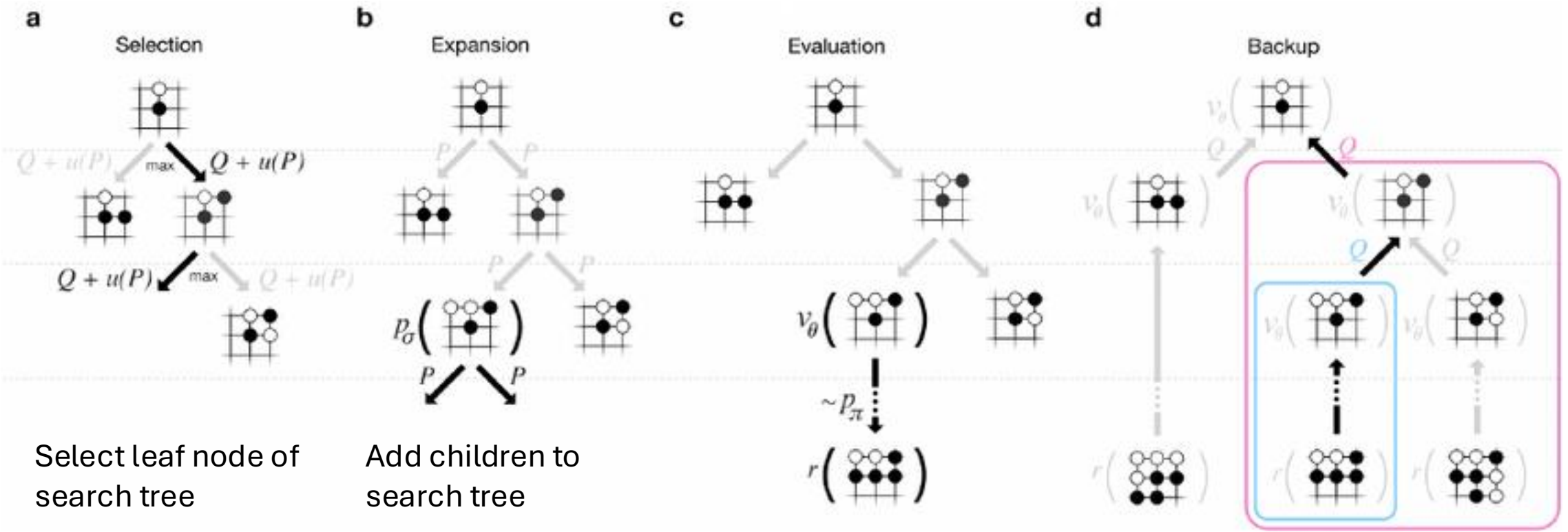
Improving Tree Search Algorithms

Review: MCTS



Select leaf node of search tree

Add children to search tree

Run simulation(s) for each child.

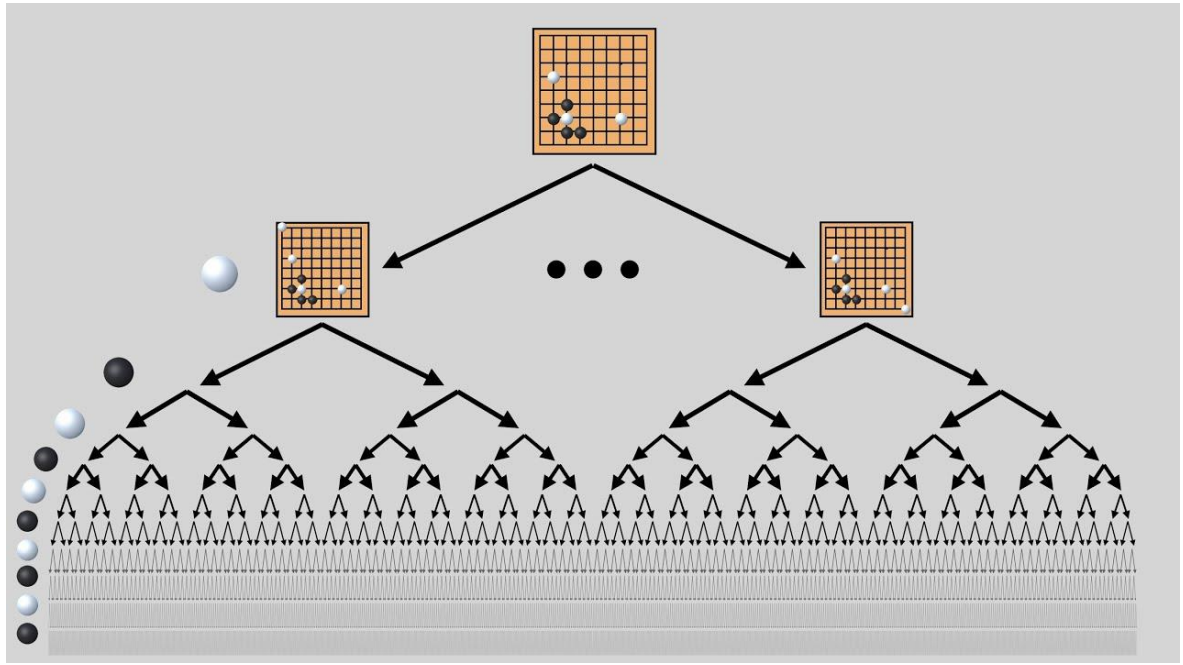
Backup results of simulation to parent nodes

MCTS TTT Visualization

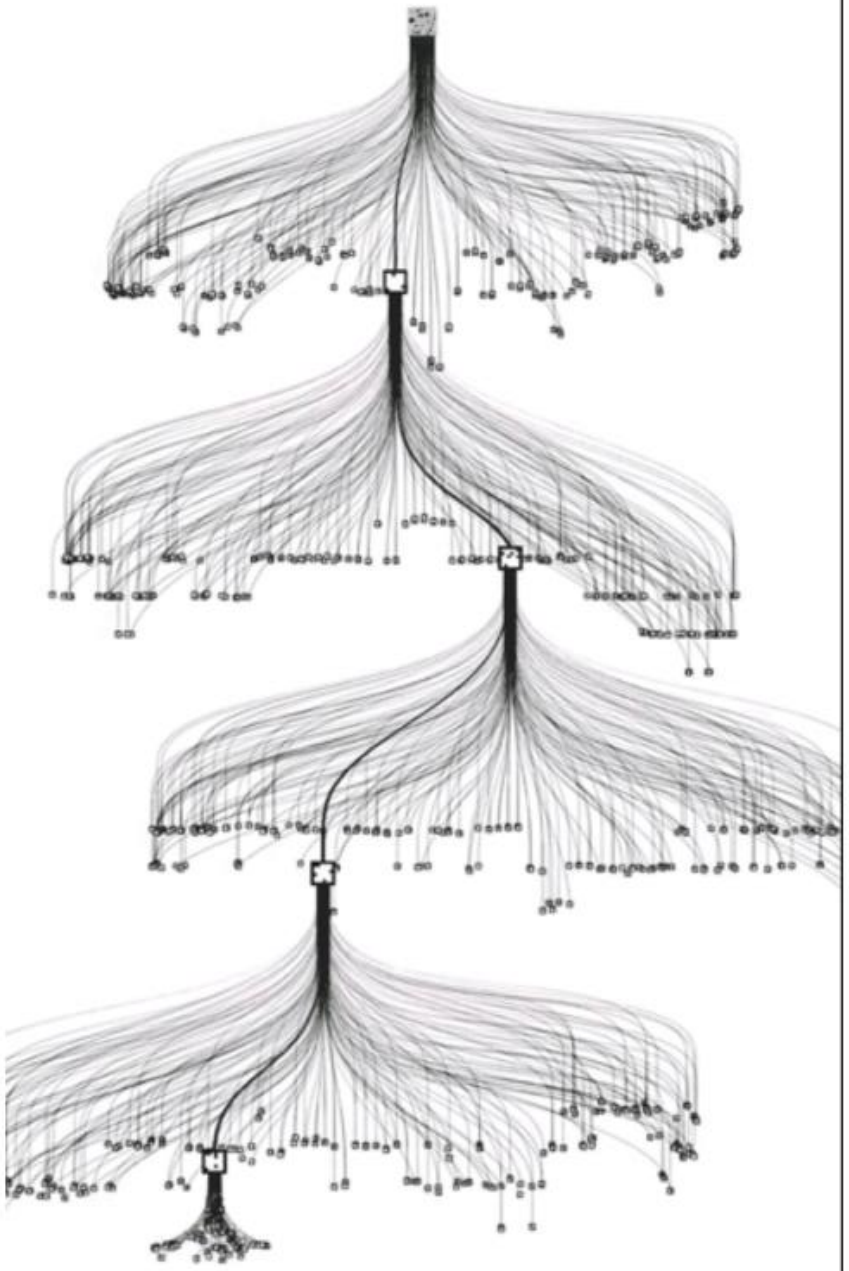
https://vgarciasc.github.io/mcts-viz/

MCTS

Exhaustive Search Tree



MCTS Search Tree



How To Improve MCTS

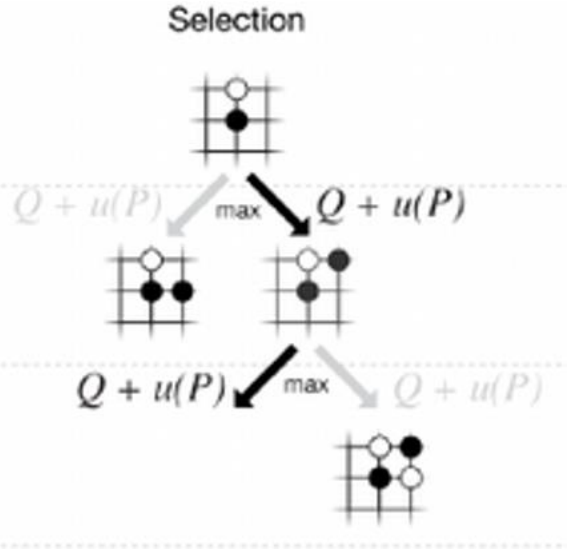
- Monte Carlo Rollout Policy
- Learning (i.e., Alpha-Go)
- Transposition Table
- Timing
- Opening Table
- Re-using search trees

Rollout Policy

- “Vanilla” MCTS uses purely random simulations/rollouts (i.e., agents take random actions)
- The advantage is that we are not reliant on unreliable heuristics
- The disadvantage is that results may be unreliable or not representative of ”good” play
- Adding too much bias (i.e., using a well informed heuristic) can actually make MCTS perform worse…
- But you can make some changes without any loss of performance
  - Don’t take a random action if one of the available actions wins immediately

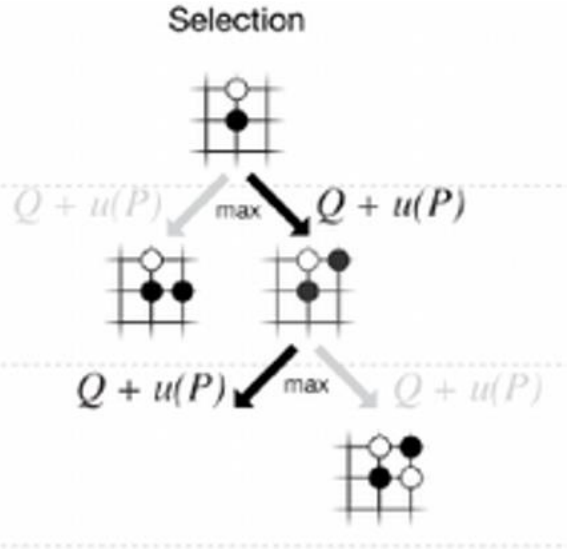
Alpha-Go

- Uses Deep Learning to learn selection policy $\pi_s(a|s)$, value function $V(s)$, and rollout policy $\pi_r(a|s)$
- $\pi_s(a|s)$ controls which action is selected at each level of the tree in the selection step of MCTS
- $\pi_s(a|s)$ is trained with Reinforcement Learning to try and match the policy of UCT



Alpha-Go

- Uses Deep Learning to learn selection policy $\pi_s(a|s)$, value function $V(s)$, and rollout policy $\pi_r(a|s)$
- $V(s)$ is the estimate of which player will win from a given state
- $V(s)$ is trained to match the results of simulations gathered by MCTS



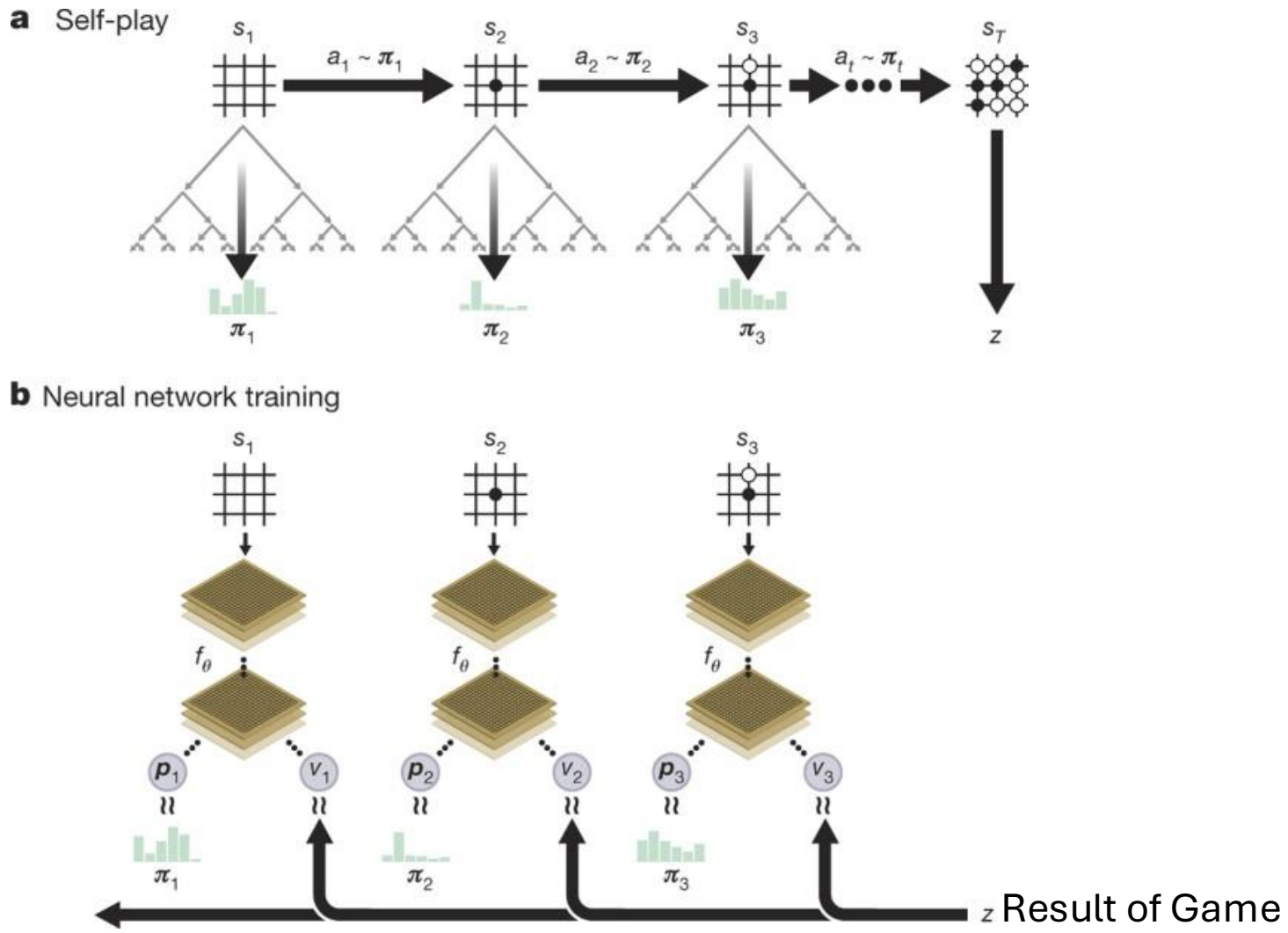
AlphaGo

- Uses Deep Learning to learn selection policy $\pi_s(a|s)$, value function $V(s)$, and rollout policy $\pi_r(a|s)$
- $\pi_r(a|s)$ is the policy used for simulations and is trained to play the best from any state
- $\pi_r(a|s)$ has to be ***fast***… MCTS gets most of its strength from the large number of simulations it can run, not the quality of the simulations
- $\pi_r(a|s)$ in the original AlphaGo paper is actually a shallow network using just a few handcrafted features

AlphaGo

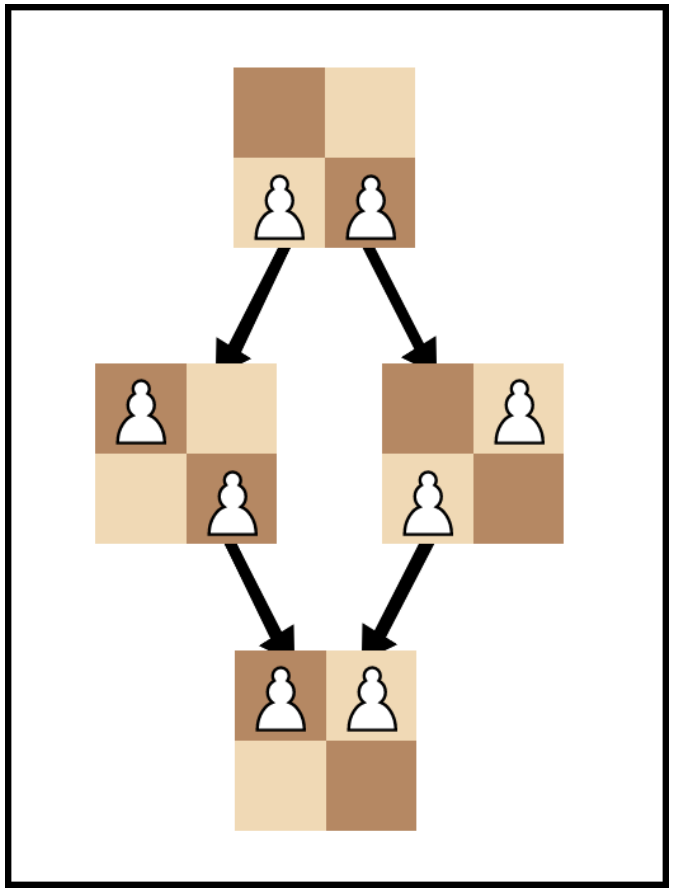
AlphaGo uses *Self-Play* to train its policy and value estimates

Self-play is repeatedly playing games against itself ***many*** times



Transposition Tables

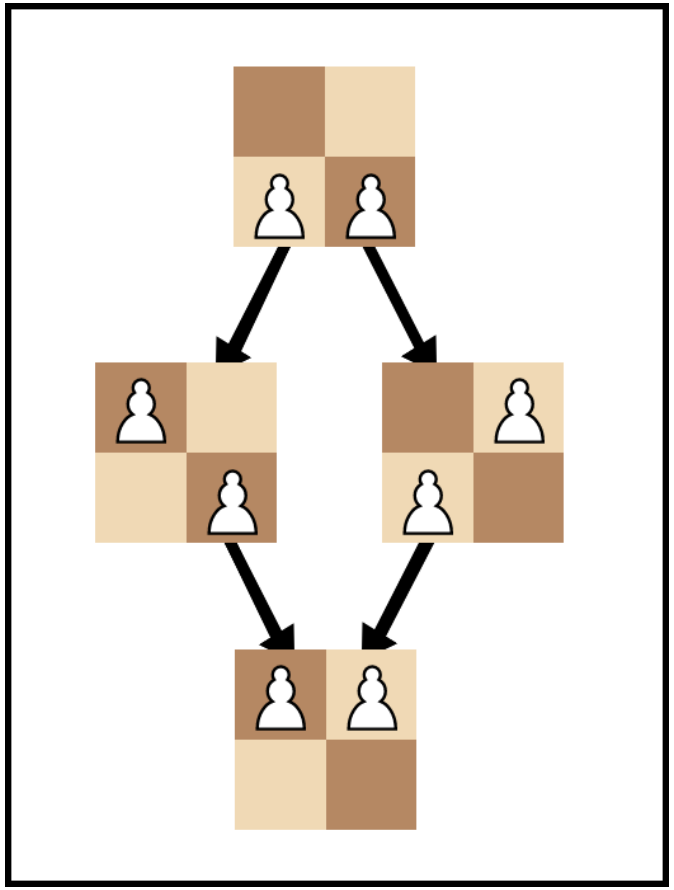
- We typically talk about these problems as search ***trees,*** but that's not exactly true…
- The same state may be reached through different action orders
- A child node in our search problem may have two (or more) parent nodes



Works for any tree search algorithm

Transposition Tables

- We typically talk about these problems as search ***trees,*** but that's not exactly true…
- A Transposition Table is a lookup table (i.e., dictionary or hashmap)
- Keys are the state, and the value is the node in the search tree
- When a state is encountered, check the transposition table to see if it is already in the search tree



Works for any tree search algorithm

Opening Table

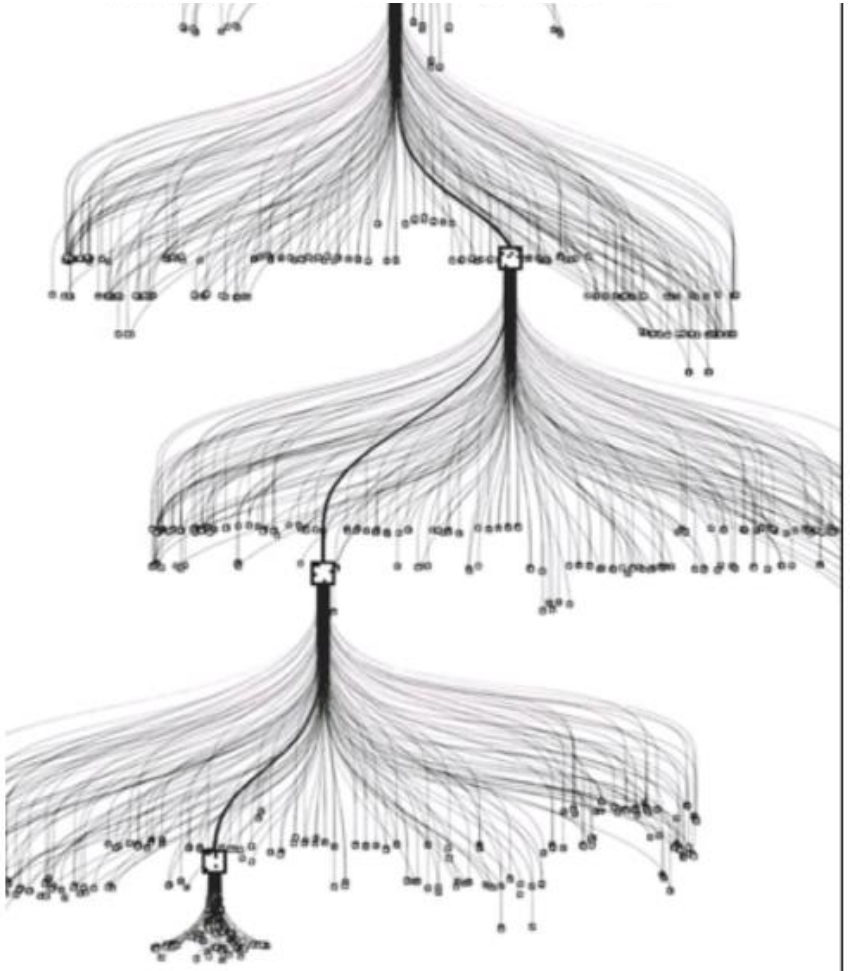
- The first move is, in one sense, the most complicated move.
  - Highest Branching factor, most possibilities, largest search tree
- On the other hand, should you really have to think at all?
- An Opening Table/Book is a (relatively) small set of memorized opening sequences and memorized responses.
- Store states and a memorized action for that state. If a state is in the table, play the action immediately
- When a state comes that is not in the opening book, start running search.

Time Management

- MCTS is an *anytime* algorithm. The longer you run it the better the results will be.
- But at a certain point there are diminishing returns.
- How do you know when you should take the action and move on?
- You can develop a heuristic for when to take an action
  - Equal amount of time per action?
  - Confidence of each possible action from the root node?
  - Many options…

Reusing Search Trees

- Do you need a completely new search tree after you take an action?
- How can you reuse part of the search tree for a previous move?



Pre-registration is coming! Quick Run through of AI courses in the Fall

1420: **Machine Learning**

- Theory and application of ML methods

2470**: Deep Learning**

- Theory and application of training Neural Networks

1640**: AI and Security**

- How secure are ML systems? Can you recover data used to train a model? What if that data should be private?

1952A: **Human-AI Interaction**

- How do we develop the right amount of *trust* in AI? How can we AI with humans in the loop?

2951F: **Learning and Sequential Decision Making**

- Reinforcement Learning (HW 8, 9)

2952C: **Learning with Limited Labeled Data**

- How do we do ML with limited labels or no labels? (HW 6, 7, but with less data)

2952G: **Deep Learning in Genomics**

- Applying DL to problems in Genomics