A Review of:

A Survey of Monte Carlo Tree Search Methods

CAMERON BROWNE, EDWARD POWLEY, DANIEL WHITEHOUSE, SIMON LUCAS, PETER I. COWLING, PHILIPP ROHLFSHAGEN, STEPHEN TAVENER, DIEGO PEREZ, SPYRIDON SAMOTHRAKIS AND SIMON COLTON

IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES, VOLUME 4, PP 1-43, 2012.

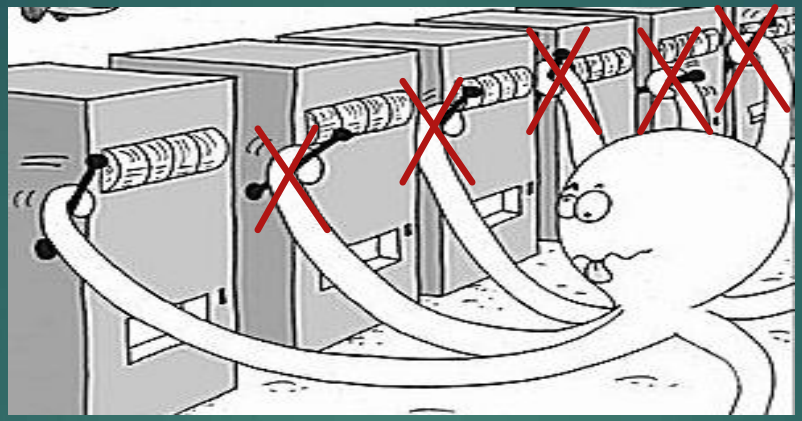
Presentation by Bobak Pezeshki
2018 – 02/12

Sources

- As the focus of this presentation, the majority of the information and images in these slides are from the mentioned "A Survey of Monte Carlo Tree Search Methods" publication
- Thank you to Tsan-sheng Hsu (Academia Sinica, Institute of Information Science) for his slide on AMAF (All Moves As First)
- Thank you to Dr. Rina Dechter (UC Irvine) and Dr. Kalev Kask (UC Irvine) for their support, instruction, and guidance

K-Armed Bandit

K-Armed Bandit



- Want to choose only the best arm!!!
  - …but we don't know what it is
    - Try a bunch of times all over the place to figure out?
    - Focus on the one that's been best so far?

We're Online!

- Our actions are not for free…
  - We could be missing out on chances of reward
  - In some cases, we can even take serious penalties
- We don't want to "regret" our search strategy
  - We want it to do as good as it can

minimize regret:

$$R_N = \mu^\star n - \mu_j \sum_{j=1}^{K} \mathbb{E}[T_j(n)]$$

basically, the difference between the best we could possibly do, and what we do

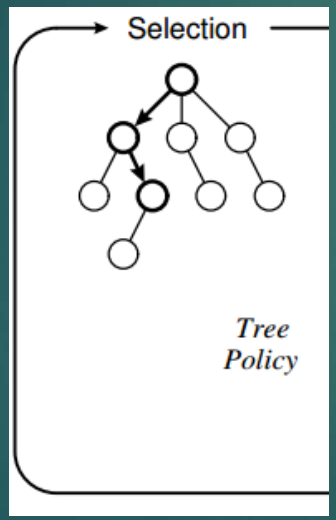
I'll Find The Best Arm!

- …Better explore branch with best arm
  - need to make sure non-zero probabilities for exploration
- But exploring can lead to regret!
  - There is no policy with slower growing regret than O(ln n)
    - If we do within a constant factor of this = pretty darn successful!
- One such policy: play arm that maximizes $\mathrm{UCB1} = \overline{X}_j + \sqrt{\frac{2 \ln n}{n_j}}$

Monte-Carlo Tree Search: Main Idea

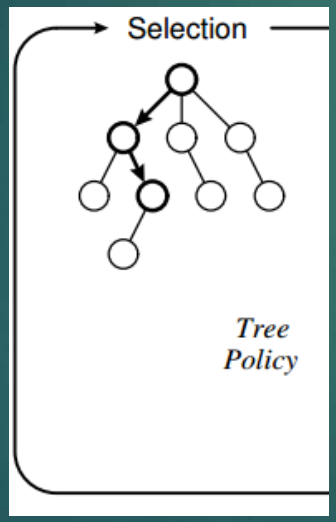
Monte-Carlo Tree Search: Main Idea

- Explore decision tree by going through the next "urgent" node
  - Uses certain "statistics" kept about the nodes



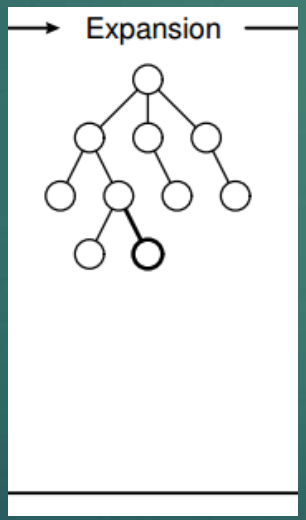
Main Idea

- ***Tree Policy*** is a method of choosing which node is most urgent to explore from



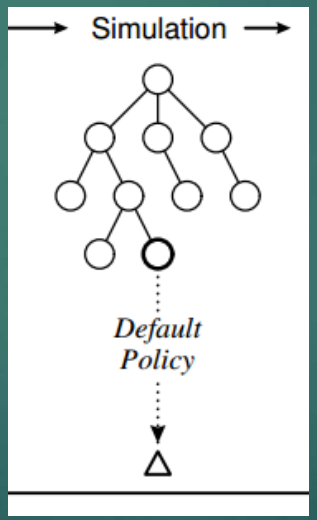
Main Idea

- Take turns expanding the children of the selected node



Main Idea

- Run a simulation to gain information about that node's quality/value

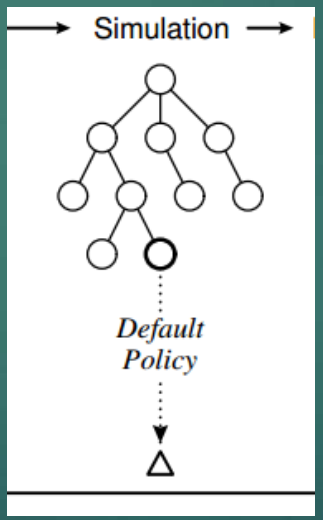


How To Choose Simulation Action?

- Simplest method = uniformly from set of actions available!
- Useful?
  - => world champion level Scrabble Play
  - => world champion level Bridge Play
- Can we do better?
  - One area of research regarding MCTS

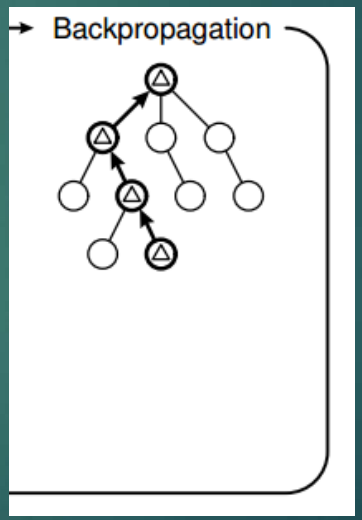
Main Idea

- ***Default Policy*** is a method of traversing through the rest of the tree until a terminal node is reached that has a determinable value



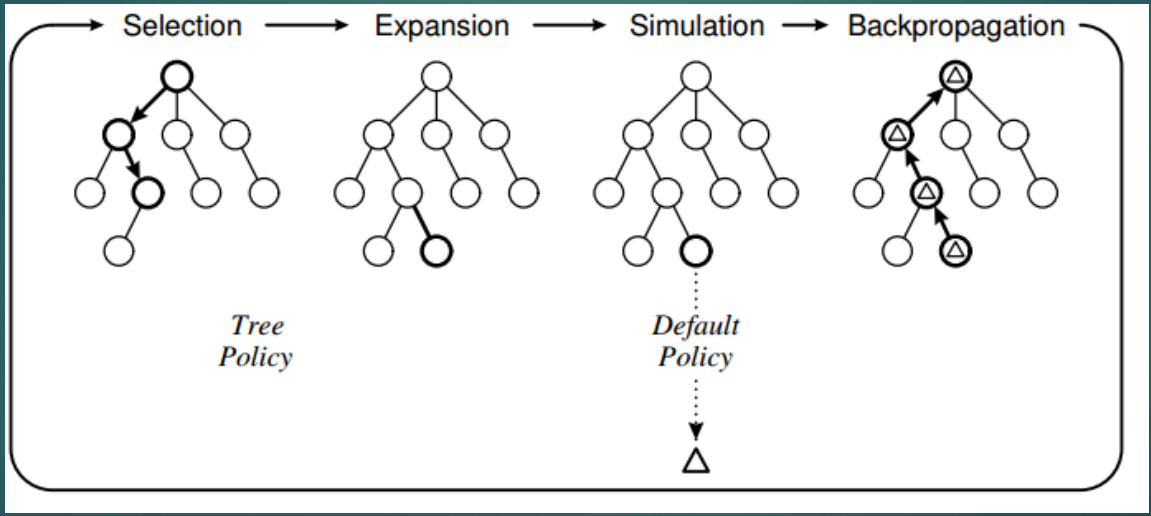
Main Idea

- Update information from the results of the simulation back up through the tree's nodes!



Main Idea

- Over time, we generate information about which paths are preferable!

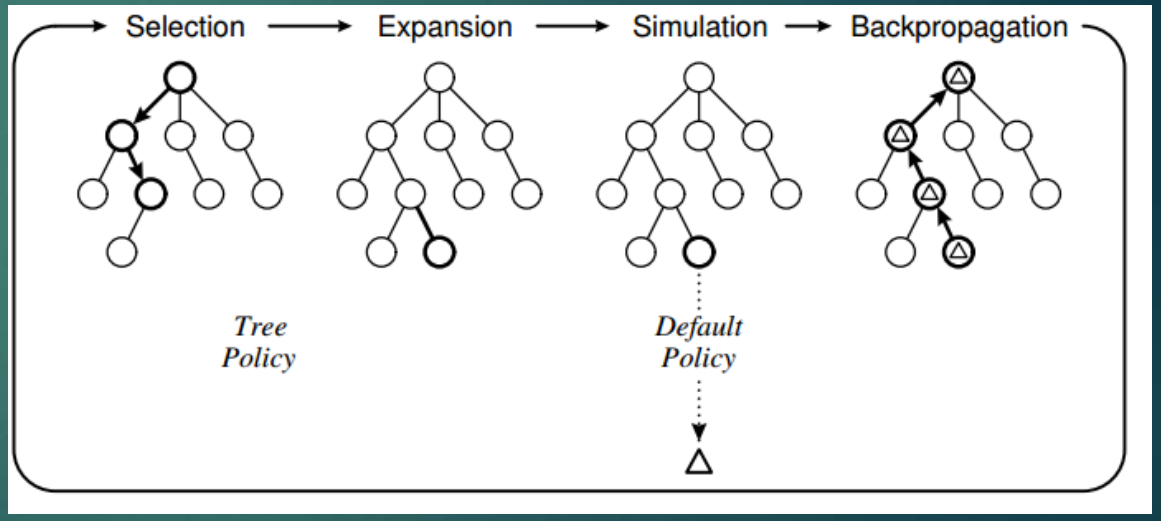


Main Idea

- Over time, we generate information about which paths are preferable!

Algorithm 1 General MCTS approach.

function MCTSSEARCH(s_0)
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow$ TREEPOLICY(v_0)
    $\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$)
    BACKUP(v_l, Δ)
  return a(BESTCHILD(v_0))



Quality of an Action

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

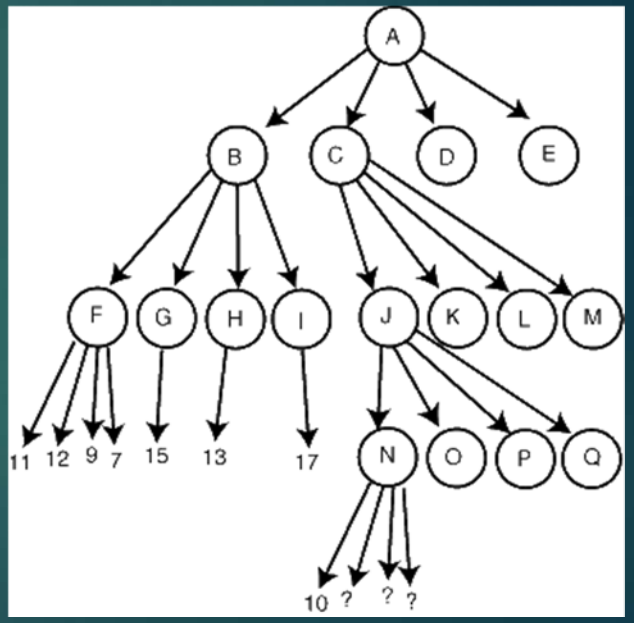
z_i = reward
I_i = wheter action ***a*** was taken from state ***s***

.
.
.

Q(s,a) = the current expectation of its reward

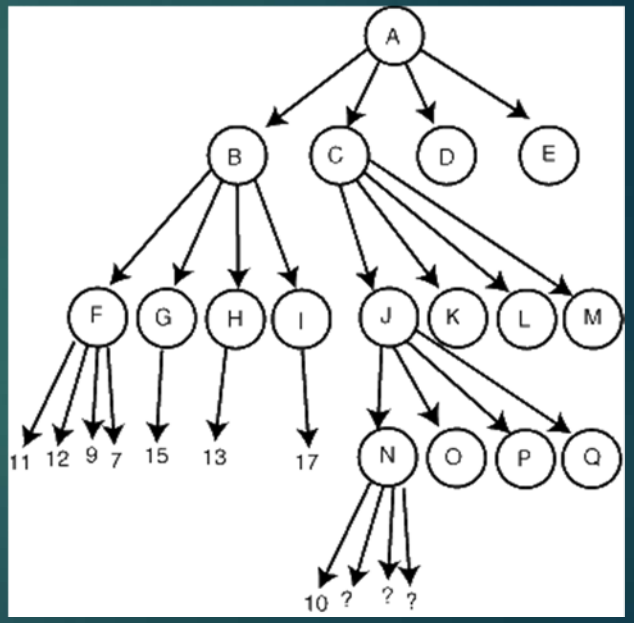
Times-Up! (How to chose a move)

- Times up and we've been simulating from node A…
  - what action should we take?
- Some methods of choosing:
  - Chose action leading to child with max value
  - Choose action leading to child we've visited most
  - Choose action that maximizes each of the above
    - (don't stop until reach such a point)
  - Choose action leading to child that maximizes a lower confidence bound



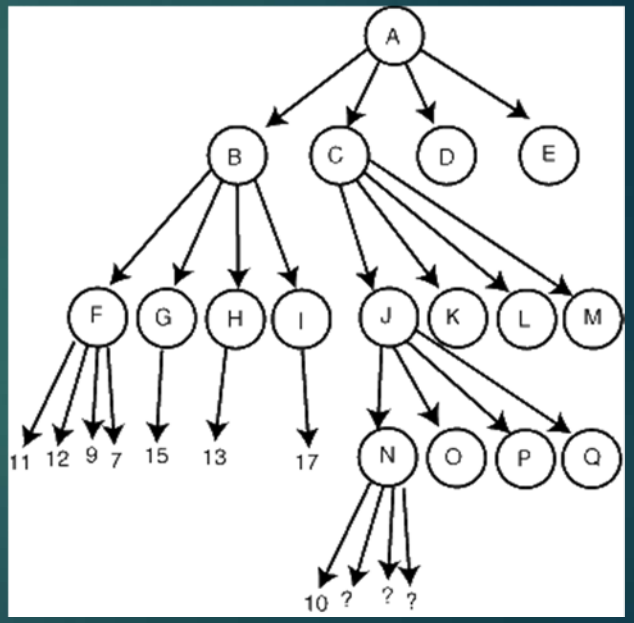
Times-Up! (How to chose a move)

- Times up and we've been simulating from node A…
  - what action should we take?
- Some methods of choosing:
  - Max Child
  - Choose action leading to child we've visited most
  - Choose action that maximizes each of the above
    - (don't stop until reach such a point)
  - Choose action leading to child that maximizes a lower confidence bound



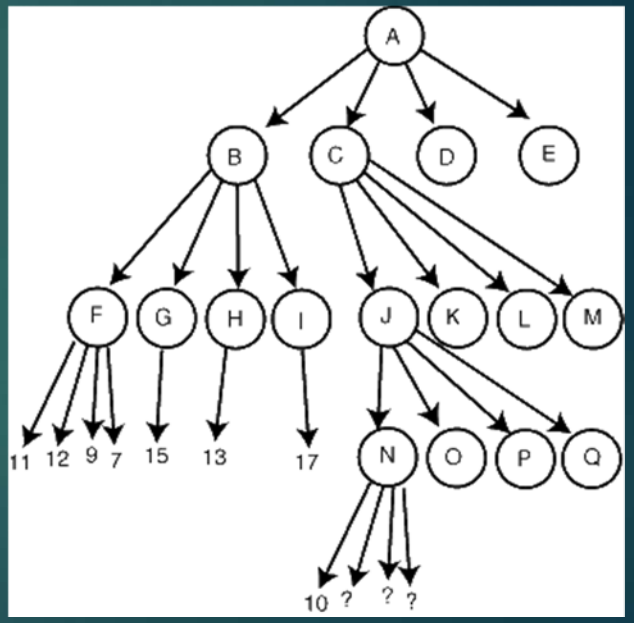
Times-Up! (How to chose a move)

- Times up and we've been simulating from node A…
  - what action should we take?
- Some methods of choosing:
  - Max Child
  - Robust Child
  - Choose action that maximizes each of the above
    - (don't stop until reach such a point)
  - Choose action leading to child that maximizes a lower confidence bound



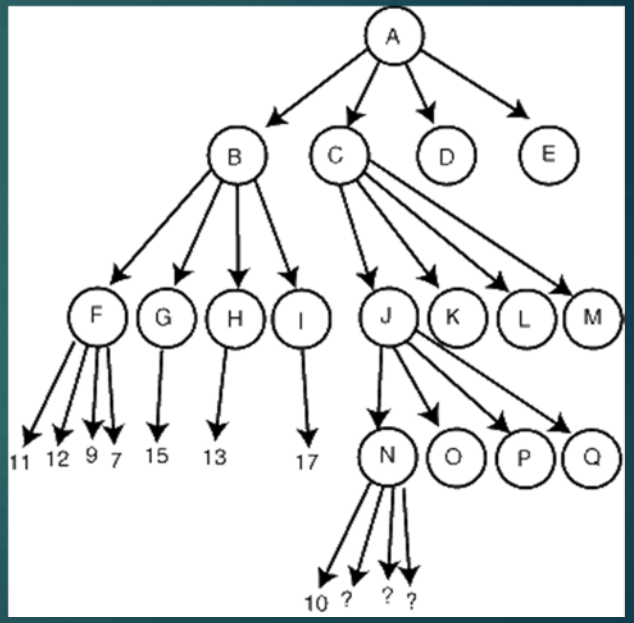
Times-Up! (How to chose a move)

- Times up and we've been simulating from node A…
  - what action should we take?
- Some methods of choosing:
  - Max Child
  - Robust Child
  - Max-Robust Child
  - Choose action leading to child that maximizes a lower confidence bound



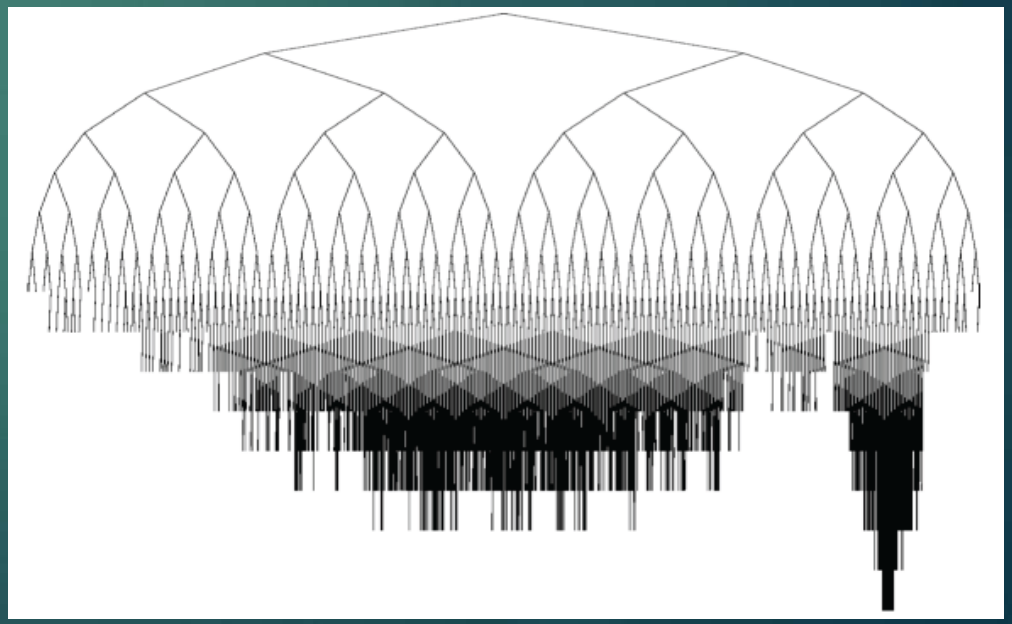
Times-Up! (How to chose a move)

- Times up and we've been simulating from node A…
  - what action should we take?
- Some methods of choosing:
  - Max Child
  - Robust Child
  - Max-Robust Child
  - Secure Child



Some Advantages

- Anytime
  - Can end the tree search whenever we choose
- Can narrow down search space in large, highly branched, trees
- Can work when we have very little domain knowledge
- Useful for games and planning
  - Utilizes sequential decisions



Some Milestones of MCTS

1990	Abramson demonstrates that Monte Carlo simulations can be used to evaluate value of state [1].
1993	Brügmann [31] applies Monte Carlo methods to the field of computer Go.
1998	Ginsberg's GIB program competes with expert Bridge players.
1998	MAVEN defeats the world scrabble champion [199].
2002	Auer et al. [13] propose UCB1 for multi-armed bandit, laying the theoretical foundation for UCT.
2006	Coulom [70] describes Monte Carlo evaluations for tree-based search, coining the term Monte Carlo tree search.
2006	Kocsis and Szepesvari [119] associate UCB with tree-based search to give the UCT algorithm.
2006	Gelly et al. [96] apply UCT to computer Go with remarkable success, with their program MoGo.
2006	Chaslot et al. describe MCTS as a broader framework for game AI [52] and general domains [54].
2007	CadiaPlayer becomes world champion General Game Player [83].
2008	MoGo achieves *dan* (master) level at 9×9 Go [128].
2009	Fuego beats top human professional at 9×9 Go [81].
2009	MoHex becomes world champion Hex player [7].

Very Interesting…

- How to explore the tree
  - Tree Search Policy
- How to expand nodes
- How to simulate the play-outs
  - Default (and other) Policies
- How to back-propagate information

Overarching Methodologies

Early Novel Tree Policy Approaches

- Nodes selected according to estimated probability that they are better than the current best move
- Minimize error probability if algorithm stopped prematurely
  - …but still converges to game-theoretic optimum given sufficient time!

Early Novel Tree Policy Approaches

- Nodes selected according to estimated probability that they are better than the current best move
- Minimize error probability if algorithm stopped prematurely
  - …but still converges to game-theoretic optimum given sufficient time!
- **Need a balance for exploration / exploitation!**

Most Popular MCTS Algorithm:

UCT

Upper Confidence Bounds for Trees

The UTC Algorithm

- Idea = use the UCB1 algorithm!!!
  - Choice of child nodes = k-armed bandit!
- Promising Properties
  - Simple
  - Efficient
  - Guaranteed to be within a constant factor of best possible bound on the growth of regret
  - Converges to minimax given sufficient time and memory

The UTC Algorithm

Maximize:

$$UCT = \overline{X}_j + 2C_p\sqrt{\frac{2\ln n}{n_j}}$$

- Notice when $n_j = 0$, value = ∞
  - promotes exploration of new nodes
- Notice as n grows, a node not visited increases its likelihood of being explored

The UTC Algorithm

Maximize:

$$UCT = \overline{X}_j + 2C_p\sqrt{\frac{2\ln n}{n_j}}$$

UCT = Exploitation Weight + Exploration Weight

$C_p \approx$ Exploration Multiplier

The UTC Algorithm

Maximize:

$$UCT = \overline{X}_j + 2C_p\sqrt{\frac{2\ln n}{n_j}}$$

$C_p = 1/\sqrt{2}$ satisfies Hoeffding inequality for rewards [0,1]

The UTC Algorithm

Maximize:

$$UCT = \overline{X}_j + 2C_p\sqrt{\frac{2\ln n}{n_j}}$$

Statistics of Nodes:

- number of times visited
- total reward of ALL playouts passing through it

The UTC Algorithm

Maximize:

$$UCT = \overline{X}_j + 2C_p\sqrt{\frac{2\ln n}{n_j}}$$

Final Choice of Move:

- Max Child
- Robust Child

The UTC Algorithm

Maximize:

$$UCT = \overline{X}_j + 2C_p\sqrt{\frac{2\ln n}{n_j}}$$

Final Choice of Move:

- Child with greatest value
- Child with highest visit count

The UTC Algorithm

Algorithm 2 The UCT algorithm.

function UCTSEARCH(s_0)
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow$ TREEPOLICY(v_0)
    $\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$)
    BACKUP(v_l, Δ)
  return a(BESTCHILD($v_0, 0$))

function TREEPOLICY(v)
  while v is nonterminal **do**
    if v not fully expanded **then**
      return EXPAND(v)
    else
      $v \leftarrow$ BESTCHILD(v, Cp)
  return v

function EXPAND(v)
  choose $a \in$ untried actions from $A(s(v))$
  add a new child v' to v
    with $s(v') = f(s(v), a)$
    and $a(v') = a$
  return v'

function BESTCHILD(v, c)
  return $\underset{v' \in \text{children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$

function DEFAULTPOLICY(s)
  while s is non-terminal **do**
    choose $a \in A(s)$ uniformly at random
    $s \leftarrow f(s, a)$
  return reward for state s

function BACKUP(v, Δ)
  while v is not null **do**
    $N(v) \leftarrow N(v) + 1$
    $Q(v) \leftarrow Q(v) + \Delta(v, p)$
    $v \leftarrow$ parent of v

Variations of MCTS

- ***Flat Monte Carlo***: A Monte Carlo method with uniform move selection and no tree growth.
- ***Flat UCB***: A Monte Carlo method with bandit-based move selection (2.4) but no tree growth.
- ***MCTS***: A Monte Carlo method that builds a tree to inform its policy online.
- ***UCT***: MCTS with any UCB tree selection policy.
- ***Plain UCT***: MCTS with UCB1 as proposed by Kocsis and Szepesvári [119], [120].

Flat UCB

- Leaves of the tree treated as multi-armed bandit
  - Try to explore leaves!
- Maintains characteristics of UCT
  - Improves regret bound over UCB1 in certain scenarios
- Led to **Bandit Algorithm for Smooth Trees (BAST)**
  - Extension of Flat UCB in which high confidence sub-optimal branches are avoided!
  - Only optimal branches expanded indefinitely given infinite time!

Single-Player MCTS

- Often include variance as factor
  - Higher σ^2 ➔ greater the chance of choosing node

Often added third term: $\sqrt{\sigma^2 + \frac{D}{n_i}}$

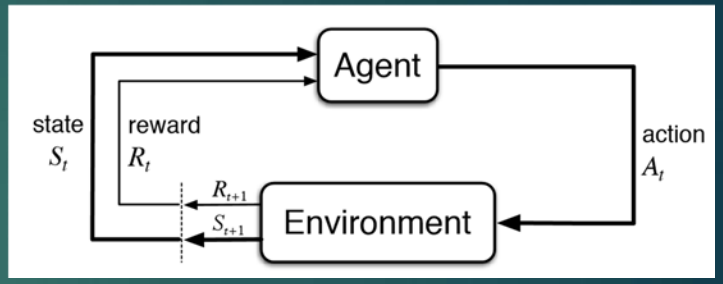
- Single player games can often afford to be more permanent when finding strong lines of play
  - No opponent to worry about needing to play sub-optimally

Multi-Agent MCTS

- Ex. Have two "agents" running simulation as opposition to each other
  - ie. Assigning different heuristics / simulation policies
- Can lead to emergent properties
- Main Challenge = choosing properties of such agents
- Two separate MCTS, then combine at end?
  - Proclaimed, but not reproduced

Reinforcement Learning!!!

- We can already see similarities
  - Attempting to "learn" optimal "paths"



- Relation to TD(λ)?

Comparison

MCTS	TD(λ)
Learn to take actions from (s,a)	Learn to take actions from (s,a)
Tree-building algorithm	Does not usually build trees
Estimates temporary state values	Learns long term state value

Do I feel a merger coming?

MCTS / TD(λ) Hybrid

4.3.2 Temporal Difference with Monte Carlo (TDMC(λ))

Osaki et al. describe the *Temporal Difference with Monte Carlo* (TDMC(λ)) algorithm as "a new method of reinforcement learning using winning probability as substitute rewards in non-terminal positions" [157] and report superior performance over standard TD learning for the board game Othello (7.3).

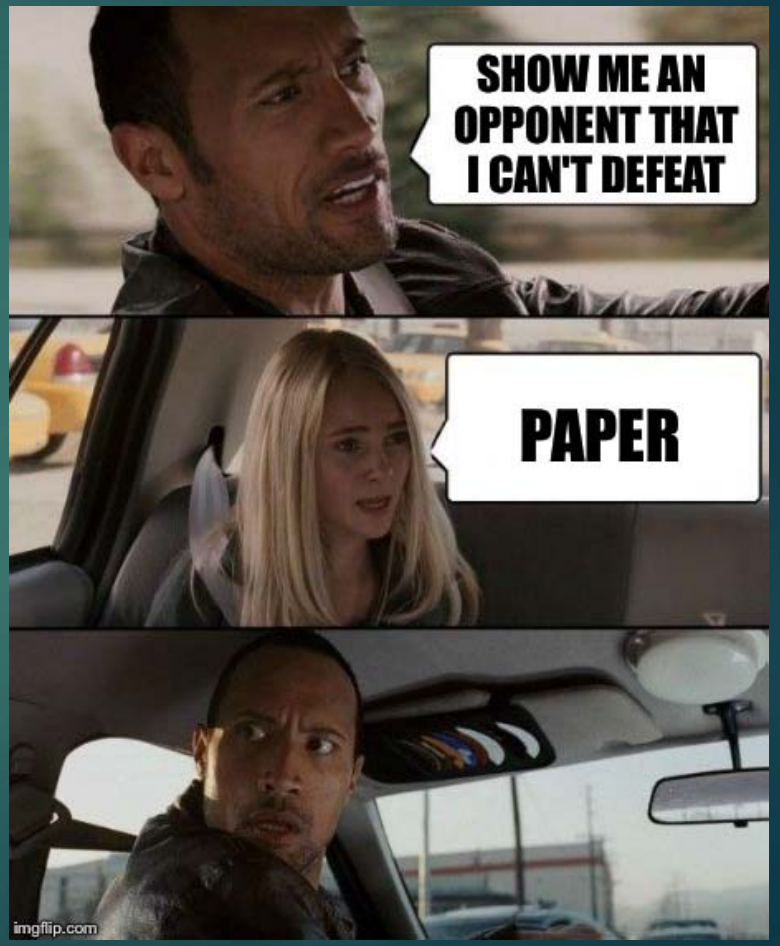
Non-Determinant Games

Determinization

- Instantiate state into a fully observable deterministic environment by using belief state
- Can perform at beginning of each iteration of MCTS
  - Then proceed as if deterministic
- Pantom Go

Opponent Modelling for MCTS

- Can use Bayesian inference / relational probability tree learning to model opponent behavior
  - Have general model and a prior
  - Continue improving estimation of parameters
- This can be used in MCTS when simulation is simulating opponent moves!



A Ton of Recursive Algorithms

- Can use variation of MCTS recursively through search iterations
- Can also combine with other tree searches such as BFS

Can be Used For Planners and SAT

- The list goes on…

Enhancing MCTS Tree Policy

Enhancing MCTS Tree Policy

- Two main categories of enhancements
  - Domain Independent
    - Can be applied to any domain without prior knowledge
    - Typically result in small improvements
  - Domain Dependent
    - Specific to certain domains
    - Exploit unique aspects / Utilize prior knowledge

UCB1-Tuned Node Selection

- Replace exploration term with:

$$\sqrt{\frac{\ln n}{n_j} \min\{\frac{1}{4}, V_j(n_j)\}}$$

where:

$$V_j(s) = (1/2 \sum_{\tau=1}^{s} X_{j,\tau}^2) - \overline{X}_{j,s}^2 + \sqrt{\frac{2 \ln t}{s}}$$

  - Implies machine j has variance at most = sample variance + √[(2 ln t)/(s)]

- Regret bound unproven, but empirically performs better than many MCTS algorithms

Bayesian UCT

- Use a Bayesian framework
  - Potentially more accurate estimations of node values and node uncertainty when number of trials is limited

- Two UCT exploration-term modifications to go with framework

$$\text{maximise } B_i = \mu_i + \sqrt{\frac{2 \ln N}{n_i}}$$

motivated by optimistic prior / independence assumptions

$$\text{maximise } B_i = \mu_i + \sqrt{\frac{2 \ln N}{n_i}} \sigma_i$$

motivated by CLT

Other Bandit Based Modifications

- EXP3
  - Probabilistic
    - Partial observability games
    - Simultaneous-moves games
- Hierarchical Optimistic Optimization for Trees
- Many More

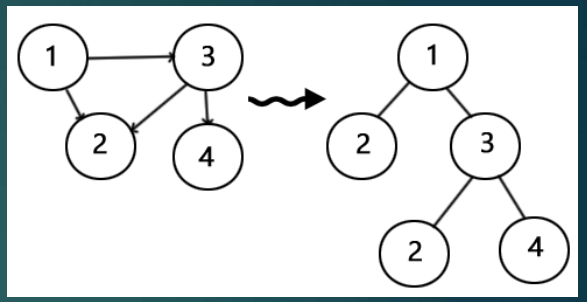
Node Selection Enhancements

Node Selection Enhancements

- First Play Urgency
  - New children immediately have some value assigned
  - Not necessarily expand all unexplored actions
    - Earlier exploitation
    - Deeper tree analysis in large branching factor trees
- Decisive and Anti-Decisive Moves
  - Must do guaranteed good moves during selection and default
- Move Groups
  - Combine “similar actions” into a group, then use UCB1
    - Can reduce large branching factors of similar moves

Node Selection Enhancements

- Transpositions
  - Model games that are in actuality more DAGs
  - Identical states via different paths are "transposed" and their information conglomerated



- Progressive Bias
  - Add domain specific knowledge as a heuristic
  - Acts kind of like a prior
  - Additional term: $$f(n_i) = \frac{H_i}{n_i + 1}$$

Node Selection Enhancements

- Opening Books
  - Use table of known positions / configurations to drive selection of nodes
- Monte Carlo Paraphrase Generation (MCPG)
  - UTC except use maximum reachable score (not average score)
- Search Seeding
  - Use experience to seed nodes with artificial values of num visits and num wins to help narrow tree search

Node Selection Enhancements

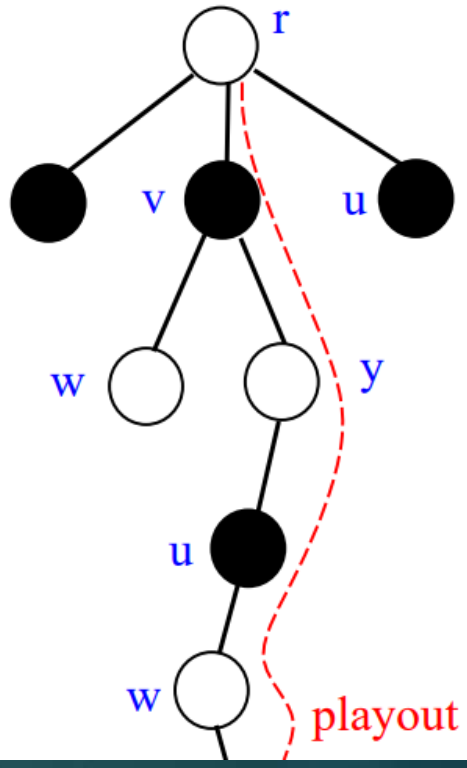
- History Heuristic
  - Use previous plays as heuristic
    - Tree-tree level = use history info to improve action selection in MCTS tree
    - Tree-playout level = use history info to improve simulation
- Progressive History
  - Progressive bias that, when history info is available, replaces H value with historical value

$$f(n_i) = \frac{H_i}{n_i + 1}$$

All Moves As First (AMAF)

- Update every state-action encountered in the playout
  - Even if the state-action did not originate in the original selection phase

- **Assume a playout is simulated with the sequence of r, v, y, u, w, $\cdots$.**
- **The statistics of nodes along this path are updated.**
- **The statistics of node u that is a child of r, and node w that is a child of v are also updated.**



Thank You
Tsan-sheng Hsu

All Moves As First (AMAF)

- Permutation AMAF
  - Also update other leaf nodes (nodes up for selection) that can lead to the same terminal state via a different order of actions
- α – AMAF
  - Keep AMAF statistics separately
  - Calculate UTC using: $\alpha A + (1 - \alpha)U$
- Some-First AMAF
  - Truncates the AMAF after m-depth of simulation moves
- Cutoff AMAF
  - AMAF only for the first k-iterations = jump start

All Moves As First (AMAF)

- Rapid Action Value Estimation (RAVE)
  - Similar to α-AMAF except α decreases and num visits increases
  - Kind of like mixing α-AMAF with Cutoff AMAF
  - For α uses: $\max\left\{0, \frac{V - v(n)}{V}\right\}$
- Killer RAVE
  - Only most important(?) moves used for RAVE updates
- PoolRAVE
  - Build pool of k best moves according to RAVE
  - Choose one move m
  - Play m with a probability p, else use the default policy

Insights About AMAF

- Random playouts provide more about the goodness of earlier moves of the playout
- AMAF updates are useful even after the playout
- More aggressive updates are often beneficial
- Combining heuristics ➔ more powerful

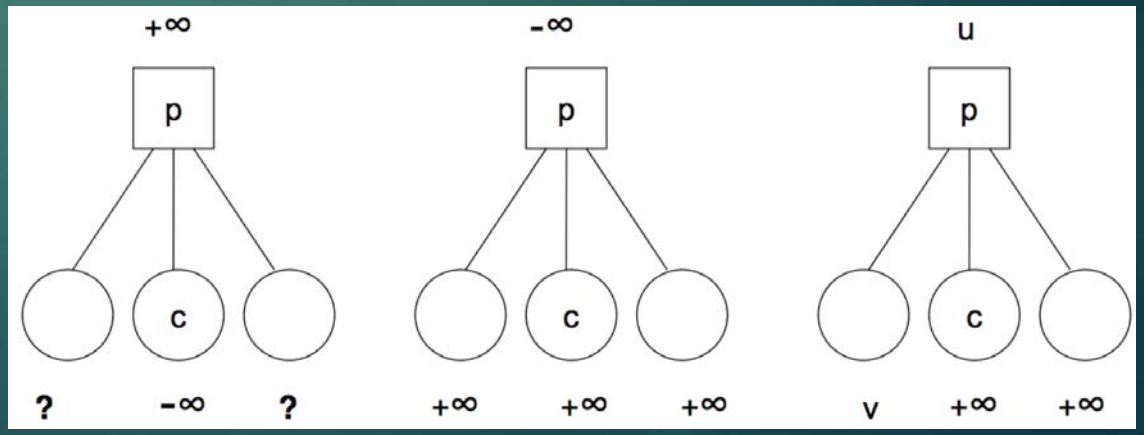
Game-Theoretic Enhancements

Game-Theoretic Enhancements

- If we know (*for sure*) what a certain node's value is, this info can be extremely useful to the nodes leading up to it
  - Looking for nodes leading to proven wins / losses

Game-Theoretic Enhancements

- MCTS-Solver (Using PNS)
  - Concept:
    - If an action leads to a state that is guaranteed to win, then the current state is guaranteed to win
    - If all actions lead to sates that are guaranteed to lose, then current state is guaranteed to lose
  - Uses Proof-Number Search (PNS)
    - Prioritize nodes whose value can be proven by exploring the fewest children
  - Used for end-game solvers



Game-Theoretic Enhancements

- Monte Carlo Proof-Number Search (MC-PNS)
  - Nodes that do not have proven value are searched via MCTS
- Score Bounded MCTS
  - Useful in games with multiple ending scores (not necessarily zero-sum)
  - Use optimistic and pessimistic bounds on score
  - Values converge to estimated score with many iterations
    - When the two bounds are equal ➔ node solved

Move Pruning

Move Pruning

- Two main categories:
  - Soft Pruning
    - These nodes may later be searched, but temporarily pruned
  - Hard Pruning
    - Never to be searched / selected again

Move Pruning

- Progressive Unpruning / Widening
  - Soft Pruning
  - Ensures that, over infinite time, all nodes are visited
  - Idea
    - Start off aggressively exploitation (in case of short time available)
    - Progressively widen search space as time becomes plentiful
- Absolute and Relative Pruning
  - When you can be sure an action a has so many more visits than its sibling actions a' that they can never overtake the number of visits a has, prune a'
- Pruning with Domain Knowledge → exactly what it sounds like

Node Expansion Enhancements

Simulation Enhancements

Simulation Enhancements

- Rule-Based Simulation Policy
  - Hand design simulation policy
- Contextual Monte Carlo Search
  - Combine simulations that reached similar parts of the tree
  - Use past simulations to drive action decisions
- Fill the Board
  - At each simulation step, choose N random intersections
    - If intersection and immediate neighbors empty, choose action
    - Else, randomly choose some legal action
  - Fills up board space quickly ➔ Patterns become more apparent

Simulation Learning

- Move-Average Sampling Technique (MAST)
  - Keep a record of average Q(a) (independent of state)
  - Bias future actions choices by Gibbs distribution
- Predicate-Average Sampling Technique (PAST)
  - Similar to MAST except keeps predicates helping to denote a certain context
  - Bias future action choices matching predicates by Gibbs distribution
    - ie. Takes into account the context of the action, and only biases using actions matching the same context

Simulation Learning

- Feature-Average Sampling Technique (FAST)
  - Similar to MAST and PAST
  - Features of the game extracted
    - how? manually?
  - TD(λ) to discern relative importance of features
  - Used to weigh the Q(a)'s

Simulation Learning

- Using History Heuristics
  - Use history of good/bad moves to inform simulation steps
    - Described as "using history information at the tree-playout level"
    - Two types
      - Internal heuristic = alters moves made during playouts
      - External heuristic = changes the move selected before the playouts
- Use of evaluation functions
  - Can be helpful in the beginning of the tree construction to avoid early bad moves
    - Then switch to greedy or some other variation

Simulation Learning

- Simulation Balancing
  - Gradient descent to bias policy during simulations
    - Does not always lead to “strong” play, but often “balanced” play
- Last Good Reply (LGR)
  - What move causes a win from the previous move? = Good Reply
    - Store this with respect to the previous move
    - Use it when the previous move pops up again
    - Overwritable by newer “Good Replies”
      - Thus “Last” Good Reply
  - With Fogretting = erase if Good Reply ever leads to a loss in a simulation

Simulation Learning

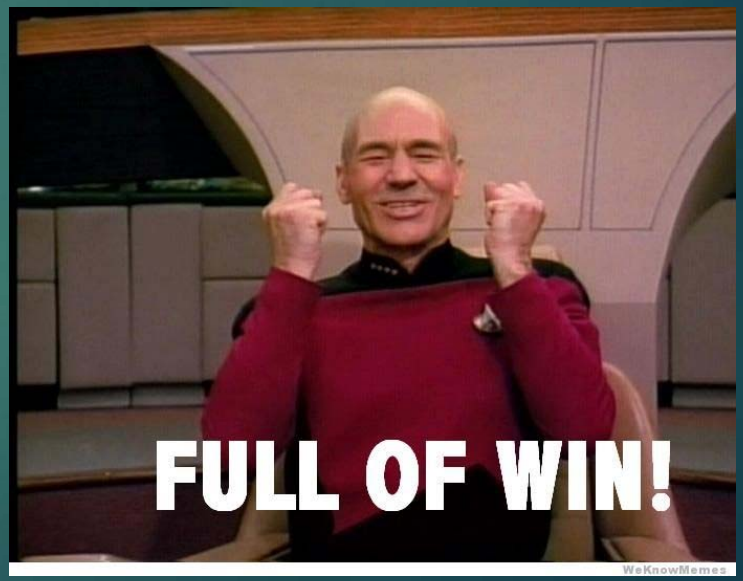
- Patterns
  - Use patterns to inform simulation

- Literature shows that spending time improving simulation is more advantageous than spending time during selection
  - Artifact of our already good advancement in selection modifications?

Backpropagation Enhancements

Backpropagation Enhancements

- Weighing Simulation Results
  - Higher weight for…
    - Shorter simulations
    - Later in game simulations
- Score bonus
  - Back propagate values [0,1], the range of which describes bad loses to strong wins

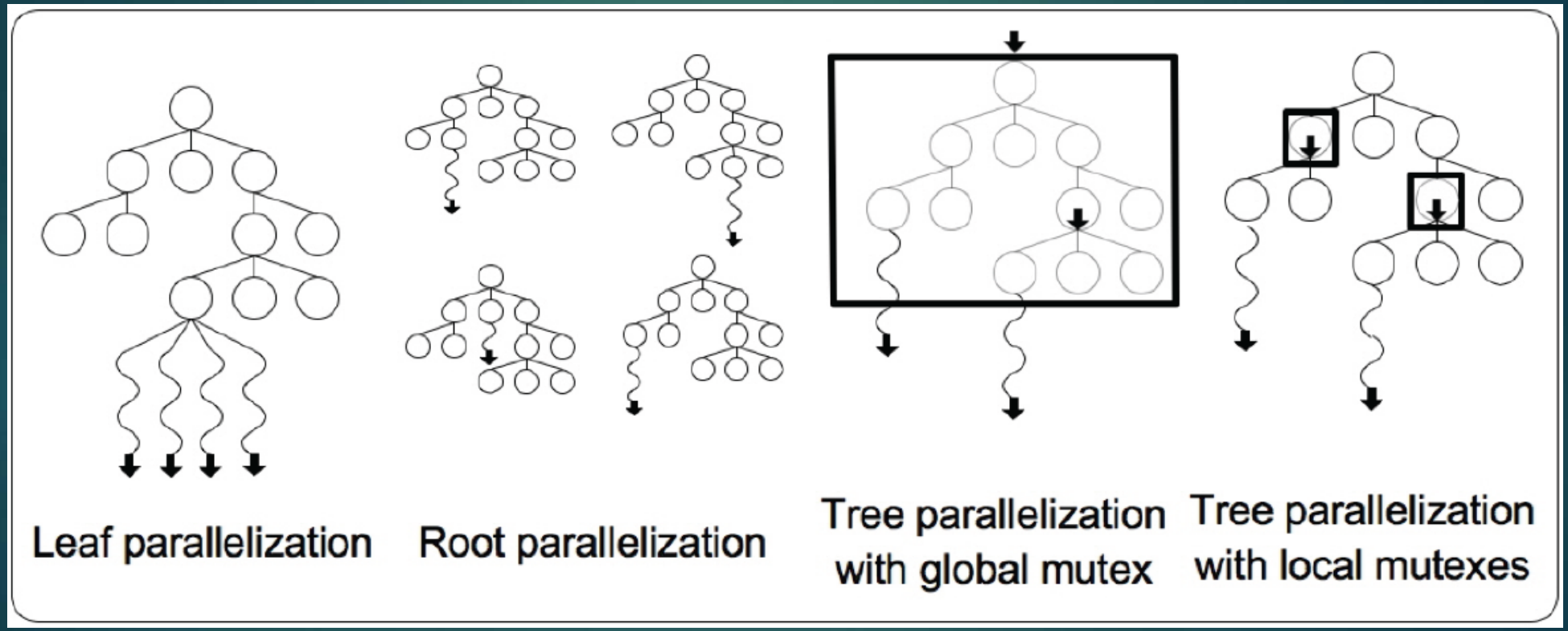


Backpropagation Enhancements

- Decaying Reward
  - Decay factor γ applied as backpropagation occurs
    - similar effect to Weighting Simulation Results by shorter wins having greater effect
- Transposition Table Updates
  - Share update information across nodes that are similar/related

Parallelization

Parallelization

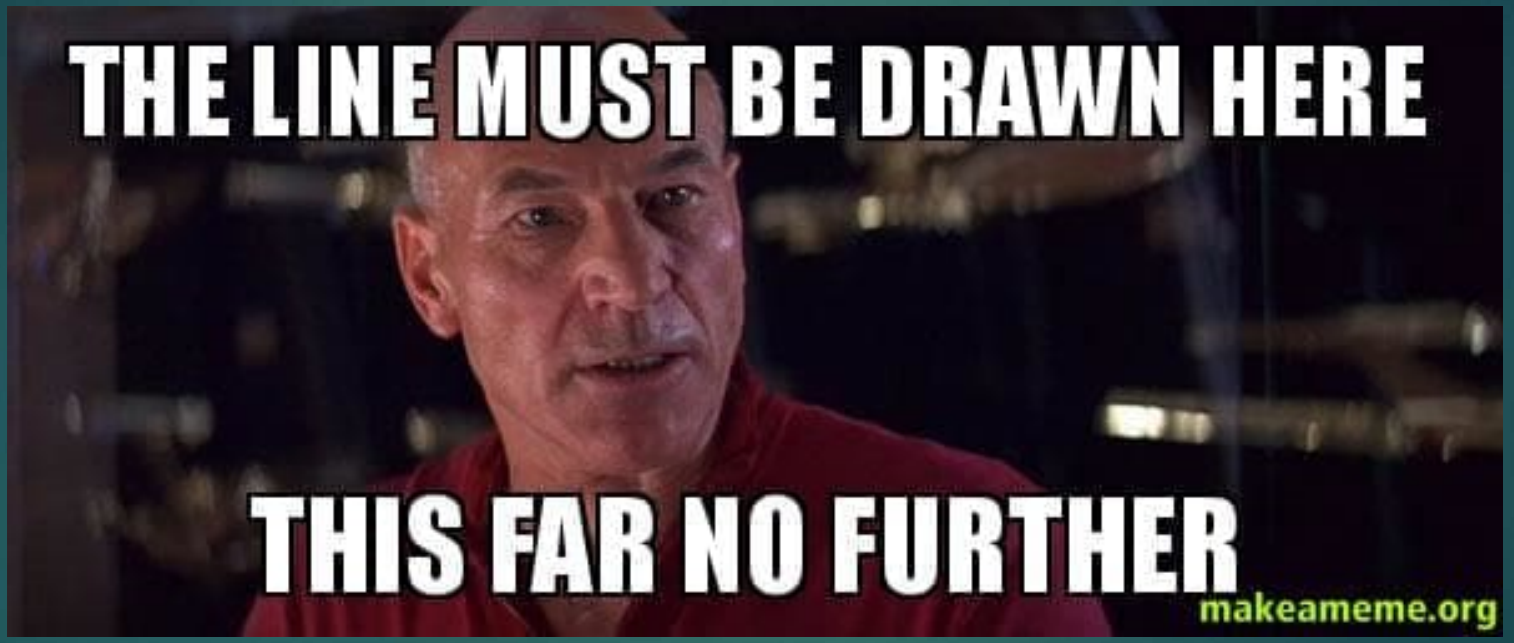


Considerations for Enhancement

Considerations for Enhancement

- Heavily modified ~~ possibly not accurate results
- Computational time vs. Better Play
  - Common metrics:
    - Win rate against particular opponents
    - Elo rating against other opponents
    - Number of iterations per second
    - Memory usage

The End


THE LINE MUST BE DRAWN HERE
THIS FAR NO FURTHER
makeameme.org