# Sequential 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
695 Park Avenue, New York, NY 10021
epstein@roz.hunter.cuny.edu
[2]Department of Computer Science
The Graduate School of The City University of New York
33 West 42nd Street, New York, NY 10036
jshih@broadway.gc.cuny.edu

**Abstract.** This paper presents and evaluates sequential instance-based learning (*SIBL*), an approach to action selection based upon data gleaned from prior problem solving experiences. SIBL learns to select actions based upon sequences of consecutive states. The algorithms rely primarily on sequential observations rather than a complete domain theory. We report the results of experiments on fixed-length and varying-length sequences. Four sequential similarity metrics are defined and tested: distance, convergence, consistency and recency. Model averaging and model combination methods are also tested. In the domain of three no-trump bridge play, results readily outperform IB3 on expert card selection with minimal domain knowledge.

## 1. Introduction

In domains where a solution is a sequence of decisions, experts often address problems with *action patterns*, contiguous subsequences of those decisions that establish subgoals (e.g.,(Korf 1990)). These macros are purposeful and often named. For example, a finesse in bridge is an action pattern, part of a plan intended to win. Our thesis is that reusable action patterns can be inductively learned from sequences of expert decisions. Specifically, in game playing, a set of sequences from previously played contests can yield action patterns that accurately select expert choices in a new contest. The principal contributions of this paper are a general method to learn action patterns called *SIBL* (Sequential Instance-Based Learning), and a demonstration of its efficacy for three no-trump card play in the game of bridge.

Game playing may be viewed as a planning problem, with an initial state, one or more goal states and a set of operators that transforms one state into another. A plan consists of an ordered set of operators that transforms the initial state into a goal state. Some planning approaches, such as non-linear and hierarchical planners, rely on complete domain knowledge to derive plans (Hendler, Tate, & Drummond 1990). Some learning methods that readily enhance planning, such as explanation-based learning (*EBL*), also assume complete domain knowledge (Minton 1985). Others, such

as case-based planners, reuse past planning experience through matching and adaptation but require large quantities of knowledge (Alterman 1988; Hammond 1989; Kambhampati 1990). SIBL, the inductive learning approach described here, addresses planning tasks but does not require extensive domain knowledge. Instead, it reuses information about past solutions to identify likely sequences of actions.

Because SIBL is intended to rely on relatively little domain knowledge, it could require substantial quantities of data. Rather than collect thousands of expertly-played bridge deals, we perturbed 72 input deals into about 90,000 training instances. We then used SIBL (and a modicum of domain knowledge, described in Section 7), to mine the resultant database for action patterns to direct expert play.

The next section provides a foundation for work with action patterns. Section 3 explains variants of SIBL and the need for good matching heuristics. Section 4 details four metrics developed for SIBL on matches between sequences of states. Section 5 describes an application of SIBL to the game of bridge, and provides results. The final sections survey related and future work.

## 2. Sequential Dependency

This section defines relevant terms and considers the extraction from problem solving experiences of sequences that could be used to support expert decisions. Consider, then, a problem solving experience

$$\begin{array}{ccccccccccccc} & & a_1 & & a_2 & & a_{i-1} & & a_i & & a_{i+1} & & a_{n-1} & \\ s_1 & \rightarrow & s_2 & \rightarrow & \ldots & \rightarrow & s_i & \rightarrow & s_{i+1} & \rightarrow & \ldots & \rightarrow & s_n \end{array} \quad \textbf{(1)}$$

represented as a sequence of $n$ states and $n$–1 actions, where the expert's action $a_i$ moves the problem state from $s_i$ to $s_{i+1}$ for $i$ = 1, 2, …, $n$-1. Each such experience contains many *transition sequences*, contiguous subsequences of states and actions within the experience that end in a particular state from which an action will be taken. In the notation of (1), a transition sequence is of the form:

$$\begin{array}{ccccccc} & a_j & & a_{j+1} & & a_{i-1} & \\ s_j & \rightarrow & s_{j+1} & \rightarrow & \ldots & \rightarrow & s_i \end{array} \quad \textbf{(2)}$$

Traditional learning methods assume that $a_i$, the action to be selected from state $s_i$, is dependent upon the nature of $s_i$, and therefore those methods seek features within $s_i$ that mandate the selection of $a_i$. *Sequential dependency*, however, postulates that the reasons for the selection of $a_i$ actually reside within a broader context, the particular sequence of states $s_j$, $s_{j+1}$, …, $s_i$ that precede $a_i$. SIBL seeks to learn that broader context.

Let a *sequential instance*, or *s-instance*, be a sequence of states followed by an action $a$, denoted:

$$s_j, s_{j+1}, \ldots, s_i \Rightarrow a \quad \textbf{(3)}$$

Observe that an s-instance describes both the current state $s_i$ and some number of consecutive, immediately prior states $s_j$, $s_{j+1}$, …, $s_i$, but that the s-instance omits the intermediate actions of the transition sequence (2). (Assuming an adequate state description, an intermediate action could be deduced from the differences between two consecutive states.) The *length* of an s-instance is the number of states it includes. Thus the length of (3) is $i–j+1$.

Given a transition sequence like (2), the *chronological expansion* of it is the set of s-instances terminating in the action that leads to *s*. For example, in the experience

$$\begin{array}{ccccccccc} & a_1 & & a_2 & & a_3 & & a_4 & \\ s_1 & \rightarrow & s_2 & \rightarrow & s_3 & \rightarrow & s_4 & \rightarrow & s_5 \end{array} \quad \textbf{(4)}$$

the chronological expansion of $s_4$ is three s-instances:

$$\begin{aligned} s_3 &\Rightarrow a_3 \\ s_2 s_3 &\Rightarrow a_3 \\ s_1 s_2 s_3 &\Rightarrow a_3 \end{aligned} \quad \textbf{(5)}$$

We call each of the s-instances in the chronological expansion of a transition sequence a *partial s-instance*. Clearly, if the transition sequence is at the *k*th state in a problem solving experience, then there are *k*-1 partial s-instances in its chronological expansion.

Each partial s-instance offers a different context that may have determined the selection of action *a*. In (5), for example, all of $s_1$ $s_2$ $s_3$ may have determined the choice of $a_3$, or perhaps only $s_2$ $s_3$, or simply the nature of $s_3$ alone. It may also be necessary, for computational efficiency, to limit consideration to the *l* most recent partial s-instances. SIBL learns and makes decisions based on sequential dependency.

## 3. SIBL

SIBL is an extension to instance-based learning (*IBL*). IB3, the version of IBL we use here, represents input examples and output concept descriptions as feature-value pairs, and ordinarily retains the prototypical examples for reuse (Aha 1992; Aha, Kibler, & Albert 1991) . When IB3 is used for planning from state *s*, it retrieves the best-matching *simple instance* of the form $s_i \Rightarrow a_i$ from its database, and applies the action of the recommended instance, $a_i$. SIBL is an IB3 approach that applies sequential dependency to decision making, using a database of s-instances like (3) instead of simple instances.

Figure 1 is an overview of learning with SIBL. It shows how a training instance is expanded into a set of one or more partial s-instances that are then matched against a database. Unless the database's best match predicts the training action, the database is updated with the partial s-instances derived from the training instance.

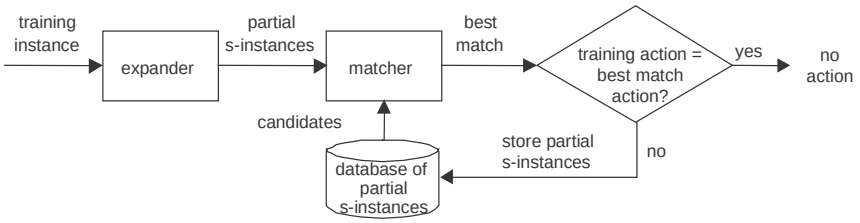


**Figure 1.** An overview of learning in SIBL

Table 1 gives high-level pseudocode for the SIBL algorithms. The length of the s-instances to be learned is an important issue. Is there a "best" length partial s-instance or does the length of the best partial s-instance vary from one decision to another? Although some actions may depend upon the entire sequence of states that preceded them, others may not require a complete chronological sequence to interpret the current situation. The relevant sequence of events may be shorter, even of length one. Table 1 therefore contains three approaches, one that assumes a parameterized best length *l*, and two that accommodate varying lengths.

Each approach incrementally constructs a database *DB* from a set of problem solving experiences, and uses it to make decisions. All three approaches begin the same way: each problem solving experience like (1) is expanded into training instances, s-instances like (3) where the expert made a decision. Here, the three methods deviate.

### 3.1 Fixed-length SIBL

For fixed-length partial s-instances, SIBL-learn-fixed in Table 1 retrieves from the database *DB* the most similar s-instance of length *l*. For example, given training data

$$s_1, s_2, s_3, s_4 \Rightarrow a \qquad \textbf{(6)}$$

and $l = 2$, SIBL-learn-fixed would select from *DB* the s-instance most similar to $s_3\, s_4$, using the distance metric described in Section 4.1 below. Unless the action the retrieved s-instance recommends is identical to the action in the training instance, the database is updated with the training instance. For $l = 1$, SIBL-learn-fixed is IB3.

Once *DB* has been constructed, SIBL makes decisions based upon it. In Table 1, SIBL-decide-fixed merely retrieves the most similar s-instance of length *l* and mandates that action. For example, given the training data in (6) and $l = 3$, SIBL-decide-fixed recommends the action taken by the s-instance in *DB* most similar to $s_2\, s_3\, s_4$. Similarity here is measured by distance, defined in Section 4.1 below.

**Table 1.** The SIBL algorithms.

**SIBL-learn-fixed** ($s_1$, $s_2$, …, $s_n \Rightarrow a$, *DB*, *l*)
**unless** FindMostSimilar ($s_{n-l+1}$, …, $s_n$, *DB*) = $a$
  append $s_1$, $s_2$, …, $s_n \Rightarrow a$ to *DB*

**SIBL-decide-fixed** ($s_1$, $s_2$, …, $s_n$, *DB*, *l*)
**return** FindMostSimilar ($s_{n-l+1}$, …, $s_n$, *DB*)

**SIBL-learn-majority-vote** ($s_1$, $s_2$, …, $s_n \Rightarrow a$, *DB*)
**for** *i* from 1 to *n*
  collect FindMostSimilar ($s_i$, $s_{i+1}$, …, $s_n$, *DB*) into *C*
**for** each candidate $c = s'_c$, $s'_{c+1}$, …, $s'_n \Rightarrow a_c$ in *C* and its recommended action $a_c$
  vote($a_c$) ← vote($a_c$) + 1
**unless** $a = a_c$ where vote($a_c$) is a maximum
  **for** *i* from 1 to *n*
    append $s_i$, $s_{i+1}$, …, $s_n \Rightarrow a$ to *DB*

**SIBL-decide-majority-vote** ($s_1$, $s_2$, …, $s_n$, *DB*)
**for** *i* from 1 to *n*
  collect FindMostSimilar ($s_i$, $s_{i+1}$, …, $s_n$, *DB*) into *C*
**for** each candidate $c = s'_c$, $s'_{c+1}$, …, $s'_n \Rightarrow a_c$ in *C* and its recommended action $a_c$
  vote($a_c$) ← vote($a_c$) + 1
**return** $a_c$ where vote($a_c$) is a maximum

**SIBL-learn-SSM** ($s_1$, $s_2$, …, $s_n \Rightarrow a$, *DB*)
**for** *i* from 1 to *n*
  collect FindMostSimilar ($s_i$, $s_{i+1}$, …, $s_n$, *DB*) into *C*
**unless** FindMostSimilarCandidate ($s_1$, $s_2$, …, $s_n$, *C*) = $a$
  **for** *i* from 1 to *n*
    append $s_i$, $s_{i+1}$, …, $s_n \Rightarrow a$ to *DB*

**SIBL-decide-SSM** ($s_1$, $s_2$, …, $s_n$, *DB*)
**for** *i* from 1 to *n*
  collect FindMostSimilar ($s_i$, $s_{I+1}$, …, $s_n$, *DB*) into *C*
**return** FindMostSimilarCandidate ($s_1$, $s_2$, …, $s_n$, *C*)

**FindMostSimilar** ($s_i$, $s_{i+1}$, …, $s_n$, *DB*)
$\sigma \leftarrow s_i$, $s_{i+1}$, …, $s_n$
**for** each partial s-instance *s* in *DB* of length $n - i + 1$
  minimize distance($\sigma$, *s*) as smallest
**return** action of smallest

**FindMostSimilarCandidate** ($s_1$, $s_2$, …, $s_n$, *C*)
**if** candidate *c* in *C* has significant minimum distance($s_1$, $s_2$, …, $s_n$, *c*),
  **then return** its recommended action $a_c$
**else if** candidate *c* in *C* has significant maximum convergence($s_1$, $s_2$, …, $s_n$, *c*),
  **then return** its recommended action $a_c$
**else if** candidate *c* in *C* has significant maximum consistency($s_1$, $s_2$, …, $s_n$, *c*),
  **then return** its recommended action $a_c$
**else if** candidate *c* in *C* has significant maximum recency($s_1$, $s_2$, …, $s_n$, *c*),
  **then return** its recommended action $a_c$
**else** select a random candidate *c* in *C*, **return** its recommended action $a_c$

### 3.2 Varying-length SIBL

For decisions rely on varying-length s-instances, there are two methods in Table 1: one relies on a voting process and the second on similarity metrics. Both methods chronologically expand each s-instance into a set of partial s-instances. Then, for each partial s-instance in each chronological expansion, both methods retrieve a *candidate* (the most similar partial s-instance of the same length as the newly expanded partial s-instance) from the partial s-instances already recorded in *DB*. Since a current experience with $k$ states gives rise to $k$–1 partial s-instances, there will be $k$–1 candidates, each a partial s-instance of a different length. For example, if (6) is a training instance and we restrict partial instances to at most length $l$ = 3, SIBL identifies a candidate of length one to match $s_4$, a second of length two to match $s_3$,$s_4$, and a third of length three to match $s_2$,$s_3$,$s_4$. Both varying-length SIBL methods collect these candidates along with the actions they recommend. They differ, however, in how they learn and decide with these candidates.

#### 3.2.1 Majority vote

As described above, for a given s-instance SIBL-learn-majority-vote retrieves the best matching candidates of varying lengths from *DB*. Then each partial s-instance retrieved by FindMostSimilar casts one vote for the action it recommends. If the action that receives the *majority vote* (the most support) among the different length candidates matches the action to be learned, no change is made to *DB*. Otherwise, each of the partial s-instances in the expansion is appended to *DB*. Given the training data in (6) and $l$ = 3, for example, SIBL-learn-majority-vote would retrieve three candidates and vote for the actions they indicate. Unless the best supported action were $a$, all three partial s-instances would be added to *DB*.

Once the database *DB* has been constructed by SIBL-learn-majority-vote, to make a decision SIBL-decide-majority-vote retrieves the candidates from *DB*. Each candidate casts one vote for the action it recommends, and the action that receives the majority vote is selected. Given the training data in (6) and $l$ = 3, for example, SIBL-learn-majority-vote would retrieve three candidates, vote for the actions they indicate, and select the best supported action. In the event of a tie, both the training and testing majority vote algorithms choose at random from among the top-ranked actions.

#### 3.2.2 Sequential similarity metrics

As described above, for a given s-instance SIBL-learn-SSM retrieves the best matching candidates of varying lengths from *DB*. Next, SIBL-learn-SSM uses FindMostSimilarCandidate in Table 1 to identify the *best-matching* candidate, the one whose state sequence is most similar to the training instance. The metrics that measure the quality of a match between two partial s-instances are called *sequential similarity metrics* and defined in the next section. In the example of (6) with $l$ = 3, some candidate of length $1 \le l \le 3$ is identified. If the best-matching candidate's recommended action already corresponds to the action taken in the training instance, no change is made to the database. When the database's best match, however,

recommends an action different from that taken by the expert in the training instance, each of the partial s-instances in the expansion is appended to *DB*.

Once the database *DB* has been constructed by SIBL-learn-SSM, to make a decision SIBL-decide-SSM retrieves the candidates from its database. Then the algorithm selects the most similar candidate and returns its recommended action. The next section describes the sequential similarity metrics that underlie this method.

## 4. Metrics for Sequential Similarity

FindMostSimilarCandidate in Table 1 applies up to four metrics to select the best matching partial s-instance of length *n* in *DB* to the training s-instance of length *n*. Each metric is defined for two same length, partial s-instances whose states are represented by a finite set of *v* features. To avoid bias toward shorter s-instances, every metric is normalized to return values in [0,1]. For clarity, normalization is omitted from the examples shown here.

A metric *m* is said to *distinguish* between two s-instances *s* and *s′* if and only if $m(s) \neq m(s')$ to some precision, say, the nearest tenth. In the order of presentation below, each metric is a refinement of the one that precedes it. For example, convergence is a way to distinguish between two partial s-instances with equal distance. FindMostSimilarCandidate applies these metrics one at a time to a set of same-length candidates. The first metric to distinguish among the different-length candidates returns its choice. We report the frequency to which they were resorted in Section 5.

Throughout, we let one partial s-instance be

$$s = s_p \rightarrow s_{p+1} \rightarrow \ldots \rightarrow s_k \tag{7}$$

and the second be

$$s' = s'_p \rightarrow s'_{p+1} \rightarrow \ldots \rightarrow s'_k \tag{8}$$

### 4.1 Distance

The *distance metric* quantifies the feature-based differences between the individual states in *s* and *s′* as:

$$distance(s, s') = \sqrt{\sum_{i=1}^{v} difference_i^2(s, s')} \tag{9}$$

The *difference* between the values of feature $f_i$ in *s* and *s′* is defined as:

$$difference_i(s, s') = \begin{cases} |f_i(s) - f_i(s')| & \text{if } f_i \text{ is numeric} \\ 0 & \text{if } f_i \text{ is discrete and } f_i(s) = f_i(s') \\ 1 & \text{otherwise} \end{cases} \tag{10}$$

In FindMostSimilarCandidate, the candidate $s'$ with minimal distance to the same length, partial s-instance $s$ of the current experience is selected.

Table 2 is a hypothetical example where $E = s_1 s_2 s_3$ is the current experience, and $Y_i$ is the stored s-instance of length $i$ = 1, 2, 3, the best candidate of length $i$ for the partial s-instance $s_{3-i+1} \ldots s_3$ of $E$. If the distances between the partial s-instances of $E$ and the stored s-instances were given as 0.67, 0.69 and 0.68, FindMostSimilarCandidate could select action $A_1$ for $E$, because $Y_1$ is the most similar (least distant) stored partial s-instance to $E$.

**Table 2.** Applying sequential similarity metrics.

| | $Y_1 \Rightarrow A_1$ | $Y_2 \Rightarrow A_2$ | $Y_3 \Rightarrow A_3$ |
|---|---|---|---|
| **Distance to *E*** | 0.67 | 0.69 | 0.68 |
| **Convergence to *E*** | 0.55 | 0.58 | 0.56 |
| **Consistency with *E*** | 1 | 1 | 2 |
| **Recency to *E*** | 1 | 1 | 2 |

### 4.2 Convergence

Although the distance metric in Table 2 shows $Y_1$ as most similar to $E$, all three values round to 0.7, so FindMostSimilarCandidate would not distinguish among the candidates to the nearest tenth with the distance metric. The algorithm therefore resorts to a second metric. The *convergence metric* quantifies the change in the distance between partial s-instances, measured from their first states to their last states. Convergence is the change in distance between s and s′ from one end to the other:

$$convergence(s, s') = distane(s_p, s'_p) - distance(s_k, s'_k) \quad (11)$$

In SIBL, the candidate $s'$ with maximal convergence to the same length, partial s-instance $s$ of the current experience is selected. Continuing the example in Table 2, since the convergence values between $E$ and the candidates are 0.55, 0.58 and 0.56, FindMostSimilarCandidate could select action $A_2$ for $E$, because $Y_2$ is the most similar (most convergent) stored partial s-instance to $E$.

### 4.3 Consistency

Convergence, like distance, may not adequately discriminate among the candidates, as shown in Table 2 to the nearest tenth. Therefore, in the ongoing example FindMostSimilarCandidate would now resort to a third metric. For each partial s-instance of the current experience and the candidate retrieved for it, the *consistency metric* tallies the maximum number of consecutive non-increases in distance between the states in the current experience and the states in each candidate. Consistency between s and s′ is measured by:

$$consistency(s, s') = t \text{ such that}$$
$$distance(s_j, s'_j) \geq distance(s_{j+1}, s'_{j+1}) \text{ for } j = m \text{ to } m + t \text{ in } [p, k] \text{ where} \quad (12)$$
$$p \leq j \leq k \text{ and } t \text{ is the largest such value}$$

In FindMostSimilarCandidate, the candidate $s'$ with maximal consistency to the same length, partial s-instance $s$ of the current experience is selected. In Table 2, continuing the example, since the consistency values between $E$ and the candidates are 1, 1, and 2, FindMostSimilarCandidate could select action $A_3$ for $E$, because $Y_3$ is the most similar (most consistent) stored partial s-instance to $E$.

### 4.4 Recency

When consistency fails to discriminate adequately among candidates, FindMostSimilarCandidate resorts to the *recency metric* that identifies the latest point where distance did not increase between consecutive states. Recency between s and s′ is measured by:

$$recency(s, s') = \text{the largest } j \text{ in } [p, k] \text{ such that}$$
$$distance(s_j, s'_j) \geq distance(s_{j+1}, s'_{j+1}) \quad (13)$$

If, for example, all the consistency values in Table 2 had been 1 but the recency values between $E$ and the candidates 1, 1, and 2, FindMostSimilarCandidate would select action $A_3$ for $E$, because $Y_3$ is the most similar (most recent) stored partial s-instance to $E$.

## 5. Experimental Design and Results

The domain of investigation for this work is card play in the game of bridge, a four-player planning domain. A bridge *deal* distributes 52 distinct cards equally among the players into four *hands*. There are 13 cards in each of four suits; in increasing order of strength: 2, 3, 4, 5, 6, 7, 8, 9, 10, jack, queen, king, ace. The game has two phases: bidding and play. During *bidding*, the *contract* (a specific number of tricks for winning and a trump suit) is determined. During *play*, one contestant, identified by the bidding, is the *declarer* and another contestant, sitting opposite the declarer, is the *dummy*. The declarer tries to achieve the contract, controlling both declarer's and dummy's cards, 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 tricks; a *trick* is constructed when each contestant in turn plays a single card. The first card's suit in the trick is the suit *led*. This work is restricted thus far to declarer play for three no-trump contracts, where the highest card in the suit led takes the trick, and the declarer is expected to take at least nine tricks. The problem addressed here is to select a sequence of actions (card plays) that enables the declarer to reach the contract.

A bridge state is a situation in which someone is expected to select a card, the *target feature*. We represent such a state as a set of 23 feature-value pairs. The features describe the suit strengths held by declarer and dummy, the suit strengths

played by each of the opponents in all previous tricks, the cards played thus far in the current trick, who must play the next card, the target feature, and whether or not the declarer wins the trick. Since high cards are most likely to win tricks, cards below 9 are considered indistinguishable and are represented by the symbol X.

A bridge deal is a problem-solving experience, a sequence of 53 states and 52 actions, 26 of which are situations where the declarer had to play a card from its own or the dummy's hand. Our program calculates the chronological expansion of each training instance as a set of partial s-instances of various lengths in the context of the deal where the training instance appears. Every s-instance of length five or less is then represented as a sequence of one or more states plus a selected card. (Five was chosen as an upper bound based on preliminary experiments.) Thus an s-instance represents the current state and zero or more (up to four) states immediately prior to the current state.

Our data began as 108 three no-trump bridge deals, each fully played by human experts who successfully made the contract. We first separated the deals into training (72 deals) and test (36 deals) sets. In any given deal, however, there are 26 decision situations, and 325 partial s-instances of lengths from 1 to 25. Furthermore, each partial s-instance can be permuted (by varying its suits, declarer and dummy) into 48 partial s-instances. At this point, when every deal yields 1248 ($26 \cdot 48$) training instances and 15,600 partial s-instances, computing resources become a consideration. We therefore divided the training set into three smaller ones.

Each smaller training set of 24 deals was used to produce a database, which we call here a *model*. Thus each model was based on almost 30,000 ($24 \cdot 26 \cdot 48$) training instances, and could be tested on the $26 \cdot 36 = 936$ withheld instances in the testing set. A *run* randomized the order of the training instances, and then a decision maker learned on them. To gauge the learner's development and the value of continued training, after each tenth of the training set was presented, the program was tested on the 936 testing instances with learning turned off. An *experiment* averaged the results over 10 runs for a decision maker.

We ran experiments for decision makers based upon each of the following models:

- Length-one instances (IB3).
- Fixed-length instances for $l = 2$, 3, 4 and 5
- Majority vote among s-instances of lengths $l \leq 5$ learned with SIBL-learn-majority-vote for $l \leq 5$.
- *Model averaging*, where a single model built with SIBL-learn-SSM was used for decision making, but performance reported as the average of the performance of the three individual models.
- *Model combination*, where the three models built with SIBL-learn-SSM voted equally to make a decision whose performance was monitored.

All fixed-length models were built with SIBL-learn-fixed. As a benchmark, we also tested random legal (suit-following) play.

We tested each SIBL decision maker on the 936 withheld instances, noting how often it selected the correct (expert's move) card. If the program selected a card in a different suit, the result was scored as 0 (error). If the program selected the identical card, the result was scored as 1 (correct). If the program selected a card in the same suit as the correct card and consecutively adjacent to the target feature, the result was also considered correct and scored as 1. (For example, 10, jack, and queen in the same

suit are consecutively adjacent and have the same effect on a trick.) Because the database is constructed from different deals, the recommended action may not always be legal in the current state, that is, SIBL may recommend a card not held but in the correct suit. Therefore, if the card SIBL selected was in the same suit as the correct card but neither identical nor consecutively adjacent, the result was assigned a numeric value between 0 and 1 based on the distance between the correct card and the card recommended by SIBL. In this case, the program plays the closest holding to the card SIBL recommends, and if two cards are equally close, the higher. With six possible actions in a suit (ace, king, queen, jack, 10, 9 and X), the difference to the adjacent card was scored as 5/6, to two adjacent cards as 4/6, and so on.

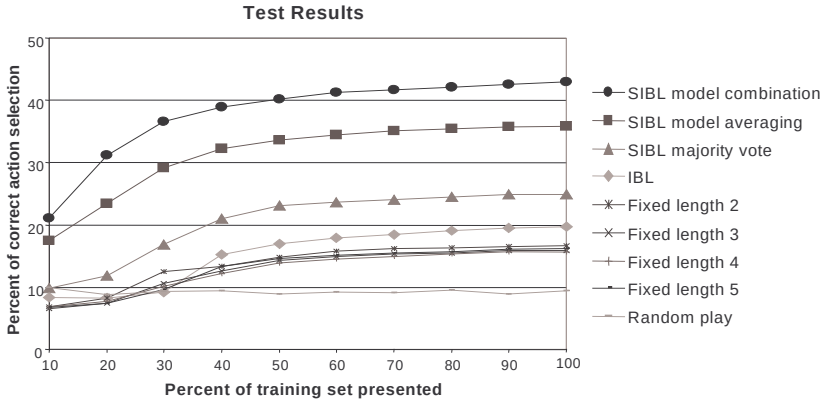


**Figure 2.** SIBL test results

Figure 2 shows how often each decision maker selected the correct action. The horizontal axis indicates the percentage of the training set seen before testing; the vertical axis shows the percentage of correct decisions. All the instance-based learners were substantially better than random play. Among the fixed-length SIBL experiments, after learning on all the training instances, $l = 1$ (IB3) outperformed all longer fixed-length sequences at the 99% confidence level. All three versions of SIBL outperformed IB3 at the 99% confidence level. Both model averaging and model combination with sequential similarity metrics were consistently better than majority vote. In addition, model combination outperformed (43% correct) model averaging (36% correct).

The sequential similarity metrics did indeed refine action selection. 80% of the decisions were made on distance alone. 17% of the time, distance did not distinguish among the candidates but convergence did. 2% of the decisions were made on consistency, and the final 1% relegated to recency or random selection among the most recent. Would more training (i.e., additional hands) have provided even better performance? The curves in Figure 2 (except the random play benchmark) appear to flatten in the last 10-20% of training. This suggests that 72 hands, as permuted, may be enough.

## 6. Related Work

Projective Visualization (*PV*) projected future states from the current one by following a set of state links (Goodman 1994) . The link between two states was defined by a set of state features that associated the two states. Initially, the state features were built one at a time with decision tree algorithms. At runtime these decision trees were used to find matching states with the same features. Like SIBL, PV tried to learn the state transition patterns. PV, however, used only lookahead information, with the risk of compounding projection errors. PV did not take advantage of historical state transition information, nor did it exploit the relative strengths of transition sequences of various lengths. The primary domain of investigation was Bilestoad, a computer game between two gladiators.

CAP learned action patterns with constructive induction. CAP's representation was a subset of Horn clause logic (Hume 1990) . It defined primitive properties of objects, relations, and actions for the target action pattern. A logical expression described a sequence of actions. Like EBL, given the basic definitions of a domain, CAP could learn domain concepts from a few descriptive examples. Unlike SIBL, CAP relied on a well-defined logical representation to assist learning domain concepts. The primary domain of investigation was mountaineering.

Moore's system used a form of reinforcement learning on a sequence of situation-action pairs to learn the effect of an action for robot arm control (Moore 1990) . It divided the search space into hierarchical segments to limit the search. It matched 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, that is sensitive to sequential relationships, a sequence of states is more advantageous.

GINA was a program that learned to play Othello from experience, with a sequential list of situations encountered, actions taken, and final outcome (DeJong & Schultz 1988) The representation of its experience base was a subtree of the min-max game tree. Bridge, however, has concealed information (the opponents' hands), that makes its branching factor much higher than Othello's, and GINA's approach less practical.

SIBL is a general method to learn action patterns, that is, to reuse stored expert knowledge to make decisions in similar subsequent situations. An expert bridge program should also rely on substantial domain knowledge and a model of its opponents. SIBL could ultimately be one component in such a program. For these reasons, we do not compare our program's decision making skill to other expert bridge programs' here.

## 7. Conclusion

Although there are about $5 \cdot 10^{28}$ possible decision states in bridge, the strongest version of SIBL described here achieves more than twice the accuracy of IB3 on card play after training on only 72 deals. The only domain knowledge provided to the program was the need to follow suit, the ranking of the cards in a suit, and a preference for a higher card in the same suit when SIBL's recommendation was not present in the hand.

Bridge play includes units of four decisions (tricks), yet no fixed-length SIBL method performed as well as IB3 (length-1 sequences) did. This is probably due to the nature of the domain, that is, bridge play is dependent on recent experience, but the extent of the relevant recent experience varies. This theory is supported by the fact that the varying-length SIBL methods substantially outperformed IB3. Majority vote from a database of varying-length, rather than fixed-length, sequences is correct 27% more often than IB3. With the similarity metrics defined here and model combination, SIBL makes correct decisions 118% more often than IB3.

These experiments demonstrate that SIBL is an effective way to detect action patterns in bridge play, and represents a substantial improvement over IB3. As described here, SIBL seeks a match in its database for past states. In that sense, it only completes plans that are already underway. Current research enhances SIBL with planning-oriented lookahead.

## References


Aha, D. W. 1992. Tolerating Noisy, Irrelevant and Novel Attributes in Instance-Based Learning Algorithms. *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.

DeJong, K. A. and Schultz, A. C. 1988. Using Experience-Based Learning in Game Playing. In *Proceedings of the Fifth International Machine Learning Conference*, 284-290. Ann Arbor, Michigan: Morgan Kaufmann, San Mateo.

Goodman, M. 1994. Results on Controlling Action with Projective Visualization. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1245-1250.

Hammond, K. J. 1989. *Case-based Planning: Viewing Planning as a Memory Task* . Boston: Academic Press.

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

Hume, D. V. 1990. Learning Procedures by Environment-Driven Constructive Induction. In *Proceedings of the Seventh International Conference on Machine Learning*, 113-121. Austin: Morgan Kaufmann.

Kambhampati, S. 1990. Mapping and Retrieval during Plan Reuse: A Validation Structure Based Approach. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 170-175. Boston: AAAI Press.

Korf, R. 1990. Real-Time Heuristic Search. *Artificial Intelligence*, 42 (2-3): 189-211.

Minton, S. 1985. Selectively Generalizing Plans for Problem-Solving. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 596-599. Los Angeles.

Moore, A. W. 1990. Acquisition of Dynamic Control Knowledge for a Robotic Manipulator. In *Proceedings of the Seventh International Conference on Machine Learning*, 244-252. Austin.