B-Tree Database for the Relative Strength of Chess Positions

Abstract

Chi-square (χ^2) statistics for chess positions are stored in a data structure as a binary tree (B-tree) in the model supplied by TrueBASIC. The record size is a compact, variable length of 11- to 26-bytes. The index key is of a fixed length of 10-bytes. The key contains the χ^2 statistic followed by a bit-mask of the chessboard squares as occupied by pieces. Because the key separator is preset in the model, the statistic key is physically stored in inverse order with the decimal part followed by the integer part. The order is logically reversed for radix sorting. The length of the data portion varies as 1- to 16-bytes to store 1- to 32-pieces at 4-bits per piece.

Introduction

From previous work ([James 2004, 2005]), Chi-square (χ^2) statistics are calculated for respective chess positions. This paper describes how the chess positions and statistics are stored in a binary tree database for rapid access.

Problem and Solution of Compact Representation

The problem is to represent the data required to describe a chess position and the associated χ^2 value in a compact representation that is rapidly accessible. The most compact way to represent a chess position is with a mask for the chessboard and a mask for the respective pieces.

The mask for a chessboard has 64 switches. The most compact mask is a 64-bit string representing which squares are occupied as a "1" bit and not occupied as a "0" bit. Such a mask describes only which squares are occupied or not occupied but does not specify the piece occupying a respective square.

The mask for the respective pieces represents 12 possible pieces as the black (b) and white (w) bishop (B), king (K), knight (N), pawn (P), queen (Q), and rook (R). The most compact description for these 12 pieces fit into the 4-bits of $2^4 - 1$ or 0 …15, namely in the range of 4-bits as 0000 … 1111. The white and black groups of pieces do not necessarily fall into consecutively numbered groups. For example, wB … wR may be numbered 1 … 6 in 4-bits as 0001 … 0110. Similarly, bB … bR may be numbered as 9 … 14 in 4-bits as 1001 … 1110. The reason for making the group of black numbers not consecutive to the group of white numbers is intuitive. The group of 4-bits for black in the range of 1001 … 1110 is in fact the negative number complement of the group of 4-bits for white in the range of 0001 … 0110. In other words, the value of a black bishop is the negative number of the value of a white bishop such that bB = -1 * wB. This makes the implementation of the representation even more compact if embedded in hardware

where the numbering scheme is not trivial because it may increase performance for repeatedly executed code.

Therefore a chess position location in its most compact format consists of the board mask of 64-bits or 8-bytes plus the piece string of 32 potential pieces at 4-bits per piece or up to 128-bits or 16-bytes. The board and piece string may vary in length and occupy a total of 11- to 24-bytes. The length may as small as 11-bytes from the 2-bytes for the statistic plus 8-bytes for the board plus 4-bits or ½ byte for only one piece on the board, rounded up to 1-byte and totaling 11-bytes.

The χ^2 value is in a range of about 39.82 … 173.36. The value of 0 is possible if the statistic is not known. The integer value of that range fits into 1-byte of 8-bits of $2^8 - 1$ or 0 … 255, namely bits 0000 0000 … 1111 1111. The rounded, integer value of the fractional part of the decimal also fits into 1-byte in the range of 0 … 99. This statistic adds 2 bytes to the total record length. The smallest representation of a compact position and its statistic is 9-bytes plus 2-bytes or 11-bytes. Hence the largest representation of a compact position and its statistic is 24-bytes plus 2-bytes or 26-bytes.

Problem and Solution of Storing the Compact Representation

The next problem is how to store the 26-byte record in a compact database. The data items must be accessible by some fast indexed structure.

The solution to storing the compact representation in a database is to choose the most compact data structure such that the data itself forms the index of the structure. The binary tree or B-tree serves this purpose. A simplified implementation of the binary tree is found in a toolkit of TrueBASIC. A limitation of that particular implementation is a key length of not more than 100 bytes. A further constraint is that the first byte of an index key may not be the separator value in 1-byte of CHR$(127) as bits 0111 1111. To guard further against the separator value being that of the first byte in the key index, the statistic value occupies the leading two bytes of the key as follows. The statistic is physically stored with the decimal part preceding the integer part. The value of the rounded, integer of the decimal part is in the range of 0 … 99 which is always less than the separator value of 127. To sort the statistic as a radix key, the values of the two bytes are logically reversed.

Therefore the key length is always in the range of 11- to 24-bytes, blocked as 2-bytes for the statistic (in reversed order), 8-bytes for the board mask. The data portion of the record is 1- to 16-bytes, making a total record length of 11- to 26-bytes.

The total indexed portion of the record is extracted in TrueBASIC by rec$[1: 10]. The χ^2 statistic is extracted by rec$[1: 2] and placed in correct string order by rec$[2: 2] & rec$[1: 1]. The board mask is extracted by rec$[3 : 10]. The variable sized data portion of the record, the piece string, is extracted by rec$[11: LEN(rec$)].

The board mask and piece string are built by the following code using only built-in language features where the piece values are the respective 4-bit values from above:

```
LET rows = 8
LET columns = 8
LET square_bit$ = Repeat$( CHR$( 0), 8)      ! 1-bit per square x 64 squares = 64-bits = 8-bytes
LET piece$ = Repeat$( CHR$( 0), 16)     ! 4-bits per piece x 32 pieces max = 128-bits = 16-bytes
LET piece_count = 0

FOR row_index = 1 to rows
   FOR column_index = 1 to columns

      ! Build board mask
      SELECT CASE pieces$( row_index, column_index)
      CASE "  "            ! pieces$() is size 2-bytes per element
         ! Leave as bit "0"
      CASE ELSE
         ! put bit "1" into bit col_idx + ( row_idx - 1) * 8
         CALL PACKB( square_bit$, ( ( ( row_index - 1) * 8) + column_index), 1, 1)
         LET piece_count = piece_count + 1
      END SELECT

      ! Build piece string
      SELECT CASE pieces$( row_index, column_index)
      CASE "wB"
         CALL  PACKB ( piece$, ( (piece_count - 1) * 4) + 1, 4, wB)
      CASE "wK"
         CALL PACKB ( piece$, ( (piece_count - 1) * 4) + 1, 4, wK)
      CASE "wN"
         CALL PACKB ( piece$, ( (piece_count - 1) * 4) + 1, 4, wN)
      CASE "wP"
         CALL PACKB ( piece$, ( (piece_count - 1) * 4) + 1, 4, wP)
      CASE "wQ"
         CALL PACKB ( piece$, ( (piece_count - 1) * 4) + 1, 4, wQ)
      CASE "wR"
         CALL PACKB ( piece$, ( (piece_count - 1) * 4) + 1, 4, wR)
      CASE "bB"
         CALL PACKB ( piece$, ( (piece_count - 1) * 4) + 1, 4, bB)
      CASE "bK"
         CALL PACKB ( piece$, ( (piece_count - 1) * 4) + 1, 4, bK)
      CASE "bN"
         CALL PACKB ( piece$, ( (piece_count - 1) * 4) + 1, 4, bN)
      CASE "bP"
         CALL PACKB ( piece$, ( (piece_count - 1) * 4) + 1, 4, bP)
      CASE "bQ"
         CALL PACKB ( piece$, ( (piece_count - 1) * 4) + 1, 4, bQ)
      CASE "bR"
         CALL PACKB ( piece$, ( (piece_count - 1) * 4) + 1, 4, bR)
      CASE ELSE
         ! Fall through
      END SELECT

   NEXT column_index
NEXT row_index
```

The data portion of the record is built by the following code:

```
LET chi2$ = Repeat$( CHR$( 0), 2)  ! 8-bits per part of chi2 * 100 with no decimal
! This correctly rounds the decimal chi2 value into chi2$[ 1: 1] and [ 2:2]
LET chi2$[ 1: 1] = CHR$( INT( ( chi2 + 0.005 - INT( chi2 + 0.005)) * 100)) ! fractional part
LET chi2$[ 2: 2] = CHR$( INT(    chi2 + 0.005)) ! integer part
```

The complete record for database entry is built by the code:

```
LET db_entry$ = chi2$ & square_bit$ & pieces$
```

Move Order Not a Problem

The move order of pieces is not a problem because the purpose of the database is to store the statistical values for discrete positions. A given position may be obtained by any number of move variations. Therefore the move order leading to a position is not stored.

The length of the key may be delimited by parsing the record as follows. Bytes 3 … 10 of the record show the squares of the chessboard that are occupied by pieces. A count of the number of bits set as 1 shows the number of pieces on the board. This is in the of a function Num_1_bits that takes as input the board mask of square_bit$ and returns a value, such as in this code:

```
DEF FN Num_1_bits( square_bit$)
   LET bits_per_byte = 8
   LET num_bits = 0
   FOR bit_idx = 1 TO bits_per_byte * LEN( square_bit$)
     IF UNPACKB( square_bit$, bit_idx, 1) = 1 THEN
          LET num_bits = num_bits + 1
     END IF
   NEXT bit_idx
END DEF
```

An numerical alternative to reading bits with the UNPACKB function that is built into TrueBASIC is to count the number of divisions by 2 of the number represented by each of the bytes in the 8-byte string. This type of programming operation is effectively also known as bit shifting, a term for fast multiplication and division. This is in the form of a function Num_bits that takes as input the board mask of square_bit$ and returns a value, such as in this code:

```
DEF FN Num_1_bits( square_bit$)
   LET num_bits = 0
   FOR byte_idx = 1 TO LEN( square_bit$)
     LET temp = ORD( square_bit$[ byte_idx: byte_idx]) + 1
     FOR bit_idx = 1 TO bits_per_byte
          LET temp = temp / 2
          IF temp >= 1 THEN
             LET num_bits = num_bits + 1
          END IF
     NEXT bit_idx
   NEXT byte_idx
```

```
END DEF
```

It is also necessary to determine the variable number of bytes needed to store the pieces at 4-bits per piece. This is in the form of a function Num_bytes that takes as input the number of pieces as num_bits from above and returns a value, such as in this code:

```
DEF FN Num_bytes (num_bits)
   LET num_bytes = INT( num_bits / 2 + 0.5)
END DEF
```

However, the move order may be stored optionally as the data portion added to the record described above. This requires viewing that entire record of bytes [1: 11] to bytes [1: 26] as a variable length key with the unlimited move order stored effectively as N-moves x 5-bytes per move in the variable length data portion.

Conclusion

What remains is to populate the database automatically from previous and continuous data results and to interface this database directly into computer chess programs such as those of the largest manufacturer ChessBase GMBH, Hamburg, Germany.

Acknowledgments

Thanks are due to Tom Kurtz and Christopher Sweeney of TrueBASIC for producing the TBTree toolkits as a teaching framework and to chess masters Mikhail Ponomarev and Philipp Ponomarev for helpful discussion at each stage of this continuing project.

References

[James 2004.] PRIA-7-2004, Poster “Statistical Analysis of the Relative Strength of Chess Positions”, Seventh Annual International Conference on Pattern Recognition and Image Analysis: New Information Technologies, St. Petersburg, Russian Federation.

[James 2005.] “Statistical Analysis of the Relative Strength of Chess Positions”, Pattern Recognition and Image Analysis, MAIK Nauka/Interperiodica Publishing, Moscow, Russian Federation, Abstract text 2005, Vol. 15, No 1, and Full text 2005, Vol. 15, No. 2.