A Group-Theoretic Zobrist Hash Function

Antti Huima

September 19, 2000

Abstract

Zobrist hash functions are hash functions that hash go positions to fixed-length bit strings. They work so that every intersection i on a go board is associated with two values B_i and W_i. To hash a position, all those values of B_i are XORed together that correspond to an intersection with black stone on it. Similarly, all W_i's are XORed together for those intersections that have a white stone. Then the results are XORed together.

We present a Zobrist hash function with the extra property that if z is the hash value of a position p, then the values z' for the positions p' that are obtained from p by exchanging the colors, by rotating the position and my mirroring can be efficiently calculated from z alone. Still, z and z' are different.

1 Theory

The *dihedral group of a square* is the eight-element group corresponding to the rotations and mirrorings of a square. Let a denote rotation by 90 degrees, say, clockwise, and b denote mirroring along, say, the vertical axis. Let 1 denote the identity translation. The dihedral group is defined by the following equations:

$$a^4 = 1, b^2 = 1, aba = b.$$

Every element of the group can be written down as

$$ba^{n_1}ba^{n_2}b \cdots ba^{n_k}b$$

where $0 \leq n_i < 4$ because $b^2 = 1$. Now all the b's can be moved to the left because $a^{n_k}b = a^{n_k-1}ba^{-1} = a^{n_k-1}ba^3$. Thus every group element can be written as $b^x a^n$, where $x \in \{0, 1\}$ and $n \in \{0, \ldots, 3\}$. Therefore there can be at most eight elements, namely

$$1, a, a^2, a^3, b, ba, ba^2, ba^3,$$

and these elements are actually distinct.

Let $\mathbb{S}_4$ denote the symmetric group of degree four. Then the dihedral group can be implemented as a subgroup of $\mathbb{S}_4$ by choosing

$$a = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \end{pmatrix}$$

and

$$b = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \end{pmatrix}.$$

Namely, letting ϵ denote the identity permutation, obviously $b^2 = \epsilon$ and $a^4 = \epsilon$, and

$$aba = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \end{pmatrix} = b.$$

Let N be a board size. Let $B = \mathbb{Z}_N \times \mathbb{Z}_N$ be the set of intersections on a board. Let $C : B \mapsto \{e, b, w\}$ be the set of board positions, i.e. functions mapping the intersections to empty, black and white, respectively. Let $\rho : C \mapsto C$ be the function implementing 90-degree rotation counterclockwise, when $(0, 0)$ denotes the lower-left corner:

$$\rho(c)(x, y) = c(18 - y, x),$$

$\mu : C \mapsto C$ the function implementing mirroring:

$$\mu(c)(x, y) = c(18 - x, y),$$

and $\kappa : C \mapsto C$ the function implementing the color exchange:

$$\kappa(c)(x, y) = E(c(x, y))$$

where $E(e) = e$, $E(w) = b$ and $E(b) = w$.

We want now to develop a specific 64-bit Zobrist hash function $Z : C \mapsto \mathbb{Z}_{64}$ that (1) is a Zobrist hash function, and (2) has three associated functions $Z_\rho, Z_\mu, Z_\kappa : \mathbb{Z}_{64} \mapsto \mathbb{Z}_{64}$ such that

$$Z_x(Z(c)) = Z(x(c))$$

for $x \in \{\rho, \mu, \kappa\}$.

We will define $Z(c)$ as

$$Z(c) = \bigoplus_{(x,y) \in \mathbb{Z}_N \times \mathbb{Z}_N} c(x, y) = \begin{array}{l} e \to 0, \\ c(x, y) = b \to Z_B(x, y), \\ Z_W(x, y), \end{array}$$

thus as a normal Zobrist hash function with the empty board mapping to zero. Bit permutations are linear. Thus, if $\pi : \mathbb{Z}_{64} \mapsto \mathbb{Z}_{64}$ is a bit permutation,

$$\pi(Z(c)) = \bigoplus_{(x,y)} c(x, y) = \begin{array}{l} e \to 0, \\ c(x, y) = b \to \pi(Z_B(x, y)), \\ \pi(Z_W(x, y)). \end{array}$$

We are going to define Z_ρ, Z_μ and Z_κ as bit permutations. Let $z = z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8$ be a 64-bit hash value written down as eight consecutive bytes. We then define

$$\begin{aligned} Z_\rho(z) =& z_2, z_3, z_4, z_1, z_6, z_7, z_8, z_5 \\ Z_\mu(z) =& z_4, z_3, z_2, z_1, z_8, z_7, z_6, z_5 \\ Z_\kappa(z) =& z_5, z_6, z_7, z_8, z_1, z_2, z_3, z_4. \end{aligned}$$

We pose the following restrictions upon the choice of Z_B and Z_W:

$$\begin{aligned} Z_B(18-x, y) &= Z_\mu(Z_B(x, y)) \\ Z_B(18-y, x) &= Z_\rho(Z_B(x, y)) \\ Z_W(x, y) &= Z_\kappa(Z_B(x, y)). \end{aligned}$$

That these restrictions can be fulfilled stems from the fact that Z_μ and Z_ρ form the dihedral group for the first and last four bytes, and Z_κ just corresponds to exchanging the halves.

THEOREM. $Z_x(Z(c)) = Z(x(c))$ for $x \in \{\rho, \mu, \kappa\}$.

PROOF. Immediate because Z_ρ, Z_μ and Z_κ are bit-wise permutations. It is therefore enough to show that the claim holds for positions with exactly one white or black stone. But for those positions the claim holds directly because of the previous restrictions.

2 Implementation

It remains to ponder how to actually implement this hash function.

For those intersections that do not lie on any of the four symmetry axes it is easy to find the values of Z_B: choose randomly $Z_B(x, y)$ for some points, and calculate the values of the symmetrical images of those points by using the formulae above.

For the remaining cases, the tengen must have $Z_B = z_1 z_1 z_1 z_1 z_2 z_2 z_2 z_2$ for some z_1, z_2.

On the diagonal axes the hash value must be $Z_B = z_1 z_2 z_3 z_2 z_4 z_5 z_6 z_5$ for a stone in the lower-left quadrant for some z_i. This is because in addition to the equations of the dihedral group, for these elements rotation and mirroring become either equivalent or inverses depending on the position of the stone.

On the axes through tengen, the hash value must be $Z_B = z_1 z_1 z_2 z_2 z_3 z_3 z_4 z_4$ for a stone on the horizontal axis. This is because in addition to the equation of the dihedral group, for these elements mirroring is either identity operation (on the vertical axis) or equivalent to two rotations (on the horizontal axis).

3 Notes

The implementation of Sec. 2 has the undesirable property that for example the position where there are four black stones on the four central side hoshi points hash the hash value zero. This is an anomaly.

This anomaly can be removed if instead of using the idempotent XOR operator the operator for combining hash values is chosen differently. We could use for example byte-wise addition modulo 256.