Learning from Sequential Examples: Initial Results with Instance-Based Learning

Susan L. Epstein[1] and Jenngang Shih[2]

[1]Department of Computer Science,
Hunter College and The Graduate School
of The City University of New York, New York, NY
epstein@roz.hunter.cuny.edu
[2]Department of Computer Science,
The Graduate School
of The City University of New York, New York, NY
jshih@broadway.gc.cuny.edu

Abstract. This paper postulates an approach to planning from a sequence of instances. Sequential instance-based learning (SIBL) generates a sequential hierarchy of planning knowledge from which to formulate plans and make decisions. We report here on the application of SIBL to the game of bridge. Initial results indicate that examples applied in a sequentially dependent manner more often select correct actions than if the examples were used independently. SIBL suggests how empirical learners for classification problems may be extended to learn to plan. The contributions of this paper are the formulation of planning as a sequence of related instances, and a demonstration of the efficacy of majority vote with SIBL in the domain of bridge.

1 Introduction

The thesis of this work is that an empirical learning algorithm for classification can be systematically extended to learn knowledge for planning, where a planning problem is viewed as a sequence of related classification problems. We call the application of this idea to IBL *sequential instance-based learning* (*SIBL*), and show here how it can learn to plan from instances, also known as cases or exemplars.

In a typical implementation, *case-based reasoning* (*CBR*) retrieves a set of relevant cases of the current problem, reuses the most relevant case, revises the retrieved case according the result required by the application, and retains the problem as a new case (Aamodt and Plaza, 1994). Although CBR has been applied to planning before, most CBR planners rely heavily on domain knowledge to manage the inherent complexity (Alterman, 1988; Hammond, 1989; Marks et al., 1988; Turner, 1988).

Manual construction and maintenance of such a knowledge-intensive system is both costly and difficult. An alternative is empirical learning, where the system accepts examples as input and produces concept descriptions (Michalski, 1983; Quinlan, 1986). Most empirical learners, however, are intended for classification problems. In the context of CBR, for example, *instance-based learning*

(*IBL*) represents both the input examples and the output concept descriptions as feature-value pairs, and most IBL algorithms retain the prototypical examples for reuse (Aha, 1992; Aha et al., 1991).

The focus of our work is to extend an empirical learning paradigm for classification, such as IBL, to learn concept descriptions for planning. The next sections introduce sequential dependency, the domain of bridge, and SIBL. Subsequent sections describe the experiments, their results, and related and future work.

2 Sequential Dependency

In classical AI planning, a planning problem is specified by an initial state, one or more goal states, and a set of operators that transform one state to another. A plan is a sequence of operator instantiations (actions) that transforms the world from the initial state to a goal state (Hendler et al., 1990). Most planning algorithms rely mainly on the current state description to select an action; the dependency among the sequential events is implicit.

SIBL views sequential events as dependent, rather than independent. In a sequence of n instances (s_1, ..., s_i, ..., s_j, ..., s_n), we say that s_j *sequentially depends on* s_i for all $1 \leq i \leq j \leq n$. Here s_i is called the *predecessor* and s_j the *successor*. Because it has in part formulated the current state and the action available in it, a predecessor influences its successor. Similarly, a successor is influenced by its predecessors to take an action. Thus, while the dimension of a classical AI planner's state description is fixed, the dimension of SIBL's state description is variable, depending on how far back one looks at the predecessors.

Given this information, one way to plan is to select a correct action based on a sequence of past events of a certain length. If the selected action is based on too short a sequence, there may not be enough information. If the selected action is based on too long a sequence, the decision may overfit a unique sequence. One resolution of this issue is to use the *majority vote*, which selects the action that is the majority among a set of similar sequences of various lengths. Let $s_i \ldots s_j \rightarrow a_k$ denote that the sequence of events from s_i to s_j recommends the action a_k. Then, for example, under majority vote the set of recommendations $\{s_a s_b s_c \rightarrow a_1, s_x s_y \rightarrow a_1, s_z \rightarrow a_2\}$ would select a_1.

Majority vote has been one of the primary parameters for the IBL family of algorithms (Aha, 1992; Aha et al., 1991; Cost and Salzberg, 1993; Dasarathy, 1991). It is an intuitive approach; if the majority of sequences under consideration point to the same action, the likelihood that the action is correct should be good. This paper applies majority vote in SIBL to the game of bridge.

3 Bridge

Bridge is a four-player planning domain. To begin, the 52 distinct cards are dealt out, that is, distributed equally among the players. The game then has two phases: bidding and play. During *bidding*, a specific number of tricks for

winning (the *contract*) is determined. During *play*, one of the contestants (identified during bidding) is the *declarer* and another is the *dummy*. The declarer tries to achieve the contract, controlling both his or her cards and the dummy's, while the other two contestants try to defeat it. (After the first card is played, the dummy's cards are exposed on the table for all to see.) Play consists of 13 sequential tricks; a *trick* is constructed when each contestant in turn plays a single card. The problem addressed here is to design a sequence of actions (card plays) that guides a bridge player to reach a specific goal. Play is viewed here as a planning task.

The search space for bridge is large. There are 13 cards in each of four suits, and during a trick each player must play a card matching the suit of the first card (the lead) in the trick whenever possible. Thus the branching factor is approximately $n(n/4)^3$ for each trick, where n is the number of cards each player holds at the beginning of the trick, and n/4 is the average number of choices the other players have in the suit. For example, on the first trick, when all players still have 13 cards, the branching factor is roughly $13(13/4)^3 = 446$. For a complete deal, the number of possible plays will be $\prod_{n=1}^{13} n/(n/4)^3 = 5 * 10^{15}$. Since there are

$$\binom{52}{13}\binom{39}{13}\binom{26}{13}\binom{13}{13} = 5 * 10^{28}$$

possible ways to deal the cards, there are roughly $3 * 10^{44}$ possible decision states in bridge.

As humans see it, however, the correct decisions in all those states are not independent of each other. How one decides which card to play in a given state depends in part on what cards have been played previously. In so large a space, domain knowledge is required to play well. The knowledge that we propose to learn is how to link decision sequentially.

4 Representation

We represent each input example as a set of feature-value pairs. The features represent the state of the world (the situation in which one is expected to play a card); the associated action is the card selected. State features describe which players hold which cards, as well as bookkeeping information, such as the trick number and how many tricks have been won by each side. Action features represent the associated action. In the work on three no trump deals described here, the highest card in the suit led takes the trick. (Card precedence is 2, 3, 4, ., Queen, King, Ace.) Since the most likely cards to win tricks are the high cards, such as Ace, King, Queen, Jack and 10, those below 10 are considered indistinguishable and represented by a common symbol.

The output concept descriptions can be structured in an AND/OR tree-like sequential hierarchy as in Figure 1. Figure 1 shows a contract of "three no trump" planned in three components: four tricks in which spades are led, three club tricks, and two heart tricks. (The other tricks are disregarded in this plan.) In

turn, each trick has components that describe behavior. A *sequential hierarchy* is a set of sequentially ordered partial sequences organized in a hierarchical structure. A partial sequence, such as "four spades" in Figure 1, is a partial solution that may contain other partial sequences or single actions at the lowest level. Partial sequences, such as "four spades," that are higher in the hierarchy are said to be *strategic* because they concern the general direction of a problem solver; those lower in the hierarchy are said to be *tactical* because they involve maneuvering on a smaller scale.

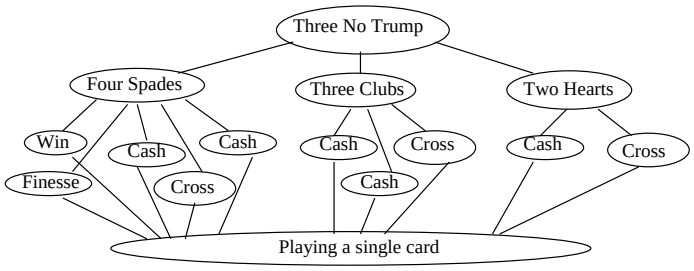

Fig. 1. A Sequential Hierarchy in Bridge

5 SIBL

Formally, an *instance*, denoted σ, is a pair that includes the description of a state s and an action a, such that action a is taken in state s. A *sequence* $(\sigma_1 \ldots \sigma_n)$ is a set of ordered instances of length n. The *empty sequence* e is a sequence with no instances. A sequence v is a *partial sequence* of z if and only if there are sequences x and y such that $z = xvy$ where x, y or both can be e. An instance *spawns* a set of all the partial sequences involving its predecessors in reverse sequential order. For example, the instance σ_3 in the sequence $(\sigma_1\sigma_2\sigma_3)$ spawns $\{(\sigma_3), (\sigma_2\sigma_3), (\sigma_1\sigma_2\sigma_3)\}$. These partial sequences are used as additional input examples for SIBL.

Like most IBL algorithms, SIBL learns a set of prototypical instances from the examples. SIBL is currently based on IB4, which is used to compute similarity between instances. IB4's variable attribute weight produces better instance selection than IB3 does. Similarly, IB4's instance reference count retains more relevant instances than either IB2 or IB1 does (Aha, 1992). Although SIBL is based on IB4, we see no reason why other empirical learning algorithms cannot be adapted to learn from sequences.

If an instance σ has f features and p predecessors, the number of features for the instance after spawning is $f(p+1)$. This linear increase in the number of features increases exponentially in the instance space. In other words, if S is the instance space for σ and σ has p predecessors, the instance space after spawning will be S^{p+1}. To control this, SIBL has a *window*, a constant number of predecessors it learns from, instead of the fully spawned set. For example, for the sequence $(\sigma_1\sigma_2\sigma_3\sigma_4)$, the instance σ_4 with window $w = 3$ produces the set $\{(\sigma_4), (\sigma_3\sigma_4), (\sigma_2\sigma_3\sigma_4)\}$. In bridge, a short sequence of steps has been routinely used by experts to explain playing techniques (Goren, 1963).

```
Set window constant w
Store the first example
While there are more examples in the training set, do
   For each instance, i, in the example, do
      Let current action, a, be the action of i
      Compute from i the set P of all partial sequences
      If a = FindMajorityAction(P), then
         Update the reference count of the partial sequences referenced
      Otherwise,
         Store P
```

Fig. 2. Main Routine that Processes Input Examples, each of which Contains a Sequence of Instances

```
FindMajorityAction(P) :
Initialize the list of majority actions, M, to the empty list
For each partial sequence, p, in P, do
   For each stored partial sequence, q, of the same length as p, do
      Calculate similarity of q to p
   Let majority action, m, be the most frequently recommended
      action by the most similar q's
   Append m to M
Return the majority action in M
```

Fig. 3. The Function that Returns the Majority Action

Pseudocode for SIBL appears in Figures 2 and 3. Figure 2 outlines the processing of input examples. For each partial sequence, SIBL maintains a *reference count*, the number of times the sequence was retrieved and recommended the correct action. Figure 3 shows the computation of the majority action. If no majority exists, the tie is broken by the reference count or, failing that, by random selection. This is similar to the strategy used by IB4.

6 Experimental Design and Results

The goal of the three experiments described here was to test the hypothesis that sequential instances can produce better action selection knowledge than non-sequential ones for planning bridge play. To demonstrate this, the number of correct action selections in a sequence of events is measured, both before and after learning. After learning, more action selections in a sequence of events should be correct. For q correct action selections in a sequential problem with n sequences, the fraction correct is calculated as $r = q/n$. Before learning, r will be small when q is small; after learning, r should increase.

Twenty-four three no trump bridge deals were used in these experiments. All twenty-four deals were fully played by experts who successfully made the contract. Each deal consisted of 26 instances, times when the declarer had to play a card from the dummy or from his or her own hand. Two thirds (16) of the deals were used for training and one third (8) for testing. Every experiment used three-fold cross validation, each with 10 trials (Kibler and Langley, 1988).

The first experiment established a baseline for performance. A *random selector* chose an action to perform in a sequential problem based on the action's probability distribution. That is, for each possible action a, an empirical mass function $f(a)$ is defined to be the proportion of all actions in the training set that are equal to a, where $\sum_{i=1}^{\infty} f(a_i) = 1$.

The second experiment tested the effectiveness of selection based on fixed sequences. A *fixed-sequence selector* was trained with instances of a fixed sequence length using IB4 (Aha, 1992). The *minimum fixed-sequence selector* used instances of length one, i.e., non-sequential instances. The *maximum fixed-sequence selector* used instances of length defined by the window constant $w = 5$.

The third experiment tested SIBL as described above. The *variable-sequence selector* chose among instances of varying lengths from one to the window constant w. It selected an instance using majority vote and the reference count. All three selectors broke ties with random selection.

Figures 4 and 5 depict the portion of the time that the correct action was selected during learning and testing, respectively. In Figure 4, during training the maximum fixed-sequence selector chose the correct action with a higher frequency than the minimum fixed-sequence selector did. In Figure 5, however, the performance of the maximum fixed-sequence selector is actually worse during testing. This anomaly occurred because, although both had the same number of input examples, the maximum fixed-sequence selector has a much larger instance space (S^{p+1}).

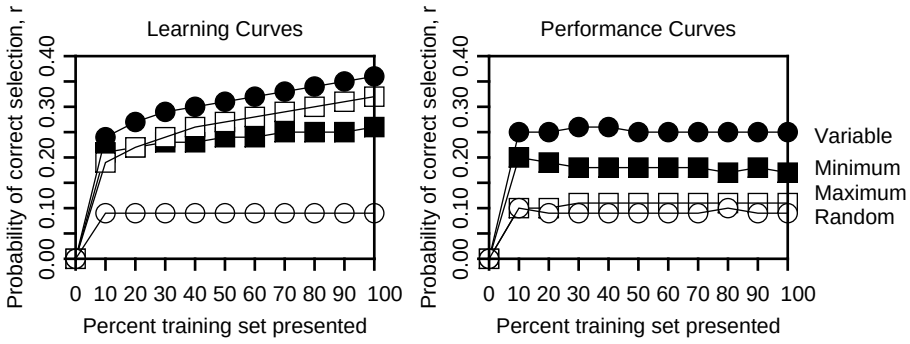

Fig. 4. Learning and Performance Curves

As Figures 4 and 5 indicate, the variable-sequence selector was consistently better than both the minimum fixed-sequence selector and the maximum fixed-sequence selector during learning ($r = 0.36$) and during testing ($r = 0.25$). It also substantially outperformed the baseline random selector ($r = 0.09$). The

performance gain, however, comes at the price of additional computation space and time. On average, the resulting instance base is about w (size of window) times larger than that of the fixed-sequence selectors; each run took about w times longer on a Cray 6400.

7 Related and Future Work

PRODIGY/ANALOGY retains plans from prior planning problems, and uses derivational analogy (Carbonell, 1986) to guide the search of similar planning problems (Veloso, 1994). When PRODIGY's general planning method (Carbonell et al., 1992) was enhanced by derivational analogy, it could solve more problems and solve them with less effort. PRODIGY/ANALOGY, however, relied heavily on domain-specific planning heuristics and annotated planning examples. As indicated earlier, our work is based on a weak method (e.g., IBL) that facilitates initial adaptation to a problem domain.

OBSERVER (Wang, 1996) learned the description of an operator (action) as a set of literals with a variation of the Version Space algorithm (Mitchell, 1977). OBSERVER could both create and repair plans with a STRIPS-like representation. It relied on explicit specification of the states in the search space, including the goal states. It is not clear, even to professional bridge players, however, just what features would exhaustively catalogue the 10^{44} states in bridge play.

Moore's system (Moore, 1990) uses a form of reinforcement learning on a sequence of situation-action pairs to learn the effects of an action for robot arm control. It divides the search space into hierarchical segments to limit the search. It matches only the current state description to select an action, whereas SIBL matches a sequence of state descriptions to select an action. In a domain, such as bridge, which is sensitive to sequential relationships, we demonstrate here that a sequence of states is more advantageous.

GINA (Dejong and Schultz, 1988) was a program that learned to play Othello from experience, with a sequential list of situations encountered, actions taken, and final outcome. The representation of its experience base, however, is a subtree of the min-max game tree. Because the representation for the average branching factor in bridge is so much higher than Othello's, this approach seems impractical here.

This is work in progress. These initial experiments demonstrate that sequential instances produce action selection knowledge, and that variable-length sequence selection outperforms minimum or maximum fixed- sequence selection. Our current research includes examining the impact of more than 24 examples on performance, and replacing majority voting with a relevance selection scheme that is based on *decision theory* (Russell and Norvig, 1995).

Acknowledgements

Thanks to David Aha and Cullen Schaffer for their comments on earlier drafts of this paper. This research was partly supported by the Bank of America Corporation.

References

Aamodt, A., and Plaza, E. (1994). Case-based reasoning: Foundation issues, methodological variations, and system approaches. *AI Communications*, 7(1).

Aha, D. W. (1992). Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithm. *International Journal of Man-Machine Studies*, 36, 267-287.

Aha, D. W., Kibler, D., and Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning*, 6, 37-66.

Alterman, R. (1988). Adaptive planning. *Cognitive Science*, 12, 393-421.

Carbonell, J. G., Blythe, J., Etzioni, O., Gil, Y., Joseph, R., Kahn, D., Knoblock, C., Minton, S., P'erez, A., Reilly, S., Veloso, M., and Wang, X. (1992). *PRODIGY4.0: The manual and tutorial* (TR CMU- CS-92-150), CarnegieMellon University.

Carbonell, J. G. (1986). Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. in Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., eds., *Machine Learning, An AI Approach*, V. II, 371-392. Morgan Kaufmann.

Cost, S., and Salzberg, S. (1993). A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning*, 10, 57-78.

Dasarathy, B. V. (1991). *Nearest Neighbor (NN) Norms: NN Pattern Classification Technique*. IEEE Computer Society Press.

DeJong, K. A., and Schultz, A. C. (1988). Using experience-based learning in game playing. *Fifth International Conference on ML*, Ann Arbor, Michigan, 284-290.

Goren, C. H. (1963). *Goren's Bridge complete: a major revision of the standard work for all bridge players*. London: Barrie and Rockliff.

Hammond, K. J. (1989). *Case-based planning: viewing planning as a memory task*, Academic Press, Inc., San Diego, CA.

Hendler, J., Tate, A., and Drummond, M. (1990). AI planning: Systems and techniques. *AI Magazine*, 11(2), 61-77.

Kibler, D., and Langley, P. (1988). Machine learning as an experimental science. *Machine Learning*, 3(1), 5-8.

Marks, M., Hammond, K. J., and Converse, T. (1988) Planning in an open world: A pluralistic approach. *Workshop on CBR*, Pensacola Beach, FL, 271-285.

Michalski, R. S. (1983). A theory and methodology of inductive learning. *Artificial Intelligence*, 20(2), 111-162.

Mitchell, T. M. (1977). Version spaces: A candidate elimination approach to rule learning. *Fifth International Joint Conference on AI*, Cambridge, MA, 305-310.

Moore, A. W. (1990). Acquisition of dynamic control knowledge for a robotic manipulator. *Seventh International Conference on ML*, Austin, TX, 244-252.

Quinlan, R. J. (1986). Induction of decision trees. *Readings in Machine Learning*, J. W. Shavlik and T. G. Dietterich, eds., Morgan Kaufmann, San Mateo, CA.

Russell, S. and Norvig, P. (1995). *Artificial Intelligence - A Morden Approach*. Englewood Cliffs, NJ: Prentice-Hall.

Turner, R. M. (1988). Opportunistic use of schemata for medical diagnosis. *Tenth Annual Conference of the Cognitive Science Society*.

Veloso, M. M. (1994). Flexible strategy learning: Analogical replay of problem solving episodes. *Proceedings of the Twelfth National Conference on Artificial Intelligence*, AAAI Press/MIT Press, Cambridge, MA, 595-600.

Wang, X. (1996). Planning while learning operators. *Third International Conference on Artificial Intelligence Planning Systems*, Edinburgh, Scotland.