A Survey of Monte Carlo Tree Search Methods

Outline

1) Introduction
2) Basics via Tic-Tac-Toe
3) Variations
4) Use Case
5) Conclusion

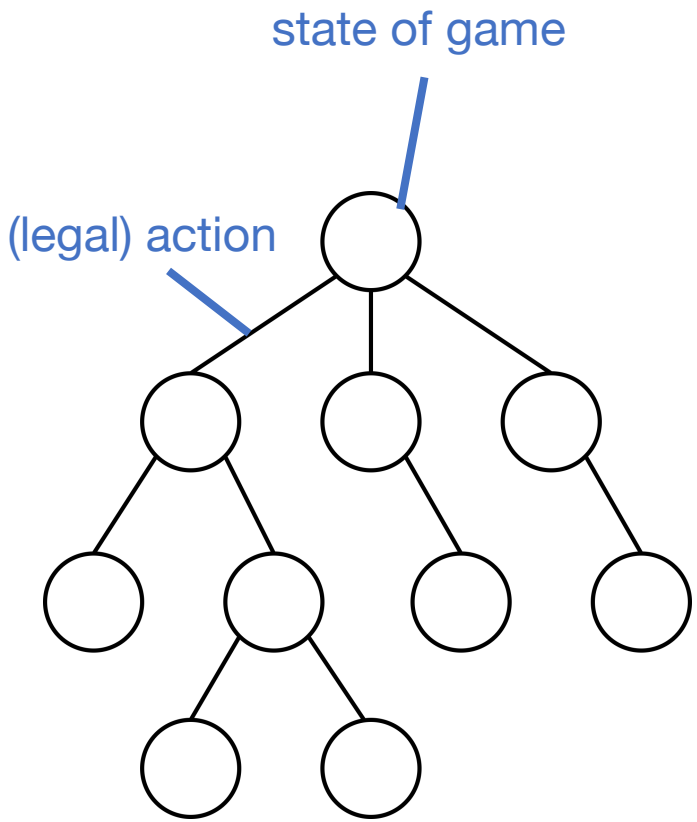
Introduction

- 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

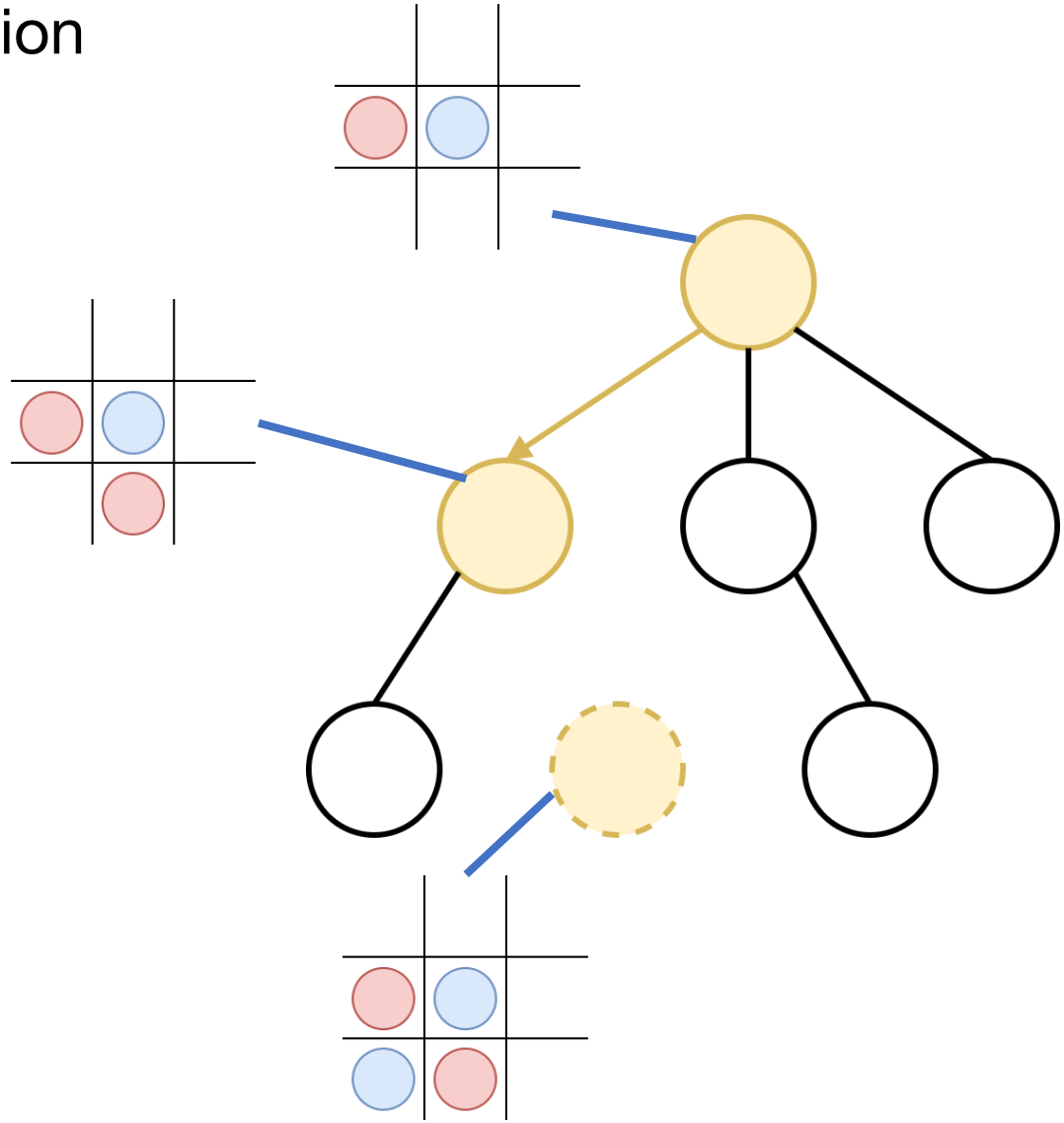
- MCTS-Idea: Combine sampling and tentative rewards
  - refine “view” of the current game tree iteratively

- Don’t overthink the term “game”
  - Generally applicable search algorithm



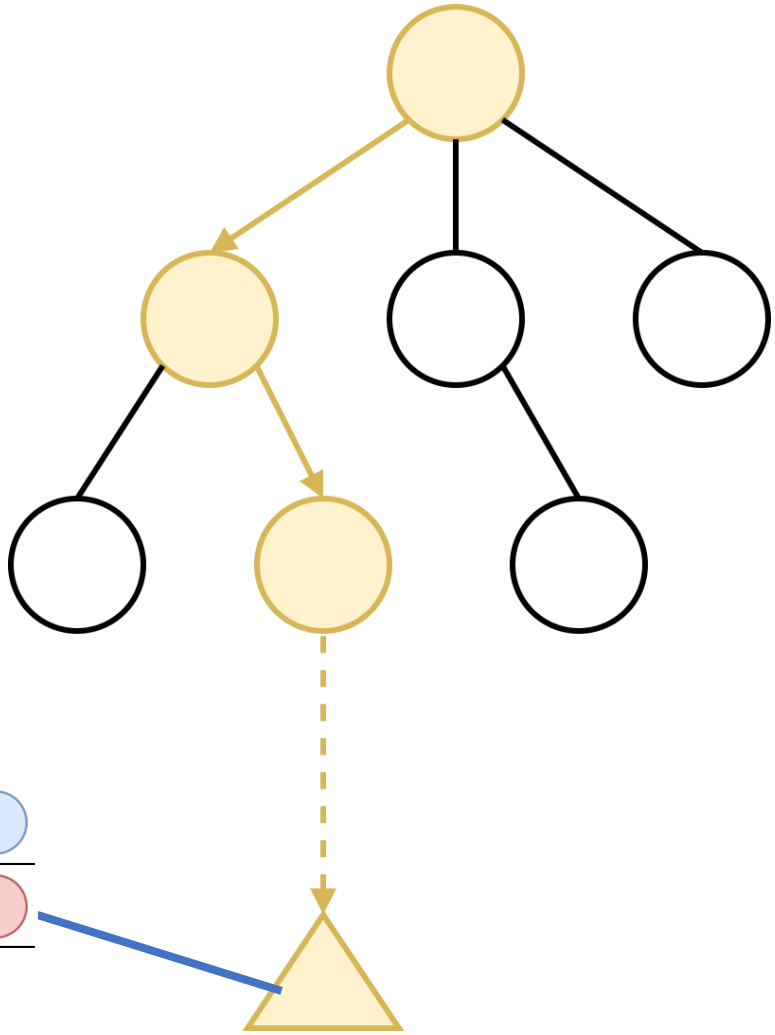
MCTS in Tic-Tac-Toe

- Step 1 – Selection



MCTS in Tic-Tac-Toe

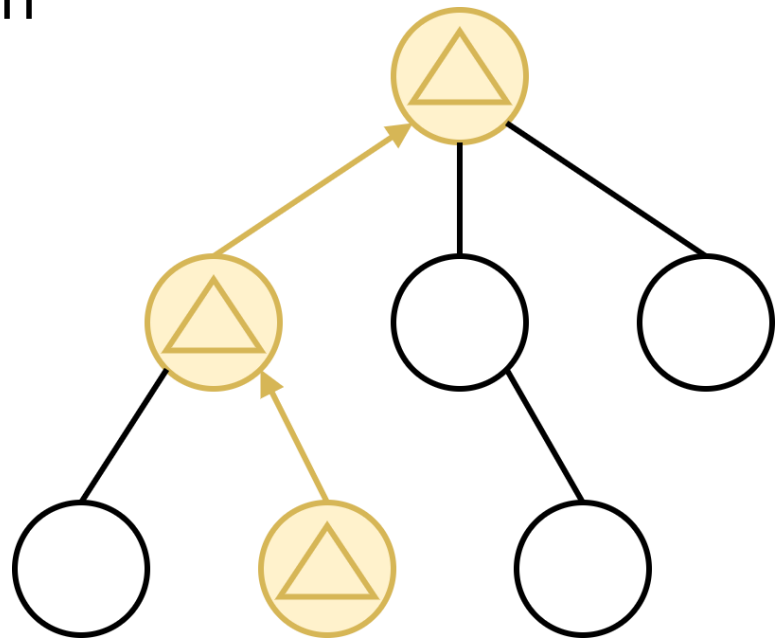
- Step 2 – Expansion
- Step 3 – Simulation



MCTS in Tic-Tac-Toe

- Step 4 – Backpropagation

- … on to the next iteration



MCTS - Variations

- Tic-Tac-Toe:
  - shallow game tree: **short paths**] many simulations
  - narrow game tree: **few paths**]
  - meaningful states: **actually different game states**] useful results

- Need to generalize:
  - **I. Selection Policy**
    - a) UCT [1]
    - b) AMAF and (G)RAVE [2,3,4]
  - **II. State Evaluation**
    - a) Loss Avoidance [5]
    - b) Novelty-based Pruning [5]
    - c) Knowledge-Based Evaluations [5]

[1] Kocsis and Szepesvári: *Bandit based Monte Carlo Planning,* 2006

[2] Gelly and Silver: *Combining online and offline knowledge in UCT,* 2007

[3] Gelly and Silver: *Monte-Carlo tree search and rapid action evaluation in computer Go,* 2011

[4] Cazenave: *Generalized Rapid Action Evaluation,* 2015

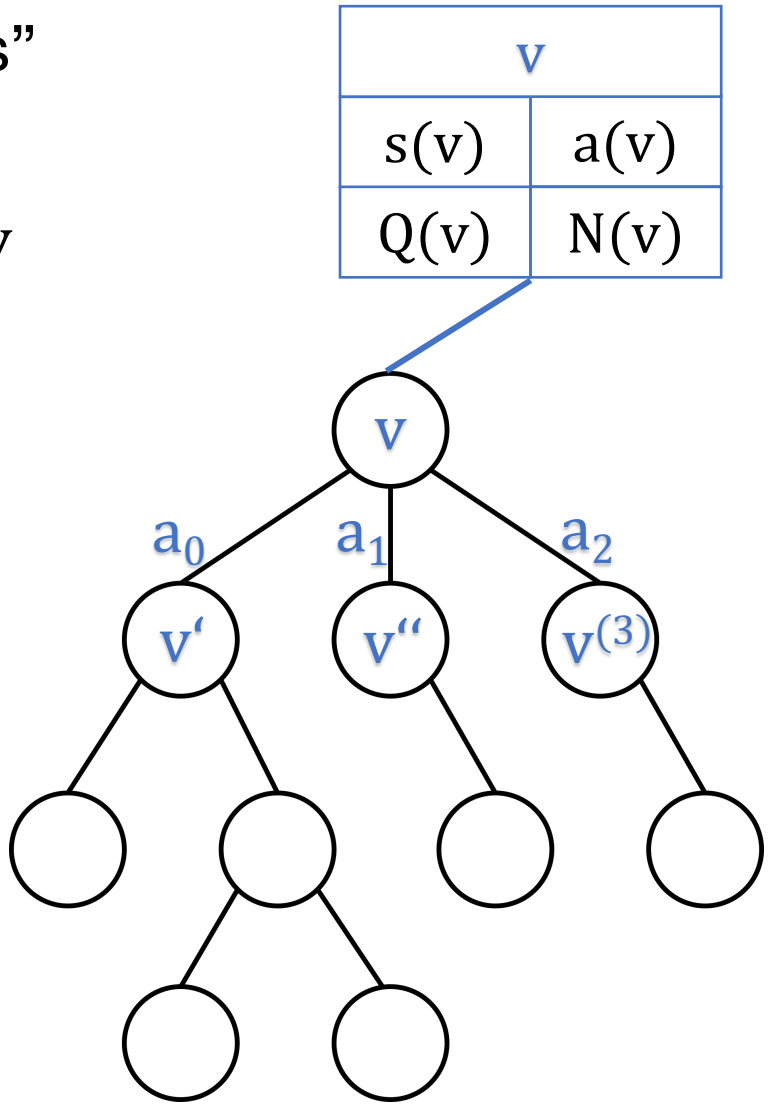
[5] Soemers et al.: *Enhancements for Real-Time Monte-Carlo Tree Search in General Video Game Playing,* 2016

UCT

- “Upper Confidence Bound for Trees”

- $s(v)$ – 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. $N(a)$



UCT

- Child node selection: $v^* = \underset{v' \in v.children}{\arg\max} \; UCT(v)$
- Transfer: Multi-Armed-Bandits

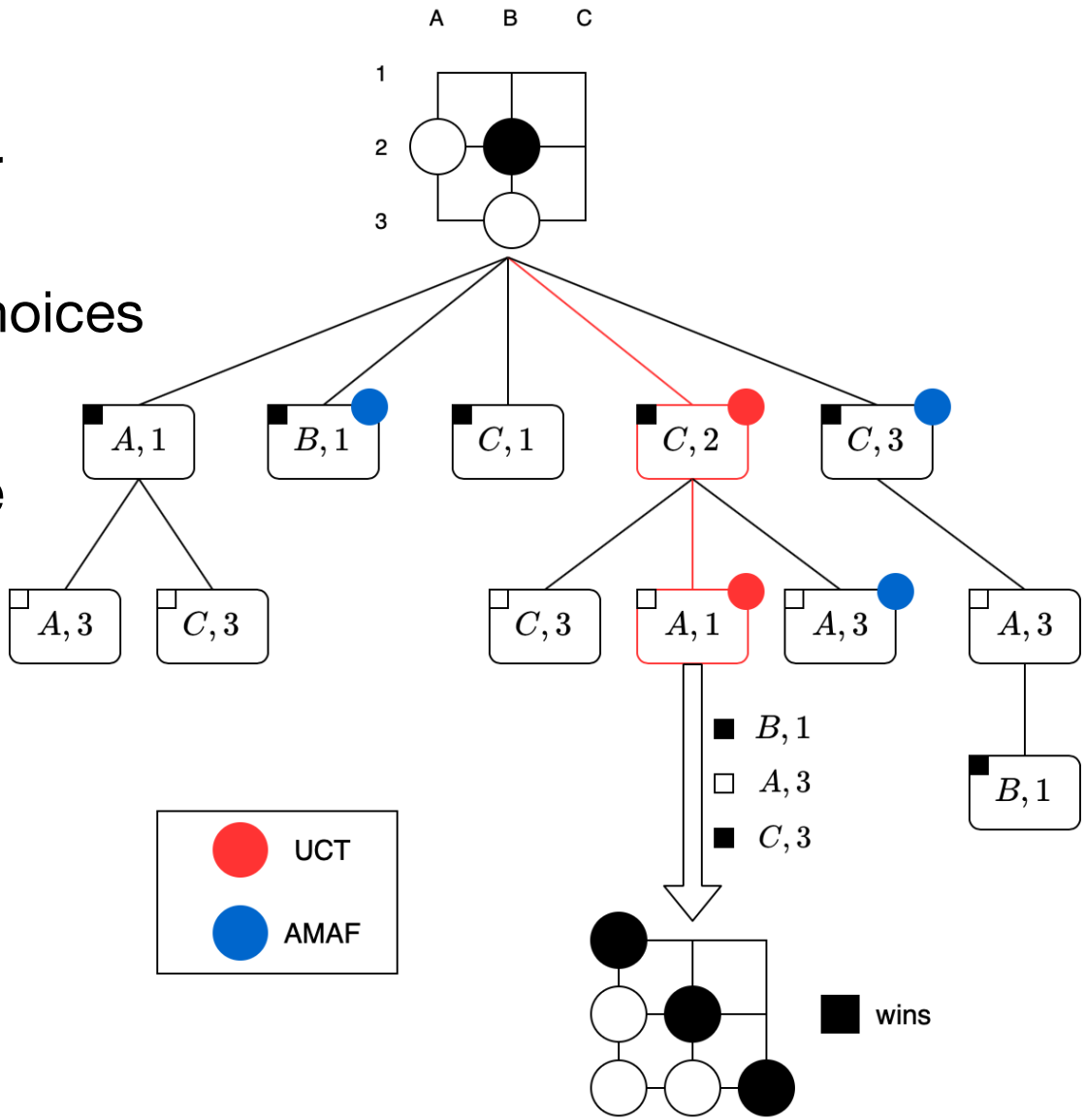
Exploitation – follow paths deemed valuable

Exploration – discover new paths

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

AMAF

- “all moves as first”
- Update more nodes per simulation
  - possible and actual choices
- New value: AMAF-score



AMAF

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

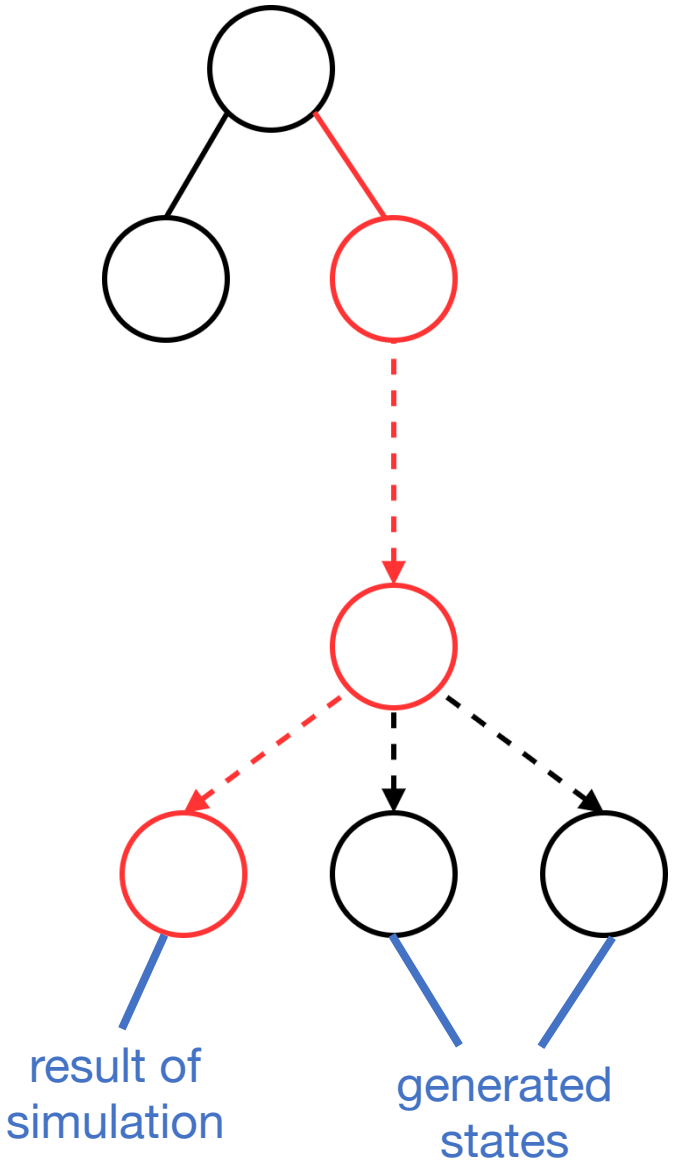
ancestor of v: which one?

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

- Generalization: GRAVE – modify ancestor selection procedure

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



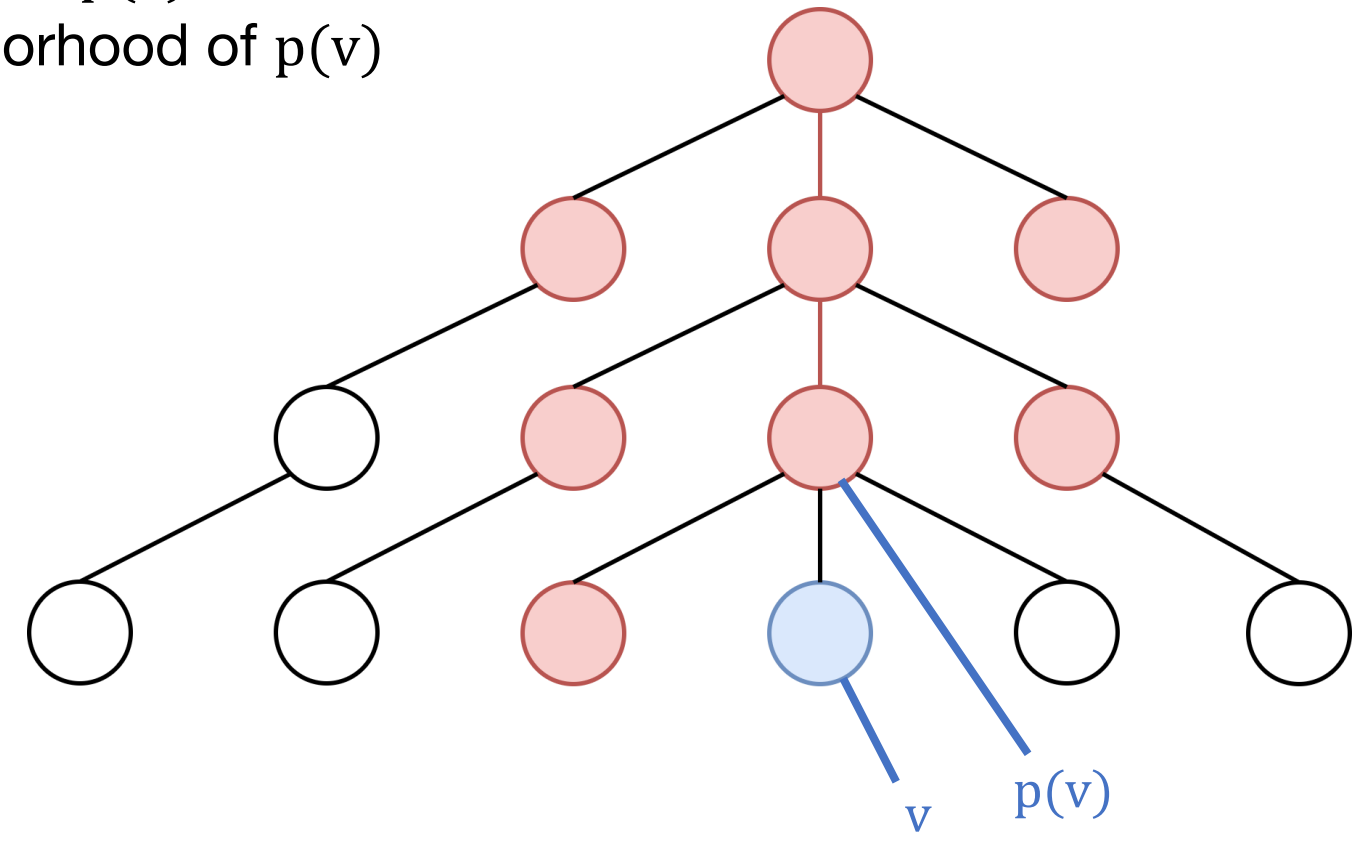
Novelty-based Pruning

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

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

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 $p(v)$



Novelty-based Pruning

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

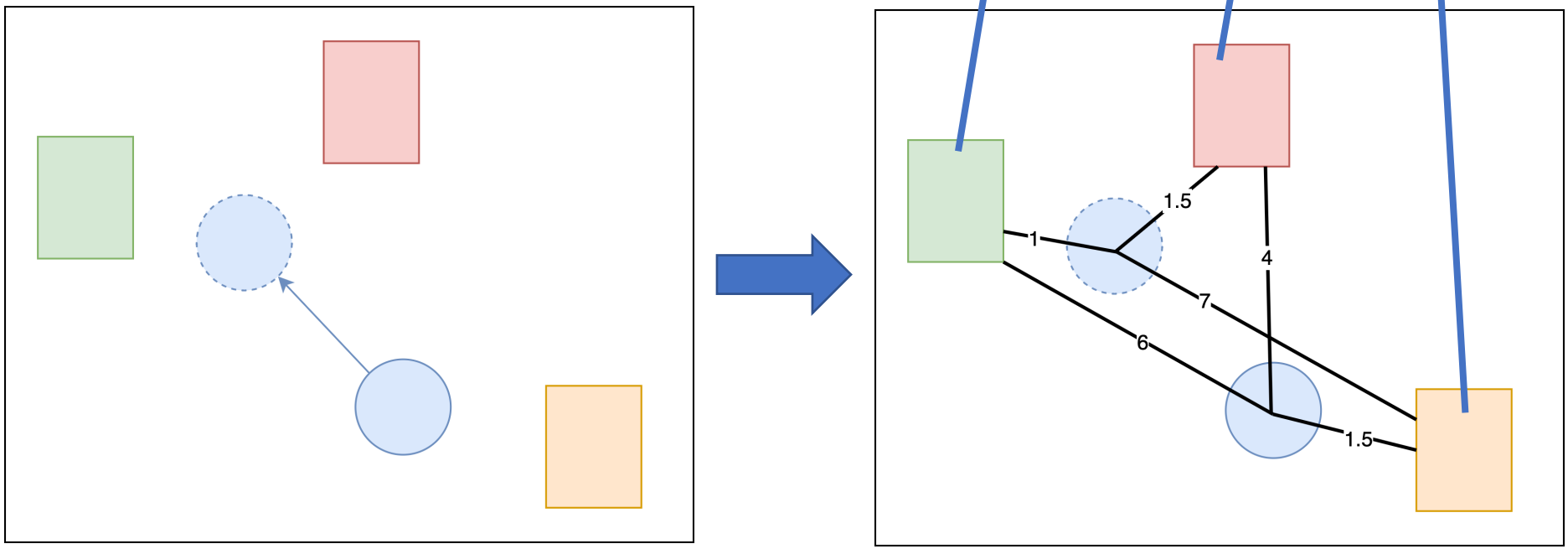
- (Somewhat) more precisely: $nov(v, N(v))$ is
  - the size of the smallest set of predicates that are true for v and false for all other nodes in N(v)

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

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

Knowledge-based Evaluations

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



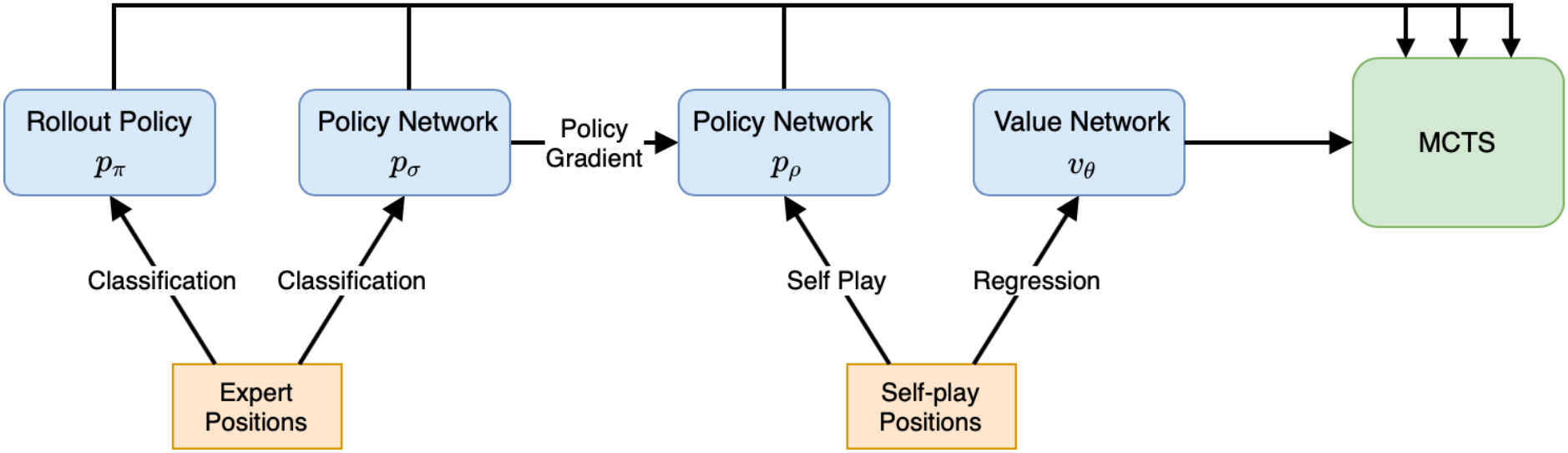
Use Case: AlphaGo

Source: https://cdn.vox-cdn.com/thumbor/nQ5aqmJ_DA5-Dsm82McDmyHr2dM=/0x84:1600x984/1600x900/cdn.vox-cdn.com/uploads/chorus_image/image/49054581/akrales_160307_0970_a_0127.0.0.0.png, (30.07.2020)

Use Case: AlphaGo

- ML-System that beat some of the world's greatest players in the game of Go (2016)

- MCTS used in combination with neural networks
  - Simulation 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