Análise da Simetria no Jogo da Velha

MAC5788 Planejamento em Inteligência Artificial

Renato Lui Geh, NUSP: 8536030

1 Simetria

Diz-se que duas configurações do jogo da velha são simétricas quando os dois tabuleiros são idênticos a menos de rotação e reflexão. Podemos considerar o tabuleiro como uma matriz 3×3, onde a marcação do primeiro jogador é representado por 1, o do segundo por 2 e uma posição vazia por 0.

1	0	0
0	2	0
0	0	1

X		
	O	
		X

0	1	2
3	4	5
6	7	8

Tabela 1: Configuração do tabuleiro em matriz (esquerda), no seu formato tradicional (meio), e os índices de cada posição no tabuleiro (direita).

Desta forma, toda configuração do jogo é única e pode ser representada por um *hash* único

$$H(T) = \sum_{i=0}^{8} T_i \cdot 3^i, \tag{1.1}$$

onde T_i é a i-ésima posição do tabuleiro. É fácil de ver que $H(T)$ é apenas uma representação da matriz em ternário.

Nosso objetivo é achar todas as configurações de tabuleiro simétricas e atribuirmos o mesmo valor de hash a elas. Para isso, precisamos achar todos os possíveis jogos válidos, e para cada um desses, acharmos seus equivalentes simétricos. Esta tarefa é fácil: basta acharmos todas as possíveis combinações de reflexão e rotação. Vamos usar a notação $H(T)$ para representar o hash único do tabuleiro T, dado pela Equação 1.1. Além disso, considere um dicionário global A que guarda todos os hashes vistos até agora.

$\hat{H}(T)$: acha todas configurações simétricas de um tabuleiro T

Entrada configuração T do tabuleiro
Saída hash único de todos os tabuleiros simétricos a T

$m \leftarrow \infty$
$h \leftarrow H(T)$
se $h \in A$ **então**
    retorna $A[h]$
Seja U um vetor de hashes ainda não vistos
Seja M a matriz 3×3 dada por T
para todo $i \leftarrow 0..3$ **faça**
    Rotaciona M em 90 graus
    $h \leftarrow H(M)$
    se $h \notin A$ **então**
        Adiciona h em U
    $m \leftarrow \min\{m, h\}$
    Reflete M
    $h \leftarrow H(M)$
    se $h \notin A$ **então**
        Adiciona h em U
    $m \leftarrow \min\{m, h\}$
    Reflete M
para todo todo $u \in U$ **faça**
    $A[u] \leftarrow m$
retorna m

Por meio de $\hat{H}(T)$, calculamos todas as configurações simétricas do tabuleiro T, guardamos o hash de menor valor, garantindo que seja único a menos de rotações e reflexão, e em seguida usamos programação dinâmica para evitar recomputar os hashes novamente. Deste modo, quando o agente avaliar um estado correspondente ao tabuleiro T, ele irá considerar todas as simetrias ao mesmo tempo.

2 Código

O código original pode ser encontrado em [2], enquanto que o código alterado para simetria em [1]. A função em Python equivalente a $\hat{H}$ é o método `State.hash`. Para a função H de hash único, usa-se a função `State.unique_hash`. O dicionário A com todos os hashes apontando para seus simétricos é a variável global `all_hashes`.

Além da função de hash e funções auxiliares, nada mais foi alterado.

3 Experimentos

O jogo da velha possue uma estratégia vencedora para o primeiro jogador, vencendo ou empatando se jogar otimamente.

X				X				X		X
			→				→			
						X				X

Tabela 2: Estratégia vencedora do primeiro jogador. Todas estratégias vencedoras simétricas podem ser reduzidas à estratégia acima por meio de rotações ou reflexões.

Na versão sem exploração da simetria do código original, o segundo jogador ou empatava ou perdia quando confrontado com a estratégia vencedora do primeiro jogador. Em contrapartida, o agente treinado com exploração da simetria sempre empatava contra um primeiro jogador ótimo.

Em questão de tempo de convergência, enquanto que a versão sem simetria convergiu na época 65000, a com simetria conseguiu a mesma convergência na época 6000, o que indica que o agente exaustou todas as jogadas mais rapidamente, o que era esperado.

Referências

[1] *Symmetric Tic Tac Toe*. 2019. URL: https://github.com/RenatoGeh/mac5788/blob/master/ep1/ttt.py.

[2] Shangtong Zhang et al. *Tic Tac Toe*. 2018. URL: https://github.com/ShangtongZhang/reinforcement-learning-an-introduction/blob/master/chapter01/tic_tac_toe.py.