Algorithm 2 Heuristic MC–RAVE

```
procedure MC–RAVE(s_0)
    while time available do
        SIMULATE(board, s_0)
    end while
    board.SetPosition(s_0)
    return SELECTMOVE(board, s_0, 0)
end procedure

procedure SIMULATE(board, s_0)
    board.SetPosition(s_0)
    [s_0, a_0, ..., s_T, a_T] = SIMTREE(board)
    [a_{T+1}, ..., a_D], z = SIMDEFAULT(board, T)
    BACKUP([s_0, ..., s_T], [a_0, ..., a_D], z)
end procedure

procedure SIMDEFAULT(board, T)
    t = T + 1
    while not board.GameOver() do
        a_t = DEFAULTPOLICY(board)
        board.Play(a_t)
        t = t + 1
    end while
    z = board.BlackWins()
    return [a_{T+1}, ..., a_{t-1}], z
end procedure

procedure SIMTREE(board)
    t = 0
    while not board.GameOver() do
        s_t = board.GetPosition()
        if s_t ∉ tree then
            NEWNODE(s_t)
            a_t = DEFAULTPOLICY(board)
            return [s_0, a_0, ..., s_t, a_t]
        end if
        a_t = SELECTMOVE(board, s_t)
        board.Play(a_t)
        t = t + 1
    end while
    return [s_0, a_0, ..., s_{t-1}, a_{t-1}]
end procedure

procedure SELECTMOVE(board, s)
    legal = board.Legal()
    if board.BlackToPlay() then
        return argmax_{a∈legal} EVAL(s, a)
    else
        return argmin_{a∈legal} EVAL(s, a)
    end if
end procedure

procedure EVAL(s, a)
    b = pretuned constant bias value
    β = Ñ(s,a) / (N(s,a)+Ñ(s,a)+4N(s,a)Ñ(s,a)b^2)
    return (1 − β) Q(s, a) + β Q̃(s, a)
end procedure

procedure BACKUP([s_0, ..., s_T], [a_0, ..., a_D], z)
    for t = 0 to T do
        N(s_t, a_t) += 1
        Q(s_t, a_t) += (z − Q(s_t, a_t)) / N(s_t, a_t)
        for u = t to D step 2 do
            if a_u ∉ [a_t, a_{t+2}, ..., a_{u-2}] then
                Ñ(s_t, a_u) += 1
                Q̃(s_t, a_u) += (z − Q̃(s_t, a_t)) / Ñ(s_t, a_t)
            end if
        end for
    end for
end procedure

procedure NEWNODE(board, s)
    tree.Insert(s)
    for all a ∈ board.Legal() do
        N(s, a), Q(s, a), Ñ(s, a), Q̃(s, a) = HEURISTIC(board, a)
    end for
end procedure
```

The value function learnt from local shape features, Q_{rlgo}, outperformed all the other heuristics and increased the winning rate of *MoGo* from 60% to 69%. Maximum performance was achieved using an equivalent experience of $M = 50$, which indicates that Q_{rlgo} is worth about as much as 50 simulations using all-moves-as-first. It seems likely that these results could be further improved by varying the heuristic confidence according to the particular position, based on the variance of the heuristic evaluation function.

5.2. Exploration and exploitation

The performance of Monte-Carlo tree search is greatly improved by carefully balancing exploration with exploitation. The UCT algorithm significantly outperforms a greedy tree policy in computer Go [19]. Surprisingly, this result does not appear to extend to the heuristic UCT–RAVE algorithm: the optimal exploration rate in our experiments was zero, i.e. greedy MC–RAVE with no exploration in the tree policy.

We believe that the explanation lies in the nature of the RAVE algorithm. Even if an action a is not selected immediately from position s, it will often be played at some later point in the simulation. This greatly reduces the need for explicit exploration, because the values for all actions are continually updated, regardless of the initial move selection.

However, we were only able to run thorough tests with tens of thousands of simulations per move. It is possible that exploration again becomes important when MC–RAVE is scaled up to millions of simulations per move. At this point a substantial number of nodes will be dominated by MC values rather than RAVE values, so that exploration at these nodes should be beneficial.