattern matches a state during future play, the duet sequence <X to 1, O to 3, X to 7> can be suggested as a possible course of action, whether or not the state is among the original six. A solo would include only one player's moves, ignoring the opponent's responses.

1.3 Using the learned sequences

SeqPlayer is a game-playing program created to test SeqLearner. SeqPlayer begins with no knowledge about a game other than the rules. It learns to play a game during training, where it observes programs of various skill levels in competition. During training, SeqPlayer uses SeqLearner to learn and weight context pairs. During testing we evaluate SeqPlayer's performance while it uses the learned context pairs as competitive strategies to make decisions. There are many ways the data acquired by SeqPlayer might be used; we report here on two.

Version 1 of SeqPlayer's decision-making module is a two-tier, hierarchical framework. Tier 1 is a single procedure, Enforcer, which either continues a sequence whose execution has already begun or discards one that becomes inapplicable because of the other contestant's response. Tier 2 is those context pairs whose weights had reached or exceeded a pre-set threshold (.55 in the work reported here) by the end of training. When it is SeqPlayer's turn to choose a move, control begins in tier 1, where Enforcer suggests the next move in an ongoing sequence. If no sequence is being executed or one has just been discarded, control passes to tier 2. Each context pair compares its context with the current state; if they match, it supports the first action in its sequence with the context pair's learned weight. The action with the highest total support is chosen.

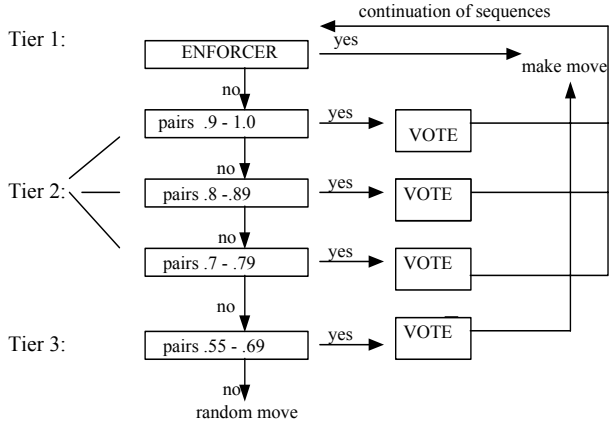


Figure 4: Version 2 of SeqPlayer's Decision-making Module

Thus, actions that begin more than one sequence and actions proposed by context pairs with higher weights are more likely to be selected. If a sequence's first action is chosen, the rest of the sequence is sent to Enforcer, which merges the individual sequences it receives into one sequence tree. As the contest continues, Enforcer recommends a next action from the branch in the sequence tree that the opponent followed. If there is no such branch, Enforcer discards the sequence tree. If no move is recommended in tier 2, a random move is chosen.

Version 2 of SeqPlayer's decision-making module has a more sophisticated structure, shown in Figure 4. It uses tier 1 from Version 1, followed by two new tiers. Control passes through the tiers one at a time; if no move is selected, then one is chosen at random. Tier 2 partitions the best of the context pairs into three subclasses by weight: [.9, 1.0], [.8, .9], and [.7, .8]. (Partition criteria were empirically selected.) Each subclass supports actions the way Version 1's tier 2 does. Together, however, they work as a hierarchy: as long as a sequence is not yet selected, the next subclass is consulted. Tier 3 includes all context pairs with weights in [.55, .7]. Like the tier-2 context pairs, those in tier 3 find applicable sequences and select an action based upon them, but tier 3 does not send the continuation of the chosen sequences to Enforcer. Tier 3's one-move advice from low-weight context pairs is expected to be better than choosing a move at random. Rather than enforce the remainder of the sequences associated with these low-weight context pairs, however, the program returns to the higher-weight context pairs on the next move, in anticipation that a better decision can be made, and a more correct sequence will be identified. Because Version 2 considers only one subclass of context pairs at a time, and stops when an action is suggested, the program no longer has to consult the full set of context pairs on every decision. This should decrease execution time.

2. Experimental Design and Results

Variously-skilled computer programs were used in our experiments. A *perfect agent* is a hand-crafted external program that plays a randomly-chosen best move in any state. A *reasonable* play makes a winning move one ply away, avoids a losing move two ply away, and otherwise makes a random, non-losing move. An *x-reasonable agent* makes reasonable plays *x*% of the time, and perfect moves 100-*x*% of the time. For purposes of comparison, baseline experiments were conducted for lose tic-tac-toe and five men's morris. In each *baseline experiment* in Table 1, a random agent competed in four 1,000-contest tournaments against four different opponents: a perfect agent, an expert (10%-reasonable), a novice (70%-reasonable), and a random agent.

To evaluate SeqLearner, two experiments were conducted on each game: Experiment 1 with Version 1 of SeqPlayer and Experiment 2 with Version 2. SeqPlayer breaks voting ties at random, so each experiment reported

Table 1: Percentage of contests won (power) and won +drawn (reliability) by Baseline and SeqPlayer (Experiments 1 and 2) when tested against four opponents. Statistically improved performance over the previous experiment is highlighted in bold, with standard deviations in parentheses. (Power is not shown against the perfect player, since one cannot win a draw game against perfect play.)

	Perfect	**Expert**		**Novice**		**Random**	
	Reliability	**Reliability**	**Power**	**Reliability**	**Power**	**Reliability**	**Power**
Lose tic-tac-toe							
Baseline	20.50	25.80	9.00	49.20	37.70	58.40	46.10
	(4.14)	(5.33)	(4.12)	(7.57)	(6.67)	(7.28)	(7.84)
Experiment 1	**66.00**	**70.80**	**12.80**	**83.50**	**42.70**	**85.30**	**55.40**
	(20.96)	(16.22)	(4.71)	(6.38)	(8.08)	(4.79)	(5.52)
Experiment 2	**82.00**	**83.80**	12.10	85.10	46.70	87.00	57.40
	(13.24)	(9.73)	(4.22)	(5.08)	(7.08)	(4.17)	(7.02)
Five men's morris							
Baseline	0.20	1.60	1.20	22.40	14.20	54.00	36.00
	(0.60)	(1.20)	(0.98)	(4.27)	(5.90)	(7.27)	(6.81)
Experiment 1	**39.60**	**34.20**	1.00	**28.80**	9.20	54.20	17.40
	(13.11)	(11.29)	(1.34)	(4.66)	(2.23)	(7.18)	(6.39)
Experiment 2	**57.80**	**45.00**	0.80	**45.00**	**29.00**	**73.00**	**51.20**
	(13.22)	(10.05)	(1.33)	(4.58)	(5.81)	(8.35)	(7.44)

here averages data from ten runs. A *run* is a training stage followed by a testing stage. A *tournament* is a set of contests between two agents. During training, SeqPlayer observed a tournament that pitted a perfect agent against a random agent, followed by a tournament against a novice, one against an expert, and one against another perfect agent. This cycle was then repeated (8 tournaments in all) to provide enough exposure to the space to form the context pairs and weight them properly. A training tournament was 20 contests for lose tic-tac-toe, and 40 for five men's morris. The testing stage for both games consisted of four 50-contest tournaments that pitted SeqPlayer against the same four agents used in training. We assumed that shorter sequences would be more frequently applicable in a broader range of unfamiliar situations, and therefore set SeqLearner to learn only shorter sequences, of length 3 for lose tic-tac-toe and lengths 3 and 5 for five men's morris. Although SeqLearner can learn any length sequence, the lengths chosen for the final experiments were hand-tuned. Context pairs were extracted from SeqTable after every training tournament.

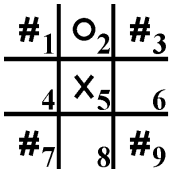
Experimental results appear in Table 1, where *reliability* is the number of wins plus the number of draws, and *power* is the number of wins. Recall that SeqPlayer begins without any built-in knowledge, either of game-playing in general or of the particular game itself. SeqPlayer uses only the knowledge it learns from watching sequences in actual play. In light of that, one does not expect SeqPlayer alone (i.e., without other knowledge or learning methods) to lead to superior play. These results demonstrate that much knowledge about a game is contained in experienced sequences; they measure SeqPlayer's ability to capitalize on this knowledge.

For lose tic-tac-toe, Experiment 1 (using context pairs with weights above .55) shows that SeqPlayer was able to win or draw 66-70% of the time against the stronger players and 83-85% of the time against the weaker players. This is a significant improvement in both reliability and power over Baseline for all four opponents. In Experiment 2, recall, context pairs with weights at least .55 were distributed among tier 3 and the subclasses of tier 2. This hierarchical approach enabled SeqPlayer to win or draw the game 82-87% of the time against all opponents, and improved reliability against the perfect and expert players.

Five men's morris, with its significantly larger space, suggests how this method scales. In Experiment 1, SeqPlayer learns enough knowledge to significantly improve reliability over Baseline for the perfect, expert and novice players. Experiment 1 actually demonstrated decreased power against the novice and random players. Basing moves on sequence knowledge alone is

Table 2: Number of learned context pairs by tiers and subclasses

	Lose Tic-Tac-Toe		**Five Men's Morris**	
Tier 2				
Subclass 1 (weight .9 - 1.0)	17.80	(2.79)	575.50	(48.79)
Subclass 2 (weight .8 - .89)	3.80	(1.54)	210.40	(8.44)
Subclass 3 (weight .7 - .79)	6.65	(1.84)	277.90	(20.97)
Tier 3 (weight .55 - .69)	3.20	(1.44)	496.30	(20.79)



< X to 8, O to 6, X to 4 >

Figure 5: With this context pair, SeqLearner has learned to reflect through the center in lose tic-tac-toe, a technique people rarely discover for this game.

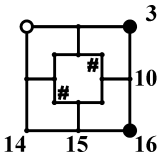
problematic for two reasons: it does not give a player enough power to win, and it may not provide enough applicable knowledge when competing against weaker players who follow few, if any sequences. Experiment 2 produced better reliability than Experiment 1 against all four opponents, and improved power against the novice and random players. As a result, SeqPlayer was able to win or draw against a perfect player 57.80% of the time, and slightly less against the mid-level opponents.

Table 1 shows that SeqLearner is most useful for learning techniques against better opponents. Realistically, the better the opponent, the more one should expect to rely on learning to produce an expert program. Hence an algorithm like SeqLearner, that leads to better performance against stronger opponents, is an important step in game learning, and a prospective component for multiple-method learning programs.

The average number of sequences collected in SeqTable in Experiment 2 was 78.2 for lose-tic-tac-toe, and 12,234 for five men's morris. These sequences yielded 31.45 context pairs with weights above .55 for lose tic-tac-toe, and 1560.1 for five men's morris. Table 2 examines the quality of the context pairs learned in Experiment 2, by their weights. In both games, context pairs with weights in [.9, 1] form the largest subclass of tier 2, larger than all the other included context pairs together. This means that many retained context pairs give excellent advice. In five men's morris, the second largest group is in tier 3. As a result, Version 2 runs 21% faster than Version 1, since tier 3 is rarely reached in the decision-making process.

3. Discussion and Related Work

Inspection indicates that SeqLearner learns salient concepts for each game. Figure 5 shows a context pair learned for lose tic-tac-toe that achieved a perfect weight of 1.00 after 160 training contests. It advises Player X to reflect O's move horizontally or vertically through the center. Although proved to be correct play (Cohen 1972), few people discover it (Ratterman and Epstein 1995).



< W to 10, B to 14, W to 15 >

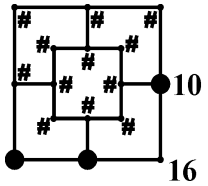
Figure 6: With this context pair, SeqLearner has learned a duet for five men's morris that anticipates and defends against the formation of two mills by black.

Figure 6 is a context pair learned in the placing stage of five men's morris; it advises White to block Black from creating a mill twice. By moving to position 10, White blocks Black from creating a mill in positions 3-10-16, and by moving to position 15 after Black moves to 14, White blocks Black from creating a mill in 14-15-16. Figure 7 shows a solo sliding context pair learned to create a mill and shuttle in and out of the mill once it is achieved. This is a crucial sequence for winning five men's morris. (An equivalent context pair was also learned for White.) The context pairs in Figures 6 and 7 achieved weights of 1.00 after 320 training contests. All context pairs learned are applicable elsewhere on the board, when a symmetrically equivalent context arises. This shuttle is a sophisticated exploitation of the rules to achieve the goal of reducing the opponent to two pieces.

SeqLearner learns action sequences for generalized states. Results show that SeqLearner, beginning with no knowledge, can learn a great deal about playing the game of lose tic-tac-toe, and scales to the more difficult game of five men's morris. Surprisingly, relying only on learned sequences, SeqPlayer can draw against a perfect five men's morris player more than half the time. Results also show that the most dramatic improvement gained from applying SeqLearner occurs when testing against the better players. This correlates with cognitive science results that better chess players rely on move-sequence structures stored in their long-term memories (Chase and Simon 1973).

Instead of all possible legal sequences, SeqLearner induces contexts from the limited number of sequences seen during experience. This should enable it to scale to larger spaces. However, for induction to be correct in a large space, many training examples will be needed. Therefore, for games with many more moves per contest and much larger boards, the SeqLearner approach may require some adaptation. Instead of collecting all sequences, additional game-specific knowledge could specify when sequences should be collected. For example, one can define locality for a game, and collect only sequences that fall in the same locality. In Go, sequences of moves in the same geographic vicinity are usually related; this could be used to indicate the length of a sequence to be learned. The algorithm that looks for the underlying context would also concentrate on a vicinity instead of the entire board. In a game like chess, however, locality would be defined both by proximity to the locations moved to, and by the squares those moved pieces attack and defend.

Although the algorithm in SeqLearner for the maximal common pattern is efficient, it does not consider the value



< B slide from 10 to 16
W anywhere
B slide from 16 to 10 >

Figure 7: With this context pair, SeqLearner has learned a powerful solo for five men's morris that shuttles in and out of a mill.

of more than one board position at a time as a feature, which may be important to the actual context. Hence the algorithm may over-generalize, that is, drop locations whose values were not identical, but whose combination of values was important. If the feature representation for learning contexts were *complete* (able to express every combination of the locations on the board as a feature) this would not be an issue. Completeness, however, comes at the expense of efficiency. In general, one limitation of SeqLearner is that context learning depends on human-selected features. The wrong features will form incorrect contexts. Learned features are a topic for future work.

Like SeqLearner, case-based planners index a sequence under a context to retrieve for future use. Unlike SeqLearner, however, case-based planners require large quantities of domain knowledge (Hammond, Converse et al. 1993). PARADISE linked chess concepts to formulate plans that guided a small search tree to select the best plan (Wilkins 1980). PARADISE's concepts, however, were taken from a manually constructed chess knowledge base, and not learned. Sequential Instance Based Learning (SIBL) learns to appropriately select actions in new bridge contests based on action patterns (i.e., sequences of consecutive states) played in previous contests (Shih 2001). Rather than learn the context, SIBL treats the sequence that led to an action as the context for the action itself, and trains a database of <sequence, action> pairs based on similarity metrics that measure a new sequence's similarity to known sequences. Kojima's evolutionary algorithm that learns patterns for Go, can also learn sequences of moves (Kojima and Yoshikawa 1999). Kojima's system, however, like SIBL, learns sequences as part of the context for the next single action to take, unlike SeqLearner which learns sequences to follow.

SeqLearner is not restricted to two-agent-domains; it could learn sequences that involve one agent or more than two agents. Although SeqLearner extrapolates a context from limited experience inductively, searching more broadly would be costly. SeqLearner takes a computationally reasonable approach; once it narrows the focus, a deductive method can further refine the acquired knowledge. For example, a narrow search from an induced context could be attempted to prove correctness.

Meanwhile, the work outlined here has demonstrated that previously experienced sequences are an important resource for programs learning expertise in a particular domain. Experience with sequences can support learning their contexts. Future work will experiment with other domains, such as truck-routing, other context extraction methods, and with using domain-specific knowledge to guide the sequence gathering of SeqLearner, rather than gathering all sequences of various lengths.

Acknowledgements

This work was supported in part by the National Science Foundation under #IIS-0328743 and #9423085.

References

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

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

Epstein, S. L. (1994). Identifying the Right Reasons: Learning to Filter Decision Makers. *Proc. of the AAAI 1994 Fall Symposium on Relevance*, Palo Alto.

Fikes, R. E., Hart, P. E., and Nilsson, N. J. (1972). "Learning and Executing Generalized Robot Plans." *Artificial Intelligence* **3**: 251-288.

Finkelstein, L. and Markovitch, S. (1998). "A Selective Macro-learning Algorithm and its Application to the *N* x *N* Sliding-Tile Puzzle." *JAIR* **8**: 223-263.

Hammond, K. J., Converse, T., and Marks, M. (1993). "Opportunism and Learning." *Machine Learning* **10**: 279-309.

Iba, G. (1989). "A heuristic approach to the discovery of macro-operators." *Machine Learning* **3**: 285-317.

Kojima, T. and Yoshikawa, A. (1999). Knowledge Acquisition From Game Records. *ICML Workshop on Machine Learning in Game Playing*.

Korf, R. (1985). "Macro-Operators: A Weak Method for Learning." *AIJ* **26**: 35-77.

Lock, E. (2003). Learning and Applying Temporal Patterns through Experience. Ph.D. thesis, The Graduate School of The City University of New York.

McGovern, A. (2002). Autonomous Discovery of Temporal Abstractions From Interaction with an Environment. Ph.D. thesis, University of Massachusetts.

Moore, A. W., Baird, L., and Kaelbling, L. P. (1998). Multi-Value-Functions: Efficient automatic action hierarchies for multiple goal MDPs. *NIPS'98 Workshop on Abstraction and Hierarchy in Reinforcement Learning*.

Precup, D. (2000). Temporal abstraction in reinforcement learning. Ph.D. thesis, University of Massachusetts.

Ratterman, M. J. and Epstein, S. L. (1995). Skilled like a Person: A Comparison of Human and Computer Game Playing. *Proc. of the Seventeenth Annual Conference of the Cognitive Science Society*, Hillsdale, NJ, Lawrence Erlbaum Associates.

Shih, J. (2001). Sequential Instance-Based Learning for Planning in the Context of an Imperfect Information Game. *International Conference on Cased Based Reasoning*.

Sutton, R. S., Precup, D., and Singh, S. (1999). "Between MDPs and Semi-MDPs: A framework for temporal abstraction in reinforcement learning." *AIJ* **112**: 181-211.

Tadapelli, P., and Natarajan, B. K. (1996). "A formal framework for speedup learning from problems and solutions." *JAIR* **4**: 419-433.

Wilkins, D. (1980). "Using Patterns and Plans in Chess." *Artificial Intelligence* **14**: 165-203.