THE AMATEURS' BOOK-OPENING ROUTINE

John F. White

Workingham
England

ABSTRACT

An easy, memory-efficient method for storing readable book-openings is described. It achieves its two objectives: to be easy to input and to be convenient to alter. The book has the additional advantage of being presented in a form easily edited for adequate perusal by human beings. It is especially applicable when memory is tight

1. INTRODUCTION

A book-opening library forms a valuable part of the armoury of most modern chess programs. Moves made by reference to the library have the advantages that·

i) they are made almost instantly, saving time for subsequent moves;

ii) non-obvious moves of strategic significance can be made;

iii) deep opening traps can be avoided without calculation

This article describes the routine that I use for my own chess programs. Book openings have previously been described in this journal by K. Spracklen (1983), using a principle ascribed to K. Thompson. The significant difference between a professional book-opening library and that of an amateur lies in the trade-off between time and cost.

A professional programmer's labour will be divided among many units of many similar chess programs, for which the savings of computer memory may be significant. The amateur programmer, if he sets any store on the value of his time at all, will certainly find it cheaper to buy more computer-memory space (or disk storage) than to write the complicated additional utilities needed to make efficient use of memory space.

2. BOOK STRATEGIES

A number of observations can be made about computer book-opening libraries.

From my own experience, it is apparent that *breadth* of the library is more important than *depth* of individual openings. This is because knowledgeable human players, and wily computer programmers, aim to take their machine opponent out of its openings' book as soon as possible. The advantage of this approach is that the machine then spends valuable time thinking about developing pieces, which are frequently placed on the wrong squares anyway.

A second requirement is that the book opening should be capable of easy revision, so that the program does not keep falling into the same errors. Probably most chess programmers are familiar with the unnerving experience of seeing their creation make a disastrous blunder immediately after emerging from a long opening sequence

I do not share the view that all free memory should be used to store book openings. A balanced book is more important than a full book, especially with regard to the ease with which the program can be taken out of book Rather, I believe that it is more important to leave space for revising the book library as necessary. Naturally, this consideration does not apply to commercial chess computers where unused computer memory is wasted memory.

Condon and Thompson (1983) have recorded that Belle's enormous book-openings library has conferred on Belle very little competitive advantage. My own chess program has only 175 book lines (some 2,000 moves) and occupies less than three-quarters of the allocated space.

Finally, it is desirable that a chess program should be able to pick between several possible alternative book continuations.

3. REPRESENTATION OF BOOK-OPENING MOVES

Book openings may be stored in two ways: as complete positions with which the associated book response is recorded and as lengthy strings of moves.

3.1. Storage of Complete Positions

This method has the great advantage of permitting book moves to be found after transposition of moves. That is to say, no matter what sequence of moves is used to reach a position, a stored response will always be available from that position.

The difficulty with this approach for the amateur is that of automation: creating an auxiliary program into which book positions may be fed, the resulting board position encoded and the subsequent book move stored.

A secondary problem is that it takes a great deal of computer memory to store complete board positions, although it may be deduced that 40 bits may suffice to define most opening board positions uniquely (see Nelson, 1985). In the latter case, a hashing technique will be needed to reduce the board position, and another routine will subsequently be required to check all the hashed positions for uniqueness. Alternatively, standard data compression techniques such as Huffman coding could presumably be used.

Levy (1985) has stated that the storage of board positions is "an opening feature of doubtful value" in view of the cost in programming time and extra memory.

3.2. Storage of Move Strings

The Spracklen-Thompson system for storing move strings aims at the association of one move with *one* byte of memory. This is accomplished by storing, not the moves, but the offset of the required move into the legal list of moves generated by the program at each position.

The difficulty of this approach for the amateur programmer lies, as with storing board positions, in the creation of a separate routine to calculate the move offsets. If moves are instead stored in the conventional FROM-TO manner, the Spracklen-Thompson system requires the use of *two* bytes of memory for each stored book response.

The Spracklen-Thompson system additionally permits a limited degree of move transpositions, since short move strings may be followed by a 'JUMP' command into the middle of a longer move string. Such jumps are difficult to calculate, and need revision whenever the book library is expanded.

Finally, the Spracklen-Thompson system stores book moves in an efficient, but highly unreadable form. For example, the book opening lines of Table 1 would be stored as:

(e4(e5(Nf3(Nc6(Bb5a6)Bc4)Nf6)f4) (c5Nf3(d6d4)Nc6)e6)

It requires little imagination to picture the prospect of debugging such a mixture for a 2,000 move book opening, while sitting in one's cold, dark spare bedroom at midnight.

e2e4e7e5g1f3b8c6f1b5a7a6
e2e4e7e5g1f3b8c6f1c4
e2e4e7e5g1f3g8f6
e4e4e7e5f2f4
e2e4c7c5g1f3d7d6d2d4
e2e4c7c5g1f3b8c6
e2e4e7e6

Table 1: Example of book opening moves in the FROM-TO notation.

4. AN AMATEUR BOOK ROUTINE

The book opening method which I have evolved for my own use is also based on the storage of move strings. The strings are entered into a word processor exactly as shown in Table I. It is *very* important that similar strings, i.e. those starting with identical moves, are stored adjacent to one another.

A short BASIC program next examines each string to see whether the front part of the string, counting every four characters, is identical to its predecessor. If so, the front (matched) part of the string is replaced with an asterisk followed by the number of bytes (FROM-TO) that will be replaced.

For example, the string data of Table 1 would be reduced to the string data of Table 2:

e2e4e7e5g1f3b8c6f1b5a7a6
*8f1c4
*6g8f6
*4f2f4
*2c7c5g1f3d7d6d2d4
*6b8c6
*2e7e6

Table 2: Reduced form of book-opening moves of Table 1.

A second BASIC program then examines the reduced data as per Table 2 and POKEs the corresponding moves as hex values (FROM-TO) into memory. Special tokens represent the '*n*' code and the end of each move string is marked by having the most significant bit of the last move byte set to 1.

5. USE OF THE AMATEUR OPENING BOOK

In operation, the moves played by both sides are stored contiguously in a buffer (BUFFER0) which maintains a record of the actual game as played. The same moves are additionally stored in BUFFER1. The reason for using BUFFER1 will become apparent in Section 6.1. (Transpositions).

When the program looks for any move in the book opening, it first extracts the first string of the book and stores it in BUFFER2. If BUFFER1 matches the front part of BUFFER2, counting bytes in pairs, the next two bytes from BUFFER2 (if they exist) contain the book response. The following two bytes after that provide a hint for the program's opponent.

If, however, there is no match, the program copies the next book string into BUFFER2, but with the important proviso that the first n bytes of BUFFER2 are skipped if the new string begins with a '*n*' token. Thus, from the example of Table 2, the second book string would be copied into BUFFER2 starting at byte 8 of BUFFER2 (skipping bytes 0 to 7).

The new BUFFER2 contents are compared with BUFFER1. This process is continued until the whole book string has been traversed. BUFFER2 will be completely rewritten only when the next considered book string lacks the '*n' token.

This method of book-opening compression can be extremely efficient since an average of less than one extra byte - relative to the Spracklen-Thompson system - is needed for each opening line stored. Thus, considering the data in Table 1, there are 29 moves requiring 58 bytes of storage in the FROM-TO representation; the Spracklen-Thompson scheme requires 30 bytes and the Amateur Book routine requires 36 bytes. This assumes the use of two bytes of memory for each move stored, as previously explained; however, all the systems are amenable to use of the offset method to find the move from the current legal list.

It can be seen that the Amateur Book routine is particularly efficient with closely-related deep opening lines, but poorer with a broad opening library (a feature shared by the Spracklen-Thompson system). It has the advantage that the original book opening text is easy to read and to correct on the word-processor, while the modifying programs in BASIC are short and simple to write.

6. ENHANCEMENTS

Four further enhancements to the Amateur Book routine are possible.

6.1. Transpositions

When the opening sequence of moves is copied into BUFFER1, it is possible to make transpositions in BUFFER1 by reference to a look-up table. For example, the string c2c4e7e6d2d4... is changed after copying to d2d4e7e6c2c4....

This method is limited by the size of the look-up table, while an excessive number of comparisons would be expected to slow the book routine too much. Nevertheless, I have managed to pack in a lot of such transpositions without noticeable effect on the speed of book search. Instead, my limit is that of available free memory.

6.2. Book Window

An additional feature of storing similar book strings together is that use of a 'book window' becomes feasible. Instead of 'walking' the whole book opening to match moves, two 'book limits' are set (initially to zero and + infinity). When matches are found with the book strings, the 'book limits' are reset to mark the highest and lowest book strings with which matches were found. This creates the 'book window'.

When the book routine is next called, the book opening library is searched only within the memory area defined by the book window. The book window will then again be narrowed to reflect the decreasing range of book choices.

Transposition routines, when present, have to be considered carefully for their effect on the book window. The easiest solution is to re-initialize the window whenever a transposition is made to BUFFER1.

6.3. Random Choice

It often happens that several alternative book strings provide different continuations from the present move sequence. The continuations could be stored in yet another buffer (BUFFER3) and a choice subsequently made among them. However, in practice – for reasons of economy of memory – I prefer to accept the first book response encountered and then to replace the first choice randomly – with high probability of acceptance – with any subsequent alternatives as they arise. Accordingly, the move strings are listed in *increasing order of merit*, always subject to the constraint that similar move strings are stored together.

6.4. Universal Book Opening

It is possible to play the move 1. ... e5 as a response to any opening move by White other than d4, f4 or Nf3. Consequently the move e5 can be used as a 'universal' book response for any opening move not found in the book opening library, provided that replies to d4, f4 and Nf3 have been stored.

In practice, this universal book move should be stored separately from the book-opening library, and considered only if the program exits from the library without a match after White's first move.

My own chess program opens a game as White with a move selected at random from e4, d4 and c4 without reference to the book-opening library.

7. PERFORMANCE

The Amateur Book routine described here works perfectly on an Atari 130XE domestic microcomputer (6502 CPU at 2 MHz; 64K RAM). Moves are made, inclusive of transpositions, within half a second per book move without the 'book window'; virtually instantly with the 'book window'.

Changes to the book opening are easily made by altering the word-processor text file. The same file can conveniently be read for checking or proofreading.

8. REFERENCES

Condon, J.H. and Thompson, K. (1983). BELLE. *Chess Skill in Man and Machine* (Ed. P.W. Frey), 2nd edition, pp. 201-210. Springer-Verlag, New York.

Levy, D.N.L. (1985). Quoted in: Harding, T.D. (1985). *The New Chess Computer Book*, pp. 238-239. Pergamon Press, London.

Nelson, H.L. (1985). Hash Tables in Cray Blitz. *ICCA Journal*, Vol. 8, No. 1, p. 6.

Spracklen, K. (1983). Tutorial: Representation of an Opening Tree. *ICCA Newsletter*, Vol. 6, No. 1, pp. 6-8.

INFORMATION FOR CONTRIBUTORS

Contributors may be interested to know that the *ICCA Journal*, as of Vol. 10, No. 1, is a source for the Institute for Scientific Information® (ISI) for inclusion in the CompuMath Citation Index® (CMCI®), the Automatic Subject Citation Alert (ASCA®) and SCISEARCH®, ISI's on-line database. The Journal is also a source for the Information Company R.R. Bowker for inclusion in the International Serials Database which is a source for Ulrich's International Periodicals Directory and the DIALOG on-line service.

Being included in the CMCI®, the *ICCA Journal* is one of the 400 Journals in mathematics, computer science, statistics, operations research, and related disciplines which is abstracted and/or indexed and/or available as tearsheets; this means that the Journal now is accessible in (on-line) database form.

Submission of material

Contributions to the Journal are welcomed in any form, although preferably by E-mail or on a MS-DOS formatted 5.25 inch diskette. In case contributors prepare their manuscripts with high-quality wordprocessors, it should be noticed that text-files in VENTURA, TEX or TROFF format are processable directly by the Editors, thereby alleviating their task considerably.