Encoding Chess Positions and Moves

by Wayne Hayes, wayne@cs.toronto.edu

This is an informal discussion of encoding chess positions and moves. It is very rough, drawn from an assignment in a graduate course in information theory taught at the University of Toronto Department of Computer Science.

A. I will use a 64-bit-vector to encode whether each square on the board has a peice on it (0=vacant, 1=occupied), and then encode which piece is in the square. For checkers, we need $P = 2$ bits to store the piece (4 possibilities: red, black, red crown, black crown). For chess, we need $P = 4$ bits (12 possibilities: white and black for each of King, Queen, Bishop, Knight, Rook, Pawn). Near the end-game, when the board is almost empty, we could also run length encode the number of unoccupied squares. This adds a bit to the encoding telling us whether RLE is in effect or not. I would guess that a good field length for the RLE would be $L = 3$ or $L = 4$ bits, allowing us to encode 8 or 16 squares (1 or 2 rows worth) of unoccupied squares. If N is the number of pieces on the board (max $N = 32$ for chess, or $N = 24$ for checkers), then this encoding takes

$$65 + PN$$

bits for RLE= 0, or less for $RLE = 1$ (by default, we'd only use RLE= 1 if it was shorter). This gives a maximum encoding length for a position of $65+2\times 24 = 113$ bits for checkers, or $65+4\times 32 = 193$ bits for chess. For example, if only half the pieces are left on the board, and RLE= 0, then the encoding length would be $65+2\times 12 = 89$ bits for checkers, or $65+4\times 16 = 129$ bits for chess. With $RLE = 1$, N half the total, and $L = 3$, the 65 would be reduced to about $NL \approx 24$ for checkers or ≈ 48 for chess. Note that this allows all valid chess positions, including ones where there is more than 1 queen, or more than 2 rooks, bishops, or nights. It also allows invalid ones, like more than 32 pieces total, two kings both in check, *etc.*

If we wanted to get *really* silly, we could group the 4-bit chess pieces since only 12 of the 16 possibilities are used, but I think that would be overly complicating things. Another possible over-complication could be to Huffman encode the pieces. However, I don't think that would help at all in checkers, and the only piece that we can be sure to exist in chess is the King, and the other non-pawn pieces, from my experience, all frequently appear in the end-game, so a variable-length encoding is unlikely to help.

An alternate encoding (instead of RLE) near the end-game, since N will be very small, is to resort to encoding only the squares that actually have pieces on them. This adds one more bit to the encoding, telling us whether we're using the "bit-vector" method or this "only-occupied" method. A square's position can be encoded in exactly $S = 6$ bits (3 bits = 8 squares for each dimension), followed by the $P = 2$ or $P = 4$ bit encoding of the piece for checkers or chess, respectively. The encoding length would then be exactly

$$(S + P)N.$$

To compute the approximate break-even point between the bit-vector method and this only-occupied method, I'll ignore RLE in the bit-vector method and solve for N in

$$(S + P)N = 64 + PN$$

giving $N = 64/S = 64/6 = 10.7$. For $N = 11$, then, the break-even is 86 bits for checkers and 104 bits for chess. Above $N = 11$, we'd use the bit-vector, otherwise we'd use "only-occupied", and so if there are less than 11 pieces, we use less than 104 bits (for chess).

The average number of bits for my encoding depends critically on the distribution of N, the number of pieces present in a particular position. To estimate the average N, I consulted my local chess expert, Luis Dissett (my office mate, who plays chess alot). He pointed me at the URL

`ftp://chess.onenet.net/chess/Game-Databases/PGN/Events/`,

which contains databases of all World Chess Championships since 1834, plus about 50 other international tournaments. There are 4267 games containing 353,000 positions giving about 80 positions per game. I spent some time computing a histogram of the number of positions having a prescribed number of pieces. The histogram, converted to probability of a random position having N pieces in it, is presented in Figure 1. From this histogram we can compute the mean number of pieces in a position,

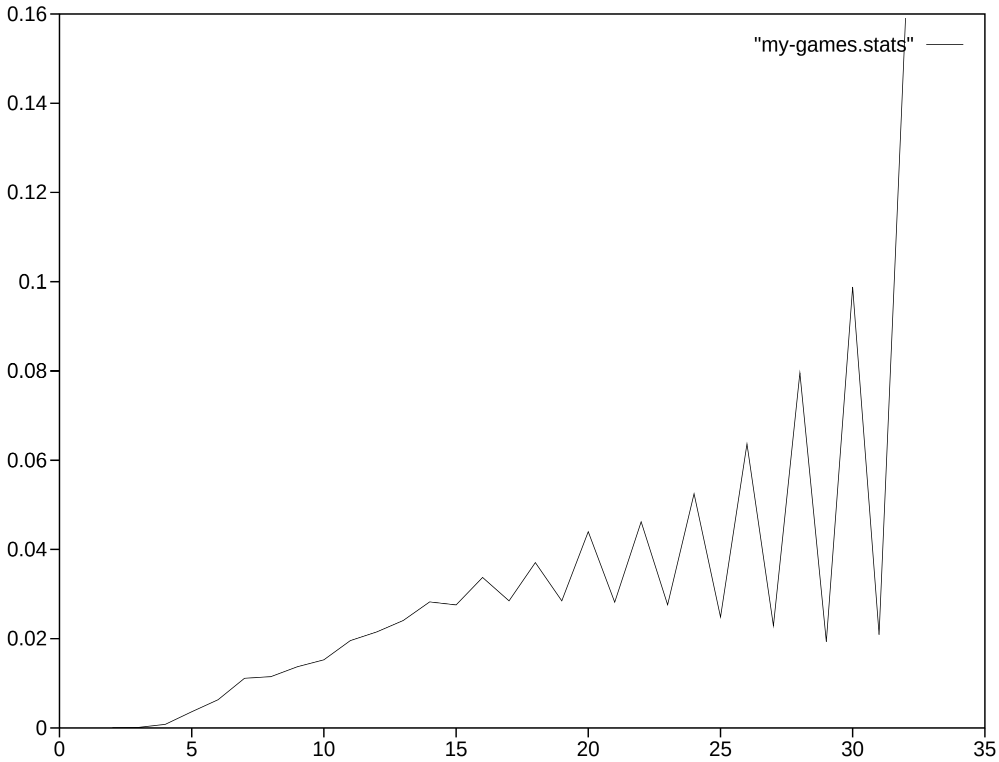

Figure 1: Normalized histogram of the number of positions having a prescribed number of pieces. There were 353138 samples. The mean is 23.1, meaning the average number of pieces during a chess game is about 23. The up-and-down near the top is due to the fact that pieces are often "exchanged". Thus, for example, the first exchange is usually a pawn for a pawn, so most games only have 1 position (out of 80) that have exactly 31 pieces on the board — .

and it turns out to be about $N = 23$. I would estimate that for a large database, my encoding scheme would take, on average,

$$66 + 23P$$

bits per position, with $P = 4 \Longrightarrow 158$ bits for chess. Although the mean $N = 23$ for chess is probably not valid for checkers, we can fudge it a bit and say that 23 is about 3/4 of 32, and 18 is about 3/4 of 24, so let's say $N = 18, P = 2 \Longrightarrow 102$ bits on average for checkers.

Finally, an absolute (but loose) upper bound on the entropy of a position can be estimated. For chess, there are a maximum of 32 squares out of 64 occupied (24 for checkers), and each of those can have 13 values (12 pieces plus empty; 5 values for checkers), giving a maximum possible number of positions of

$$\binom{64}{32} 13^{32} \approx 10^{54} = 2^{179} \text{ for chess,}$$

$$\binom{64}{24} 5^{24} \approx 10^{34} = 2^{113} \text{ for checkers.}$$

So, my maximum encoding of 193 bits for chess and 113 bits for checkers isn't too far off the mark. Similar numbers for the average of 23 pieces for chess give entropy$\approx 10^{43} = 2^{142}$, compared to my average encoding of 158 bits. This gives a relative redundancy of less than 10% (based on a loose bound, though).

There are still more optimizations that could be done. For example, 1 bit can be taken away for storing a bishiop's position, since it stays on the same colour for the entire game; a pawn never appears in the top or bottom row; certain configurations are impossible, like having two Kings simultaneously in check; *etc.* But I think I've reached the point of diminishing returns.

B. To encode a complete game, we start with an encoding of a valid initial position, using the above encoding. (I assume that we may want to encode an end-game, so we can't just assume that the initial position is the beginning of a game.) From that point, we can reduce redundancy by encoding only moves, rather than complete positions. A simple encoding of a move would be simply to specify the starting and ending position of the piece. There are some special cases to handle: "castling" in chess involves moving both the King and the Rook in one move; when a pawn gets to the opponent's side, it becomes a different piece; and in checkers, one "move" can involve more than one jump, because a piece is allowed to jump over as many opposing pieces as it can. Modulo these exceptions, the above encoding would require 12 bits per move. However, since it is *precisely* 12 bits to specify a square-to-square move, there's no room for an exception flag, and we must use an extra bit. Thus to encode all valid moves would require at least 13 bits per move in this encoding.

A more compact encoding could be based on the fact that, given a position, only certain moves are legal. We could specify an algorithm for enumerating all legal moves from a given position, and then simply encode the legal move that we want to make. For example, once the starting position of the piece we want to move is specified (6 bits), we can enumerate all the places to which it can legally move, radially outward from it's current position, starting with the "forward" direction and moving clockwise. For example, the King usually has 8 valid positions it can get to, so we'd number them from 0 to 7 starting with the direction towards the opponent. The Queen has more options: we'd number the ones "forward" until we hit the edge of the board or another piece; then the ones diagonally forward and to the right; then the ones directly to the right; *etc.* We'd skip any positions that were occupied with other pieces, thus saving some integers. Since we have a full context, both the encoding and the decoding algorithms would be able to count the available destinations and enumerate them unambiguously. This would allow us to use a variable-length bit-field to enumerate the valid moves that this piece can take, without having to specify the length of the bit-field in the encoding. If there was only one valid move the piece could take, the bit-field could even be of length zero. Thus, a move could conceivably be coded using only the 6 bits required to specify the starting square (say, a King surrounded by pieces or squares that are checked, leaving only one destination). The worst case would be the queen sitting in a very open board, where she would have about 20-25 possible moves (certainly no more than 32). So the worst-case would be $6 + 5 = 11$ bits per move—and there is plenty of space left to encode exceptions without using more bits. Checkers would be a bit more complex in this encoding, since we need to be able to specify multiple jumps, but it's certainly not impossible.

Any code that specified the pieces by name rather than position would only need to distinguish 6 names in chess, rather than 12, because we know who's turn it is. Thus only 3 bits, rather than 4, would be needed, and we'd have 2 values left for exception flags. Similarly, checkers would require only 1 bit (pawn or crown). However, the code I'm suggesting doesn't encode pieces, only positions; the piece is stored implicitly in the context[1]

Finally, we would also need to specify an extra message that specifies end-of-game. This is needed over and above an algorithmic (*i.e.,* context-based) conclusion of "check-mate" in chess, for example, because players can resign at any time.

[1] On the other hand, this may not be a very good idea, because there can, in principle, be more than 2 rooks, bishops, knights, (up to 10), and more than 1 queen (up to 9). So, encoding by piece identity may not be so hot.