Monte-Carlo Tree Search and Rapid Action Value Estimation in Computer Go

Artificial Intelligence, volume 175, pp 1856-1875, 2011.

Authors: Sylvain Gelly, David Silver

Presenter: Andy Thai (andy.thai@uci.edu)

Outline

- **Introduction**
- Background
  - Computer Go
  - Two-player games
  - Monte-Carlo Simulation
- MCTS
  - Monte-Carlo Tree Search
  - UCT
  - All-moves-as-first
- RAVE
  - MC-RAVE
  - UCT-RAVE
- Heuristic MC-RAVE
  - Heuristic UCT-RAVE
- Question

Introduction

- Monte-Carlo tree search (MCTS) algorithms have revolutionized search algorithms, particularly in problems **on solving computer Go programs.**
  - Replacing traditional search methods
- Simulates thousands of random games from current position with self-play.
  - Positions added into a search tree
  - Nodes are average outcome of all simulated games that reach the position
- Search tree goes along path with **highest potential values**
- **MCTS evaluation function**
  - Depends only on observed simulation outcomes, not handcrafted
  - Continuously improves from additional simulations
- **Converges on optimal search tree**
  - Given infinite memory and computation
  - Develops best first manner, expands promising regions more deeply

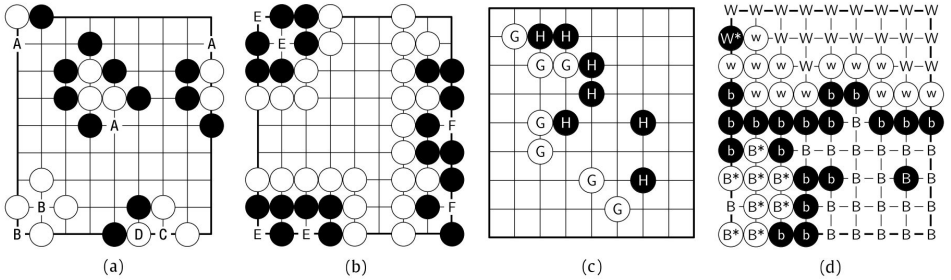
Introduction

- Two major **enhancements** to Monte-Carlo tree search
  - **Rapid Action Value Estimation (RAVE) for MCTS**
    - Shares value of actions across each subtree of search tree
    - Fast, rough estimate of action value
      - Compared to normal Monte-Carlo values which is slower but more accurate
    - **MC-RAVE** combines these two value estimates to minimize mean squared error
  - **Heuristic Monte-Carlo tree search**
    - Uses heuristic function initializes values of new positions in search tree
  - Enhancements implemented in **MoGo program** (9 x 9 Go)
    - First program to achieve master level (dan) and to defeat a professional human player

Outline

- Introduction
- **Background**
  - Computer Go
  - Two-player games
  - Monte-Carlo Simulation
- MCTS
  - Monte-Carlo Tree Search
  - UCT
  - All-moves-as-first
- RAVE
  - MC-RAVE
  - UCT-RAVE
- Heuristic MC-RAVE
  - Heuristic UCT-RAVE
- Question

Computer Go



- Chess was previously used as a metric.
  - Alpha-beta search algorithms already outperform human players.
- Why use **computer go**?
  - Challenging task with many state spaces
  - More than 10^{170} states and 361 legal moves
  - Long-term effect of a move may not come up until 50-100 turns later
- **Rules**
  - 19x19 grid, but 13x13 and 9 x 9 are popular alternatives.
  - Players take turns putting a stone on an intersection in grid.
    - You can’t move stones, only capture them.
  - A group of stones can be captured by completely blocking off adjacent intersections.
  - Game ends when both players pass.
  - Winner is the one with the most stones that cannot be captured.

Two-player games

- Game is a **two-player, perfect-information, zero-sum game**
  - Black (first) and white (second) alternate turns, selecting an action $\mathbf{a_t \in A(s_t)}$
    - $\mathbf{s_t \in S}$**:** current state, where **S** is a finite state space
    - **A(s):** finite set of legal actions in state **s**
    - Game finishes when you reach a terminal state with outcome **z**
  - Black wants to **maximize z**, while white wants to **minimize z**.
- **A two player policy:** $\pi(s, a) = Pr(a|s)$
  - Stochastic action selection strategy; determines the probability of selecting actions in any given state.
  - **Policy for black:** $\pi_B(s, a)$; **policy for white:** $\pi_W(s, a)$; $\boldsymbol{\pi} = <\pi_B, \pi_W>$
- **Value function:** $Q^{\pi}(s, a)$
  - Expected outcome after playing action **a** in state **s**, then following policy **π** for both players until termination
- Starts from a root state s_0 and sequentially samples states and actions without backtracking.
  - Given a state s and an action a, at every step t, the two player policy $\pi(s, a)$ selects an action a_t.
  - The next state s_{t+1} is generated based on the game's rules.
  - The outcome of the game z updates the values of states or actions encountered during the simulation.

Monte-Carlo Simulation

- From a **given root state s_0**, **evaluate candidate actions** using a simulation-based search
  - Simulate complete games from s_0 until completion
- Value of each action **a** from **s_0** is estimated by **mean outcome of all simulations starting with candidate action a**
- **Monte-Carlo value (MC value): Q(s, a)** is the mean outcome of all simulations where action a was selected in state s.
  - N(s) complete games are simulated with policy π from state s

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{N(s)} \mathbb{I}_i(s,a) z_i$$

Monte-Carlo Simulation

- **Monte-Carlo value (MC value):** Q(s, a) is the mean outcome of all simulations where action a was selected in state s.

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{N(s)} \mathbb{I}_i(s,a) z_i$$

  - z_i **:** outcome of the ith simulation
  - $\mathbb{I}_i$ **:** Returns 1 if action a was selected in state s at the ith simulation, otherwise returns 0
  - **N(s, a):** total of number of simulations in which action a was selected in state s.

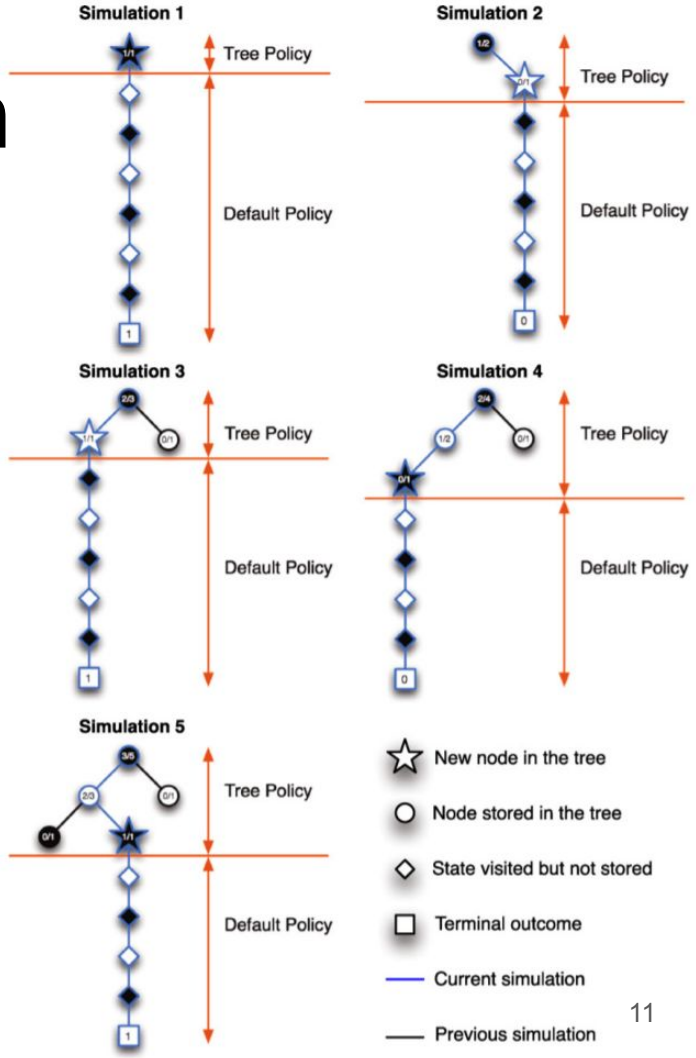
$$N(s,a) = \sum_{i=1}^{N(s)} \mathbb{I}_i(s,a)$$

Outline

- Introduction
- Background
  - Computer Go
  - Two-player games
  - Monte-Carlo Simulation
- **MCTS**
  - Monte-Carlo Tree Search
  - UCT
  - All-moves-as-first
- RAVE
  - MC-RAVE
  - UCT-RAVE
- Heuristic MC-RAVE
  - Heuristic UCT-RAVE
- Question

Monte-Carlo Tree Search

- Uses **MC simulation to evaluate search tree nodes**
- Values in search tree used to **select best action during later simulations**
- MCTS is sequentially **best first**
  - Selects best child at each step of the simulation
  - Refocus attention to highest value regions of state space
- As search tree grows:
  - Values of nodes approximate the **minimax value**
  - Simulation policy approximates **minimax policy**
- A search tree T has one **node n(s)** for every **state s** seen during simulations.
  - Each node n(s) contains:
    - Total count for the state: **N(s)**
    - Action value: **Q(s, a)**
    - A count for each action a ∈ A: **N(s, a)**



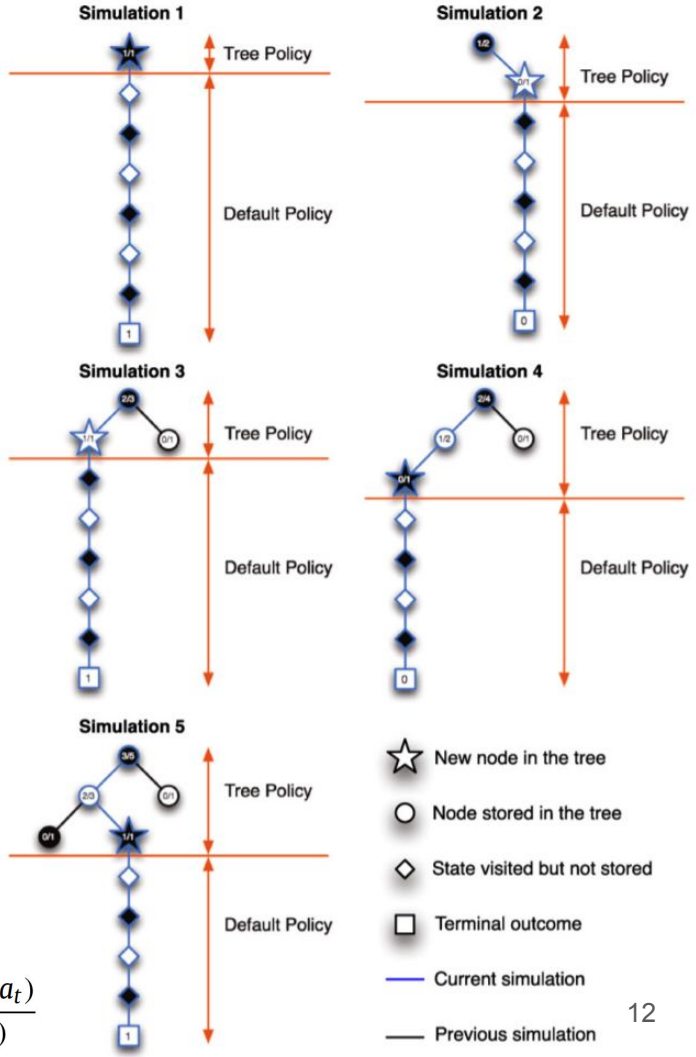
Monte Carlo Tree Search

- Simulations start from root s_0 and go into 2 stages
- When state s_t is represented in the search tree, a tree policy selects the actions, otherwise a default policy is used.
- Simplest version of algorithm: **greedy MCTS**
  - **First stage:** selects the greedy action with highest value during first stage: $\text{argmax}_a\ Q(s_t, a)$
  - **Second stage:** selects actions uniformly at random
- Every state and action is evaluated by the **mean outcome**
  - After every simulation $s_0, a_0, s_1, a_1, \ldots, s_T$ with outcome z, every node updates its count and action value to the new MC value.

$$N(s_t) \leftarrow N(s_t) + 1,$$

$$N(s_t, a_t) \leftarrow N(s_t, a_t) + 1,$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \frac{z - Q(s_t, a_t)}{N(s_t, a_t)}$$



UCT

- Greedy algorithms are **inefficient**
  - They stop exploring after poor outcomes, even with lots of uncertainty
- **Optimism in the face of uncertainty**
  - Select the actions with the greatest potential value
  - Uncertain actions get a value **bonus based on amount of uncertainty**
- **UCT for MCTS**
  - Treats each state in tree as a multi-armed bandit problem
  - Tree policy selects actions using UCB1 (maximizes upper confidence bound)
  - Converges to minimax value function

$$Q^{\oplus}(s,a) = Q(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}},$$

$$a^* = \underset{a}{\operatorname{argmax}}\, Q^{\oplus}(s,a)$$

- **$Q^{\oplus}(s, a)$:** UCB1
- **c:** scalar exploration constant
- **a*:** action selected from policy

All-moves-as-first heuristic

- Value of playing action **a** immediately is **average outcome of all simulations** where **a** is played **at any time.**
    - Have a general value for each move, **regardless** of when it is played
  - Biased estimate of the true action value function
  - Assume value of a move is usually unaffected by moves elsewhere on board

$$\tilde{Q}(s,a) = \frac{1}{\tilde{N}(s,a)} \sum_{i=1}^{N(s)} \tilde{\mathbb{I}}_i(s,a) z_i$$

This function is the **mean outcome** of all simulations in which action a is selected at any turn after s is encountered.

- $\tilde{\mathbb{I}}_i$**(s, a):** returns 1 if state **s** was encountered at any step **t** of the **i**th simulation and action **a** was selected at any step **after t**, 0 otherwise
- $\tilde{N}$**(s, a):** total number of simulations to estimate AMAF value
- z_i**:** outcome of ith simulation

Outline

- Introduction
- Background
  - Computer Go
  - Two-player games
  - Monte-Carlo Simulation
- MCTS
  - Monte-Carlo Tree Search
  - UCT
  - All-moves-as-first
- **RAVE**
  - MC-RAVE
  - UCT-RAVE
- Heuristic MC-RAVE
  - Heuristic UCT-RAVE
- Question

RAVE

- MCTS cannot generalize between **related positions or related moves**
  - MCTS **separately estimates** value of each state and action in search tree
- **Rapid action value estimation** combines MCTS with AMAF heuristic.
- Instead of computing MC value of each node in search tree, **RAVE computes the AMAF value**.
- Every state in the search tree is the root of a subtree.
  - If a simulation visits state s_T at step t, all subsequent states in that current simulation are in the subtree for s_T.
- Idea: use RAVE to **generalize over subtrees**
  - Assume that value of an action a in state s will b similar from all states within the subtree for s.
  - The value of an action a is estimated from all simulations starting with s, regardless of when a occurs.
  - Shares knowledge between related nodes in search tree
  - Rapid, biased estimate of action values
  - Works very fast, can find best move after a few simulations

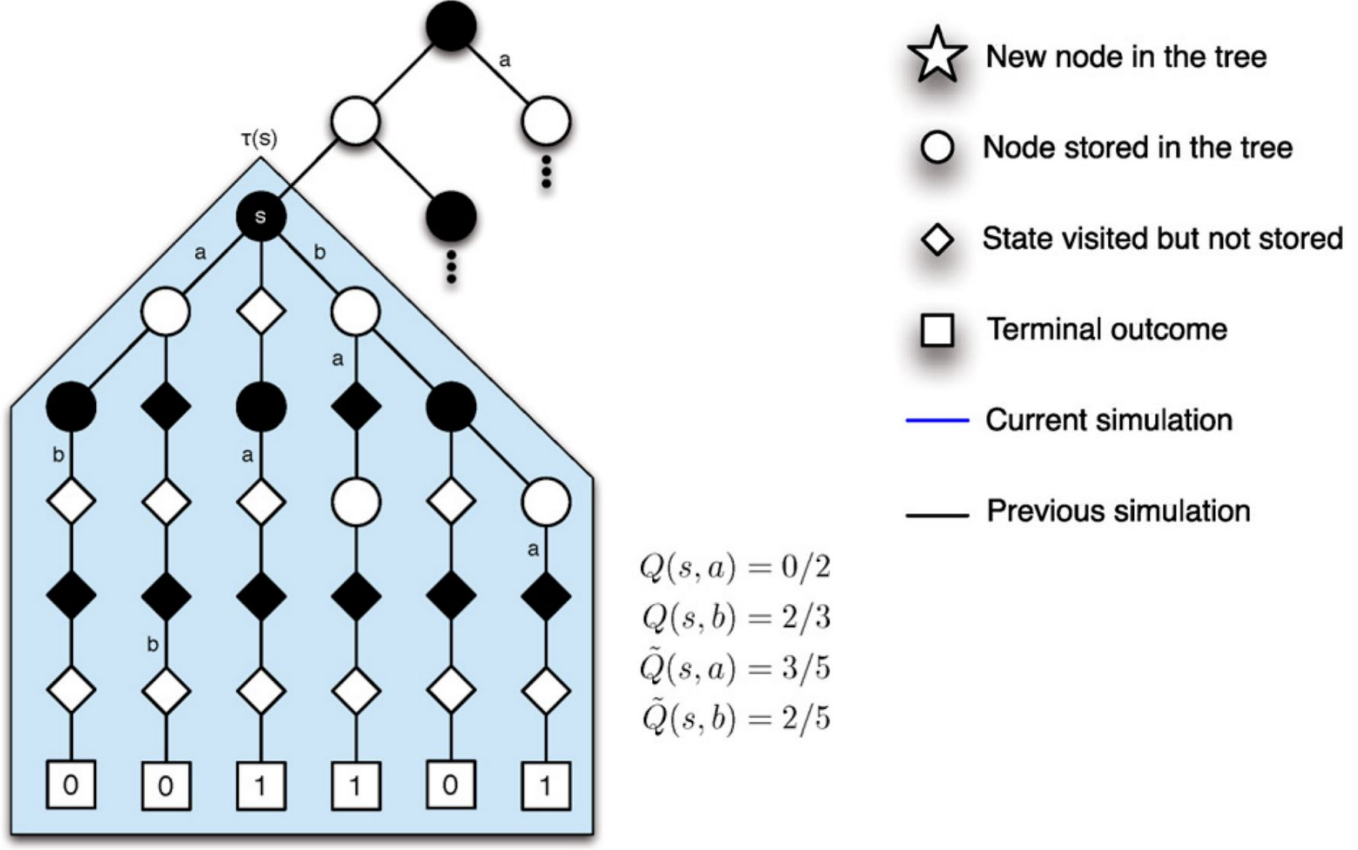


Fig. 4. An example of using the RAVE algorithm to estimate the value of Black moves a and b from state s. Six simulations have been executed from state s, with outcomes shown in the bottom squares. Playing move a immediately led to two losses, and so Monte-Carlo estimation favours move b. However, playing move a at *any* subsequent time led to three wins out of five, and so the RAVE algorithm favours move a. Note that the simulation starting with move a from the root node does not belong to the subtree $\tau(s)$ and does not contribute to the AMAF estimate $\tilde{Q}(s,a)$.

MC-RAVE

- **Downsides of RAVE:** learns quickly, often wrong
  - Assumes a particular move has the same value across an entire subtree
    - Often false! Nearby changes can change the value of a move.
- MC-RAVE **combines RAVE with MCTS** for better accuracy and convergence.
  - It uses a **weighted sum** for an action value a in state s.

$$Q_{\star}(s,a) = \big(1 - \beta(s,a)\big)\, Q(s,a) + \beta(s,a)\, \tilde{Q}(s,a)$$

β(s, a): weighting parameter for state s and action a

- **Few** simulations done: more weight on **AMAF** value, β(s, a) is closer to 1
- **More** simulations done: more weight on **MC** value, β(s, a) is closer to 0

MC-RAVE

- hand-selected schedule for **β(s, a):**

$$\beta(s,a) = \sqrt{\frac{k}{3N(s)+k}}$$

k: the equivalence parameter, or number of simulations at which the MC value and AMAF value are given equal weight, s.t. β(s, a) = ½

- Minimum MSE schedule for **β(s, a): minimizes MSE for MC-RAVE's $Q_{\star}$(s, a)**

$$\beta = \frac{\tilde{n}}{n+\tilde{n}+4n\tilde{n}\tilde{b}^2}$$

n: total number of simulations to compute MC value

ñ: total number of simulations to compute AMAF value

b: unknown constant, the RAVE bias, solve empirically or with ML

MC-RAVE

- Each simulation is divided into two stages:
  - **Stage 1:** actions are greedily selected for states within the search tree to maximize combined MC and AMAF value
  - **Stage 2:** actions are selected by a default policy for states beyond the search tree
- After every simulation s_0, a_0, s_1, a_1, …, s_T, **update both** MC and AMAF values.
- For every state s_T in the search tree, **update** the values and counts for node $n(s_T)$

MC-RAVE Results

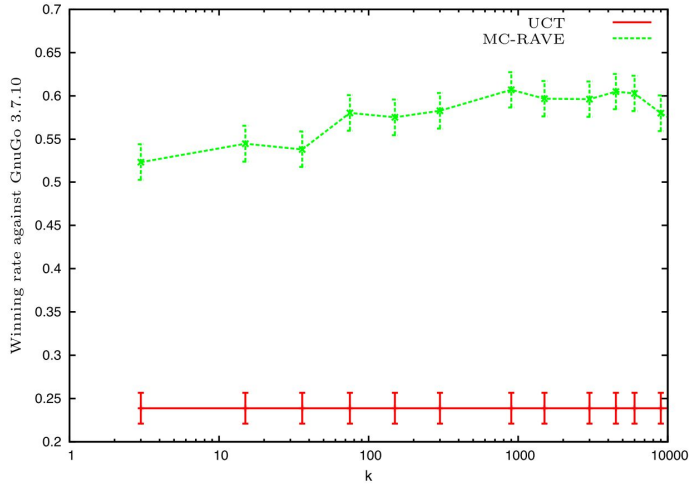


Fig. 5. Winning rate of MC–RAVE with 3000 simulations per move against GnuGo 3.7.10 (level 10) in 9×9 Go, for different settings of the equivalence parameter k. The bars indicate the standard error. Each point of the plot is an average over 2300 complete games.

Table 1

Winning rate of *MoGo* against GnuGo 3.7.10 (level 10) when the number of simulations per move is increased. *MoGo* competed on CGOS, using heuristic MC–RAVE and the hand-selected schedule, in February 2007. The versions using 10 minutes per game modify the simulations per move according to the available time, from 300,000 games in the opening to 20,000 in the endgame. The asterisked version competed on CGOS in April 2007 using the minimum MSE schedule and additional parameter tuning.

Schedule	Computation	Wins vs. GnuGo	CGOS rating
Hand-selected	3000 sims per move	69%	1960
Hand-selected	10,000 sims per move	82%	2110
Hand-selected	10 minutes per game	92%	2320
Minimum MSE	10 minutes per game	97%	2480*

UCT-RAVE

- Extension of MCTS
- Uses **optimism-in-the-face-of-uncertainty** principle
  - Adds an **exploration bonus** based on upper confidence bound for current value
- Incorporated into MC-RAVE:

$$Q_{\star}^{\oplus}(s,a) = Q_{\star}(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}}$$

Outline

- Introduction
- Background
  - Computer Go
  - Two-player games
  - Monte-Carlo Simulation
- MCTS
  - Monte-Carlo Tree Search
  - UCT
  - All-moves-as-first
- RAVE
  - MC-RAVE
  - UCT-RAVE
- **Heuristic MC-RAVE**
  - Heuristic UCT-RAVE
- Question

Heuristic MCTS

- Suppose there is a state s and an action s with a **high uncertainty value.**
  - So, incorporate prior knowledge using:
    - **Heuristic evaluation function:** H(s, a)
    - **Heuristic confidence function:** C(s, a)
- When first adding a node, **initialize according to heuristic function**.
  - Q(s, a) = H(s, a)
  - N(s, a) = C(s, a)
    - Confidence is measured by **equivalent experience**:
      - Number of simulations required to get an MC value of similar accuracy to heuristic value.
- After initialization, **update the value and count as usual using MC simulation.**

Heuristic MC-RAVE

$$Q(s,a) \leftarrow H(s,a),$$
$$N(s,a) \leftarrow C(s,a),$$
$$\tilde{Q}(s,a) \leftarrow H(s,a),$$
$$\tilde{N}(s,a) \leftarrow \tilde{C}(s,a),$$
$$N(s) \leftarrow \sum_{a \in \mathcal{A}} N(s,a).$$

- Heuristic MC-RAVE: **combining** heuristic MCTS with MC-RAVE.
- When a node n(s) is added to the search tree, initialize MC and AMAF to heuristic evaluation. Initialize both counts to the heuristic confidence functions.
- Types of heuristic functions:
  - **Even-game heuristic:** $Q_{even}(s, a) = 0.5$
    - Assumes most positions between strong players are close in value.
  - **Grandfather heuristic:** $Q_{grand}(s_t, a) = Q(s_{t-2}, a)$
    - Assumes the value of a played move is similar to the value of that same move the last time it was made.
    - Sets value of each node in the tree to the value of its grandfather
  - **Handcrafted heuristic:** $Q_{mogo}(s, a)$
    - Based off of handcrafted patterns used in MoGo's default policy. Moves are matched to patterns labeled good, bad, or unlabeled.
  - **Local shape heuristic:** $Q_{rlgo}(s, a)$
    - Computed from linear combination of local shape features from the RLGO 1.0 model.

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
```

Heuristic Results

- Compared the four heuristic evaluations in 9 x 9 Go using heuristic MC-RAVE.
- Heuristic confidence is set to C̃(s, a) = M, where the equivalent experience is some constant M.

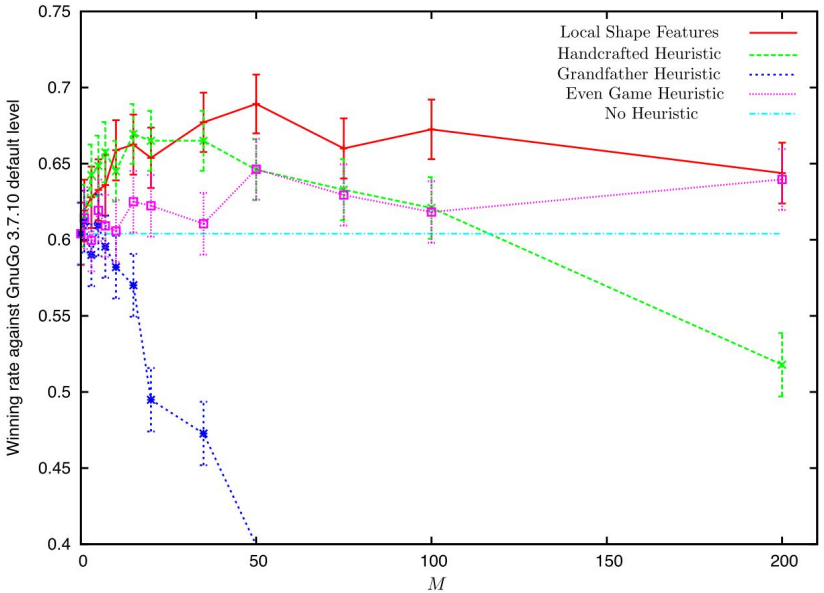


Fig. 6. Winning rate of *MoGo*, using the heuristic MC–RAVE algorithm, with 3000 simulations per move against GnuGo 3.7.10 (level 10) in 9 × 9 Go. Four different forms of heuristic function were used (see text). The bars indicate the standard error. Each point of the plot is an average over 2300 complete games.

Heuristic UCT-RAVE

- Authors attempted **extending UCT** to the **heuristic MC-RAVE function**.
- Did not produce satisfactory results:
  - **Optimal exploration rate was zero** for tree policy.
- Explanation
  - In the RAVE algorithm, if an action isn't selected, it'll still likely be played later.
  - No need to explore, values for all actions are continuously updated.
- Caveat
  - Tests were done on thousands of simulations per move.
  - Exploration may be more important when scaled to millions of simulations per move.
    - Large number of nodes will be dominated by MC values instead of RAVE
      - Exploration may become more useful

Question

In this paper, the authors reviewed MCTS and introduced two extensions to Monte-Carlo tree search: RAVE/MC-RAVE and heuristic MCTS.

What problems or cases do these algorithms address that a normal MCTS tree search may have difficulty with in a Computer Go game?

What are some flaws of RAVE and heuristic MCTS?