Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Српски / srpski
Srpskohrvatski / српскохрватски
Edit links

Zobrist hashing

From Wikipedia, the free encyclopedia

Zobrist hashing (also referred to as **Zobrist keys** or **Zobrist signatures** [1]) is a hash function construction used in computer programs that play abstract board games, such as chess and Go, to implement transposition tables, a special kind of hash table that is indexed by a board position and used to avoid analyzing the same position more than once. Zobrist hashing is named for its inventor, Albert Lindsey Zobrist.[2] It has also been applied as a method for recognizing substitutional alloy configurations in simulations of crystalline materials.[3]

Contents [hide]

Calculation of the hash value [edit]

Zobrist hashing starts by randomly generating bitstrings for each possible element of a board game, i.e. for each combination of a piece and a position (in the game of chess, that's 12 pieces × 64 board positions, or 14 if a king that may still castle and a pawn that may capture *en passant* are treated separately). Now any board configuration can be broken up into independent piece/position components, which are mapped to the random bitstrings generated earlier. The final Zobrist hash is computed by combining those bitstrings using bitwise XOR. Example pseudocode for the game of chess:

```
constant indices
    white_pawn := 1
    white_rook := 2
    # etc.
    black_king := 12

function init_zobrist():
    # fill a table of random numbers/bitstrings
    table := a 2-d array of size 64×12
    for i from 1 to 64:  # loop over the board, represented as a linear array
        for j from 1 to 12:      # loop over the pieces
            table[i][j] = random_bitstring()

function hash(board):
    h := 0
    for i from 1 to 64:      # loop over the board positions
        if board[i] != empty:
            j := the piece at board[i], as listed in the constant indices, above
            h := h XOR table[i][j]
    return h
```

Use of the hash value [edit]

If the bitstrings are long enough, different board positions will almost certainly hash to different values; however longer bitstrings require proportionally more computer resources to manipulate. Many game engines store only the hash values in the transposition table, omitting the position information itself entirely to reduce memory usage, and assuming that hash collisions will not occur, or will not greatly influence the results of the table if they do.

Zobrist hashing is the first known instance of tabulation hashing. The result is a 3-wise independent hash family. In particular, it is strongly universal.

As an example, in chess, each of the 64 squares can at any time be empty, or contain one of the 6 game pieces, which are either black or white. That is, each square can be in one of 1 + 6 x 2 = 13 possible states at any time. Thus one needs to generate at most 13 x 64 = 832 random bitstrings. Given a position, one obtains its Zobrist hash by finding out which pieces are on which squares, and combining the relevant bitstrings together.

Updating the hash value [edit]

Rather than computing the hash for the entire board every time, as the pseudocode above does, the hash value of a board can be updated simply by XORing out the bitstring(s) for positions that have changed, and XORing in the bitstrings for the new positions. For instance, if a pawn on a chessboard square is replaced by a rook from another square, the resulting position would be produced by XORing the existing hash with the bitstrings for:

```
'pawn at this square'      (XORing out the pawn at this square)
'rook at this square'      (XORing in the rook at this square)
'rook at source square'    (XORing out the rook at the source square)
'nothing at source square' (XORing in nothing at the source square).
```

This makes Zobrist hashing very efficient for traversing a game tree.

In computer go, this technique is also used for superko detection.

Wider usage [edit]

The same method has been used to recognize substitutional alloy configurations during Monte Carlo simulations in order to prevent wasting computational effort on states that have already been calculated.[3]

See also [edit]

- Alpha-beta pruning

References [edit]

1. ^ Zobrist keys: a means of enabling position comparison.
2. ^ Albert Lindsey Zobrist, *A New Hashing Method with Application for Game Playing*, Tech. Rep. 88, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, (1969).
3. ^ *a* *b* Mason, D. R.; Hudson, T. S.; Sutton, A. P. (2005). "Fast recall of state-history in kinetic Monte Carlo simulations utilizing the Zobrist key". *Computer Physics Communications* **165**: 37. Bibcode:2005CoPhC.165...37M. doi:10.1016/j.cpc.2004.09.007.

Categories: Computer chess | Computer Go | Game artificial intelligence | Hash functions