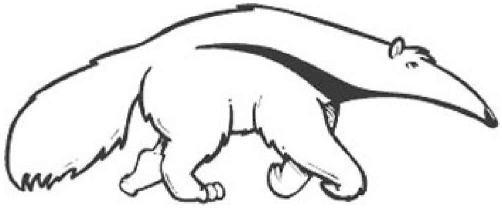
MCTS (Checkers) AI Discussion

Disclaimer : We discuss a few ideas, there are many others… We will not go into detailed technical solutions (actual code) …

Kalev Kask



Some pictures from publications referenced

BREN:ICS
INFORMATION AND COMPUTER SCIENCES
UNIVERSITY of CALIFORNIA IRVINE

Resources

- “A Survey of Monte Carlo Tree Search Methods”
  - http://ccg.doc.gold.ac.uk/ccg_old/papers/browne_tciaig12_1.pdf
- many others
- .....

Basics

- Board size
  - M = number of rows
  - N = number of columns
- StudentAI class
  - This is where your code will go …
- get_move()/GetMove()
  - called by the system, when it is your turn to move …
- Time limit
  - E.g. 8 min per game per player …

Persistent Board

- There is a board (member) object (of type Board) in StudentAI class, for your convenience
- You need to update it yourself
  - e.g. when get_move() is called, opponent's move is passed in as input, you need to call make_move() to keep the board up to date …

Keeping track of time

- Time is of the essence
  - There will be a timeout
- You have to keep track of time
- Avg # of moves per game ? 25
  - 8min = 480sec -> about 20 sec per move
  - MCTS iterations slower early, faster later

```
double total_time_elapsed = 0.0
...
Move get_move(...)
{
    double remaining_time = ...
    if (remaining_time < some_small_number_eg_3)
      make_random_move
    else {
            tS = time_stamp_now

            // do your normal stuff
            ..............................

            tE = time_stamp_now
            dt = tE - tS // time used for this get_move() call
            total_time_elapsed += dt
            }
}
```

MCTS

- Basic question :
  - How long does each iteration take
    - this determines how many iterations you can do
  - What to you get out of each iteration
- If # iterations -> ∞ then learned win-rate -> true win-rate
  - The more iterations you can do, the better quality play
  - Are you doing enough iterations?

MCTS tradeoff

- Make each iteration faster?

- Get more out of each iteration?
  - Only “good quality” iterations really contribute
  - Naïve MCTS will execute many “poor quality” iterations -> slow convergence

- Tradeoff : is the extra effort you put into each iterations (e.g. heuristic) paying off

MCTS how many iterations?

- How many iterations is enough?
  - Depends of what you get out of each iteration
  - For checkers
    - 100 probably not enough
    - 1000 probably enough

MCTS how to make iterations faster ?

- When doing selection/simulation you need the board along the way
  - For blind/uninformed simulation, can make a copy of "main" board in the beginning, and then just call getAllMoves()/makeMove()
  - If you heuristic, need to do 1-move lookahead at each state (simulation)
    - Do you do (deep)copy of the board, and then just makeMove() off of the (deep)copy?
    - Do you skip (deep)copy and do makeMove()/Undo()?

MCTS how to make iterations faster ?

- When entering getMove()
  - getAllPossibleMoves() returns just 1 single move, don’t need to be MCTS
  - Just immediately return that 1 single move

MCTS how to make iterations faster ?

- When simulation is taking too long?
  - You can cut if off, but the terminal value is unknown
  - Can apply heuristic
  - But heuristic needs to be sufficiently accurate
  - May need 3 states
    - Win
    - Loss
    - TooHardToTell

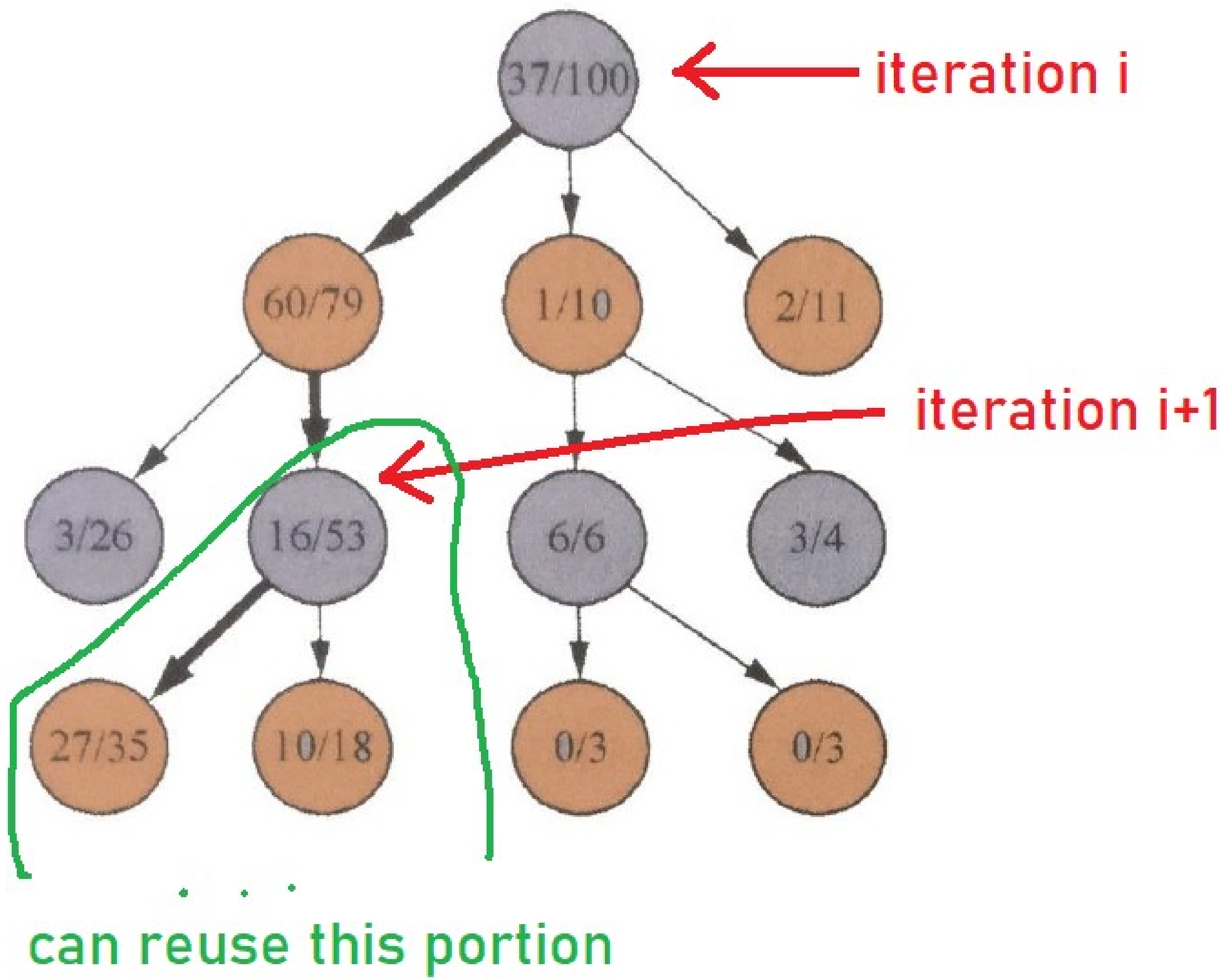
Heuristic

- Heuristic needs to be
  - Accurate
  - Discriminative
- E.g. H(s) = #B(s) - #W(s) is not discriminative
  - Vast majority of time, either no capture (H is same for all children) or 1-piece capture (H is same for all children)
  - Most of the time material value does not change
  - So this H is uninformative and useless for guiding MCTS

Heuristic : testing

- Experimentally evaluate how discriminative/accurate it is
- Implement simple MiniMax (a few moves lookahead) and run your H against Ramdom/Poor/Average. What is your winrate?

MCTS reuse subtree



MCTS modified UCT

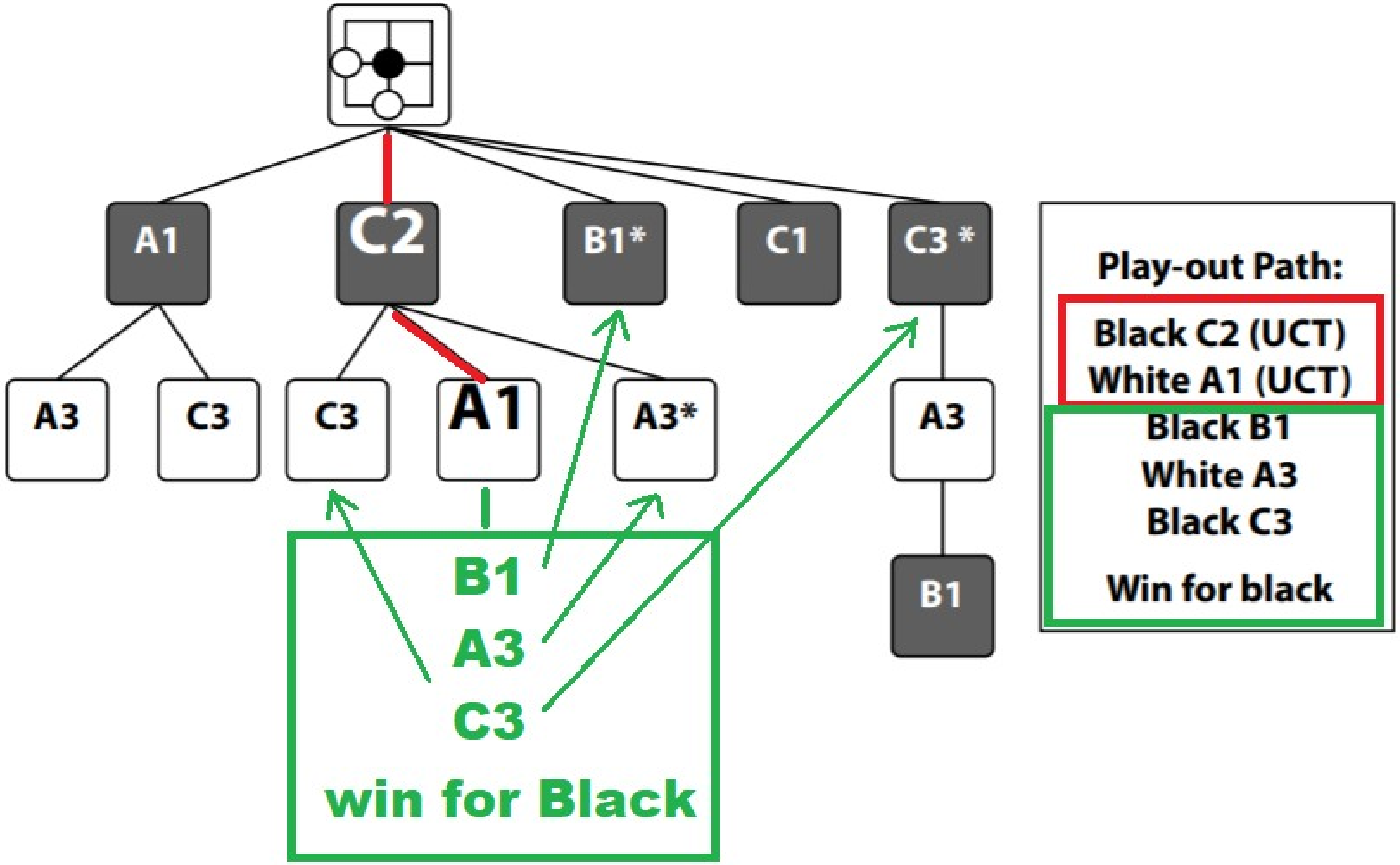
$$\text{UCT} = \frac{w_i}{s_i} + c\sqrt{\frac{\ln s_p}{s_i}} + \frac{b_i}{s_i}$$

when adding node to MCTS tree can initialize w_i and s_i

what is c? can determine experimentally

add a term $\frac{b_i}{s_i}$

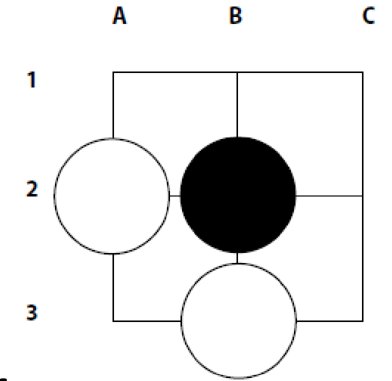
MCTS : AMAF heuristic



MCTS : AMAF heuristic

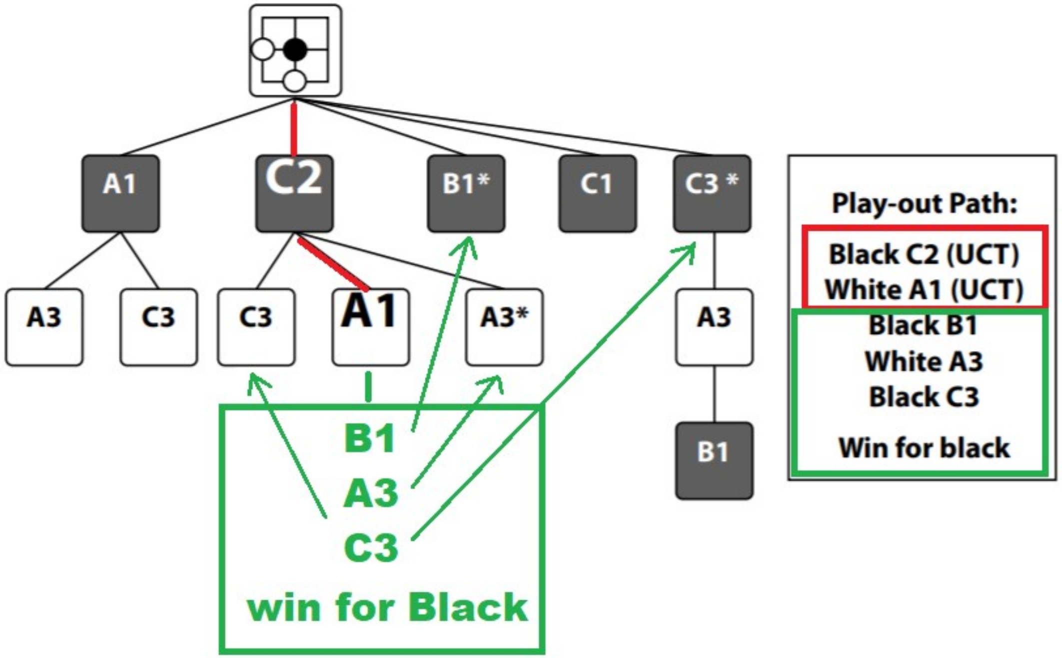
- AMAF = All Moves As First
- Value of making move **a** immediately is **average outcome of all simulations** where **a** is played **at any time**
  - Have a general value for each move, **regardless** of when it is played
- Biased estimate of true move value
- Based on independence assumption – value of move is unaffected by other moves before/after
- Advantage = quick estimates
- Disadvantage = biased

MCTS : AMAF nodes values



- For nodes on the selection path,
- Update those siblings of the selection path,
- That correspond to moves selected during simulation

- Update siblings of Start,C2,A1 based on the path Start → C2 → A1 → B1 → A3 → C3



MCTS : α-AMAF

- Compute/keep 2 sets of counts for each node
  - Standard MCTS estimates
  - AMAF estimates
- α-AMAF estimate is $\alpha \cdot \text{AMAF} + (1-\alpha) \cdot \text{Standard}$
  - $\alpha=0$ is Standard MCTS
  - $\alpha=1$ is (pure) AMAF
  - α-AMAF is blend of the two

MCTS : RAVE

- RAVE = Rapid Action Value Estimates
- Each node in MCTS tree has its own α that starts at 1 and then decreases to 0, as more iterations go through the node
  - If you have very few sampled paths through the node, you rely on its AMAF estimate
  - If you have many sampled paths through the node, you rely on its own Standard MCTS estimate

$$UCT_{RAVE}(s) = \alpha(s) \cdot UCT_{AMAF}(s) + (1-\alpha(s)) \cdot UCT_{Standard}(s)$$

$$\alpha(s) = \max(0, \frac{(P - s_i)}{P})$$

- **P** is a parameter, $\mathbf{s_i}$ is count for node **S**