An Evolutionary Algorithm Extended by Ecological Analogy and its Application to the Game of Go

Takuya Kojima **Kazuhiro Ueda** **Saburo Nagano**
College of Arts and Sciences
The University of Tokyo
3-8-1 Komaba, Meguro-ku, Tokyo, 153, JAPAN
kojima@taikan.c.u-tokyo.ac.jp

Abstract

The following two important features of human experts' knowledge are not realized by most evolutionary algorithms: one is that it is various and the other is that the amount of knowledge, including infrequently used knowledge, is large. To imitate these features, we introduce an activation value for individuals and a new variation-making operator, splitting, both of which are inspired by ecological systems. This algorithm is applied to the game of Go and a large amount of knowledge evaluated as appropriate by a human expert is acquired. Various kinds of Go knowledge may be acquired such as patterns, sequences of moves, and Go maxims, part of which has already been realized.

1 Introduction

The knowledge of human experts has two important features. One is that it is full of *variety*; humans can have many types of knowledge, such as verbal or visual, and there is little limitation in size and shape. The other is that it can be a *large amount*; it is estimated that a chess master has between 10,000 and 100,000 chunks of knowledge [Simon and Gilmartin, 1973]. As a consequence, experts also have knowledge which is *infrequently used*.

In contrast, knowledge acquired by most evolutionary algorithms, such as Genetic Algorithm (GA), has two features. One is that it is *inflexible*; knowledge representation is uniform such that the length of all strings is unique. The other is that in most cases acquired knowledge is limited to one kind; all rules tend to be similar [Goldberg, 1989] in most algorithms. Moreover, algorithms can only acquire knowledge which is frequently used. These two characteristics of algorithms are far from those of human experts.

Among evolutionary algorithms, Genetic Programming (GP) overcomes the first weakness; it can use programs as representation, which are far flexible representation than that used by other major evolutionary algorithms. However, GP does not overcome the second weakness; all the individuals tend to be similar.

We choose ecological systems as a new source of ideas. In ecological systems, many species coexist in environment. We introduce an activation value for individuals and a new variation-making operator, splitting, in order that different individuals coexist.

This research aims to acquire a large number of flexible rules, including useful but infrequently used ones. This algorithm may help us to build a learning model in an evolutionary way which learns as people do.

This algorithm is applied to the game of Go, which has recently attracted considerable attention from AI researchers as the target next to chess [Kitano and others, 1993; McCarthy, 1990]. Games have been used as a testbed for AI algorithms because they are well-defined and easy to use to test the algorithms. Almost all the games studied so far have been so simple that the game playing systems are on an expert level only by using the search-intensive approach. Go is the only exception. Since Go has a much larger search space than chess, taking the search-intensive approach used for chess is not enough to make Go systems play as well as human experts. Thus, another approach in AI, the knowledge-intensive approach, should be considered. Go playing systems of this approach have a large amount of knowledge and search a small space resembling human experts [Saito, 1996]. Thus, Go is considered to be an appropriate testbed for the knowledge-intensive approach.

The knowledge-intensive approach is difficult to take because of the difficulty of acquiring a huge amount of knowledge. This algorithm aims to acquire large numbers of rules, making it possible to take the knowledge-intensive approach.

2 Explanation of this Algorithm

Since it is inspired by ecological systems, this algorithm is explained using ecological terms.

2.1 Overview of the Algorithm

This algorithm acquires a large number of rules in the form of production rules. Each rule is considered as an *individual* and has an *activation value*. There are no rules in the initial state. A training datum is considered as *food* which is eaten by any rule that matches it and the activation value of the rule increases. If there are no rules which match a given datum, a new rule is created which matches the datum. Thus, the number of rules increases at an early stage. Rules whose activation value is over a certain threshold *split* into the original rule and a more specific rule. Every rule eats food during each step and decreases one activation value. Rules whose activation value is 0 *die*. The procedure of this algorithm is shown below.

Algorithm 1 Outline of this Algorithm

```
1   step ← 1
    while step ≤ laststep
2       choose a datum from a training set
3       if no rule matches the datum
            then make a new rule
            else feed matched rules
        for all rules
4           if activation of a rule > {threshold}
                then split the rule
5           activation of a rule ← activation of a rule − 1
6           if activation of a rule = 0 then the rule dies
        end
7       step ← step + 1
    end
```

2.2 Details of the Algorithm

Rules take the form of production rules. They are Horn clauses; the condition part of each rule consists of a set of clauses and contains only $\wedge$ as a logical connective.

Matching and Feeding Rules

Matched rules are those whose condition part and action part match a given training datum. Those rules that do not have more specific rules than themselves are fed. An activation value equal to Parameter $FOOD$ is shared among the rules fed by one training datum. The following is an example.

Suppose that the following five rules are matched.

1. **IF** C_1 **THEN** A_1
2. **IF** $C_1 \wedge C_2$ **THEN** A_1
3. **IF** $C_2 \wedge C_3$ **THEN** A_1
4. **IF** $C_4 \wedge C_5$ **THEN** A_1
5. **IF** $C_2 \wedge C_3 \wedge C_4$ **THEN** A_1

Since Rules 1 and 3 have more specific rules than themselves (Rules 2 and 5), they are not fed. The others, Rules 2, 4 and 5, do not have more specific rules than themselves, so they share food and each gets one third of the food.

In summary, the more general the rules are, the more frequently they are matched, but the lower the probability is that they are fed when they are matched.

Splitting and Making Rules

A rule whose activation value is over a certain threshold splits into two rules; one is the original rule and the other is created by adding a new predicate to the condition part of the original (parent) rule. For example, a pattern, **IF** $C_1 \wedge C_2$ **THEN** A_1, splits into the original rule and a new pattern, **IF** $C_1 \wedge C_2 \wedge C_3$ **THEN** A_1. The new predicate, C_3, is randomly chosen as long as the newly created rule matches the given datum.

If no rule is matched, a new rule is created which has only one predicate chosen randomly in the condition part as long as the newly created rule matches the given datum.

When a new rule is created, its activation value is decided by a parameter, INI_ACT. When a rule is split, the activation value of the newly created rule is determined by INI_ACT and is subtracted from the activation of the parent rule.

2.3 General Description of the System Behavior

The following is the general description of the behavior of this system. In the initial state, this system has no rules. A training datum is given and if no rules match the given datum, a new rule having only one predicate in the condition part is created, thus many rules with only one condition are created in the early stages. A rule whose activation value is over a certain threshold splits into the original rule and a more specific rule created by adding a new predicate to the condition, so rules that often match data split frequently and more specific rules are created. The frequency of matching of a newly created rule is always less or equal to that of its parent rule, because the condition of the new rule is always more specific than that of its parent. Rules split and evolve into more specific rules, as long as the frequency of being fed is higher than a certain threshold. Therefore, in the stable state, all rules are expected to match data with almost the same frequency.

The number of rules is expected to be almost equal to the amount of $FOOD$. Let N be the number of rules. For each step, every rule loses one unit from its activation value, so the total loss in activation value is N; on the other hand, the total gain in activation value is $FOOD$.

3 Application to the Game of Go

We chose the game of Go as a testbed for the algorithm, because Go has a much larger search space than chess and knowledge is therefore indispensable for computer systems to play Go. Moreover, recent cognitive studies on Go players [Saito and Yoshikawa, 1996] have revealed that human experts use Go knowledge such as "patterns" and also search a game tree a little; this shows the importance of Go knowledge to human players. Therefore, studying acquisition of Go knowledge may help Go playing systems to play better.

Game knowledge can be classified into two categories: *strict knowledge* and *heuristic knowledge*. The former is the knowledge that can always be applied and the validity can easily be shown, such as the way to capture stones. The latter is the knowledge whose validity depends on situations, such as *tesuji*, a heuristic way of attack or defense. In Go, a search space is so huge and moves are so sensitive to situations that most of the knowledge used by human players is the heuristic knowledge.

In our previous studies [Kojima *et al.*, 1994; Kojima, 1995], we built a system that deductively acquires Go patterns. Although the system acquires reliable patterns, it can only acquire strict rules, not heuristic ones.

The two big problems in the present systems that acquire heuristic knowledge are that the acquired knowledge is localized and small, and that the knowledge representation is fixed. For example, one system [Sei and Kawashima, 1994] collects from master's game records patterns, which are fixed diamond shape and whose area is within five Manhattan distances of the center. Our algorithm makes it possible to acquire more flexible knowledge; patterns with any shapes and of unlimited size can be acquired. Thus, patterns acquired by our algorithm are more similar to those which human players have than those acquired by the previous systems.

3.1 Details of Application to Go

Go is a game between two players, who take turns putting black and white stones on a board. The standard board size is 19x19; sometimes a smaller board, such as 13x13 or 9x9, is used usually by beginners.

Training examples are game records between professional players. Two kinds of training sets are used: one is 19x19 board games (total 600 games) and the other is 9x9 board games (total 200 games). Training sets are chosen as follows. One game is randomly chosen and all the moves of the game are taken as training data[1] from the first move to the last move. After all the moves are used as training data, another game is randomly chosen. This process is repeated. Each move is one time step in this algorithm.

There are many kinds of Go knowledge, such as patterns, sequences of moves, and maxims. A pattern is a rule whose condition part is a part of board configuration and whose action part is a move. A sequence of moves is a rule that suggests several moves. A maxim is a rule consisting of Go terms. This paper describes the acquisition of patterns, so the rules mentioned in the most of the paper are Go patterns.

Rules are described in relative terms so they can be executed by either players. A predicate of the condition part consists of objects and their coordinates (*coordinate=object*); the action point is the point to which a stone is placed and has the coordinates [0,0]; other coordinates are relative to the action point, and objects are either stones or edges of the board. Stones are either "SAME" (i.e. belonging to the active player) or "DIFF" (i.e. belonging to the opponent). The action part of a rule is always "place a stone to [0,0]", so it is omitted in the following examples.

For example, "**If** [2,1]=SAME $\wedge$ [0,1]=DIFF $\wedge$ [5,0]=EDGE" means that **if** a *same* color stone as the active player's is in the point [2,1] (relative to the action point), and if a *different* color stone exists in point [0,1] and *edge* exists in point [5,0] **then** put a stone on [0,0].

When rules split or are newly created, a new condition about stones or edges of the board is chosen.

3.2 Parameters

This algorithm has two parameters: one is a constant, INI_ACT, which is the initial activation value for newly created rules, and the other is $FOOD$, a function of the number of games used as training data ($= i$), which is the total amount of activation given by one training datum.

In the experiment with 9x9 board games, $FOOD$ is defined as follows: if $i < 4000$, $FOOD = 400$, and if $i \geq 4000$, $FOOD = i \div 10$, and INI_ACT is 200. In the experiment with 19x19 games, $FOOD$ is defined as follows: if $i < 4000$, $FOOD = 2000$, and if $i \geq 4000$, $FOOD = i \div 2$, and INI_ACT is 1000.

The threshold of splitting is twice the value of $FOOD$.

Parameter INI_ACT is smaller than $FOOD$. A newly created rule is an assumption, thus it is often inappropriate and consequently dies, because a new condition is randomly chosen. Therefore, the activation value given to a newly created rule is smaller than that given to an existing rule.

The number of iterations is counted by the number of games used. It is 50,000 for 9x9 board games and 26,500 for 19x19 board games.

3.3 Experimental Results

The Number of Acquired Rules and the Probability of Being Fed

In the 9x9 board games, 4,684 rules were acquired, while in the 19x19 board games, 12,850 rules were acquired.[2] This shows that our algorithm, unlike the general evolutionary algorithms, enables a large number of rules to be acquired.

Rules that are fed over 100 times were selected for evaluation, because rules that are just created may be inappropriate and die after only a few steps. As a result, 3,211 rules were selected for 9x9 board games and 7,587 rules for 19x19 board games.

The average probability that the selected rules are fed[3] is calculated. The results were 1.58% for 9x9 board games and 0.188% for 19x19 board games, demonstrating that infrequently used rules, which are difficult for ordinary evolutionary algorithms to acquire, can be acquired by this algorithm.

Quality of Acquired Rules

Acquired rules from 9x9 board games were evaluated by an expert[4]. It is difficult to evaluate the patterns stored by previous studies, because they have a fixed shape and have too many indifferent stones to be called rules. The

[1] All the moves are used, because almost all the moves of professional players are taken as good moves for amateur players and only less than one percent of moves are bad.

[2] In this simulation, newly created rules that are the same as a rule already in the rule set is deleted. Thus, there is no duplication of rules.

[3] The average of the probability that each rule is fed during the learning process.

[4] 5 *dan* amateur player

rules acquired by our algorithm are sufficient for evaluation by human experts.

Of the 3,211 selected rules, the top 5% of the 161 rules (the order being determined by the number of being fed) were evaluated. Moves indicated by the rules were evaluated and categorized as good, average, or bad. Of the 161 rules, 67 (41.6%) were evaluated as good, 34 (21.1%) as average, and 60 (37.3%) as bad.

This result shows that about two thirds of the rules acquired by this algorithm are acceptable, showing that the algorithm works effectively, even without any domain specific heuristics.

Automatic Context Separation

This algorithm automatically acquires specialized rules, which are used only in a certain stage or context. Most evolutionary algorithms can acquire only general rules and require humans to classify training data into contexts in order to acquire specialized rules. Our algorithm, however, does not require the classification by humans and automatically separates situations or contexts. For example, Figure 1 shows a rule used only in the opening and Figure 2 a rule used only in the end game.

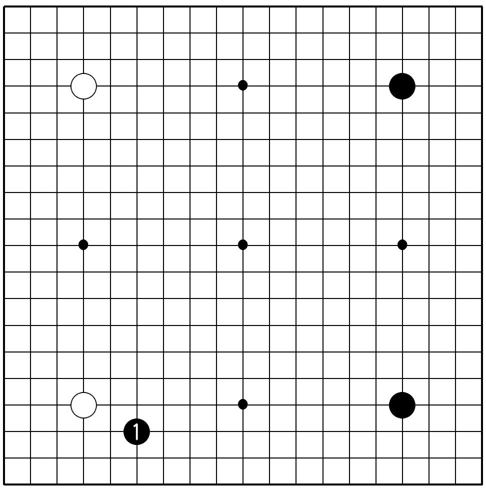

Figure 1: An example of a rule used in the opening.

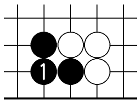

Figure 2: An example of a rule used in the end game.

The Variety of Acquired Rules

In previous studies, only small and fixed-shaped rules are acquired, whereas this algorithm permits the rules to be any size and shape.

Our algorithm acquires rules with a variety of the sizes, never realized in previous studies. Figure 1 is an example of a large (whole-board) rule. Figure 3 is an example of a small rule.

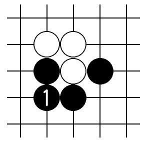

Figure 3: An example of a small rule.

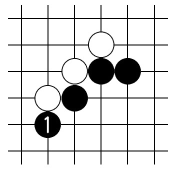

Figure 4: An example of an irregular-shaped rule.

The variety in shape of the rules acquired by our algorithm is demonstrated by the following three examples. Figure 1 is an example of the rule with stones spread over a wide area. Figure 3 is an example of a compact, square rule. Figure 4 is an example of an irregularly shaped rule.

Rules that are used in different contexts can be acquired, as shown in "Automatic Context Separation".

The examples show the great variety in size, shape, and context of knowledge acquired by this algorithm.

Learning Process

In the early trials, general rules tend to be acquired and gradually specialized rules are acquired. This is because the more trials there are, the more food that is given. While frequently used rules can be acquired when the amount of food is small, less frequently used rules are not acquired until the amount of food is large.

Take Figures 1[5] and 3[6] for example.[7] While Figure 3 can occur in all stages of the game, Figure 1 appears only in the opening. The ages[8] of the rules are 1,139,957 and 335,825 respectively, showing that the more popular the rules are, the older they are.

4 Discussion and Related Work

4.1 Advantages of Introducing an Activation Value

One of the major differences between our algorithm and other evolutionary algorithms is that our algorithm gives each individual an activation value. Two advantages of introducing the activation value are discussed below.

[5] If [10,1]=SAME [10,13]=SAME [-2,1]=DIFF [-2,13]=DIFF [0,-3]=EDGE [-6,0]=EDGE

[6] If [1,0]=SAME [0,1]=SAME [2,1]=SAME [1,1]=DIFF [0,2]=DIFF [1,2]=DIFF

[7] Since both rules have six conditions, the probability that they are acquired is the same.

[8] the time steps from the birth of the rule

Avoidance of Convergence

In almost all evolutionary algorithms, individuals tend to become similar and converge at one point or in one set of a small number of rules. This convergence is avoided by introducing activation value to individuals. The reason is as follows.

Suppose many similar rules are created and they tend to match the same data. As explained before, among matched rules, those that do not have more specific rules than themselves are fed, so those that do have more specific rules matching the same data are not fed and die. Some individuals that are fed also die if there are too many similar rules being fed at the same time, because the food is shared among them. As a consequence, the appropriate number of individuals for the given food survives and rules acquired will not be similar as long as there is variety in the training data.

Acquisition of Rules Infrequently Used and Specialized

Ordinary evolutionary algorithms acquire the rules that are used at almost every time step, not those that are infrequently used, because all individuals are evaluated at every time step. By introducing the activation value, such rules are acquired if they are fed before activation value becomes zero. This is demonstrated by the average probabilities that rules are fed being 1.58% for 9x9 board games and 0.188% for 19x19 board games.

Automatic situation separation is also the consequence of introducing activation value. Opening rules, such as Figure 1, will appear only in the opening and not in other stages. Without the activation value such specialized rules would not be acquired.

4.2 Variation-making Operator

Genetic Algorithm (GA) and Genetic Programming (GP) adopt crossover as a variation-making operator. Crossover is a good operator for spreading good building blocks (*schema*); once good schema are found, they are spread rapidly into the population by the crossover operator, and individuals in the population tend to be similar. Therefore, it is quite useful for optimizing functions.

Instead, our algorithm adopts splitting, which is a new operator. Unlike crossover, this operator does not refer to other individuals in making a variation, and each individual makes independently new rules by adding one condition to itself. As a result, the probability of making a novel rule is expected to be higher than with crossover.

While it is said that crossover tends to make individuals similar to each other, splitting is expected to make individuals different. Since our purpose is to acquire various rules, we adopted splitting.

4.3 Comparison with Other Evolutionary Algorithms

Since our algorithm is a kind of an evolutionary algorithm, we compare it with other evolutionary algorithms in this section.

Genetic Programming

Among major evolutionary algorithms, only Genetic Programming (GP) [Koza, 1992] admits flexible knowledge representation. It admits more flexible knowledge representation than our algorithm; while our algorithm admits only Horn clauses, GP admits any type of programs in describing rules.

On the other hand, while GP acquires small sets of rules, our algorithm can acquire large numbers of rules and some infrequently used rules.

Therefore, our algorithm is appropriate for acquiring a large number of rules that can be described by Horn clauses, while GP is appropriate for acquiring a small number of optimal rules that are too complex to be described by Horn clauses.

Classifier Systems

Classifier Systems (CS) [Holland *et al.*, 1986] use many classifiers to solve problems, as does our algorithm. However, CS do not acquire the thousands of rules, which our algorithm does. Moreover, the probability that rules are matched should be high and rules which are infrequently used are not acquired by CS, whereas they are with our algorithm.

CS need a uniform format (i.e. the same length of strings) because they use crossover. Our algorithm adopts splitting and does not require the same length of strings, and the acquired rules are varied in the number of predicates in the condition part.

4.4 Comparison with Humans

The learning process of this algorithm can be compared with that of humans.

In the beginning of this algorithm, there are no rules and the simplest forms of rules are created. Gradually they evolve from simple rules to more complex ones. This is similar to the human learning process. Human beginners also learn simple rules first and then gradually learn exceptions and more complicated rules.

Food might be compared with the memory capacity of humans. While novices have much less memory capacity, their capacity gradually increases; experts have much larger capacity than novices[Chase and Simon, 1973]. In this algorithm, the amount of food given for one training datum is determined by the $FOOD$ parameter. As mentioned in Section 3.2, $FOOD$ increases as the iteration continues. This is similar to the process of human learning in expanding memory capacity.

4.5 Improving the Quality of Acquired Rules

The human expert's evaluation of the acquired rules was shown above. Most rules evaluated as average or bad lacked some conditions, or stones. The complexity of the rules depends on the number of iterations; the more the algorithm iterates, the more specific the rules are. The number of the rules acquired depends on the parameter $FOOD$. Therefore, changing the parameter and the number of iterations may improve the quality of the acquired rules. We are currently studying this assumption.

4.6 Future Work: Acquisition of other kinds of knowledge in Go

This paper focuses only on the acquisition of patterns. Since this algorithm is flexible, other kinds of rules may be acquired.

Human players are said to have not only patterns but also sequences of moves, which they consider important [Saito, 1996]. Since the condition part of this algorithm can take any predicates, indexes of previous moves (for example, two moves before) can be a part of the condition. Thus, sequences of moves have already been acquired, although they are not mentioned in this paper.

Furthermore, once Go terms are defined, Go maxims using these Go terms might be acquired. Since rules of this algorithm are Horn clauses, predicates of rules can be Go terms. In splitting, a new predicate which is a Go term can be added to the rule condition to make the rule more complex. In this way, many rules using Go terms, which we call *maxims*, may be acquired by this algorithm. Acquiring Go maxims, which human experts implicitly have, may help both computers and human novices to understand Go.

The the weakest point in present Go playing systems is said to be that they do not behave differently according to the situations they face, but it is very difficult to write rules for so many situations. Once situation terms in Go are defined, Go patterns or rules may be acquired for the situations using the terms, as with the Go maxims. This may enable computers to play a much stronger game of Go.

5 Conclusions

A new evolutionary algorithm inspired by ecological systems is introduced. Although previous evolutionary algorithms acquire only a few kinds of rules, this algorithm acquires a large number of them. It can acquire rules used as infrequently as 1% of the time, which have never been acquired by other evolutionary algorithms. It acquires Horn clauses, which are flexible compared to the fixed knowledge representation of most evolutionary algorithms.

The algorithm was applied to the game of Go. A large number of appropriate rules were acquired; this may enable Go playing systems to take knowledge-intensive approach. Automatic context separation was also observed.

This algorithm should enable systems to acquire sequences of moves and Go maxims, which are said to be important for human players, because it permits flexible knowledge representation. So this study may enable Go playing systems to be much stronger.

It is shown that the learning process of this algorithm is similar to that of humans.

This algorithm enables a large number of rules to be acquired in an evolutionary way and introduces the possibility of integrating evolutionary algorithms and knowledge engineering.

Acknowledgements

The authors would like to thank Masahiro Okasaki for useful information, and Prof. Takao Terano, Masahiro Hachimori, and Kiyoshi Izumi for their helpful comments.

References

[Chase and Simon, 1973] William G. Chase and Herbert A. Simon. Perception in chess. *Cognitive Psychology*, 4:55–81, 1973.

[Goldberg, 1989] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[Holland *et al.*, 1986] J. H. Holland, K. J. Holyoak, R. E. Nisbett, and P. R. Thagard. *Induction: Processes of Inference, Learning, and Discovery.* MIT Press, 1986.

[Kitano and others, 1993] Hiroaki Kitano et al. Grand challenge AI applications. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1677–1683. Morgan Kaufmann, 1993.

[Kojima *et al.*, 1994] Takuya Kojima, Kazuhiro Ueda, and Saburo Nagano. A case study on acquisition and refinement of deductive rules based on EBG in an adversary game: how to capture stones in Go. In *Game Programming Workshop in Japan '94*, pages 34–43. 1994.

[Kojima, 1995] Takuya Kojima. A model of acquisition and refinement of deductive rules in the game of Go. Master's thesis, The University of Tokyo, 1995.

[Koza, 1992] John R. Koza. *Genetic Programming.* MIT Press, 1992.

[McCarthy, 1990] J. McCarthy. Chess as the Drosophila of AI. In T. Anthony Marsland and Jonathan Schaeffer, editors, *Computers, Chess, and Cognition*, chapter 14, pages 227–237. Springer-Verlag, 1990.

[Saito and Yoshikawa, 1996] Yasuki Saito and Atsushi Yoshikawa. An analysis of strong go-players' protocols. In *Proceedings of Game Programming Workshop in Japan '96*, pages 66–75. 1996.

[Saito, 1996] Yasuki Saito. *Cognitive Scientific Study of Go.* PhD thesis, The University of Tokyo, 1996. in Japanese.

[Sei and Kawashima, 1994] Shinichi Sei and Toshiaki Kawashima. The experiment of creating move from "local pattern" knowledge in Go program. In *Game Programming Workshop in Japan '94*, pages 97–104, 1994. in Japanese.

[Simon and Gilmartin, 1973] Herbert A. Simon and Kevin Gilmartin. A simulation of memory for chess positions. *Cognitive Psychology*, 5:29–46, 1973.