Pattern Selection Problem for Automatically Generating Evaluation Functions for General Game Player

KANEKO Tomoyuki YAMAGUCHI Kazunori
KAWAI Satoru
Graduate School of Arts and Sciences
The University of Tokyo
3-8-1 Komaba, Meguro-ku, Tokyo, 153-8902, JAPAN
{**kaneko, yamaguch, kawai**}**@graco.c.u-tokyo.ac.jp**

Abstract

Patterns are widely used to make evaluation functions, however, the identification of suitable patterns in a game-independent manner is difficult. Our approach is (1) to generate patterns by translation of such features written in logic that can be generated without assistance by expert players of a target game, and (2) to select a suitable patterns among generated ones. The problem is that we have to examine a large number of patterns in the selection, in order to generate accurate evaluation functions. We propose a lightweight selection method based on statistics and an optimization technique on pattern matching that reduces the cost of selection. Experiments on Othello showed significant improvements on accuracy of evaluation functions made of patterns selected by the proposed method.

1 Introduction

1.1 General game players

One of the most ambitious goals of artificial intelligence research is the development of a general game player that can learn and play an arbitrary instance of a certain class of games. Strong game programs must have an accurate and efficient evaluation function that estimates the result of a match of a *position*. Since an evaluation function is specific to a target game, the development of general game players requires the automatic construction of evaluation functions without assistance by human experts.

1.2 Learning of evaluation functions

A popular way to construct an evaluation function is to make it a (linear) combination of evaluation primitives called *features*, and adjust the parameters of the combination [2].

There is a general method that generates features written in logic programs (we call them *logical features*) [4], however, logical features are not practical due to slow evaluation of logic programs. On the other hand, practical evaluation functions are constructed by using a large number of patterns as features [3, 2]. However, mechanical identification of suitable patterns is difficult, and there is few game-specific techniques (e.g. [12]).

1.3 Our approach

Our goal is to construct efficient and accurate evaluation functions by the combination of general methods:

1. generation of patterns by the two steps; (a) generation of logical features, and (b) extraction of patterns from them,
2. selection of suitable patterns,
3. composition of a diagram so as to perform incremental pattern matching in position evaluation.

This paper is organized as follows. The next section reviews related researches and problems. After the introduction of basic definitions in Sect. 3, methods of pattern generation and position evaluation are briefly explained in Sect. 4 and Sect. 5. Then, Sect. 6 proposes a method of selecting patterns. Sect. 7 shows the experimental results in Othello and Sect. 8 concludes this paper.

2 Related Work and Problems

Fawcett developed a general method that constructs features written in logic programs by transformation of the specification of a target game [4]. It is general and actually applied to many games; Othello, a single-agent search problem[4, 5], symmetric chess like games [15] and a variant of Shogi [10]. However, logical features

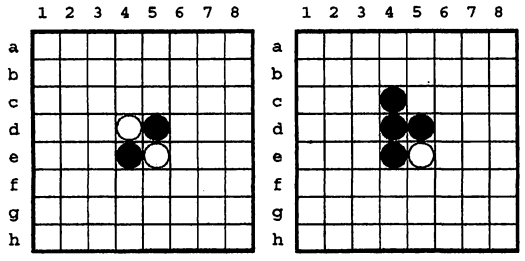

Figure 1: Othello initial position (left) and a position after black played c4 (right). Facts below each board define a position.

prohibitively cost on position evaluation. In spite of recent improvements achieved by a combination of optimization techniques [11, 8], the efficiency does not reach to that of evaluation functions made of patterns [9].

As for the evaluation, patterns can be evaluated much more efficiently than logical features. Buro developed efficient and accurate evaluation functions in Othello, made of a large number of patterns in fixed shapes [3, 2]. The use of fixed shapes contributes to both accuracy and efficiency. However, it is difficult for general game players to automatically identify such shapes.

Recently, we developed a method to generate patterns from logical features [9]. The accuracy of generated evaluation functions is much improved compared to those of logical features, but, does reach to that of Buro's evaluation functions. The difference is in the number of patterns used in evaluation functions where Buro used more than 270,000 patterns while we used only about 8,000 patterns at that time. Since it is not difficult to generate a large number of patters in our scheme, the main problem is the selection of patterns. The main contribution of this paper is a selection methods applicable to a large number of candidates and improvements on pattern matching by using cube extraction.

3 Specification of a Game

This section introduces how to write the specification of a game in logic, on which the generation method depends.

3.1 Positions and domain theory

A *position*, which is an intermediate status of a match, is described by a set of special facts. A fact is a clause without body. In Othello, `owns` and `blank` represent a position. For example, the facts defined in the initial position in Othello and the position after black played c4 are shown in Figure 1. Here, we use `x` for black, and use `o` for white. In the initial position, `owns(d5,x)`, `owns(e4,x)`, `owns(d4,o)`, `owns(e5,o)` are defined for squares with a disc. Also `blank` is defined for each empty squares.

```
legal_move(Square, Player):-
  square(Square), bs(Square,_End,Player).
bs(S1,S3,P):-blank(S1), opponent(P,Opp),
  neighbor(S1,D,S2), span(S2,S3,D,Opp),
  neighbor(S3,D,S4), owns(P,S4).
span(S1,S2,D,Owner):-
  square(S1),square(S2),
  player(Owner), owns(Owner, S1),
  neighbor(S1,D,S3),span(S3,S2,D,Owner).
span(S,S,D,Owner):-
  square(S),player(Owner),owns(Owner,S),
  direction(D).
line(S,S,D):-square(S),direction(D).
line(From,To,D):-
  neighbor(From,D,Next), line(Next,To,D).
opponent(x, o). opponent(o, x).
direction(n).direction(ne).direction(e).
direction(se).direction(s).direction(sw).
direction(w).direction(nw).
square(a1). square(a2). square(a3).
(...)
square(d2). square(d3). square(d4).
neighbor(a1, s, a2). neighbor(a2, n, a1).
neighbor(a2, s, a3). neighbor(a3, n, a2).
(...)
neighbor(c4,ne,d3). neighbor(d3,sw,c4).
```

Figure 2: A sample domain theory of Othello (4×4)

A *domain theory* is a main part of the specification of a game that specify the rules of the game and the goal conditions. It is described by a set of Horn Clauses. As an example in Othello, we use a domain theory shown in Figure 2.

3.2 Logical features

A logical feature is defined as such a Horn Clause in the predicate logic that predicates in their body are defined in a domain theory or a position. The following clause is an example of a logical feature.[1]

```
f(A):-owns(x,A).  % pieces for black
```

[1] It is written as `f2(Num) :- count([A], (owns(x,A)), Num)` in the work by [4]. In this paper, we assume counting as the default semantics of logical features and omit the predicate "count".

The *value* of a logical feature upon a state is defined as the number of solutions, where solutions are the bindings of such constants to variables that make the clause true. In the above feature, `A` is a variable, and the solutions in the initial position shown in Figure 1 (left) are `{d5, e4}` and the value is two, which is the number of squares currently owned by black.

3.3 Patterns

A pattern is defined as a conjunction of facts describing a part of position. The value of a pattern is 0 or 1 according to its Boolean value, where we consider the Boolean value of a fact 1 (0) if the fact is defined (undefined) in a given position respectively. The following conjunction is an example of a pattern.

$$\text{blank(a1)} \wedge \text{owns(x, a2)} \wedge \text{owns(o, a3)}$$

This pattern suggests white can play at a square a1 if it match a position.

4 Pattern Generation

Patterns are generated by the following steps:

1. generation of logical features by using a method of Fawcett [4],
2. translation into the equivalent set in propositional logic by unfolding, and
3. extraction of patterns.

First, logical features are generated by using Fawcett's method [4]. It generates features by means of syntactic translation of Horn Clauses which are extracted from the domain theory of a target game. For example, a mobility feature is written by using predicate `legal_move`, as follows.

```
f(A) :-legal_move(A,o).    % mobility for white
```

Complex features can be generated by taking preconditions of existing features. See Fawcett [4] for more details about automated construction.

In the next step, generated features are translated into the propositional logic by *unfolding* which is a technique in partial evaluation of logic programming [1]. Figure 3 shows a part of the results of unfolding the feature in the above example `f(A) :-legal_move(A,o)`. After the translation, features are represented in the form of conjunctions and disjunctions of facts. Detailed methods of the translation is described in [9].

Finally, patterns are extracted from the translated features by simply taking conjunctive part of their propositional formulae. Figure 4 lists patterns extracted from

```
legal_move(a1,o) :-
    blank(a1), owns(x,a2), owns(o,a3).
legal_move(a1,o) :-
    blank(a1), owns(x,b1), owns(o,c1).
legal_move(a1,o) :-
    blank(a1), owns(x,b2), owns(o,c3).
```

Figure 3: Partial result of unfolding of `f(A) :- legal_move(A,o)`

- blank(a1) $\wedge$ owns(x, a2) $\wedge$ owns(o, a3)
- blank(a1) $\wedge$ owns(x, b1) $\wedge$ owns(o, c1)
- blank(a1) $\wedge$ owns(x, b2) $\wedge$ owns(o, c3)

Figure 4: Extracted patterns

unfolded features shown in Figure 3. We can see that each pattern has a corresponding clause whose body (the right hand of a clause) is equivalent to the pattern.

5 Pattern Matching

This section propose an efficient method to compute matching of a large number of patterns. Pattern matching is required in position evaluation and also in selection of patterns. The basic ideas are (1) to perform incremental calculation and (2) to utilize partial order on patterns.

5.1 Hasse diagrams

For example, let each of a, b, and c is such a fact that used to describe a position, and consider that there are six patterns listed in Table 1. Here, ab means a conjunction of a and b, and so on. A naive way to perform incremental pattern matching is to construct such an index about dependency of patterns on facts, as sketched in Figure 5. A square node denotes a pattern, and a circle denotes a fact. Using this index, when a fact "a" changed, we can determine that matching of only a, ab and abc are required because the values of other patterns are independent to the fact a.

By using an optimized index (we call it a *diagram*) as sketched in Figure 6, we can reduce matching further because it is clear that matching of a pattern 'abc' is required only when the value of the pattern 'ab' changed.

Table 1: Six sample patterns

pattern	abc	ab	bc	a	b	c
dep	$\{ab, bc\}$	$\{a, b\}$	$\{b, c\}$	a	b	c

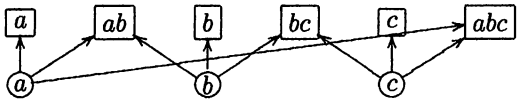

Figure 5: A naive index of sample patterns

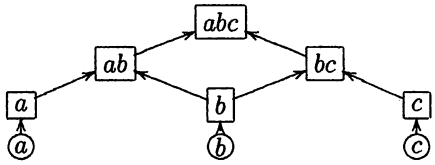

Figure 6: A Hasse diagram of sample patterns

Actually, the diagram is a Hasse diagram [6] on the partial order on sample patterns. In the diagram, a node represents a pattern, and an edge from pattern p to q represents a relation that the pattern p is covered by q. Details are discussed in [9].

5.2 Counters for matching

In order to speedup individual matching of a pattern, an integer counter $cur(p)$ is associated with each pattern p such that the matching is determined by integer comparison instead of naively computing a logical conjunction of each facts in the pattern.

Let $dep(p)$ ($upd(p)$) be children (parents) of pattern p in a diagram, respectively. We define a counter $cur(p)$ as the number of children of p whose current value is true. Then, as long as $cur(p)$ is properly maintained, the Boolean value of $cur(p) = |dep(p)|$ coincides with the value of the pattern p.

5.3 Incremental matching

Our algorithm, named *cover propagation* is listed in Figure 7, which updates the truth values of patterns by adjusting counters of patterns. In the figure, the algorithm for the case that a fact becomes true is shown. The algorithm for the case that a fact becomes false can be stated similarly, and is omitted for the sake of simplicity. It is shown that $cur(p)$ is properly managed by the algorithm [10].

5.4 Efficiency and optimization

The computational cost of the algorithm can be estimated by the number of adjustments of the counters. Because the algorithm will go along at most once for each edge, the worst cost is propositional to the number of edges. Cube extraction is performed here in order to reduce edges, as well as other optimizations.

```
// Let p be the input variable which becomes true
W ← {p}.
while W ≠ ∅ do // W keeps patterns to visit
   Pick q ∈ W. // pick a pattern to visit
   W ← W − {q}.
   for each r ∈ upd(q) do
      cur(r) ← cur(r) + 1
      // if all the covered patterns become true
      // then the pattern r itself becomes true
      if cur(r) = dep(r) then
         W ← W ∪ {r}
```

Figure 7: The cover propagation algorithm

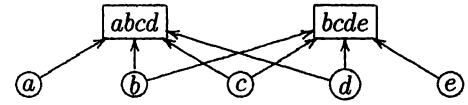

Figure 8: Diagram before cube extraction

5.4.1 Cube extraction

Cube extraction is a classic method in the area of logic circuit optimization [16], which reduces the number of edges by the identification and creation of common subexpressions. The method can directly be applied to the diagrams for pattern matching. For example, Figure 8 draws a diagram for two patterns $\{abcd, bcde\}$ which has eight edges. In Figure 9, an optimized diagram is shown. The diagram has just seven edges because edges from a set of nodes ($\{b, c, d\}$) to another set of nodes ($\{abcd, bcde\}$) in Figure 8 are replaced those from the set of nodes to a common subexpression bcd and those from the subexpression to the other set of nodes. The reduction of edges also contribute s to the reduction of the cost of construction of diagrams.

5.4.2 Elimination of redundant edges

Sometimes, we can safely eliminate an edge in a diagram. The condition is that if $upd(q)$ is constructed so that $(\bigcup_{q \in P, p \in upd(q)} base(q)) = base(p)$ holds for every p. A typical example is a case that if there are four patterns $\{abc, ab, bc, ca\}$ and three edges from ab, abc and bc to abc, then an edge from ca to abc can be eliminated. This is because if the both patterns ab and bc are true,

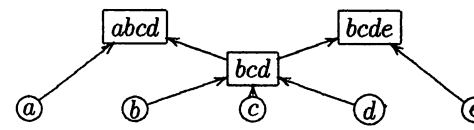

Figure 9: Diagram after cube extraction

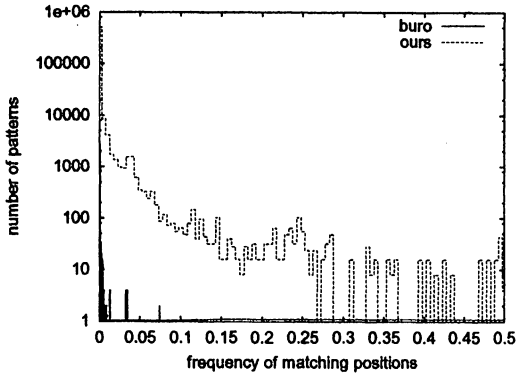

Figure 10: Histogram as to the frequency of matching positions

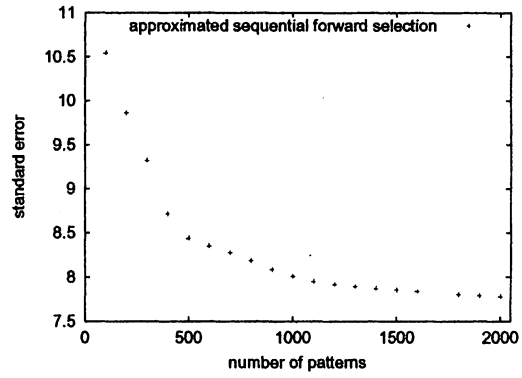

Figure 11: Accuracy of evaluation functions with patterns selected by approximated sequential forward selection

facts a, b and c are true, and the pattern abc must be true.

6 Pattern Selection

Now, our goal is to improve accuracy of evaluation functions by increasing the number of patterns to those used in practical evaluation functions. In order to select such a large number of patterns, selection methods must handle a very large number of candidates and also positions. Popular selection methods such as F-test in statistics are not applicable here. For example, the number of candidates we prepared in Othello is about 8,000,000 and we cannot store covariances of them in normal computers.

Our idea is to solve the problem by the combination of two computationally inexpensive methods. Of course, they do not guarantee the optimality of results. It should be noted that the cost of identification of the optimal pattern subset is known to be exponential to the number of candidate patterns [7], and it is not acceptable in our situation.

6.1 Selection by frequency

The first idea is to select patterns by their frequencies. There are two backgrounds: (1) if patterns of low frequencies are used, the efficiency of position evaluation by using the method explained Sect. 5 will improve, and (2) it is said that the use of patterns of extremely low frequency causes over-fitting [2]. Thus, it seems natural to drop patterns of high or extremely low frequency.

In order to determine appropriate thresholds, frequencies of part of Buro's horizontal patterns [2] are measured and shown in Figure 10. The result suggests that good evaluation functions can be generated only using patterns whose frequencies are below 0.075. Figure 10 also shows the result of measurement as to our patterns. We can see that many patterns can be dropped by the frequency.

6.2 Approximated forward selection

The second idea is to apply a selection method of explanatory variables in regression [7], by treating a pattern as an explanatory variable. We found that an approximated sequential forward selection is suitable for our purpose. It is an efficient selection method that approximates multivariate regression by accumulation of univariate regressions. This method had been mainly used in order to perform computation by hand, before computers became widely available [14].

The outline of the algorithm is listed in Figure 12. A termination criterion at the fourth line is usually the number of selected variables. Priority function at the fifth line is usually the correlation with y_i,[2] however, we used covariance with y_i instead in the hope that patterns of moderate frequency will take precedence. What function should be used as a priority function is not deeply examined yet and is one of our future work.

A preliminary experiment is performed in order to confirm suitability of the method for pattern selection. Figure 11 shows the standard error of evaluation functions made of patterns that are selected by the method among about 4,000 patterns. From the observation that the error is decreasing (accuracy is improving) as the number of patterns increases, we can conclude that the method selects useful patterns.

[2] In that case, selected variables by the method are known to be the same as variables selected by normal sequential forward selection if and only if explanatory variables do not have correlation with each other [14].

// let y_0 be a target variable,
// $X = \{x_0, x_1, , , x_p\}$ be explanatory variables
$i \leftarrow 0$
while (termination criterion is not satisfied)
  pick x_{α_i} of the highest priority
  compute $a_{\alpha_i}, b_{\alpha_i}$ by univariate regression
    s.t. $a_{\alpha_i} x_{\alpha_i} + b_{\alpha_i}$ predicts y_i
  $y_{i+1} \leftarrow y_i - (a_{\alpha_i} x_{\alpha_i} + b_{\alpha_i})$ // residuals
  $i \leftarrow i + 1$

Figure 12: Approximated forward selection algorithm

// let y'_0 be a target variable,
// $X_0, X_1, \ldots X_n$ be sets of variables
$i \leftarrow 0$
while (X is not empty)
  select variables among X_i to predict y'_i
    by approximated forward selection
  $y'_{i+1} \leftarrow$ residuals after the selection
  $i \leftarrow i + 1$

Figure 13: Iterative selection algorithm

For each iteration, the priority of each pattern can be updated without position evaluation if a table of the number of cooccurrences of patterns is available. It becomes difficult to keep such a table in memory if the number of patterns becomes large. We can still avoid frequent position evaluation by iteratively applying approximated forward selection, as listed in Figure 13. Patterns should be split into sets of moderate number of patterns $(X_0, X_1, \ldots X_n)$ in advance. The order of patterns is important because the result differs according to the order. Also, a termination criterion in each approximated forward selection should be modified because we cannot use the number of selected variables. We propose to use a test whether the priority of a selected pattern is beyond a given threshold.

7 Experimental Results

In order to show the proposed method can generate accurate evaluation functions, experiments on Othello are performed. For these experiments, a computer with Athlon MP 2100+ CPU running Linux is used and the program is implemented in GNU C++.

Positions of 60 discs 55 discs are used in the selection and training, which are extracted from matches played between LOGISTELLO and KITTY. [3] About 50,000 positions are selected by eliminating duplicate positions considering symmetry of geometry and players. Then, we generated two disjoint sets of positions expanding symmetric ones again. One set is about 800,000 positions for training and the other is about 6,000 positions for test.

[3] They are available at `ftp://external.nj.nec.com/pub/igord/IOS/misc/`.

Table 2: Attributes of selected patterns

discs		threshold	#patterns	length	match
60	(a)	$5.00 \cdot 10^{-6}$	481552	8.58	641.6
	(b)	$2.50 \cdot 10^{-5}$	326157	8.19	627.9
	(c)	$1.25 \cdot 10^{-4}$	125734	6.90	577.4
	(d)	$5.00 \cdot 10^{-4}$	46739	5.55	486.0
	(e)	$1.25 \cdot 10^{-3}$	23069	4.61	401.4
	(f)	$2.50 \cdot 10^{-3}$	13239	4.06	323.9
	(g)	$5.00 \cdot 10^{-3}$	7531	3.74	239.2
	(h)	0.01	4147	3.62	159.4
55	(i)	$5.00 \cdot 10^{-6}$	878602	9.37	-
	(j)	$1.25 \cdot 10^{-4}$	206918	7.46	831.0
	(k)	$5.00 \cdot 10^{-4}$	69101	5.91	687.9
	(l)	$5.00 \cdot 10^{-3}$	9885	3.90	322.7
	(m)	0.01	5139	3.62	216.1

Table 3: Accuracy of evaluaution functions

60 discs	error	55 discs	error
(c)	6.07	(j)	8.38
(d)	6.31	(k)	8.54
(e)	6.58	(l)	9.65
(g)	7.39	(m)	10.18
(h)	7.81		

7.1 Pattern generation and selection

First, we generated 11,079 logical features and then 8,502,664 unique patterns are extracted from the logical features. The frequency of the patterns are shown in Figure 10 in the previous section.

Then, we applied selection by frequency described in Sect. 6.1, and selected 540,724 patterns. Thresholds used are $[1.25 \cdot 10^{-5}, 0.075]$. They are sorted by their frequency. Finally, we applied iterative sequential selection described in Sect. 6.2 with various thresholds. The results are listed in Table 2. We can see that larger number of patterns are selected as lower threshold are used. In the tables, "match" means the number of patterns whose values are true in a position on average. It also increase as threshold becomes lower. The computation of whole selection takes about two hours on average.

Table 4: Accuracy of other evaluation functions

type	#patterns	error		match
		60	55	
Buro	272032	5.77	8.39	38
Basic	3952	8.17	10.7	97.0

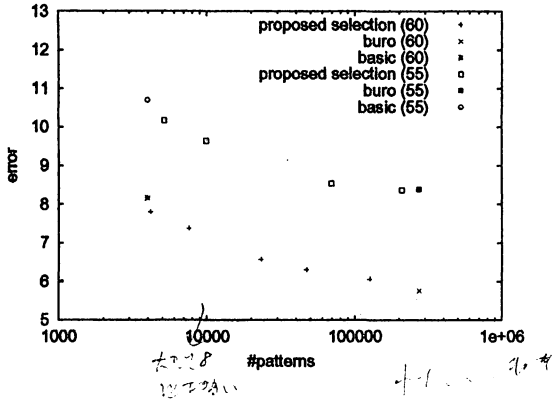

Figure 14: Accuracy of evaluation functions

7.2 Accuracy of evaluation functions

Evaluation functions made of the patterns selected by the proposed method are generated. They use linear combination and their weights are adjusted by means of least mean squares so that they predict the final score (the difference between the number of black discs and that of white ones at the end of the match after both players played in the best way).

The accuracy of evaluation functions are listed in Table 3. In the table, "error" is square root of mean squared errors. Table 4 lists the results of experiments performed in [9] about the accuracy of Buro's function (Buro) and that of our previous work (Basic) [9].[4] Figure 14 illustrates the same data. We can see that the accuracy is improved by using larger number of patterns, better than that of our previous work and approaching to those of Buro's.

7.3 Efficiency of evaluation functions

We gathered a sequence of about 3,000,000 positions that df-pn$^+$ search [13] visited whose root positions are of 49 discs which are extracted from 23 matches in IOS records.[5] The efficiency of evaluation functions become worse as the number of patterns increase, as shown in Figure 15. The improvements on efficiency is our future work.

[4]Note that different positions were used for training.

[5]They are available at `ftp://external.nj.nec.com/pub/igord/othello/ios/`.

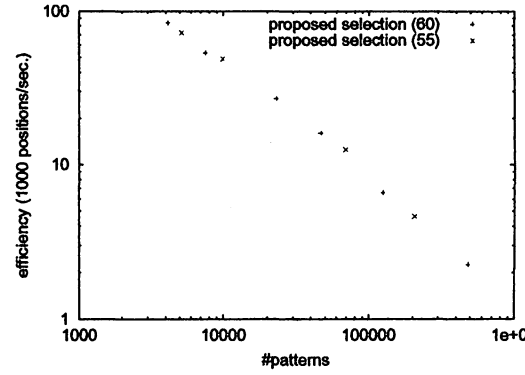

Figure 15: Efficiency of evaluation functions

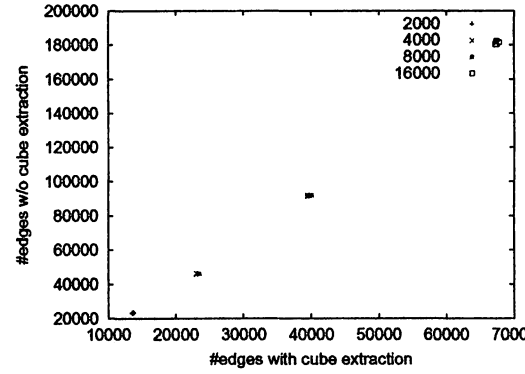

Figure 16: Edge reduction by cube extraction

7.4 Improvements by cube extraction

In order to measure improvements by cube extraction, the number of edges and efficiency of such pair of diagrams are compared that are constructed with or without cube extraction made of the same patterns. Patterns are randomly picked up from automatically generated patterns, and five sets of patterns are prepared for each size. The results are shown in Figure 16 and Figure 17. It is observed that cube extraction reduced the number of edges about a half, consequently improved the efficiency of pattern matching about two to four times.

8 Concluding Remarks

In this paper, a method of constructing accurate evaluation functions without assistance by expert players of a

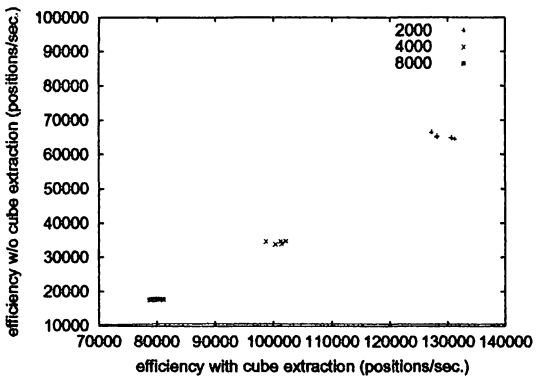

Figure 17: Improvments on efficiency by cube extraction

game is described, which is crucial to construct a general game player. We showed that accurate evaluation functions in Othello are constructed by selecting patterns among a large number of candidates. The accuracy is better than that of our previous work and approaching to that of Buro's. On the other hand, still there is a room for improvements as to the efficiency of position evaluation and it is our future work.

References

[1] A. Bossi, N. Cocco, and S. Dulli. A method for specializing logic programs. *ACM Trans. Prog. Lang. Syst.*, 12(2):253–302, Apr. 1990.

[2] M. Buro. From simple features to sophisticated evaluation functions. In *Proceedings of the First International Conference on Computers and Games*, pp. 126–145, Tsukuba, Japan, Nov. 1998. Springer-Verlag.

[3] M. Buro. Improving heuristic mini-max search by supervised learning. *Artificial Intelligence*, 134(1–2):85–99, Jan. 2002.

[4] T. E. Fawcett. *Feature Discovery for Problem Solving Systems*. PhD thesis, Department of Computer Science, University of Massachusetts, Amherst, 1993.

[5] T. E. Fawcett and P. E. Utgoff. Automatic feature generation for problem solving systems. In *Proceedings of the 9th International Conference on Machine Learning*, pp. 144–153. Morgan Kaufmann, July 1992.

[6] D. Gries and F. B. Schneider. *A Logical Approach to Discrete Math*. Springer-Verlag, New York, 1993.

[7] A. Jain, P. Duin, and J. Mao. Statistical pattern recognition: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4 – 37, Jan. 2000.

[8] T. Kaneko, K. Yamaguchi, and S. Kawai. Compiling logical features into specialized state-evaluators by partial evaluation, boolean tables and incremental calculation. In *PRICAI 2000 Topics in Artificial Intelligence*, pp. 72–82, Melbourne, Australia, Aug./Sept. 2000.

[9] T. Kaneko, K. Yamaguchi, and S. Kawai. Automatic feature construction and optimization for general game player. In *The Sixth Game Programming Workshop*, No. 14 in IPSJ Symposium Series 2001, pp. 25–32, Oct. 2001.

[10] T. Kaneko, K. Yamaguchi, and S. Kawai. Automatic construction of pattern-based evaluation functions for game programming. *IPSJ JOURNAL*, 43(10):3040–3047, Oct. 2002. in Japanese.

[11] T. Kaneko, K. Yamaguchi, and S. Kawai. Compiling logical features into specialized boolean networks with incremental propagation. *IEICE Transaction on Information and Systems*, J85-D-I(11):(to appear), Nov. 2002. in Japanese.

[12] T. Kojima, K. Ueda, and S. Nagano. An evolutionary algorithm extended by ecological analogy and its application to the game of Go. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, Nagoya, Japan, 1997.

[13] A. Nagai and H. Imai. Application of df-pn+ to othello endgames. In *Game Programming Workshop in Japan '99*, pp. 16–23, Oct. 1999.

[14] T. Okuno, H. Kume, T. Haga, and T. Yoshizawa. 改訂版 多変量解析法. 日科技連, 1981. in Japanese.

[15] B. D. Pell. *Strategy Generation and Evaluation for Meta-Game Playing*. PhD thesis, University of Cambridge, 1993.

[16] R. L. Rudell. Tutorial: Design of a logic synthesis system. In *Design Automation Conference*, pp. 191–196, Las Vegas, NV USA, June 1996.