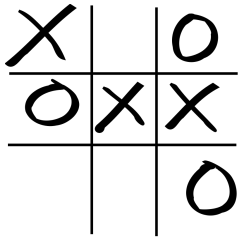
Monte Carlo Tree Search

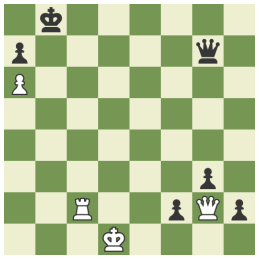
CSC384 – Introduction to Artificial Intelligence

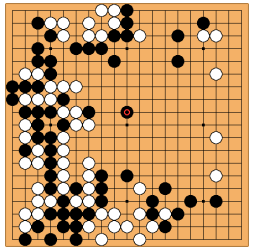
Slides borrowed/adapted from
AAAI-14 Games Tutorial, by Martin Muller:
https://webdocs.cs.ualberta.ca/~mmueller/courses/2014-AAAI-games-tutorial/slides/AAAI-14-Tutorial-Games-5-MCTS.pdf
International Seminar on New Issues in Artificial Intelligence, by Simon. Lucas:
http://scalab.uc3m.es/~seminarios/seminar11/slides/lucas2.pdf
Introduction to Monte Carlo Tree Search, by Jeff Bradberry
https://jeffbradberry.com/posts/2015/09/intro-to-monte-carlo-tree-search/

Conventional Game Tree Search

Perfect-information games

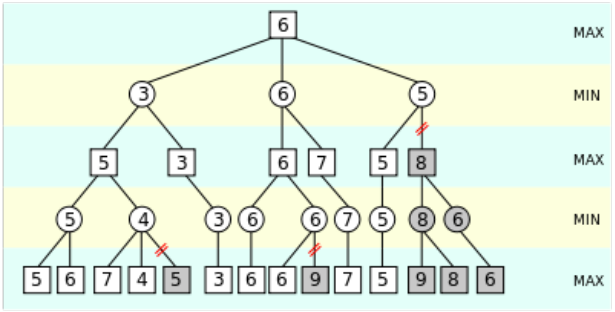






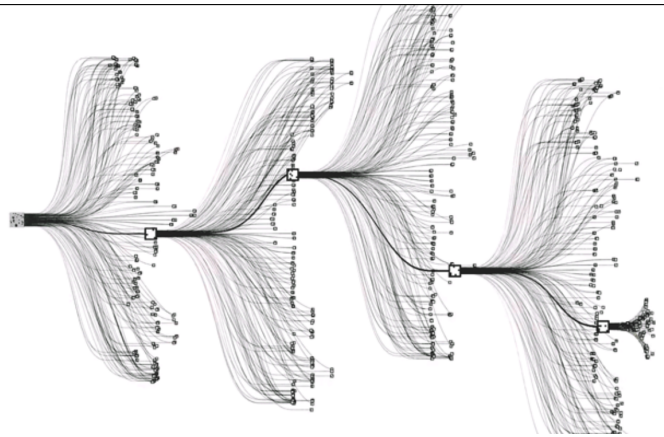
- All aspects of the state are fully observable

Minimax algorithm with alpha-beta pruning



- Effective for
  - Modest branching factor
  - A good heuristic value function is known

Go

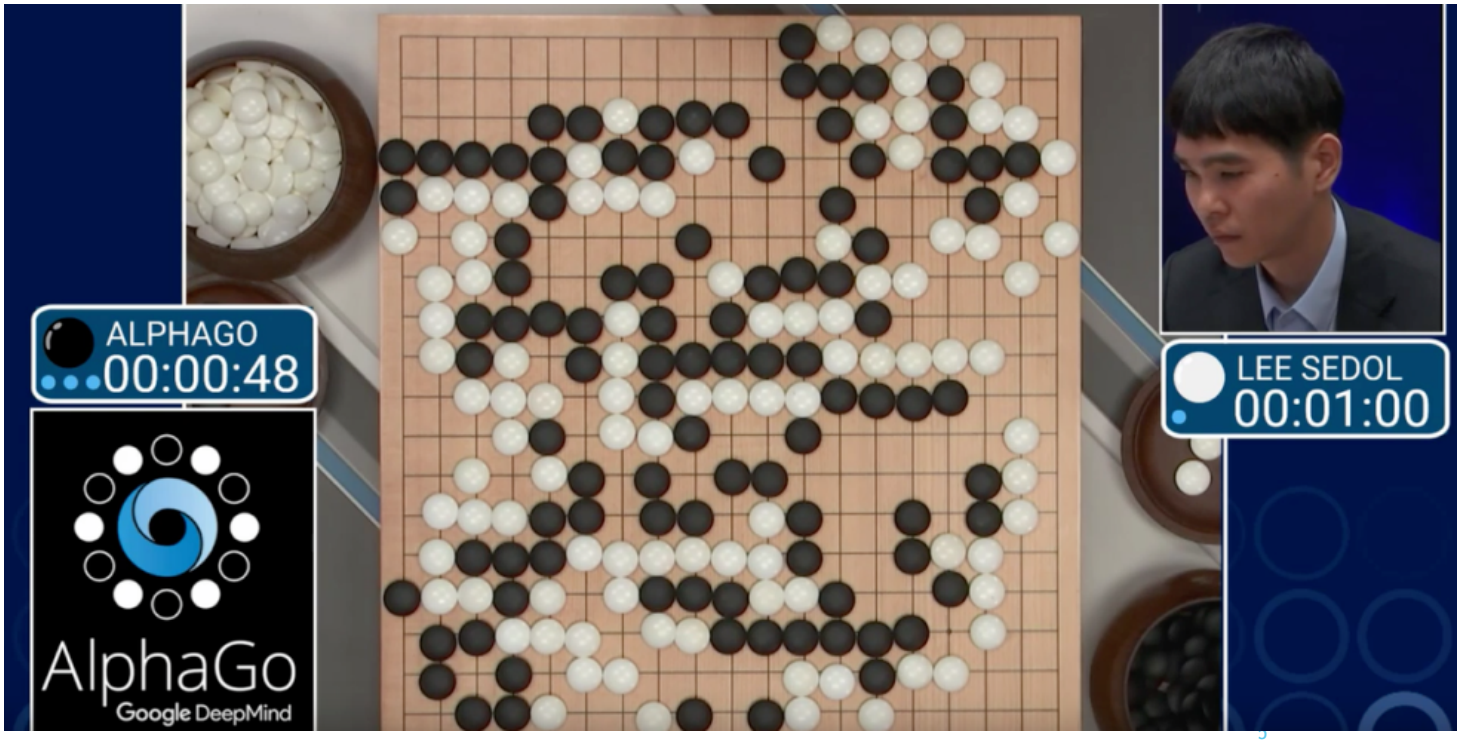


- Level weak to intermediate with alpha-beta
- Branching factor of Go is very large
  - 250 moves on average, game length > 200 moves
  - Order of magnitude greater than the branching factor of 20 for Chess
- Lack of a good evaluation function
  - Too subtle to model: similar looking positions can have completely different outcomes

Monte Carlo!

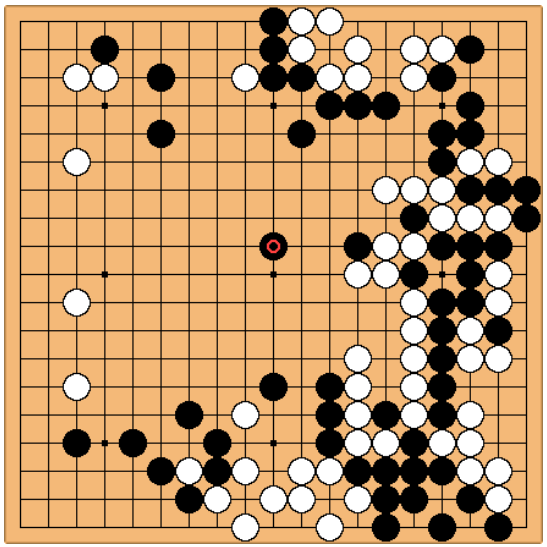
- Revolutionized the world of computer Go
- Application to deterministic games pretty recent (less than 10 years)
- Explosion in interest, applications far beyond games
  - Planning, motion planning, optimization, finance, energy management

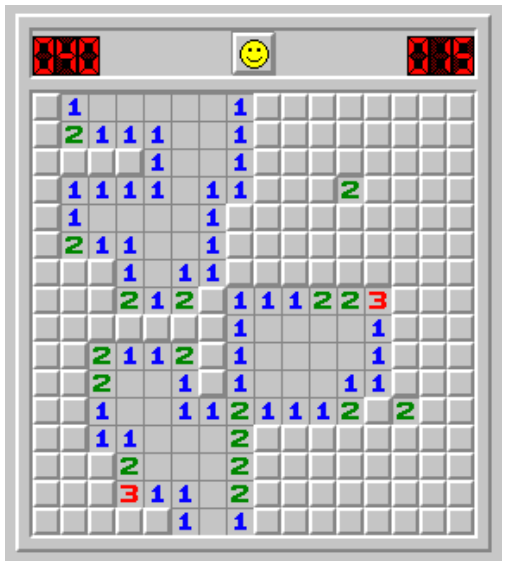
AlphaGo and Monte Carlo Tree Search



MCTS for computer Go and MineSweeper

- Go: deterministic transitions
- MineSweeper: probabilistic transitions





Basic Monte Carlo Simulation

- No evaluation function?

  → Simulate game using random moves

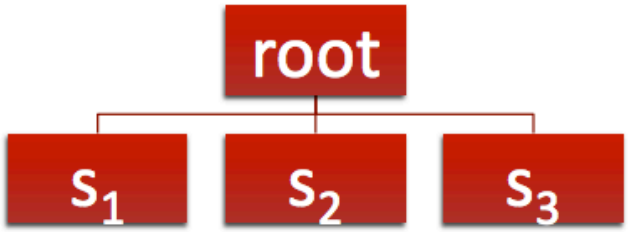
  → Score game at the end, keep winning statistics

  → Play move with best winning percentage

  → Repeat

Use this as the evaluation function, hopefully it will preserve *some* difference between a good position and a bad position

Basic Monte Carlo Search

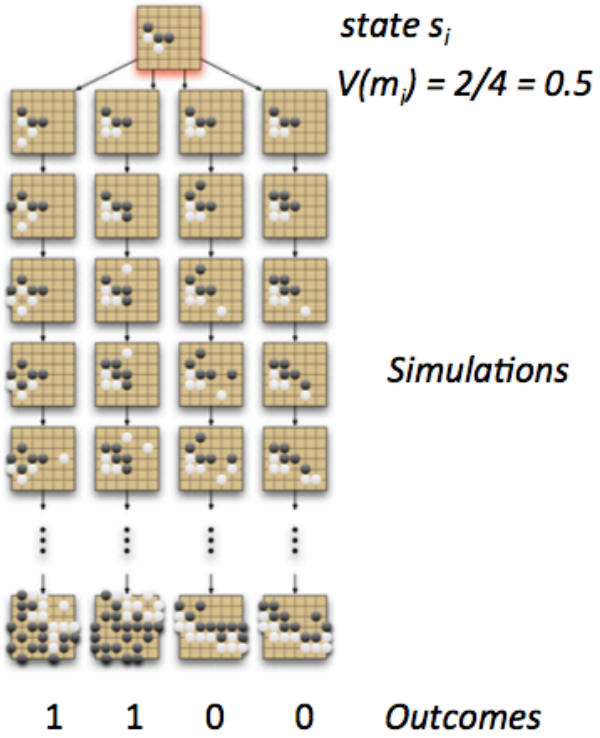


1 ply tree

root = current position

s_1 = state after move m_1

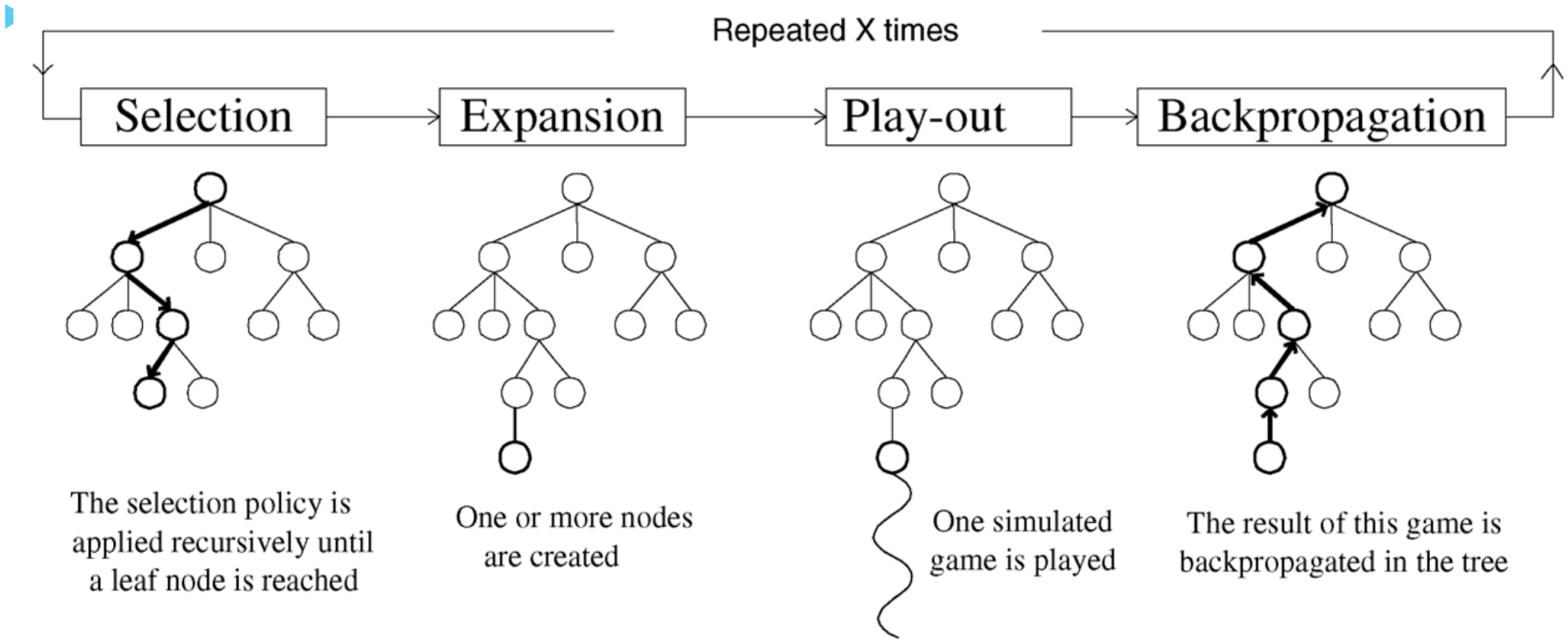
s_2 = ...



Monte Carlo Tree Search

- MCTS builds a *statistics tree* (detailing value of nodes) that partially maps onto the entire game tree
- Statistics tree guides the AI to focus on **most interesting nodes** in the game tree
- Value of nodes determined by simulations

Monte Carlo Tree Search



Real-Time Monte Carlo Tree Search in Ms Pac-Man. Pepels et al. IEEE Transactions on Computational Intelligence and AI in Games (

Naïve Approach

- Use simulations directly as an evaluation function for αß
- Problems
  - Single simulation is very noisy, only 0/1 signal
  - Running many simulations for one evaluation is very slow
  - Example:
    - Typical speed of chess programs **1 million** eval/sec
    - Go: 1 million moves/sec, 400 moves/simulation, 100 simulations/eval = **25** eval/sec
- Result: Monte Carlo was ignored for over 10 years in Go

Monte Carlo Tree Search

- Use results of simulations to guide growth of the game tree

- **Exploitation:** focus on promising moves
- **Exploration**: focus on moves where uncertainty about evaluation is high

- Seems like two contradictory goals
  - Theory of **bandits** can help

Multi-Armed Bandit Problem

- Assumptions:
  - Choice of several arms
  - Each arm pull is independent of other pulls
  - Each arm has fixed, unknown average payoff
- Which arm has the best average payoff?

Consider a row of three slot machines

A

P(A wins) = 60%

B

P(B wins) = 55%

C

P(C wins) = 40%

- Each pull of an arm is either
  - A win(payoff 1) or
  - A loss (payoff 0)
- A is the best arm → **but we don't know that**

Exploration vs. Exploitation

- Want to **explore** all arms
  - BUT, if we explore too much, may sacrifice reward we could have gotten
- Want to **exploit** promising arms more often
  - BUT, if we exploit too much, can get stuck with sub-optimal values

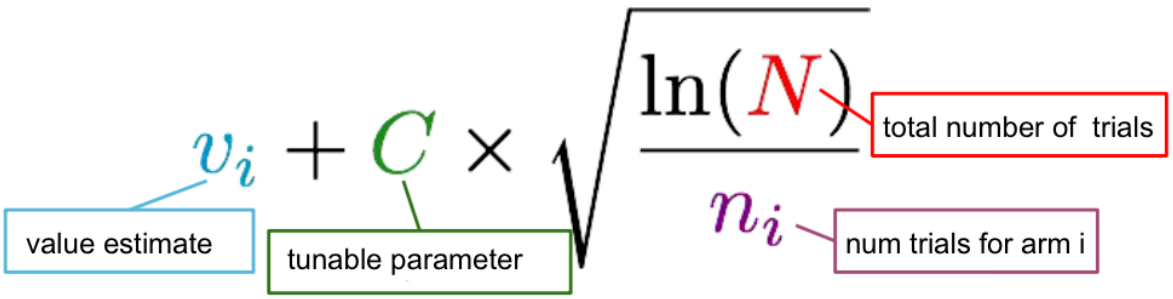
- Want to minimize regret = loss from playing non-optimal arm
- Need to balance between exploration and exploitation

How do we determine best arm to play?

- **Policy:** Strategy for choosing arm to play at some time step t
  - Given arm selections and outcomes of previous trials at times 1, …, t - 1
- Examples
  - Uniform policy
    - Play a least-played arm, break ties randomly
  - Greedy
    - Play an arm with highest empirical payoff
- What is a **good** policy?

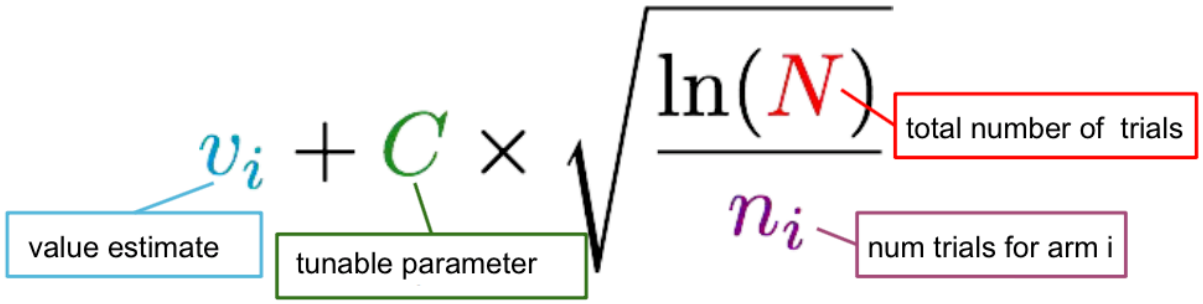
Upper Confidence Bound

- UCB1 formula (Auer et al 2002)
- **Policy**
  1. First, try each arm once
  2. Then, at each time step:
     - Choose arm *i* that maximizes the UCB1 formula for the upper confidence bound



UCB Intuition

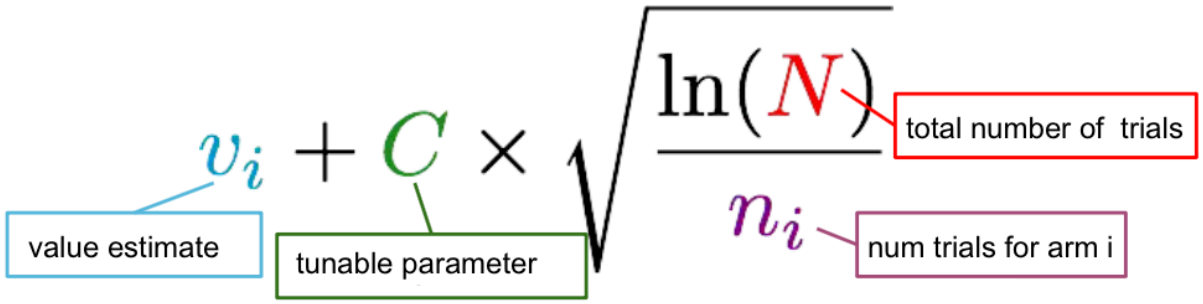
Pick each arm which maximizes



- Higher observed reward v_i is better **(exploit)**
- Expect "true value" to be in some confidence interval around v_i

UCB Intuition

Pick each arm which maximizes



- Confidence interval is large when number of trials n_i is small, shrinks in proportion to $\sqrt{n_i}$
  - High uncertainty about move, larger **exploration** term
- Explore children more if number trials is much less than total number of trials

Theoretical Properties of UCB1

- Main question: rate of convergence to optimal arm
- Huge amount of literature on different bandit algorithms and their properties
- Typical goal: regret O(logn) for n trials
- For many kinds of problems, cannot do better asymptotically (Lai and Robbins 1985)
- UCB1 is a simple algorithm that achieves this asymptotic bound for many input distributions

The case of Trees: From UCB to UCT

- UCB makes a single decision
- What about sequences of decisions (e.g. planning games?)
- Answer: use a look-ahead tree
- Scenarios
  - Single Agent (planning, all actions controlled)
  - **Adversarial** (as in games, or worst-case analysis)
  - Probabilistic (average case, “neutral” environment)

Bandits to Games

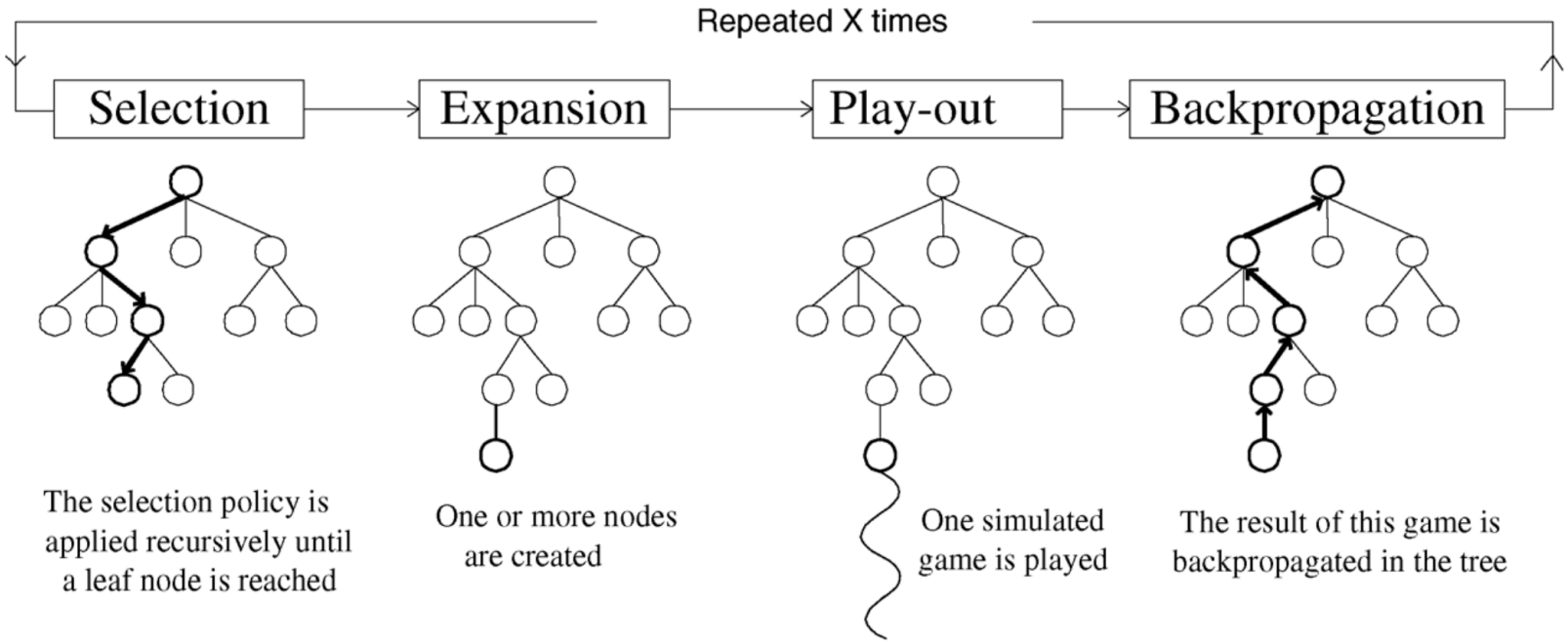
- Bandit arm ≈ move in game
- Payoff ≈ quality of move
- Regret ≈ difference to best move

Monte Carlo Tree Search

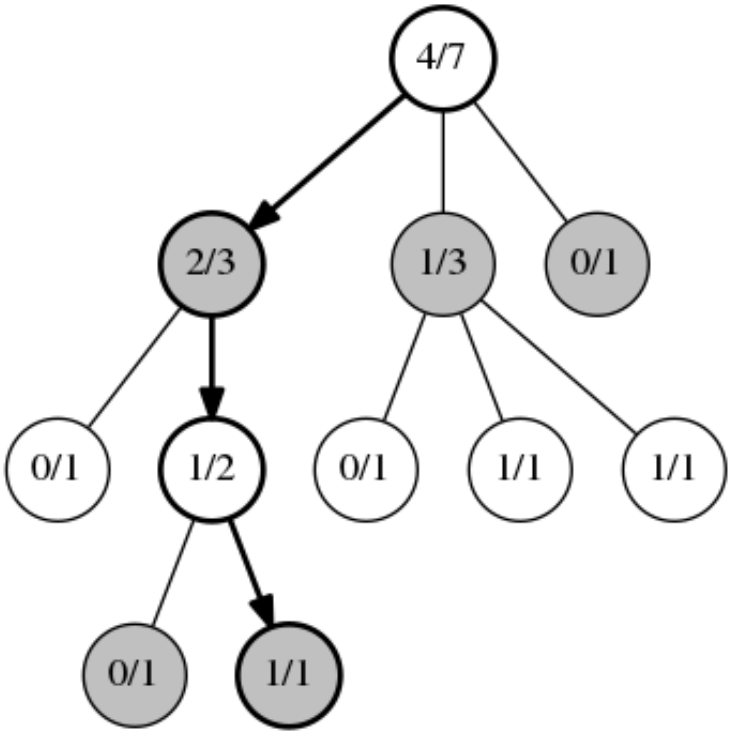
Basic Idea

- Build lookahead tree (e.g. game tree)
- Use rollouts (simulations) to generate rewards
- Apply UCB-like formula to interior nodes of tree
  - Choose “optimistically” where to expand next

Monte Carlo Tree Search

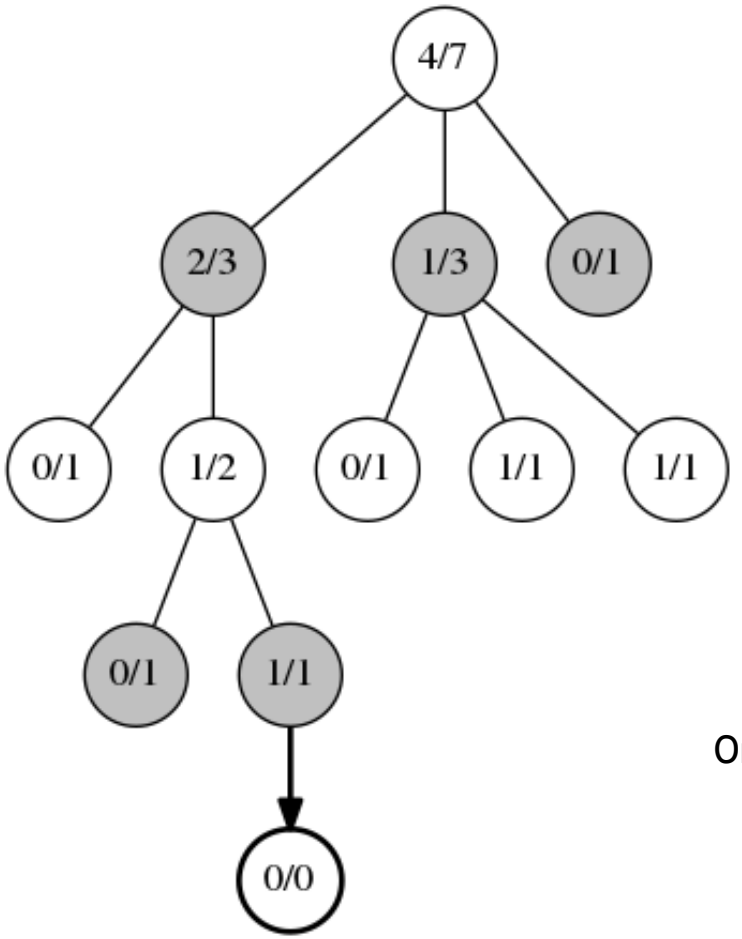


1) Selection



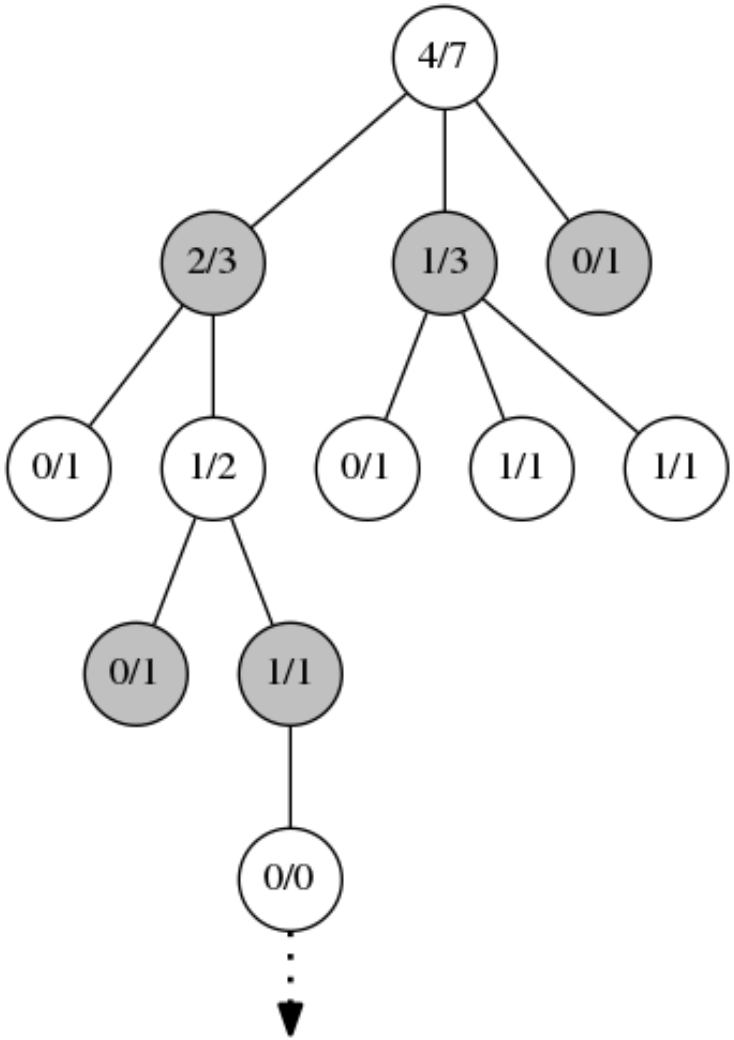
Selection policy is applied recursively until a leaf node is reached

2) Expansion



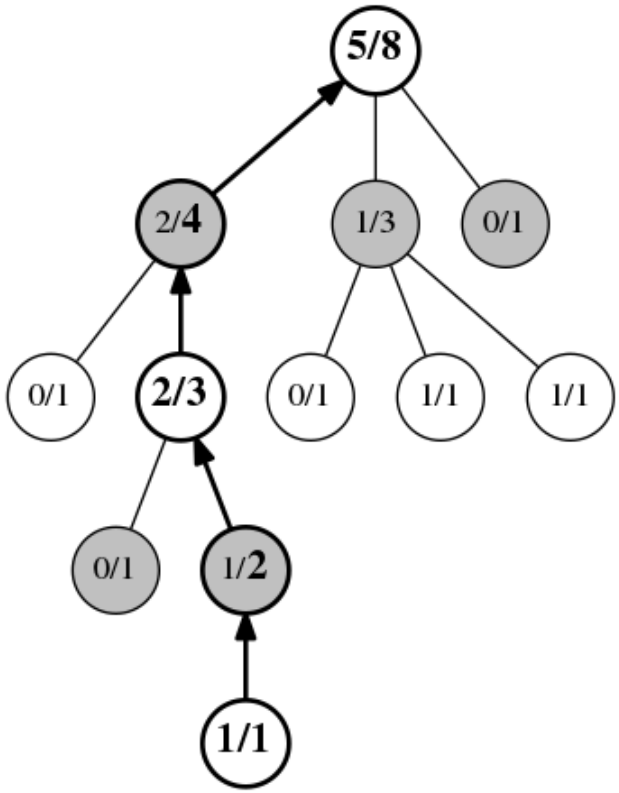
One or more nodes are created

3) Simulation



One simulated game is played

4) Backpropagation

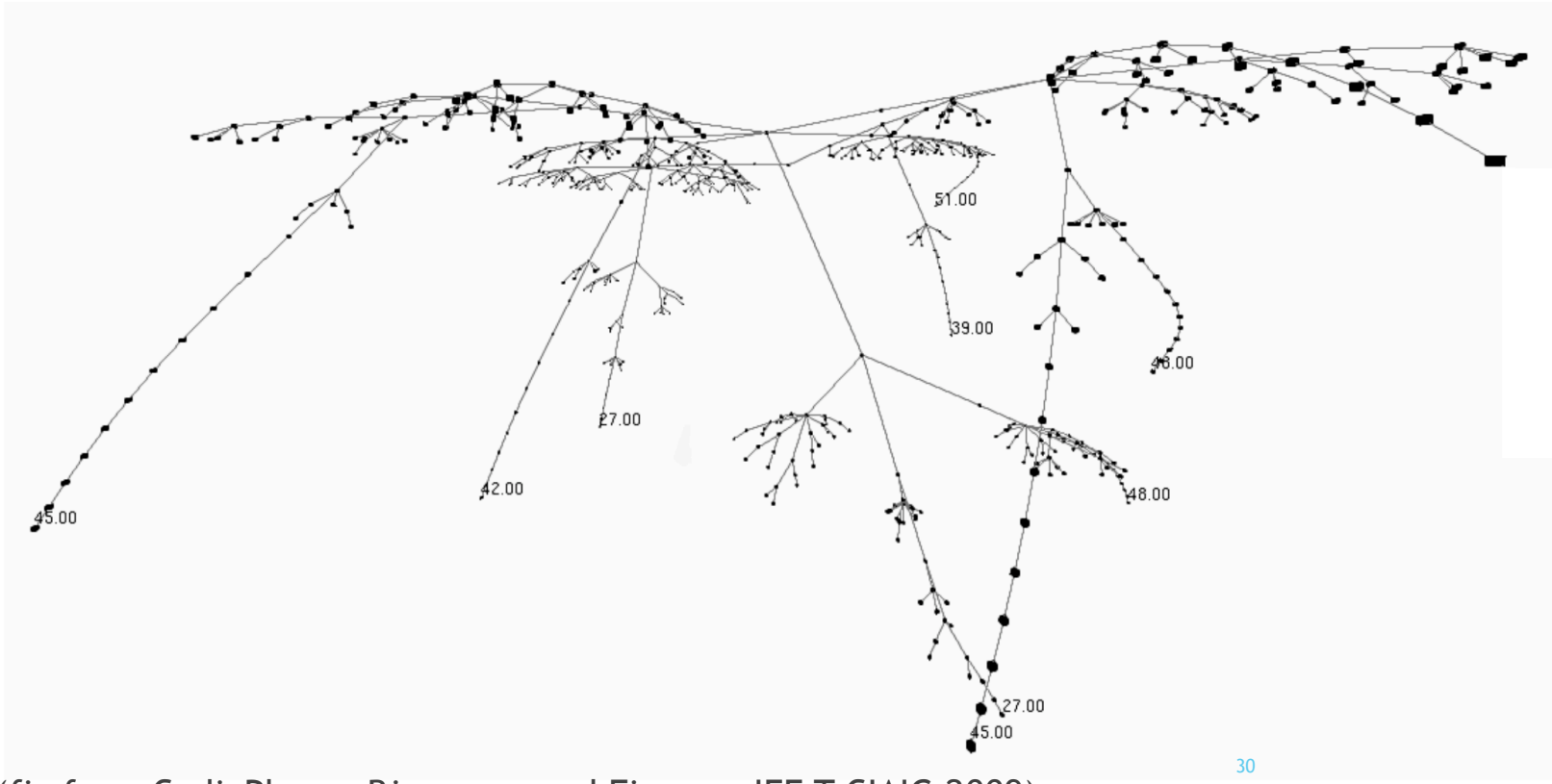


Result is backpropagated up the tree

Move Selection for UCT

- Scenario:
  - Run UCT as long as we can
  - Run simulations, grow tree
- When out of time, which move to play?
  - Highest mean
  - Highest UCB
  - **Most simulated move**
    - MCTS looks at more interesting moves more often

Sample MCTS Tree



(fig from CadiaPlayer, Bjornsson and Finsson, IEE T-CIAIG 2009)

Summary – MCTS So far

- UCB, UCT are very important algorithms in both theory and practice with well-founded convergence guarantees under relatively weak conditions
- One of the advantages of MCTS is its applicability to a variety of games, as it is **domain independent**
- Basis for extremely successful programs for games and many other applications

Impact – Strengths of MCTS

- Very general algorithm for decision making
- Works with very little domain-specific knowledge
  - Need simulator of the domain
- Can take advantage of knowledge when present
- Anytime algorithm
  - Can stop the algorithm and provide answer immediately, though improves answer with more time

Improving Simulations

- Default roll-out policy is to make uniform random moves
- Goal is to find strong correlations between initial position and result of a simulation
- Game independent techniques
  - If there is an immediate win, take it
  - Avoid immediate losses
  - Avoid moves that give opponent immediate win
  - Last Good Reply
- Using prior knowledge

Last Good Reply

- Last Good Reply (Drake 2009), Last Good Reply with Forgetting (Baier et al 2010)
- Idea: after winning simulation, store (opponent move, our answer) move pairs
  - Try same reply in future simulations
  - Forgetting: delete move pair if it fails
- Evaluation: worked well for Go program with simpler playout policy
  - Trouble reproducing success with stronger Go programs
- Simple form of adaptive simulations

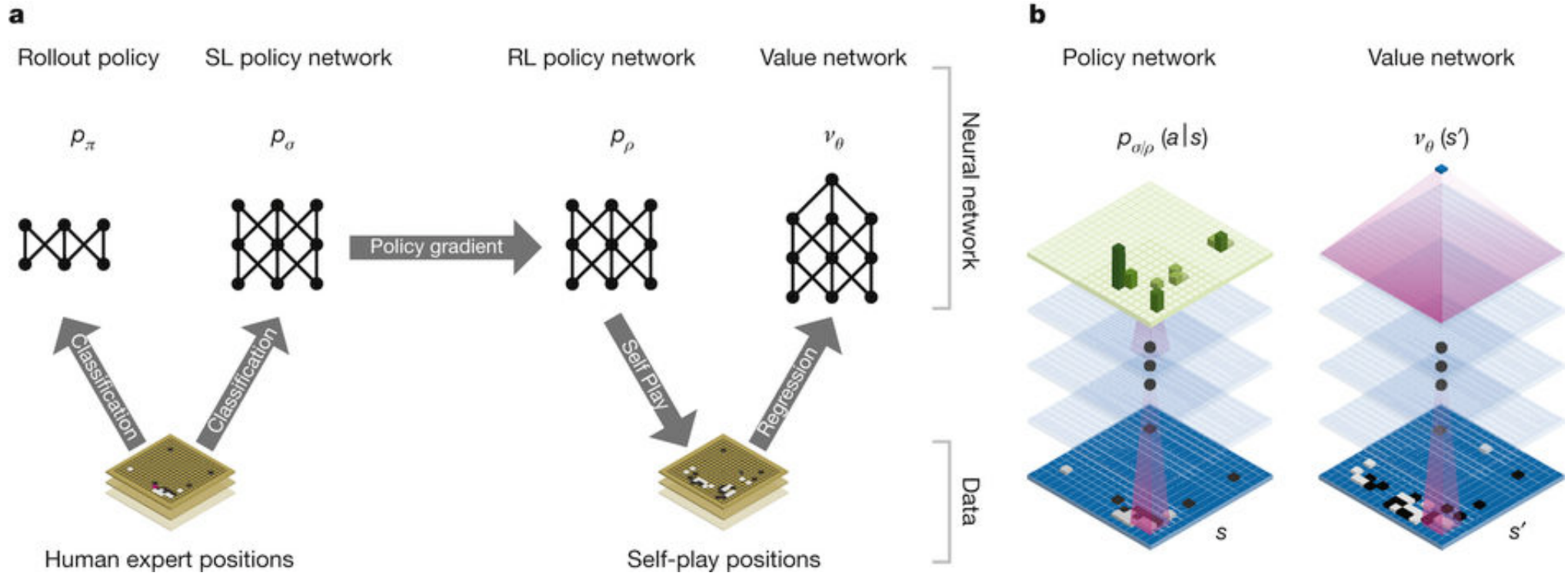
Using Prior Knowledge

- (Silver 2009) machine-learned 3x3 pattern values
- Mogo and Fuego: hand-crafted features
- Crazy Stone: many features, weights trained by Minorization-maximization algorithm (Coulom 2007)
- Fuego today
  - Large number of simple features
  - Weights and interaction weights trained by Latent Feature Ranking
- AlphaGo
  - Neural networks trained over human expert games

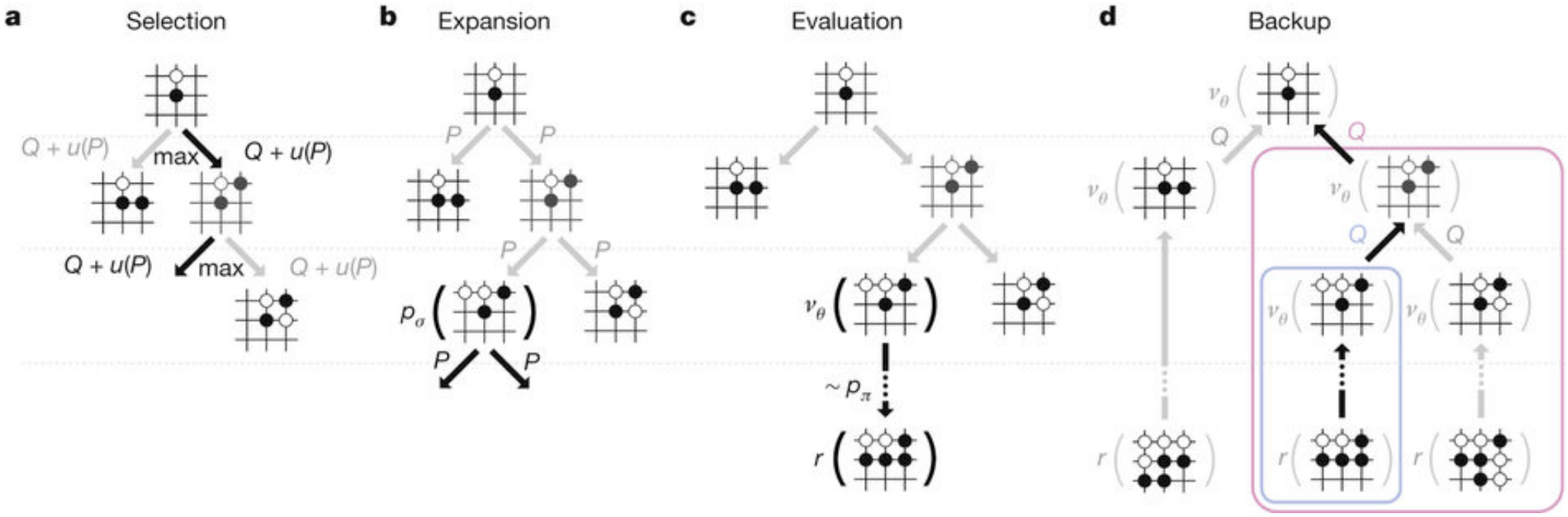
MCTS and Learning

- Learn better knowledge
  - E.g. patterns, features of a domain
- Learn better simulation policies
  - Simulation balancing (Silver and Tesauro 2009)
  - Optimize balance of simulation policies so an accurate spread of simulation outcomes is maintained, rather than optimizing direct strength of simulation policy
- Adapt simulations online
  - Dyna2, RLGo (Silver and Muller 2013)
  - Nested Rollout Policy Adaptation (Rosin 2011)
  - Use RAVE estimates (Rimmel et al 2011)

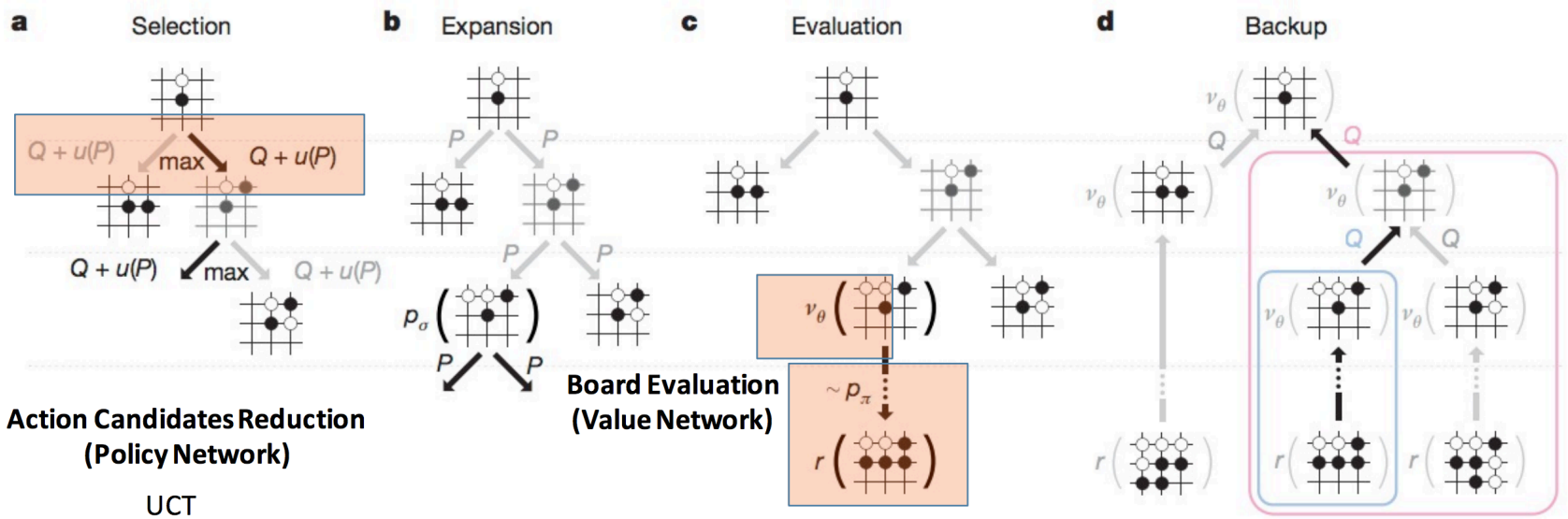
AlphaGo


a
Rollout policy
SL policy network
RL policy network
Value network
p_π
p_σ
p_ρ
v_θ
Policy gradient
Classification
Classification
Self Play
Regression
Neural network
Data
Human expert positions
Self-play positions
b
Policy network
Value network
$p_{\sigma/\rho}(a|s)$
$v_\theta(s')$
s
s'

AlphaGo


a
Selection
Q + u(P)
max
Q + u(P)
Q + u(P)
max
Q + u(P)
b
Expansion
P
P
P
P
p_σ
P
P
c
Evaluation
v_θ
$\sim p_\pi$
r
d
Backup
v_θ
Q
Q
v_θ
v_θ
Q
Q
v_θ
v_θ
r
r
r

AlphaGo



Action Candidates Reduction (Policy Network)

UCT

Board Evaluation (Value Network)

(Rollout): Faster version of estimating p(a|s)
→uses shallow networks (3 ms →2µs)

MCTS Real-Time Approach

- Requirements:
  - Need a fast and accurate forward model
  - i.e. taking action a in state s leads to state s' (or a known probability distribution over a set of states)
- We could learn this model if it doesn't exist?
- For games, such a model usually exists
  - But may need to simplify it

MCTS Real-Time Approaches

- State space abstraction
  - Quantise state space – mix of MCTS and Dynamic Programming
  - Search graph rather than a tree
- Temporal Abstraction
  - Don’t need to make different actions 60 times per second
  - Instead, current action is usually same (or predictable from) previous one
- Action abstraction
  - Consider higher-level action space