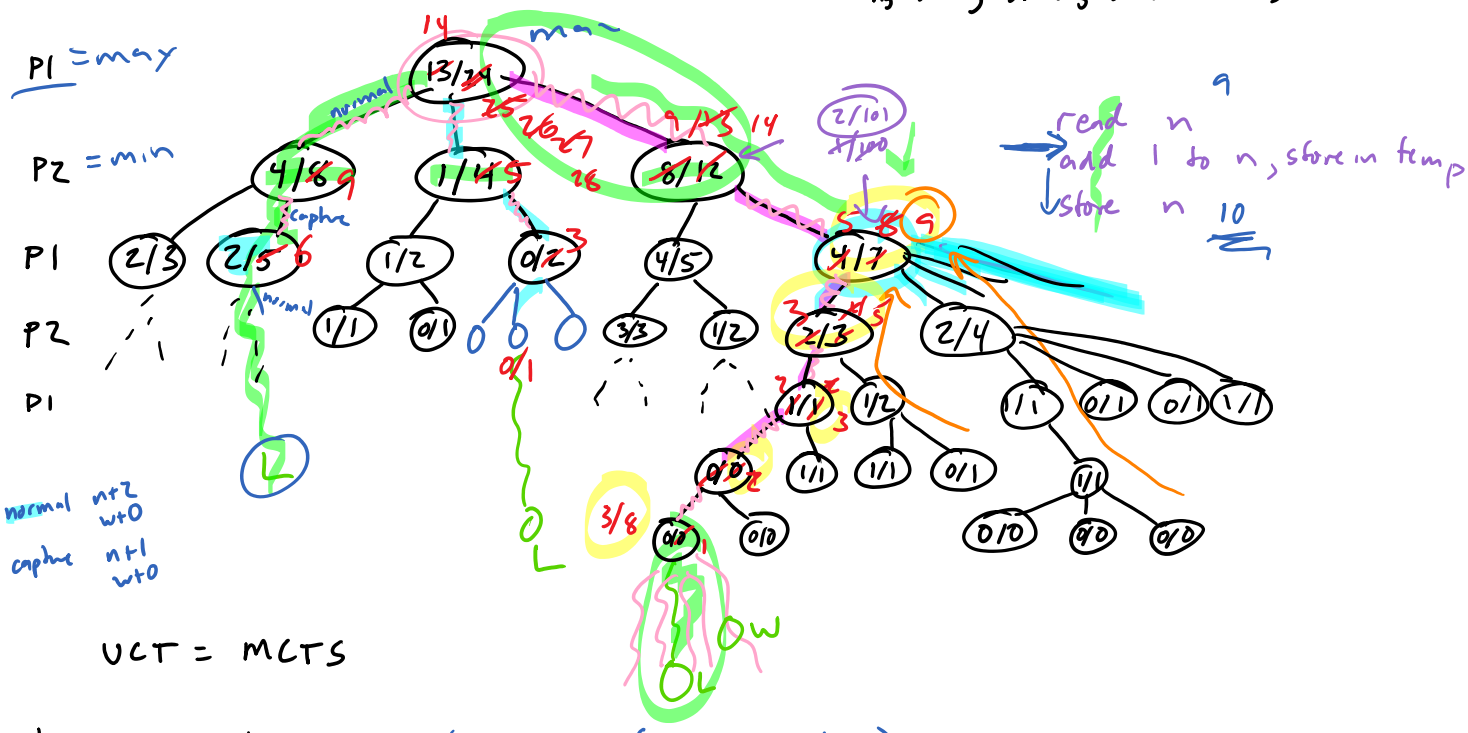
Monte Carlo Tree Search

Until out of time

- **traverse tree** — root → leaf (for example, UCB) — tree policy
- **expand** — if leaf is expandable, add its children (expandable = nonterminal and non-zero visits)
- **simulate** — play to terminal pos (from arb. selected added child if expanded) — default policy (for ex. random)
- **update** — backpropagate stats along path through tree from point simulation started → root

MCTS + UCB = UCT

Select move from root based on current stats (choose child w/ highest avg or highest visit count)



advantages: convergent — converges to minimax (given enough time)

anytime — returns a move after any number of iterations

no domain knowledge — no heuristic

easily parallelized
- leaf — multiple parallel playouts from newly added nodes
- tree — parallel traversals in same tree
- root — separate tree for each CPU, combine results (majority rule / combine stats)

disadvantages: no domain knowledge — if you know something about game, vanilla MCTS won't use that

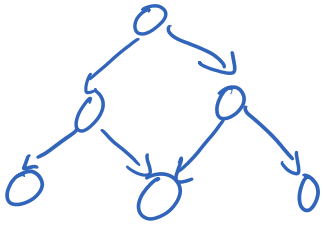
some games not amenable

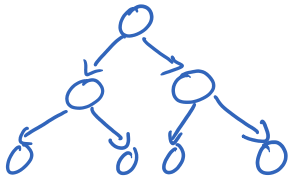
trap state — most moves bad
one good move — MCTS takes a long time

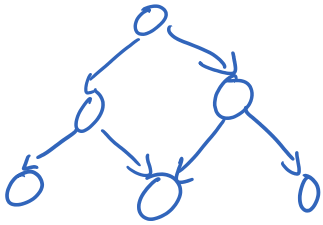
chess has trap states — MCTS not too good

Go — MCTS works well

MCTS adapted for games that aren't trees

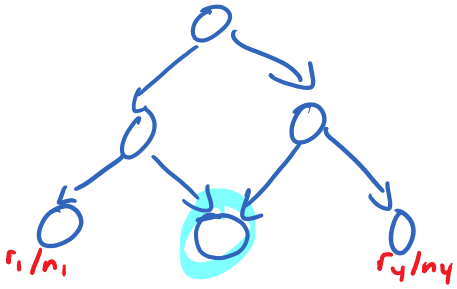


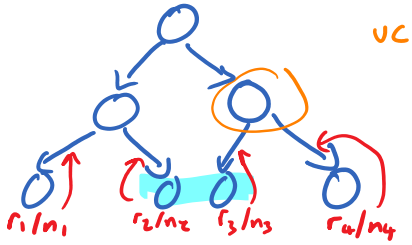




Monte Carlo Tree Search

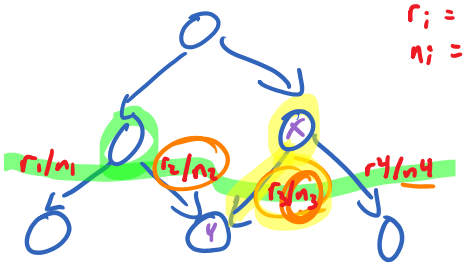
MCTS adapted for games that aren't trees





UCB0: ignore multiple paths

duplicate states reachable along different paths



r_i = total reward after traversing edge

n_i = total times edge traversed

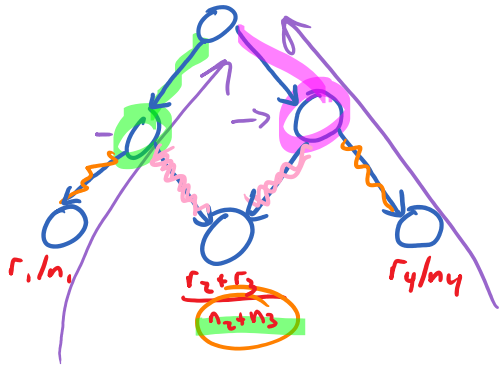
UCB1: ignore other paths when deciding where to go from X, use value

$$\frac{r_3}{n_3} + \sqrt{\frac{c \ln(n_3 + n_4)}{n_3}}$$

for Y

UCB2: combine multiple paths in exploit

$$\frac{r_2 + r_3}{n_2 + n_3} + \sqrt{\frac{c \ln(n_3 + n_4)}{n_3}}$$



no guarantee on convergence

$$\frac{r_2 + r_3}{n_2 + n_3} + \sqrt{\frac{c \ln(n_3 + n_4)}{n_2 + n_3}}$$

MAST : move averaging sampling technique

different kinds of moves

keep global stats for each kind of move

bias playouts towards kinds of moves w/ better statistics

PAST : predicate averaging ....

keep statistics for each predicate T
each predicate F

bias based on which predicates a move makes T/F

wins / traversals through states that make

20 / 30 P(s) T

15 / 40 P(s) F

25 / 50 Q(s) T

10 / 20 Q(s) F

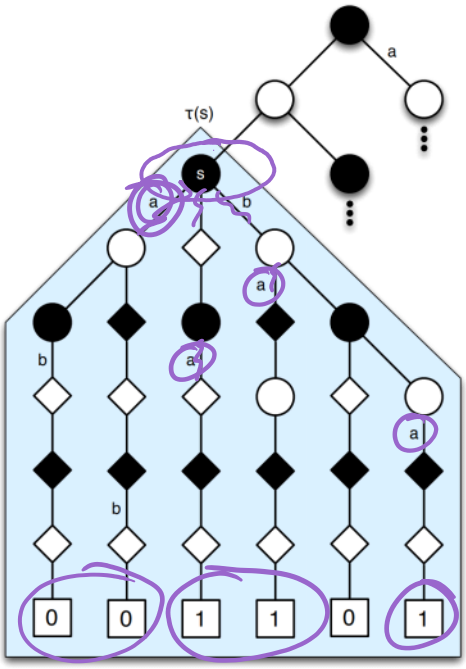
when considering a move to s'

Suppose P(s') T
Q(s') F

bias according to combination of corresponding stats

MC-RAVE

rapid averaging value estimation



$$Q(s,a) = 0/2$$
$$Q(s,b) = 2/3$$
$$\tilde{Q}(s,a) = 3/5$$
$$\tilde{Q}(s,b) = 2/5$$

all moves as first (AMAF)

$Q(s,a)$ does reward having taken action a from state s

$\tilde{Q}(s,a)$ takes AMAF heuristic into account

does reward for action a over subtree rooted at s

From Gelly and Silver, Monte-Carlo tree search and rapid action value estimation in computer Go. Artif. Intell. 175, 1856-1875, 2011

weight of $\tilde{Q}$ vs Q

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

$\beta(s,a) \approx 1$ if $N(s,a)$ small

$\beta(s,a) \approx 0$ $N(s,a)$ large

$$\sqrt{\frac{k}{3N(s)+k}}$$

for Go ($k \approx 1000$) Fig 5

$N(s)$ = # times visited s