Pattern Matching in Go Game Records

Shi-Jim Yen[1], Tai-Ning Yang[2], Jr-Chang Chen[3], Shun-Chin Hsu[4]

[1]*Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien, Taiwan. sjyen@mail.ndhu.edu.tw*
[2]*Department of Computer Science, Chinese Culture University, Taipei, Taiwan. tnyang@faculty.pccu.edu.tw*
[3]*Department of Applied Mathematics, Chung Yuan Christian University, Chung Li, Taiwan. jcchen@cycu.edu.tw*
[4]*Department of Information Management, Chang Jung Christian University, Tainan, Taiwan. schsu@mail.cju.edu.tw*

Abstract

Many professional Go games, Go life-and-death problems and others are saved as digital game records by SGF (Smart Game Format). Valuable information hides in these records. This article presents a novel Go game record information retrieval system. In this system, the most difficult part is Go pattern matching in Go game records. In this article, a Go pattern matching algorithm is given to find game records that contain a desired query pattern. Then, a proposed index structure for a Go record database integrates methods of information retrieval and domain knowledge of Go. This index can increase the speed of pattern matching in the game database.

1. Introduction

Nowadays, many professional Go game records, tsumego (life and death problems), and other minor Go problems are saved as digital game records by SGF (Standard Go Format). A lot of important information is saved in these game records, such as the sequence of stones played and positions, names of the players, results and comments. Information retrieval in Go game records is very useful. [1][11][14][15][16] Additionally, if a player wants to know the winning rate for open game strategies, the Go information retrieval system can provide that answer. For those who research computer Go, retrieving game records can be useful for correcting the content of the Go pattern database or assist in debugging Go programs.

When retrieval information in game records, the most difficult part would be to find those task game records containing a desired Go pattern from numerous game records. This problem can be considered a pattern matching problems. A Go pattern is a particular arrangement of stones caused by their relative position. A pattern can be range from 3×3 to 19×19. Each position in a pattern can be white, black or empty, or a combination of white, black and empty. Through rotation, reflection, and changing color, one Go pattern can have 16 different variations on a game board.

Pattern matching is an important problem in computer Go. Many pattern matching methods have been proposed in [1][5][12][13][14]. However, most approaches only discuss about that: given a substantial number of patterns, check which patterns are matched on a game board. This article presents a novel approach for finding game records containing a desired query pattern from a lot of game records. In this system, input is a query pattern and the output is game records containing this query pattern. From these output game records, game record information, such as the name of the game record, move sequences, player names and results, can be obtained.

This article is organized as follows. The framework of proposed idea is shown in Section 2. Section 3 presents a novel Go pattern matching algorithm. Section 4 gives an index structure to increase the speed of Go pattern matching. Finally, Section 5 gives conclusions.

0-7695-2882-1/07 $25.00 ©2007 IEEE

2. Go Game Record Information Retrieval System

The goal of this work was to develop an efficient Go game record information retrieval system that uses the proposed indexing structure. Figure 1 shows the system framework. Ability to process both pattern and text queries is a critical feature of the system, as it allows users to search information with more query forms. For example, a user may wants to identify the winning rate when a specific player uses a specific opening pattern, by sending a query contenting player name, result, and the opening pattern. The related game records will output, and the winning rate can be then be computed. It is practicable for the system to execute a pattern query with a logical operation such as *and*, *or*, *not*, etc. The proposed structure increases the speed of these operations.

Figure 2 presents the procedure and major components of the proposed system. This approach is based on a sequential matching algorithm, which is used when construction the database and during searching. In addition to the searching algorithm, two principal factors affect system performance: feature patterns, and a feature tree.

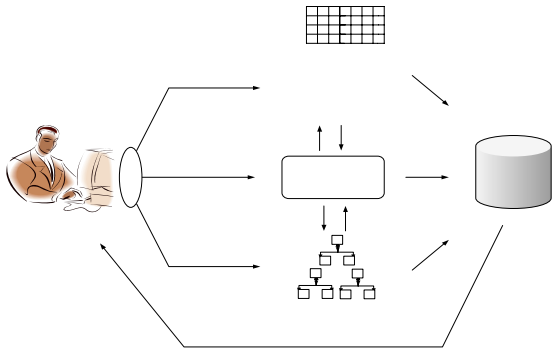


Fig. 1. Go game record information retrieval system.

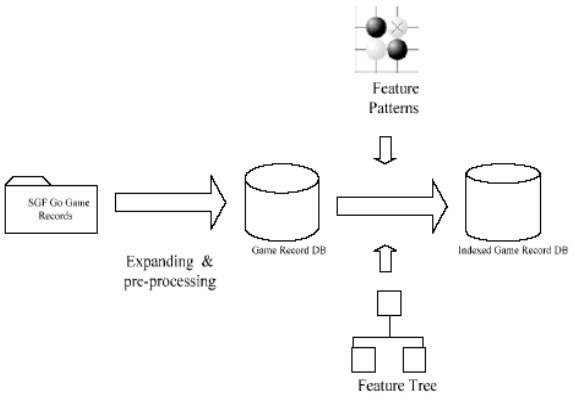
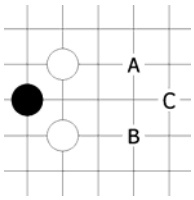


Fig. 2. Constructing an index structure.

3. A Go pattern matching algorithm

Before pattern matching, each game board and $n \times m$ query pattern is transformed into a 19×19 array and an $n \times m$ array, respectively. Each item in an array is represented by 3 bits. For example, Figure 3 shows a 5×5 query patterns. Each position denoted by A, B and C can be a black stone or empty. Three bits are used to express four possible states at every intersection. Bit 1 indicates that this point could be empty or not; bit 2 signifies that this point could be black stone or not; bit 3 denotes that this point could be white stone or not. Each number in the array is generated from 3 bits. For example, a number 3 in the array means that its relative binary number is 011, indicating that this point could be black stone or empty.



11111
14131
21113
14131
11111

Fig. 3. Format of a query pattern.

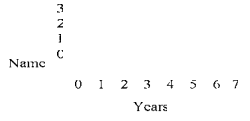
The KMP string matching algorithm avoids unnecessary matching. [10] However, the KMP algorithm is a one-dimensional string matching algorithm. Hence, a 2-D pattern should be regarded as a set of 1-D patterns. The Go pattern matching algorithm is processed along following steps.

Step1. *Choose one row from the query pattern.* This row is considered as a row pattern. It is a 1-D pattern and suitable for the KMP algorithm. Row patterns in the set are ranked according to the following criteria:

State: Choose the row pattern with a fixed state.
Length: If there are more than one fixed state row pattern, choose the longest row pattern. If the row patterns have the same length, then choose the row pattern with the *highest degree of repetition*.

Step2. Search the row pattern obtained in step 1 on the board (row by row) by using the KMP algorithm.

Step3. If the row pattern appears on the board, then check sequentially every element in



Text Query like

the query pattern with points around the row pattern by bit mapping. The bit mapping algorithm is described as follows. Let P represents the Query pattern and S represents the relative area around the matched row pattern in the board. Compute the following three statements.

Let R = P AND S
If R = S, then output the matched position
Else fail

Repeat Step 1~3 until all the rows are scanned.

If the size of query pattern is $n \times n$, only the $(2n-1) \times (2n-1)$ local area around the last move on the game board is scanned. Because new possible patterns only appear in this area of the board. If some stones were removed because of last move, the effect of these removed stones must be considered, as removed stones may generate new patterns. The size of scanned area depends on the variation of the game board. The scan area can be defined as $(2 \times (n-1) +$ *maximal length of variation of game board*$) \times (2 \times (n-1) +$ *maximal wide of variation of game board*$)$.

If the size of query pattern is nxn, time complexity on each game board is $O((2n-1) \times (2n-1))$. In many Go pattern matching application, some positions of the query pattern may be located in the board edge as in Fig. 4. When search edge pattern, it will only need to scans along an edge of board and the time complexity will be $O((2n-1) \times (2n-1) / (2n-1)) = O(2n-1)$ (only shift along the edge). Similarly, if some positions of the query pattern are located in the board corner as in Fig. 5, it will only need to consider a corner of boards and the time cost of corner matching will be $O((2n-1) \times (2n-1) / (2n-1) \times (2n-1)) = O(1)$ (without any shift during searching).

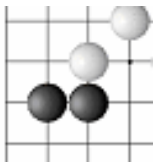


Fig. 4. An edge pattern.

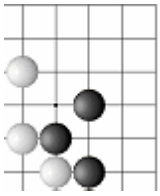


Fig. 5. A corner pattern.

For a desired query pattern, all actions (rotation, reflection, and color changes) on the query pattern on game board must be considered. The 16 related patterns generated from the query pattern through these actions must be found.

An experiment applying the Go pattern matching algorithm was performed. Table 2 shows the experimental environment. During the experiment, the pattern matching program will output all game boards containing the query pattern. When the query pattern did not include an edge, the average search time was 10－20 seconds in the environment. Figure 6 shows the searching times for the different query pattern sizes (3×3, 4×4,..., 19×19). Figure 6 also shows the relationship between searching time and pattern size. The curve first goes up and then drops down. Generally, the smaller query pattern the more time-consuming in a pattern matching algorithm. But in proposed Go pattern matching algorithm, a large query pattern will cause large searching area.

According to experimental results, average time cost is too long for real applications. This problem is solved by indexing these game boards.

Table 2. The environment of experiment.

Computer	Pentium 4, 3.0 G, 1GB RAM
Operating System	Windows XP
No. of Game record	6,292 game records(6.7MB)
No. of game board	1,341,801 game boards (790.3MB)
language& Database	C++ MySQL 4.0.15

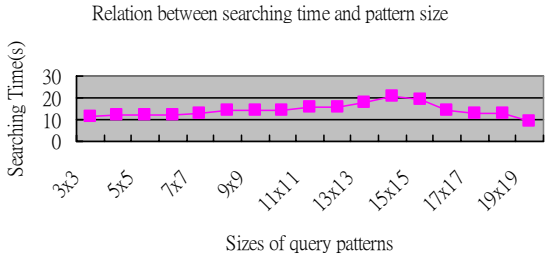


Fig. 6. Searching time at different query pattern sizes.

4. Feature Patterns and Index Structure

To construct index structure on the game record database, the features of many significant query patterns should be extracted and used as the index key. The proposed approach uses 400 featured patterns. Feature patterns are some patterns that appear frequently in Go games. These patterns are part of pattern database in the computer Go program JIMMY, which was developed from 1994[19]. The pattern database is maintained by S.J. Yen, who is a Taiwan amateur 6-dan Go player. Almost all significant moves can be recognized by the pattern database system. Each

feature pattern has its own meaning in Go games and can be considered as *jump*, *knight move*, or *diagonal* move in a Go game; however, most of them are more complicated.

These feature patterns are good indexing keys. Each has its own characteristics and meanings. By using them and a suitable indexing structure, game records can be classified systematically.

The feature patterns are integrated by a popular index structure: the *inverted list*. [4] The proposed sequential matching algorithm is applied to find the game boards containing each feature pattern, and build an inverted file for our game records.

All game boards were classified according to the feature pattern(s) they contain. This will be a good characteristic for matching processing. Almost every reasonable query pattern contains at least one feature pattern, and it is easy to perform logical operations such like "AND" or "OR" under this structure.

With a desired query pattern, we first check which feature patterns it contains. This can be done quickly by algorithm in [4]. Then we only need to apply the proposed Go pattern matching algorithm on this query pattern in those game boards containing the result feature patterns. For example, if the pattern matcher shows that the query pattern contains feature pattern 1 and feature pattern 3, we will only search those game boards in the set $(1 \cap 3)$.

5. Conclusion

This article presented a novel a Go pattern matching algorithm. If query patterns are edge patterns or corner patterns, game records containing this query pattern can be obtained immediately. A simple and efficient index structure was also proposed. The index is based on feature patterns extracted from the pattern database of the computer Go program JIMMY. This index structure increases the speed of pattern searching to a satisfactory time.

The proposed approaches can solve the most difficult part of a Go game record information retrieval system. String matching for text in Go game records is easily implemented. The matching can be considered as text information retrieval. With the proposed methods and a string matching algorithm, a Go game record information retrieval system can be developed that accommodates both pattern view and text view queries. This system will be useful for Go players and computer Go researchers.

6. References

1. B. Bouzy, and T. Cazenave, "Computer Go: An AI oriented survey," *Artificial Intelligence* 132, 2001, pp. 39-103.
2. M. Boon, "A Pattern Matcher for Goliath," *Computer Go*, No.13, 1990, pp.12-24.
3. T. Cazenave, "Generation of patterns with expernal conditions for the game of Go," *Advances in Computer Games*, Vol. 9 Uni. Of Limburg, Maastricht, 2001.
4. W. Frakes and Y. R. Baeza, *Information Retrieval: Data Structures and Algorithms*, Prentice-Hall.
5. GnuGo website. http://www.gnu.org/software/gnugo/gnugo.html.
6. Hollosi, A. SGF FF4-Smart Game Format, 2004. http://www.red-bean.com/sgf.
7. K.T. Hsu *Pattern recognization in Go game records*, M.Sc. Thesis, Department of Computer Science and Information Engineering, National Taiwan University, Taiwan, 2004. (in Chinese).
8. H. Iida, M. Sakuta and J. Rollason, "Computer shogi," *Artificial Intelligence* 134, 2002, pp. 121-144.
9. J. Karkkainen and E. Sutinen, "*An Efficient Index Structure for String Databases*," *Proc. of the 27th VLDB Conference*, pp. 351-160, 2001.
10. D.E. Knuth, J.H. Morris, and V.R. Pratt, "Fast pattern matching in strings," *SIAM Journal on Computing*, 6(2), 1997, pp323-350.
11. T. Kojima and A. Yoshikawa "Knowledge Acquisition from Game Records. Johannes Fürnkranz and Miroslav Kubat," 16th International Conference on Machine Learning (ICML-99), Bled, Slovenia, 1999.
12. Y.T. Lee, *Pattern Matching in Go based on Patricia Tree.* M.Sc. Thesis, Department of Computer Science and Information Engineering, National Taiwan University, Taiwan, 2004. (in Chinese)
13. M. Mueller, "Pattern Matching in Explorer," *Proceedings of the Game Playing System Workshop*, pp.1-3, Tokyo, Japan, 1991.
14. M. Muller "Computer Go," *Artificial Intelligence*, 134(2003). pp. 145-179.
15. T. Nakamura and T. Kajiyama, "Feature Extraction from Encoded Texts of Moves and Categorization of Game Records," *Game Informatics*, Vol. GI-2-11, 2000.
16. J. Nunn, "Extracting information from endgame databases," *International Computer Go Association (ICGA) Journal*, 1993, pp. 191-200, ISSN 1389-6911.
17. S.J. Yen, *Design and Implementation of a Computer GO Program JIMMY*. Phd. Thesis, Department of Computer Science and Information Engineering, National Taiwan University, Taiwan, 1999. (in Chinese)
18. S.J. Yen, J.C. Chen, T.N. Yang and S.C. Hsu, "Computer Chinese Chess," *International Computer Go Association (ICGA) Journal*, Vol. 27, No. 1, March 2004, pp. 3-18, ISSN 1389-6911.