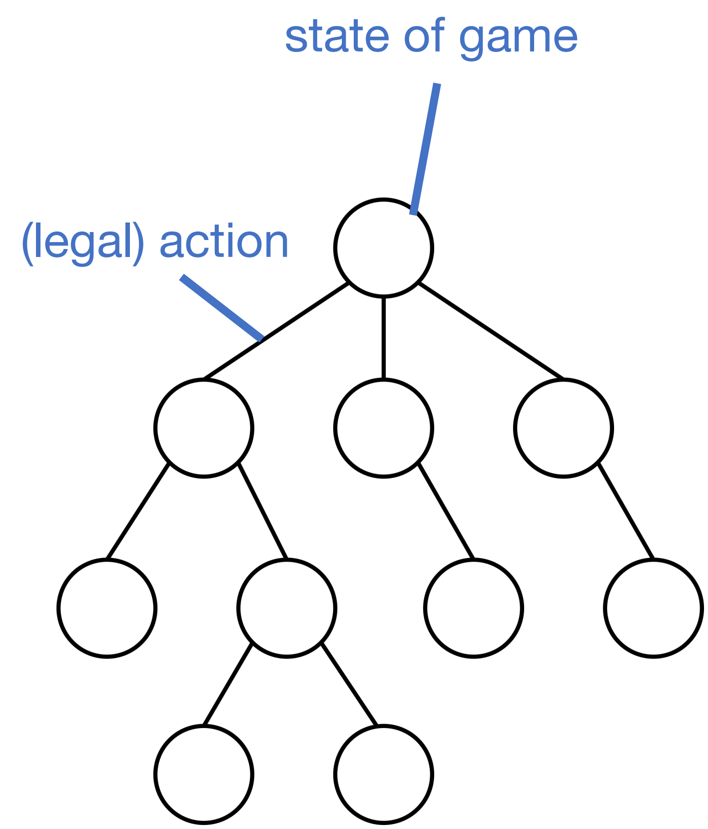
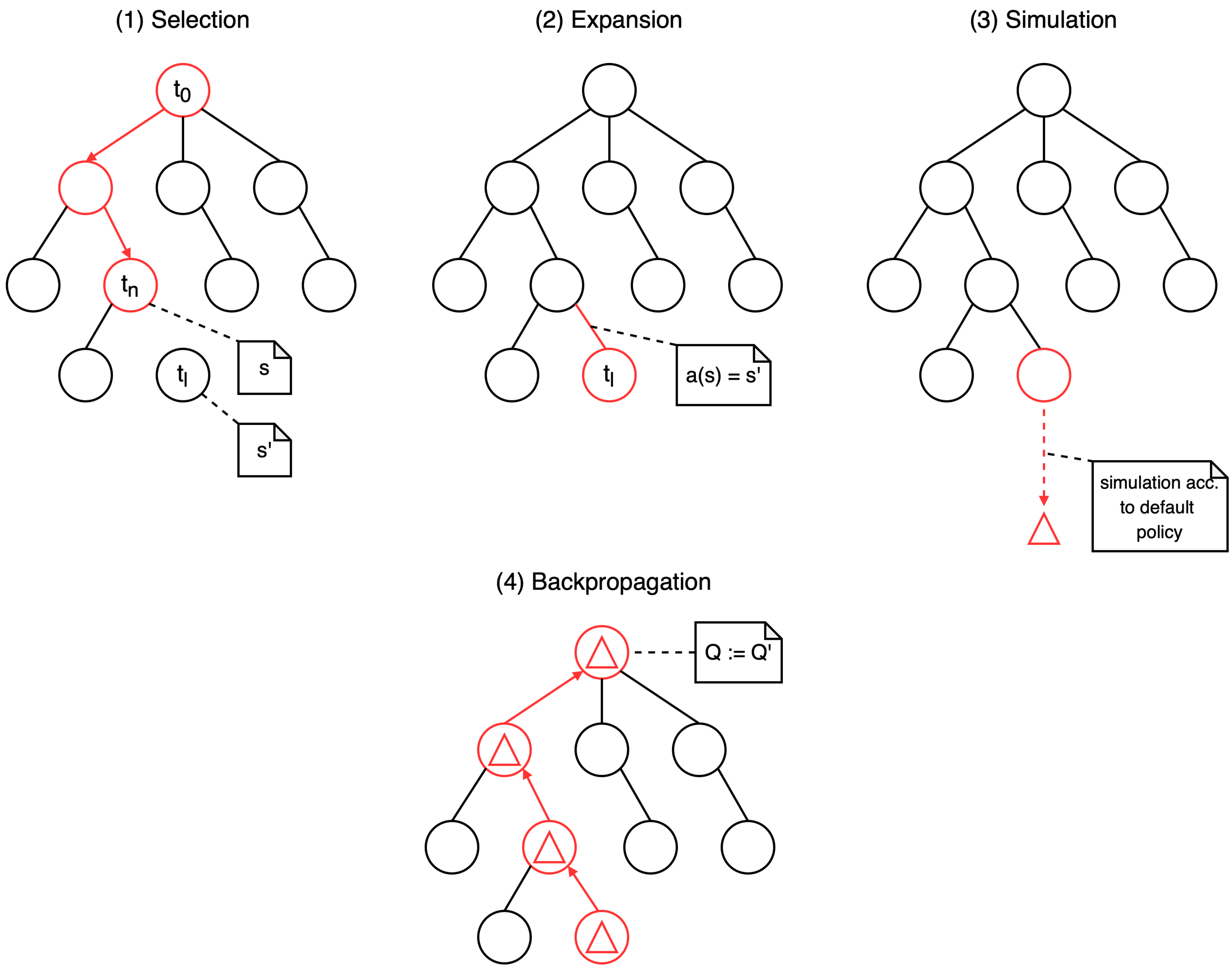
A Survey of Monte Carlo Tree Search Methods

Introduction

- Consider a game-playing agent
  - How to find good/best next move?
- Game represented as a game tree
  - Nodes represent (subset of) states
  - Still too large to check all actions
- Idea: Combine sampling and tentative rewards
  - refine “view” of the current game tree iteratively
- Don’t overthink the term “game”
  - Generally applicable search algorithm

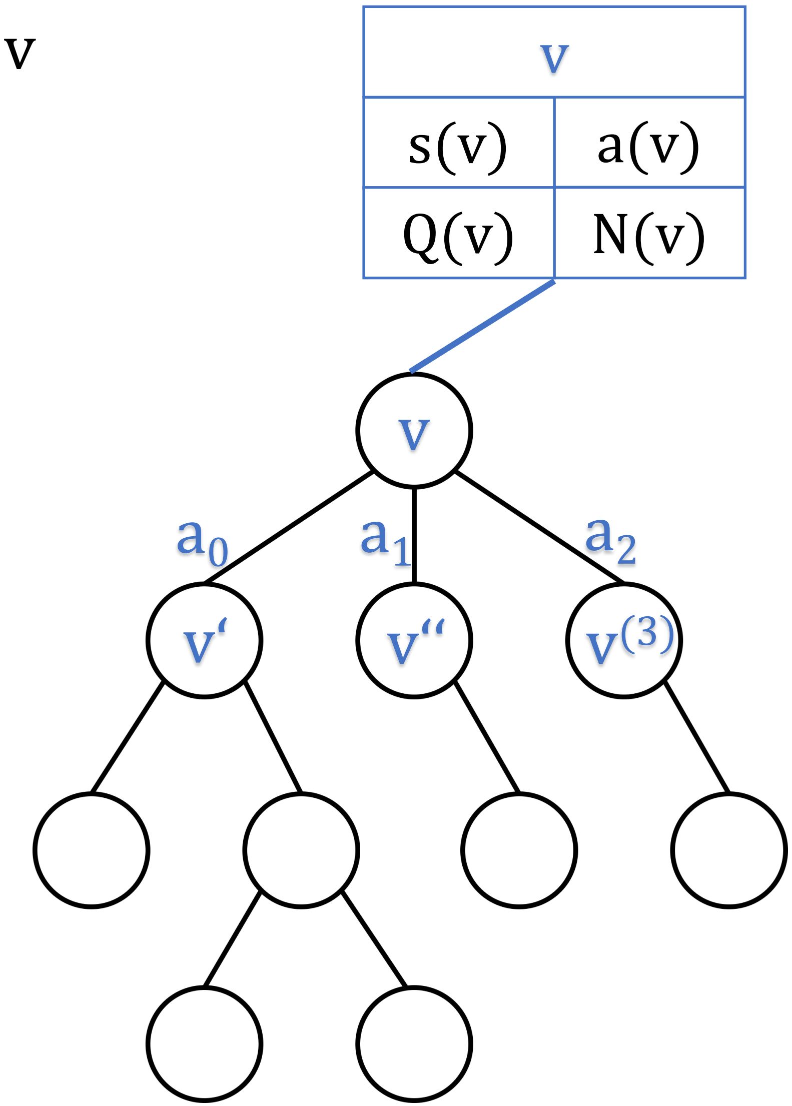


Basics: Monte Carlo Tree Search



Basics: Setting

- $s(v)$ – state s represented by node v
- $a(v)$ – incoming action a of node v
- $Q(v)$ – total simulation reward of v
- $N(v)$ – number of times v has been visited
- Add. data structures possible
  - e.g. $N(a)$



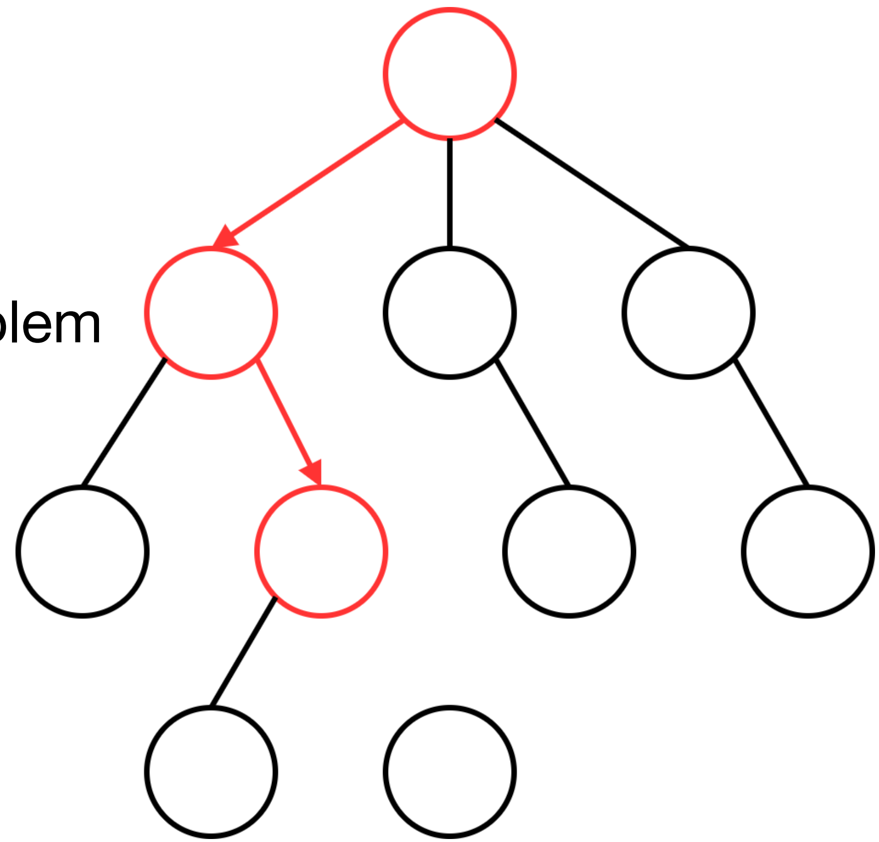
Basics: Monte Carlo Tree Search

- First Step: Selection

- Controlled by the Tree Policy

- Transfer: Multi-Armed Bandit Problem
    - T-times: pick some action
    - collect associated reward
    - maximize total reward

- Later: UCT



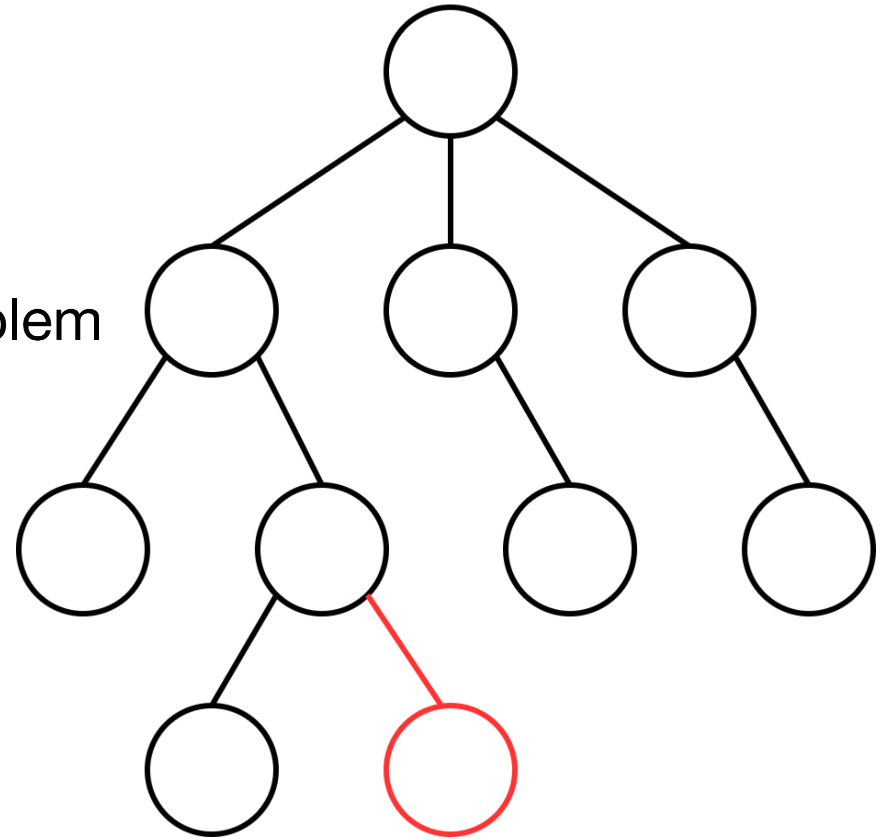
Basics: Monte Carlo Tree Search

- Second Step: Expansion

- Controlled by Tree Policy

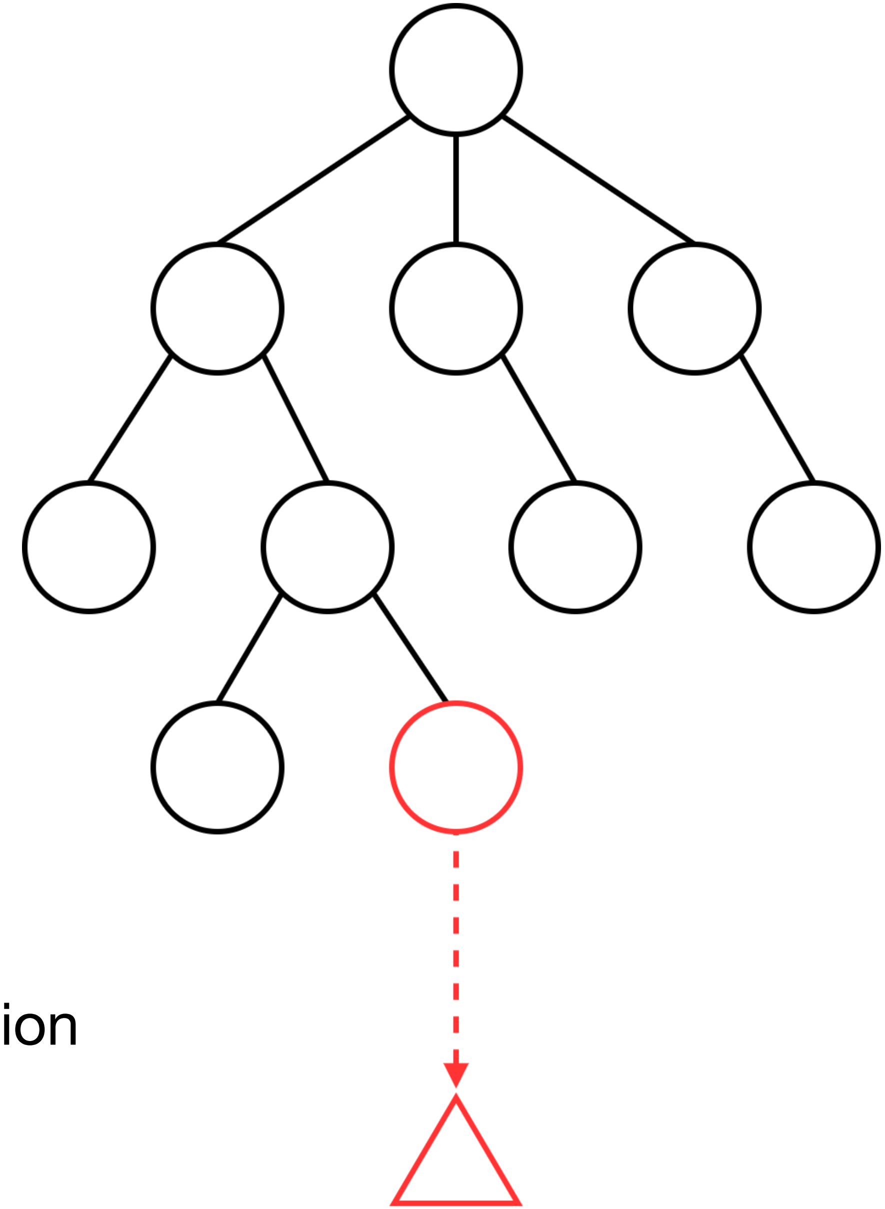
- Transfer: Multi-Armed Bandit Problem
    - T-times: pick some action
    - collect associated reward
    - maximize total reward

- Later: UCT



Basics: Monte Carlo Tree Search

- Third Step: Simulation
- Play out a game
  - i.e. execute actions until a terminal state is reached
  - evaluate result
- Controlled by Default Policy
- Details to consider:
  - speed vs. quality
  - complexity of evaluation function

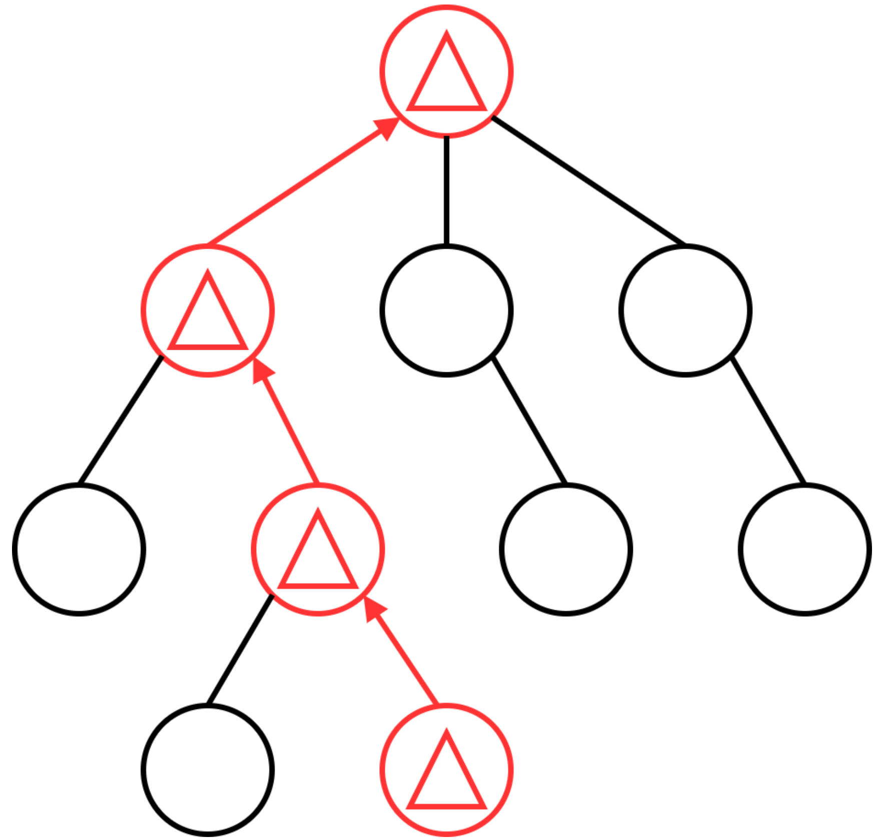


Basics: Monte Carlo Tree Search

- Fourth Step: Backpropagation

- Update values of
  - $Q(v)$
  - $N(v)$
  - ...

- Can be modified by heuristics
  - later - AMAF



Basics: Monte Carlo Tree Search

- About iterations:

- While the computation budget is not exhausted
  1) execute the four steps as many times as possible
  2) Increase tree size and number of evaluated states
  3) Finally, pick the best one

- Time-constraints are crucial
  - real-time, turn-based games

Basics: UCT

- “Upper Confidence Bound for Trees”
- Exploitation: Follow discovered path deemed valuable
- Exploration: Discover new paths

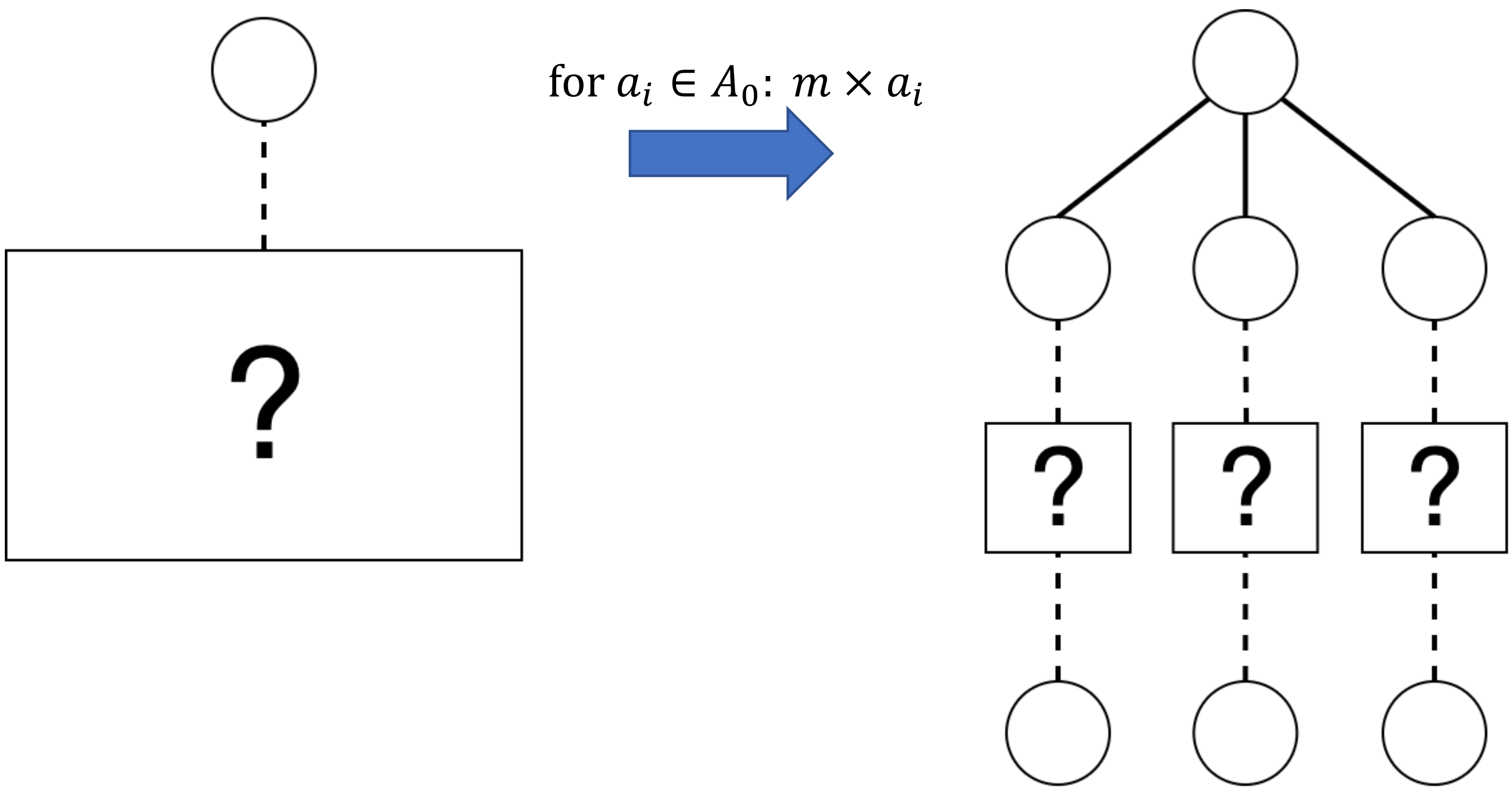
“Exploitation“

$$UCT(v) = \frac{Q(v')}{N(v')} + C_p \cdot \sqrt{\frac{2 \log N(v)}{N(v')}}$$

“Exploration“

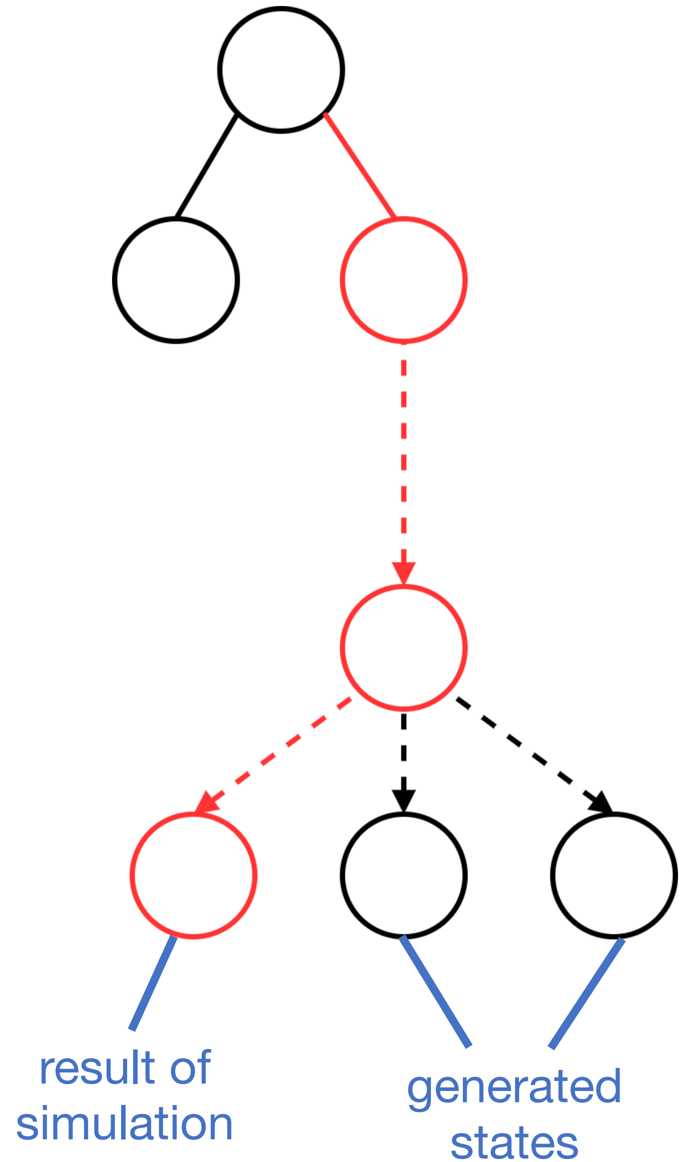
Heuristics: BFS-Tree Initialization

- Before MCTS: execute each available root-action
  - Information gained can guide algorithm



Heuristics: Loss Avoidance

- Ignore negative results when encountering new nodes
- If unvisited node represents a loss, generate neighbors
- Backpropagate only best-case scenario
- Similar (inverse) approach for overly optimistic evaluations



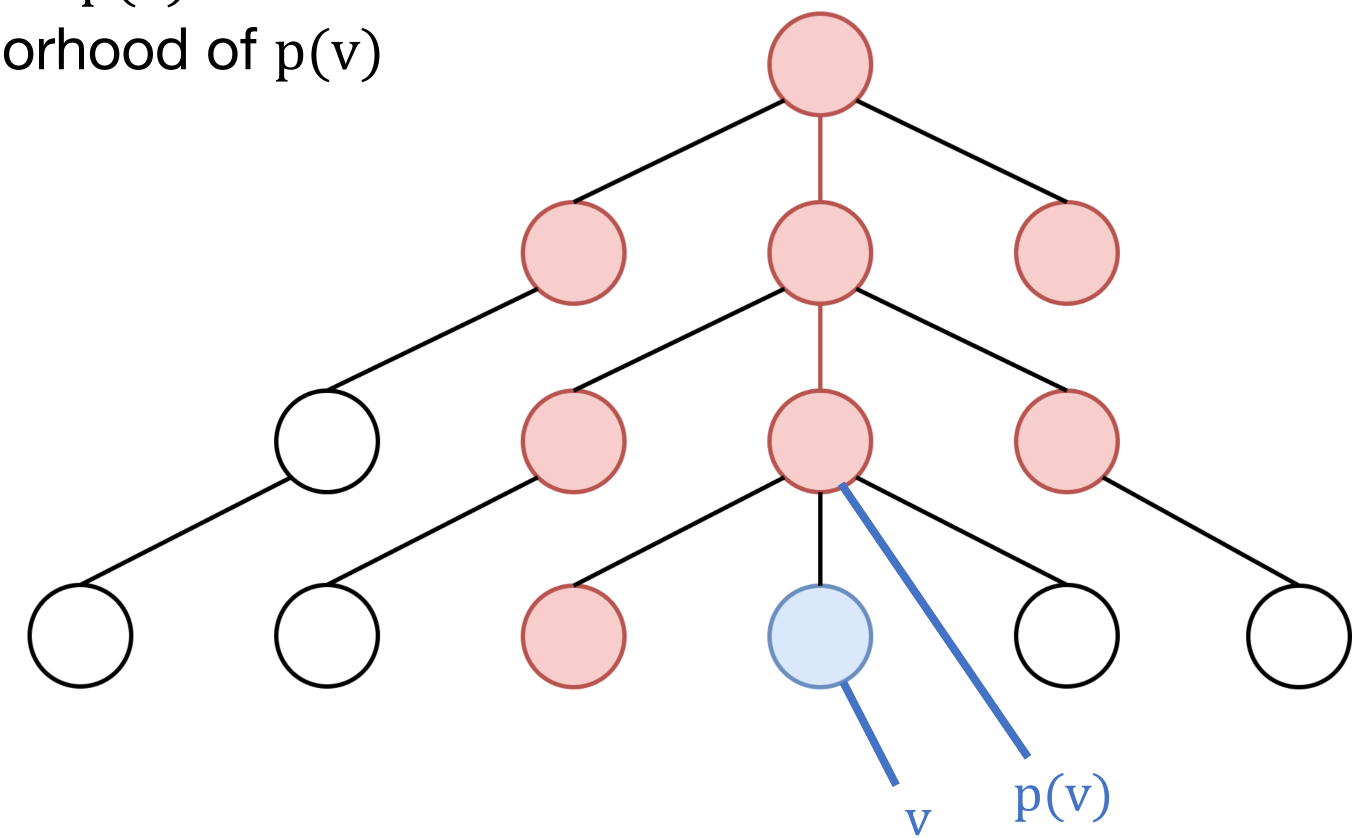
Heuristics: Novelty-based Pruning

- Problem: Many similar states and redundant paths
- Solution: Use a novelty measure $\mathrm{nov}(\mathrm{v})$, all nodes above a threshold are pruned

- Step 1: For node v, define a neighborhood of other nodes
- Step 2: Compare nodes in the neighbohood to v, giving us their similarity or novelty
- Step 3: Prune v if it is too similar to any other nodes

Heuristics: Novelty-based Pruning

- Step 1: the neighboorhood $N(v)$ is the union of
  - the *left* siblings of v
  - the parent of v: $p(v)$
  - *all* siblings of $p(v)$
  - the neighborhood of $p(v)$



Heuristics: Novelty-based Pruning

- Step 2: Compare nodes in the neighbohood to v, giving us their similarity or novelty

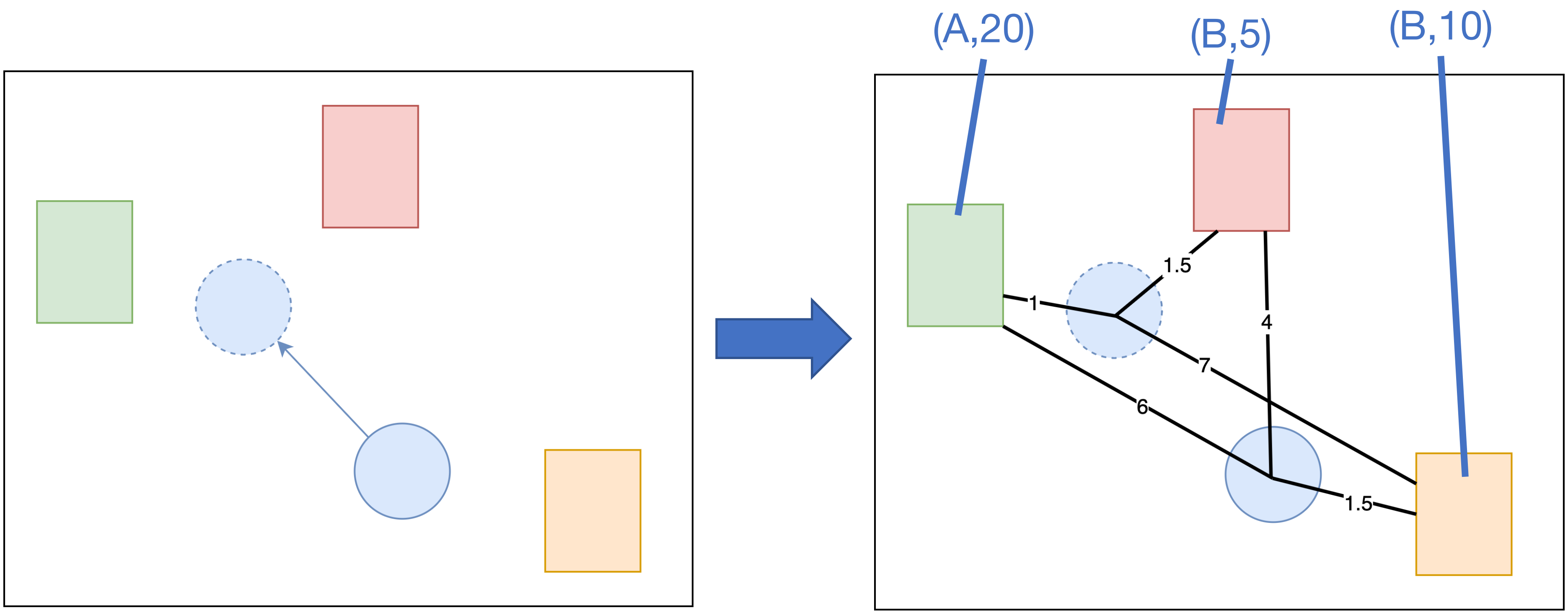
- (Somewhat) more precisely: $nov(v, N(v))$ is
  - the size of the smallest set of predicates that are true for v and false for all other nodes in $\mathrm{N(v)}$

- For example: if $save(v)$ and $\neg\, save(v')$ where $v' \in N(v)$
  - we have $nov\big(v, N(v)\big) = 1$

- Step 3: Prune all nodes with novelty larger than $T > 1$

Heuristics: Knowledge-based Evaluations

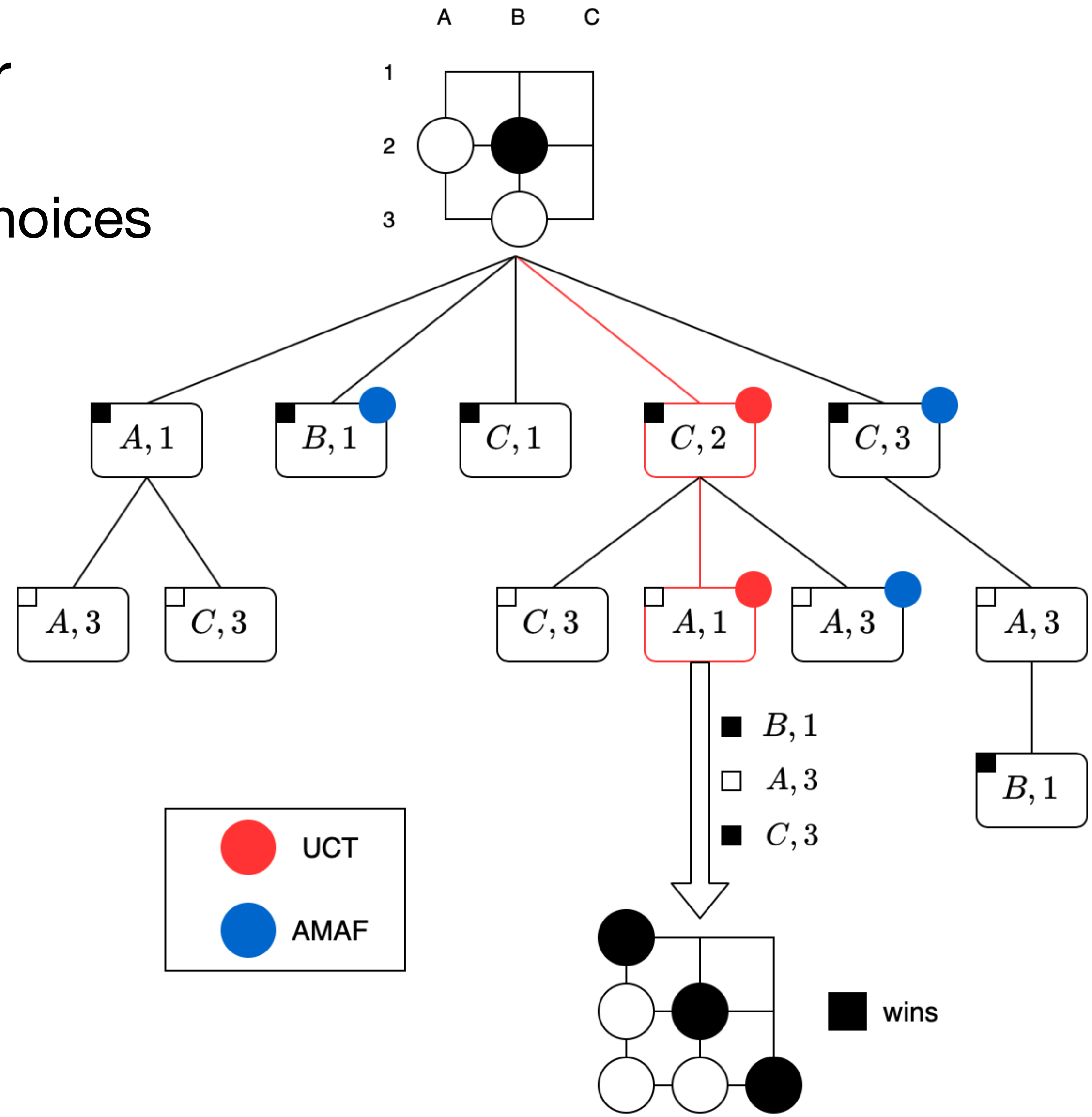
- Terminal state s_T has same value as starting point s_0
  - Is it an improvement?
  - classify nearby objects: (state i, weight $\mathrm{w_i}$)
  - A^*-Algorithm: distance to objects $\mathrm{d(i)}$
  - Add. value of $\sum w_i \cdot (d_0(i) \ - d_{T(i)})$



Heuristics: All Moves As First

- Update more nodes per simulation
  - possible and actual choices

- New value: AMAF-score



Heuristics: All Moves As First

- RAVE – Combine UCT and AMAF
  - $AMAF(v) = \beta(v) \cdot Q(v) + \big(1 - \beta(v)\big) \cdot AMAF(v')$
  - where $\beta(v) = \sqrt{\frac{K}{3 \cdot N(v) + K}}$

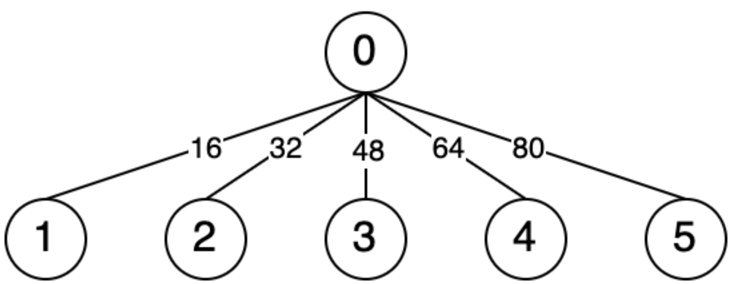
ancestor of v: which one?

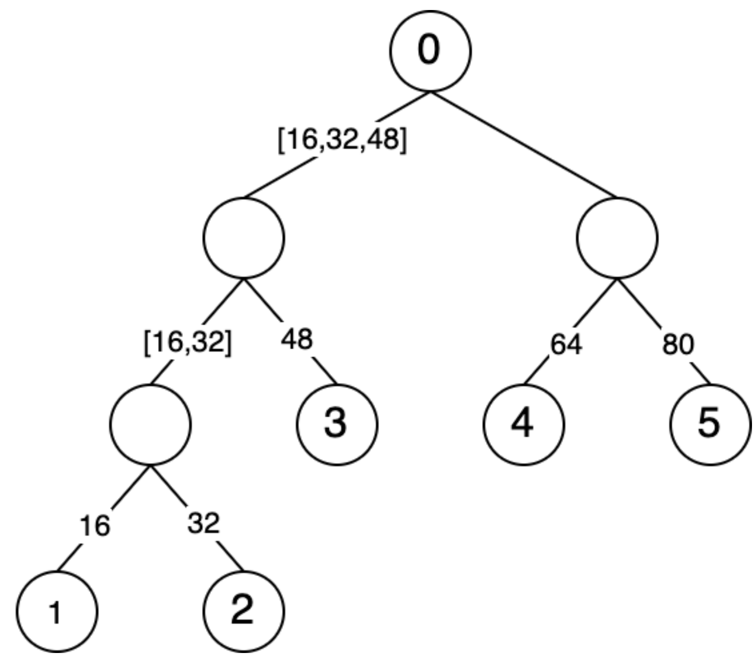
- Equivalence Parameter K: number of simulations where UCT and AMAF have equal weight

- Generalization: GRAVE – modify ancestor selection procedure

Extension: Tree Compression

- Control size and width of tree when there are many paths
  - Binary-search
- Inspired by: Deep Architect
  - (Hyper-)parameters for neural network architectures

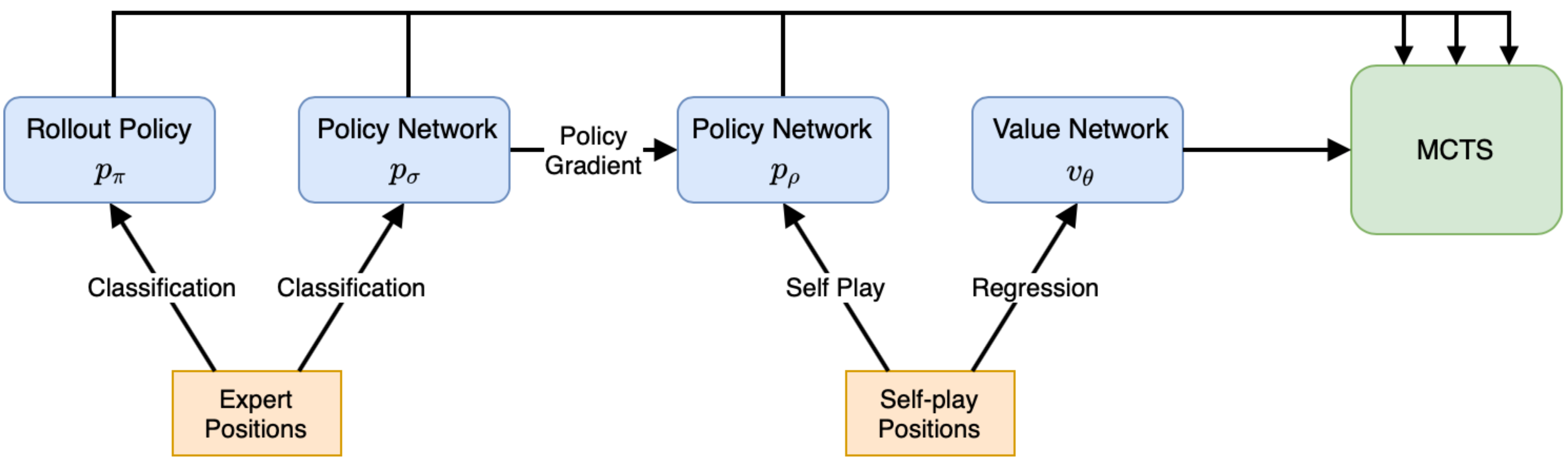




Application: AlphaGo

- ML-System that beat some of the world's greatest players in the game of Go

- MCTS used to train a neural network
  - Lookahead search combines policy networks and value networks



Conclusion

- MCTS – Search algorithm that combines
  - sampling
  - and simulations

- Addresses Exploration-Exploitation tradeoff

- Works well out of the box but can be extended and tailored to specifics of current domain
  - Heuristics
  - Transfer techniques from statistics and probability theory

- Most famous application: games
  - AlphaGo