Proceedings of the 2007 IEEE Symposium on
Computational Intelligence and Games (CIG 2007)

Automatic Generation of Evaluation Features for Computer Game Players

Makoto Miwa
Dept. of Frontier Informatics
School of Frontier Sciences
the University of Tokyo, Chiba
miwa@logos.k.u-tokyo.ac.jp

Daisaku Yokoyama
Dept. of Frontier Informatics
School of Frontier Sciences
the University of Tokyo, Chiba
yokoyama@logos.k.u-tokyo.ac.jp

Takashi Chikayama
Dept. of Frontier Informatics
School of Frontier Sciences
the University of Tokyo, Chiba
chikayama@logos.k.u-tokyo.ac.jp

Abstract— **Accuracy of evaluation functions is one of the critical factors in computer game players. Evaluation functions are usually constructed manually as a weighted linear combination of evaluation features that characterize game positions. Selecting evaluation features and tuning their weights require deep knowledge of the game and largely alleviates such efforts.**

In this paper, we propose a new fast and scalable method to automatically generate game position features based on game records to be used in evaluation functions. Our method treats two-class problems which is widely applicable to many types of games. Evaluation features are built as conjunctions of the simplest features representing positions. We select these features based on two measures: frequency and conditional mutual information.

To evaluate the proposed method, we applied it to 200,000 Othello positions. The proposed selection method is found to be effective, showing much better results than when simple features are used. The Naïve Bayesian classifier using automatically generated features showed the accuracy close to 80% in win/lose classification. We also show that this generation method can be parallelized easily and can treat large scale problems by converting these selection algorithms into incremental selection algorithms.

Keywords: feature selection, frequent closed itemset, conditional mutual information, Othello

I. Introduction

Static evaluation functions are one of the most important components of game-playing programs. They evaluate a position and assign it a heuristic value which indicates some measure of advantage.

Static evaluation functions are usually expressed as a linear combination of evaluation features and output a scalar value called an evaluation value.

Developers usually construct and tune the static evaluation functions of difficult games, such as chess, shogi, and go by hand. They extract important evaluation features from a position and assign weights to the features. They are required to have expert knowledge of the target game to extract appropriate features and spend a long time in assigning their weights. This way of construction is ad hoc, and sometimes leads to a local optimum.

Automatic construction of static evaluation functions is one approach to cut the cost to developers and avoid the local optimum problem. Today, we can obtain a lot of game records as electronic data: those between professional game players or on match servers. We can also obtain a number of self-game records overnight using a large number of fast computers. Using these game records, several studies have been made on automatic construction of the static evaluation functions of such games as backgammon [1] and Othello [2]. These studies have made great successes on these games, but they have not been applied to more complicated games, partly because the computational cost becomes unbearably high and partly because the way game knowledge is used in the original method cannot be easily applied to more complicated games.

In this paper, we show a new method of automatic generation of evaluation features based on game records, without using deep game knowledge. This method treats two-class problems which include win/lose and mated/not mated, which are common in many types of games. Once game features are generated, they can be automatically weighted to form an evaluation function through a variety of successful methods. As the proposed method can be parallelized, it can be applied to complicated games requiring large computational cost.

The structure of this paper is as follows. Section II describes related work on automatic construction of evaluation functions. Section III explains how we generate evaluation features automatically. Section IV and V discuss the experimental settings and results of our experiments. Finally, section VI concludes the paper and outlines our further research directions.

II. Related Work

A large number of studies have been made on construction of static evaluation functions for over half a century [3]. Most studies tried only to weight the evaluation features which are selected by hand and little attention was given to generate evaluation features. These methods which weight the evaluation features include neural networks, temporal difference [4], and ordinal correlation [5].

Several studies have been done on automatic construction of static evaluation functions since early 1990s. They constructed static evaluation functions from simple features which represent input positions and they used game records of past games. We now can obtain many game records, such as those electronically published matches between professional players or on match servers. We can also obtain a number of self-game records. The studies may be classified

into two groups: those with direct methods and those with hierarchical methods.

A direct method tries to construct static evaluation functions directly from simple features using mechanisms such as multi-layer neural networks. This method represents a static evaluation function as a high-level combination of simple features. This representation has high expressivity, but requires a large amount of computational resources. In addition, the resulted evaluation function is difficult to analyze. This group includes TD-Gammon [1] which used reinforcement learning; Fogel's game players (tic-tac-toe [6], checkers, namely Blondie24 [7], and chess [8]) and Kumar's checkers [9] which used genetic algorithms; Messerschmidt's tic-tac-toe [10] and Franken's checkers [11] which used particle swarm optimizers (PSO).

A hierarchical method has two phases to construct static evaluation functions. The first phase is to generate high-level evaluation features and the second phase is to weight evaluation features. Hierarchical methods represent static evaluation functions as a linear combination of high-level features. This representation is less expressive than that of the direct method, but needs less resources to construct and compute the function. The evaluation function is easy to analyze and optimize by hand because we can check each evaluation feature individually. This group includes ZENITH [12], ELF [13], GLEM [2], Kaneko's method [14], and Duminy's method[15].

These studies made great successes and many applications to make computer game players automatically were constructed [16]. They, however, focus only on such games as backgammon and Othello due to computational complexity of the methods and their use of knowledge about their target games, and thus the methods are yet to be applied to more complicated games. For complicated games, a hierarchical method is preferable to a direct method because of their low computational complexity.

III. Proposal

Our method generates evaluation features using two-class labeled training positions obtained from game records, which are easily available as mentioned earlier. The two-class problems include win/lose and mated/not mated, which are common in many types of games.

Evaluation features are built as conjunctions of the simplest features representing positions. Once game features are generated, they can be automatically weighted to form an evaluation function through a variety of successful methods, including those based on neural networks, temporal difference and ordinal correlation.

In this method, we construct evaluation features from simple binary features called *base features*. We construct *evaluation features* by selecting from all conjunctions of base features, called *patterns*, with two criteria: frequency and conditional mutual information. We first select frequent closed patterns to avoid the cost of evaluating all the patterns and the risk of over-fitting. Then we select important evaluation features with eliminating dependent patterns based on conditional mutual information. This selection process is fast and scalable because of fast algorithms and their parallelizations, which can also reduce the memory cost.

In this section, we introduce the expression of the static evaluation functions to be built on the features generated through our method. Then, we explain extraction of base features and construction of evaluation features on them.

A. Expression of Static Evaluation Function

Static evaluation functions are required to be fast and accurate. For being fast, most common static evaluation functions are represented as a linear combination of evaluation features:

$$F(p) = \Sigma_i w_i \cdot f_i + b \quad (1)$$

where F is an evaluation function, p is an input position, f_i is an evaluation feature, w_i and b are constants.

For higher accuracy, a representation which has higher expressive power than a linear combination is preferable and it may enable more economical descriptions. However, because of the computational cost of construction and the greater danger of over-fitting, the simple linear expression is usually preferred.

This linear expression has the following merits. (Note that it may not be a linear function of base features.)

- Computational cost of constructing and evaluating the function is low.
- We can check each evaluation feature individually.
- The magnitude of w_i is directly related to the importance of f_i which enables us to easily analyze, optimize the function, and tune weights by hand.

B. Base Features, Patterns and Evaluation Features

To deal with input positions, some expressions are required to represent them. Our method uses simple binary features selected by hand to represent them. Such features should be able to represent all the positions which appear in the target game.

Atomic features are selected as *base features.* For example, an atomic feature 'x is on A1' is a suitable base feature in tic-tac-toe. These atomic features should be selected so that, as a set, they precisely represent all the positions. [1]

Our method can deal with non-binary features using discretization methods [17] which transform the non-binary valued features into ordered discrete ones.

In our method, *evaluation features* are built as conjunctions of base features. We call a conjunction of base features a *pattern.* A pattern appears in a position when all the base features in the pattern are $true$ in the position. Our method does not deal with other combinations of base features such

[1] High-level and more complex features could also be selected as base features. 'x has made a line' in tic-tac-toe may be included in the high-level features. It is important to select the high-level features which represent a crucial rule or some established situation which terminate the game or change the game state drastically because they have great possibilities to be selected as evaluation features. The selection of these features improves the expressive power of evaluation features and reduces the cost to generate evaluation features.

as disjunctions, negations or arithmetic operations. Although the expressive power of the evaluation features may decrease with this limitation, it reduces the computational cost in generating evaluation features. Note that disjunctions can be expressed in the linear combination of evaluation features and negations of features can be included in base features.

Let n be the number of base features. The number of patterns is 2^n and it is hard to treat all of the possible patterns as evaluation features because of the learning and executing costs. Important patterns should be selected in minimum loss of information. To reduce the computational cost, we select evaluation features through two phases: The first phase is extracting the patterns frequent in the training positions, and the second phase is selecting the patterns useful for classification out of frequent patterns.

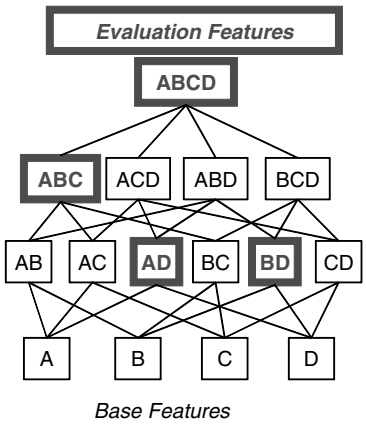

Fig. 1. Base features, patterns, and evaluation features. Patterns are conjunctions of base features. Evaluation features are selected from patterns.

Figure 1 shows the construction of evaluation features from base features. 16 ($= 2^4$) patterns (including patterns consisting only of base features and an empty pattern) are built as conjunctions of 4 base features A, B, C, and D. From the patterns, evaluation features are selected based on their frequency and conditional mutual information in training positions.

C. Extraction of Frequent Patterns

We first extract frequent patterns from training positions. The number of patterns is so large that we can not evaluate all of them. Patterns which rarely appear in training positions cause over-fitting, so they should be eliminated. By this, infrequent important patterns may be overlooked, but we can not evaluate how important such patterns are if they do not appear frequently enough in training positions.

A frequent pattern is defined as:

Definition 1 (frequent pattern): A pattern which appears in training positions α times or more.

where α is called a minimum support. If the number of positions in which a frequent pattern k appears is n_k, the number of positions in which a conjunction of the pattern k and a base feature is less than n_k. This enables us to extract frequent patterns efficiently by adding or eliminating a base feature to already found patterns.

There are patterns which appear in exactly the same set of training positions, called *fully dependent patterns.* Although fully dependent patterns will anyway be eliminated in the CMIM selection later described in III-D because they have the same information on training positions, they add some computational load. Most of these waste patterns can be eliminated by selection of closed patterns. The definition of a closed pattern is:

Definition 2 (closed pattern): Maximal element of patterns which appear in exactly the same set of training positions

In Figure 2, we show an example of extracting frequent patterns and frequent closed patterns from training positions. There are 6 training positions and 9 base features, numbered from 1 to 9. Base features which are $true$ are presented in the figure sets. With the minimum support of 3, frequent patterns are patterns appearing in 3 positions or more. A line in the center column of frequent patterns have frequent patterns appearing in exactly the same set of training positions. Maximal patterns, those not completely included in other patters in the same line, are selected as frequent closed patterns.

For extraction of the frequent closed patterns, we can use algorithms proposed in data mining field for market basket analysis by treating patterns as a set of items. We use LCM (Linear time Closed set Miner) [18]. LCM won FIMI '04 (Frequent Itemset Mining Implementations 2004) and is one of the most efficient algorithms to enumerate frequent closed patterns.

Figure 3 shows a search tree in LCM. All nodes have frequent patterns and the root node has an empty pattern. The node's children are created by appending base features in the same order to the pattern, so a child's pattern includes the parent node's pattern. The extension from a parent node's pattern to a child's pattern in the fixed order is called a *prefix preserving closure (PPC) extension.* This extension enables LCM to enumerate frequent closed patterns at low computational cost and its computational complexity is $O(n)$, where n is the number of frequent closed patterns. Each node has a compact database which has enough information to treat training positions containing its pattern. The subset of the database is delivered to their children after database reduction. The database is called an *occurrence table.* LCM searches the tree in a depth-first manner to enumerate frequent closed patterns. When LCM visits a node, it keeps the list of nodes which are on the way from the root node to the visited node and their occurrence tables. Not all the tree information is required for enumeration, so LCM has low spatial cost.

We made two extensions to LCM.

1) Selection based on information gain
2) Parallelization

When the number of selected frequent closed patterns are too large for the CMIM selection (in III-D), we can

Training positions	Frequent patterns	Frequent closed patterns
{1, 2, 5, 6, 7, 9}	{1} {1, 7} {1, 9} {1, 7, 9}	{1, 7, 9}
{2, 3, 4, 5}	{2, 7} {2, 9} {2, 7, 9}	{2, 7, 9}
{1, 2, 7, 8, 9}	{7} {9} {7, 9}	{7, 9}
{1, 7, 9}	{2}	{2}
{2, 7, 9}		
{2}		

Fig. 2. Training positions, frequent patterns, and frequent closed patterns when the minimum support is 3. A position has 9 base features from 1 to 9 and $true$ features are presented.

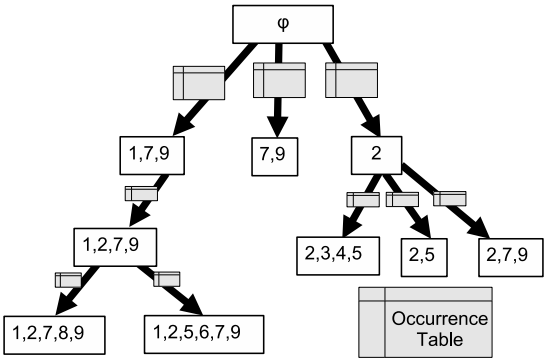

Fig. 3. A search tree in LCM. A node has a pattern, and children are made by appending base features to the pattern. The node's *occurrence table* contains the information about positions related to its pattern and is delivered to its children after database reduction.

perform prior selection based on information gain without much cost. We insert the labels of training positions into the occurrence tables and select important patterns based on information gain while enumerating patterns. Information gain of a pattern is the mutual information (see III-D) between the labels and the pattern.

TABLE I

PARALLELIZED LCM

root node

step 1,	extract frequent base features and decide the order to append for the PPC extension
step 2,	search a part of base features if the number of base features is small and inequality of clients' tasks will occur
step 3,	place the order in step 1 and frequent base features or frequent patterns in step 2 as tasks into a work queue

client nodes

step 1,	select a task from the work queue
step 2,	construct an occurrence table based on the patterns by searching from root node
step 3,	enumerate frequent closed patterns using LCM

Since sibling nodes have no dependency in LCM, we can search sibling nodes independently on different processors. Parallelization of LCM is based on a simple work-queue strategy in which idle processors get works from a queue, as shown in Table I. In this parallelization, every node has a set of training positions to extract frequent patterns from. Occurrence tables become large with many training positions. To reduce the cost of sending occurrence tables that would dominate the computation cost, patterns are sent and client nodes construct occurrence tables from the ground up using the positions and the patterns.

D. Selection of Evaluation Features Based on Conditional Mutual Information

From the extracted frequent closed patterns, we select important patterns as evaluation features, because the number of the frequent closed patterns is large and not all the frequent closed patterns are informative. For the selection, there are many useful feature extraction and/or selection methods proposed in the machine learning field. Some methods, PCA (Principal Component Analysis) and learning methods for example, select patterns by evaluating all the patterns at a time [19]. Such methods cost high and they are not suitable to selection. Some other methods select patterns by evaluating them based on criteria such as chi-squares and information gain [20]. The methods are fast but many mutually dependent patterns may be selected.

We use CMIM (Conditional Mutual Information Maximization) [21] for the selection. CMIM selects a pattern based on conditional mutual information of a pattern given the information included in already selected patterns. CMIM selects a pattern which has the largest conditional mutual information for every already selected pattern, as follows.

$$\begin{aligned} v(1) &= \arg\max_n \hat{I}(Y;X_n) \\ v(k+1) &= \arg\max_n \left\{ \min_{l\leq k} \hat{I}(Y;X_n|X_{v(l)}) \right\} \\ &\quad (1\leq k < K) \end{aligned} \tag{2}$$

where Y is a label, X is a pattern, K is the number of patterns to be selected, $\hat{I}(Y;X_n)$ is mutual information, and $\hat{I}(Y;X_n|X_{v(l)})$ is conditional mutual information. Mutual information $\hat{I}(Y;X)$ is X's information about Y and is defined as follows.

$$\hat{I}(Y;X) = \hat{H}(Y) + \hat{H}(X) - \hat{H}(Y,X) \tag{3}$$

where $\hat{H}(X)$ is entropy, meaning the randomness of a probability variable X. Conditional mutual information $\hat{I}(Y;X|Z)$ is new X's information about Y after obtaining Z's information about Y and is defined by the following equation.

$$\begin{aligned} \hat{I}(Y;X|Z) &= \hat{H}(Y|Z) - \hat{H}(Y|X,Z) \\ &= \hat{H}(Y,Z) - \hat{H}(Z) \\ &\quad -\hat{H}(Y,X,Z) + \hat{H}(X,Z) \end{aligned} \tag{4}$$

Entropy is defined as follows.

$$\hat{H}(X_1,\ldots,X_n) = -\sum_{x_1,\ldots,x_n\in\{0,1\}^n} p_{x_1,\ldots,x_n}\log p_{x_1,\ldots,x_n}. \tag{5}$$

In this process, most of dependent patterns are not selected. CMIM need not evaluate all the patterns at the same time and is relatively fast. The computational complexity of CMIM is $O(KMN)$, where M is the number of patterns, N is the number of training positions, and K is the number of patterns to be selected. The spatial complexity of CMIM is $O(MN)$. As CMIM has high computational cost in calculating mutual information and conditional mutual information, the memory space used should fit in the main memory. This becomes impossible if the number of frequent closed patterns is too large.

TABLE II
DIVIDE-AND-CONQUER-LIKE CMIM SELECTION

step 1,	divide patterns into small sets randomly (S patterns for each)
step 2,	select patterns which have a larger conditional mutual information than the given cut-off conditional mutual information from each set
step 3,	go to step 1 if selected patterns do not meet certain criteria

To overcome this spatial difficulty, patterns are selected in a divide and conquer manner. Details of the divide-and-conquer-like CMIM selection algorithm are given in Table II. In step 3, the number of repetitions or the number of patterns discarded in step 2 is used as the criteria. Selection can be performed with small memory for each divided set of pattern. The spatial complexity of the selection is $O(SN)$ and does not depend on the total number of patterns. Although patterns selected from these divided sets of patterns are not necessarily the same as ones selected considering all the patterns at a time, patterns can have almost the same information from the viewpoint of CMIM.

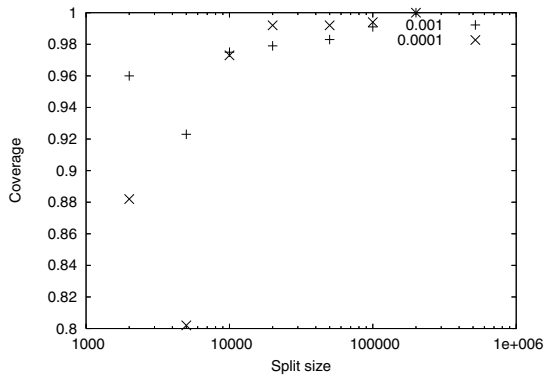

Fig. 4. Difference between selected features using CMIM and ones using the divide-and-conquer-like CMIM selection in Dorothea. The coverage shows how selected features using the divide-and-conquer-like CMIM selection covers selected features using CMIM.

To show the difference between features selected through the original CMIM selection and ones through the divide-and-conquer-like CMIM selection, a preliminary experiment on a small dataset is tried out. We used *Dorothea* which is available in feature selection challenge[2] in NIPS2003. Dorothea contains 800 training examples expressed with 139,351 features. We selected frequent closed patterns using LCM by with the minimum support of 16. Figure 4 shows the difference of the features selected from 146,380 patterns when the cut-off conditional mutual information are 0.01 and 0.001. The result shows that most of the patterns selected with the CMIM selection are covered by the divide-and-conquer-like CMIM selection.

Since each selection in step 2 in Table II has no dependency, we can assign each set of patterns to different processors and parallelize the divide-and-conquer-like CMIM selection. The divide-and-conquer-like CMIM selection is parallelized based on a simple work-queue strategy. This parallelized divide-and-conquer-like CMIM selection is called *parallelized CMIM selection.*

IV. EXPERIMENT SETTINGS

We have applied our method to a win/lose problem in *Othello* game and tried to automatically generate evaluation features to construct an evaluation function in Othello. Othello, also known as *Reversi,* is a popular Japan-origined board game played between 2 players on an 8 by 8 board using discs with white and black faces on different sides. The move is done by placing a disc on an empty square and turn over all bracketed opponent's discs. When placing on no empty square brackets opponent's disc, the player has to pass. The game ends when neither player can put a disc and the player with more discs of its color on the board wins.

In this section, we explain a Naïve Bayesian classifier and experiment settings which include base features, data sets, and experiment environments.

A. Naïve Bayesian classifier

To evaluate the selected evaluation features, we used a Naïve Bayesian classifier [22] which is one of the fastest classification algorithms.

The Naïve Bayesian classifier is a simple likelihood ratio test with the assumption of conditional independence among features. The classifier is empirically known to work well even when this assumption precisely holds. The Naïve Bayesian classifier is represented as:

$$f(\mathbf{x}) = \sum_{i=1}^{N}\left\{\log\frac{\hat{P}_{1,1}(x_i)\hat{P}_{0,0}(x_i)}{\hat{P}_{0,1}(x_i)\hat{P}_{1,0}(x_i)}\right\}x_i + b \tag{6}$$

where N is the number of the evaluation features, and $\hat{P}_{\alpha,\beta}(x)$ is the share of the training positions that a pattern x is α (1 if it appears in it and 0 if not) and its label is β.

[2] http://www.nipsfsc.ecs.soton.ac.uk/

B. Experiment Settings

1) Base Features: States of each board square are chosen as the base features. Each square is expressed by 3 base features: "black is on the square", "white is on the square", and "empty". The total number of the base features is 192. We did not use any other game knowledge, such as the exclusiveness of these three base features of the same square or the symmetry of the board.

2) Data Sets: The data set to generate evaluation features contained 200,000 game positions. The data set to test the classification performance contained 962,439 positions. These positions are already with 60 discs on the board, leaving four empty positions, and labeled as *win* or *lose*. The label denotes the black player's result of the game. These positions were extracted from game records in the Generic Game Server [23]. Each position is labeled by searching to the end of the games from the position.

3) Experiment Environments: A PC cluster system with 50 nodes is used. Each node has two 2.4GHz Intel Xeon and 2GB of RAM. Software are written in Python and C++.

V. Experiment Result

In this section, the obtained evaluation features and the results of classification using them are described and discussed.

A. Evaluation Features

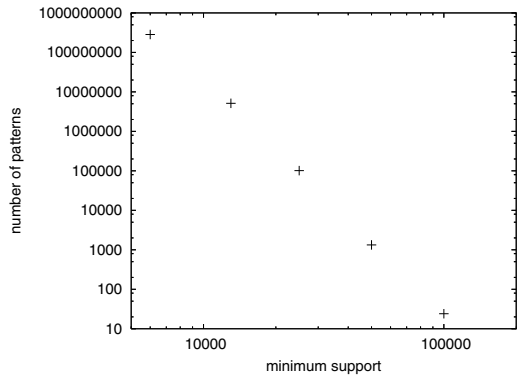

Fig. 5. Number of frequent closed patterns along the minimum support out of a total of 200,000 positions

In the first step, frequent closed patterns are selected using a parallelized version of LCM. The number of the obtained frequent closed patterns with varying minimum supports are shown in Figure 5. The number of frequent closed patterns decrease rapidly with the minimum support.

Evaluation features are then selected from the frequent closed patterns using a parallelized CMIM. Figure 6 shows the number of selected features with varying cut-off conditional mutual information in the selection of CMIM, for cases with two different minimum supports of 6,000 and 4,000. Since the number of the frequent closed patterns becomes very large with the minimum support of 4,000, prior selection on information gain is used to eliminate such patterns that

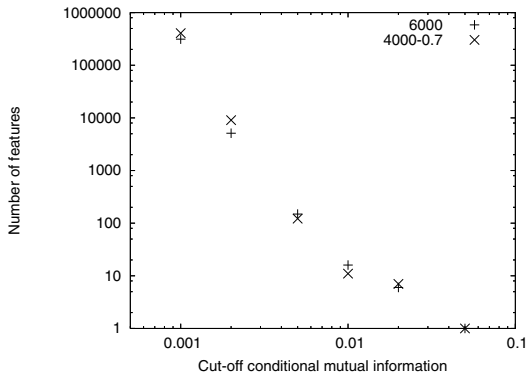

Fig. 6. Number of selected evaluation features along the cut-off conditional mutual information using a CMIM selection (6,000: minimum support=6,000, selected from 282,615,853 patterns, 4,000-0.7:minimum support=4,000, information gain ≤ 0.7, selected from 172,022,168 patterns)

neither of win nor lose ratios are less than 80% (less than 0.7 in information gain) in the positions they appear. The divide-and-conquer-like selections are made with 5 layers. One set contained 2,000 frequent patterns except for the last selection and, in the last selection, it contained 10,000 frequent patterns.

Some CMIM processing are performed concurrently with LCM processing to alleviate work imbalance of the parallelized LCM. Generation of evaluation features took about 2 days using 49 nodes (98 processors) when the minimum support was set to 6,000 and the cut-off conditional mutual information in the CMIM selection to 0.001. The task did not end in a month on a single processor. LCM selection used 140MB and the CMIM selection used 250MB of memory at their peaks. This shows our selection method requires reasonably small space and it can handle still larger data sets.

B. Classification Results

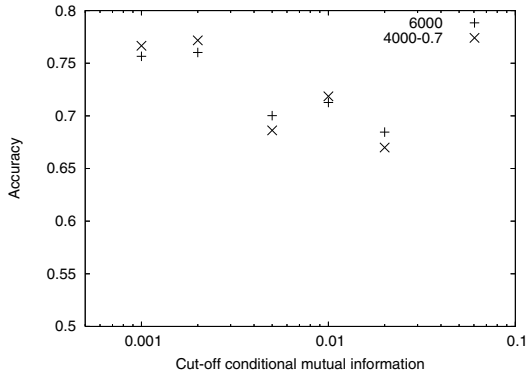

Fig. 7. Accuracy of Naïve Bayesian classifiers using selected features

The accuracy of the obtained classifiers is shown in Figure 7. We selected evaluation features as described in V-A. In Table III, the results of three learning methods directly

TABLE III
ACCURACIES OF CLASSIFIERS USING BASE FEATURES

classifier	accuracy
Naïve Bayesian	73.9%
3-layer neural network	∼75.2%
Linear Discriminant Classifier	73.4%

using base features are shown. We performed a 5-fold cross validation in each test. The best of them had 77.2% in its accuracy, 0.773 in its precision, 0.788 in its recall, and 0.780 in its F1-value: precision = a / b; recall = a / c; F1-value = 2 $\times$ precision $\times$ recall / (precision + recall), in which a is the number of correctly predicted positions labeled *win*, b is the number of positions predicted to be *win* by the classifier, and c is the number of positions labeled *win*. This result shows the Naïve Bayesian classifier using generated features with our method performs better than the three classifiers directly using the base features: a Naïve Bayesian classifier, a 3-layer neural network using the RPROP learning rule [24] and a linear discriminant classifier [22]. The Naïve Bayesian classifier and the linear discriminant classifier converged, but the 3-layer neural network did not. As the result of the 3-layer neural network, we show its best result here. We also tried a Support Vector Machine (SVM) to learn directly from the base features, but its learning did not terminate in a week. These results show that the generated features are more expressive than the base features.

VI. CONCLUSION AND FUTURE WORK

In this paper, we proposed a new efficient and scalable method to construct evaluation features automatically using game records. The method constructs evaluation features from a set of base features through selections based on frequency and conditional mutual information. We applied this method to a win/lose problem in Othello and generated evaluation features. Several thousands of evaluation features were generated from 192 base features, and we obtained a Naïve Bayes classifier with 77.2% in classification accuracy. The classifier performed better than three classifiers using the original base features. Parallelization of the algorithm enabled the method to handle a large data set, which conventional feature extraction and machine learning methods cannot handle, with tractable spatial cost. The method can also find important features. The evaluation features generated by our method may also provide hints to developers constructing their evaluation functions by hand.

In the future, we would like to improve the proposed feature selection method further. GLEM's *pattern* [2] and approximate methods may be useful for this purpose. There is no general way to apply GLEM's *pattern* to other games, and we should try to generate the way to cluster base features automatically. We also would like to apply our method to more complicated games like chess and shogi. We have shown that a method using LCM and CMIM without the parallelization is effective in generating evaluation functions in mate problems in shogi [25]. With the parallelization, we will be able to apply our method to larger data sets and will be able to show that our method is not limited to simple games such as Othello end games.

REFERENCES

[1] G. Tesauro, "TD-Gammon, A Self-teaching Backgammon Program, Achieves Master-Level Play," 1993, pp. 19–23.

[2] M. Buro, "From Simple Features to Sophisticated Evaluation Functions," in *Proceedings of the First International Conference on Computers and Games (CG-98)*, H. J. van den Herik and H. Iida, Eds., vol. 1558. Tsukuba, Japan: Springer-Verlag, 1998, pp. 126–145. [Online]. Available: citeseer.ist.psu.edu/buro99from.html

[3] J. Fürnkranz, "Machine Learning in Games: A Survey," in *Machines that Learn to Play Games*, J. Fürnkranz and M. Kubat, Eds. Huntington, NY: Nova Science Publishers, 2001, ch. 2, pp. 11–59.

[4] J. Baxter, A. Tridgell, and L. Weaver, "Learning to Play Chess Using Temporal Differences," *Machine Learning*, vol. 40, no. 3, pp. 243–263, 2000.

[5] M. B. D. Gomboc, T. A. Marsland, "Evaluation fuction tuning via ordinal correlation," in *Advances in Computer Games*, H. van den Herik, Iida, Ed. Kluwer, 2003, pp. 1–18.

[6] D. B. Fogel, "Using Evolutionary Programming to Construct Neural Networks that are capable of playing Tic-Tac-Toe," in *Proceedings of the IEEE International Conference on Neural Networks (ICNN-93)*, San Francisco, 1993, pp. 875–879. [Online]. Available: http://www.natural-selection.com/Library/1993/EP_NN_tic_tac_toe.pdf

[7] ——, *Blondie24: Playing at the Edge of AI*. Morgan Kaufmann, September 2001.

[8] D. B. Fogel, T. J. Hays, S. L. Hahn, and J. Quon, "A self-learning evolutionary chess program," in *Proceedings of the IEEE*, vol. 92, no. 12, 2004, pp. 1947–1954.

[9] K. Chellapilla and D. B. Fogel, "Evolving an Expert Checkers Playing Program Without Using Human Expertise," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 4, pp. 422–428, 2001. [Online]. Available: http://www.natural-selection.com/Library/2001/IEEE-TEVC.pdf

[10] L. Messerschmidt and A. P. Engelbrecht, "Learning to play games using a pso-based competitive learning approach." *IEEE Trans. Evolutionary Computation*, vol. 8, no. 3, pp. 280–288, 2004.

[11] N. Franken and A. Engelbrecht, "Comparing pso structures to learn the game of checkers from zero knowledge," in *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*, R. Sarker, R. Reynolds, H. Abbass, K. C. Tan, B. McKay, D. Essam, and T. Gedeon, Eds. Canberra: IEEE Press, 8-12 December 2003, pp. 234–241.

[12] T. E. Fawcett, "Feature Discovery for Problem Solving Systems," Ph.D. dissertation, Department of Computer Science, University of Massachusetts, Amherst, MA, 1993. [Online]. Available: ftp://ftp.cs.umass.edu/pub/techrept/techreport/1993/UM-CS-1993-049.ps

[13] P. E. Utgoff and D. Precup, "Constructive Function Approximation," in *Feature Extraction, Construction and Selection: A Data Mining Perspective*, ser. The Kluwer International Series in Engineering and Computer Science, H. Liu and H. Motoda, Eds. Kluwer Academic Publishers, 1998, vol. 453, ch. 14.

[14] T. Kaneko, K. Yamaguchi, and S. Kawai, "Automated Identification of Patterns in Evaluation Functions," in *Advances in Computer Games 10*, H. J. van den Herik, H. Iida, and E. A. Heinz, Eds. Kluwer Academic Publishers, 2004, pp. 279–298. [Online]. Available: http://www.cs.ualberta.ca/~mburo/ps/ordinal.ps.gz

[15] W. H. Duminy and A. P. Engelbrecht, "Composing linear evaluation functions from observable features." *South African Computer Journal*, vol. 35, pp. 48–58, 2005.

[16] I. Althöfer, "Computer-aided game inventing," 10 2003.

[17] U. M. Fayyad and B. Keki, "Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning," in *IJCAI-93*, 1993, pp. 1022–1027.

[18] T. Uno, M. Kiyomi, and H. Arimura, "LCM ver 3.: Collaboration of Array, Bitmap and Prefix Tree for Frequent Itemset Mining," Chicago, IL, 8 2005.

[19] A. Hyvärinen, "A survey on independent component analysis," *Neural Computing Surveys*, vol. 2, pp. 94–124, 1999.

[20] S. Eyheramendy and D. Madigan, "A novel feature selection score for text categorization," in *Proceedings of the International Workshop on Feature Selection for Data Mining: Interfacing Machine Learning and Statistics (in conjunction with 2005 SIAM International Conference on Data Mining)*, Newport Beach, CA., April 2005.

[21] F. Fleuret, "Fast Binary Feature Selection with Conditional Mutual Information," in *JMLR Vol.5*, 11 2004, pp. 1531–1555.

[22] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification (2nd ed.)*. Wiley Interscience, 2000.

[23] M. Buro and I. Durdanovic, "An overview of neci's generic game server," 2001, http://citeseer.ifi.unizh.ch/600506.html.

[24] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in *Proc. of the IEEE Intl. Conf. on Neural Networks*, San Francisco, CA, 1993, pp. 586–591. [Online]. Available: citeseer.ist.psu.edu/riedmiller93direct.html

[25] M. Miwa, D. Yokoyama, and T. Chikayama, "Automatic construction of static evaluation functions for computer game players," in *Proceedings of the 9th International Conference on Discovery Science*, L. Tpdorovski, N. Lavrač, and K. P.Jantke, Eds. Springer, October 2006, pp. 332–336.