Monte Carlo Tree Search

Cmput 366/609 Guest Lecture

Fall 2017

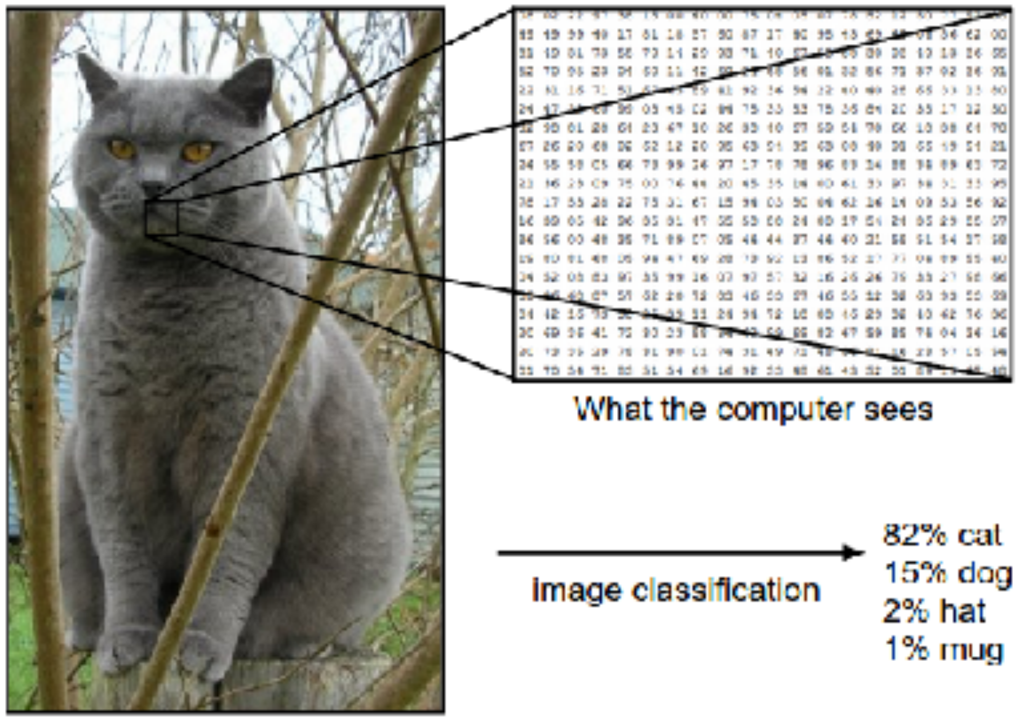
Martin Müller

mmueller@ualberta.ca

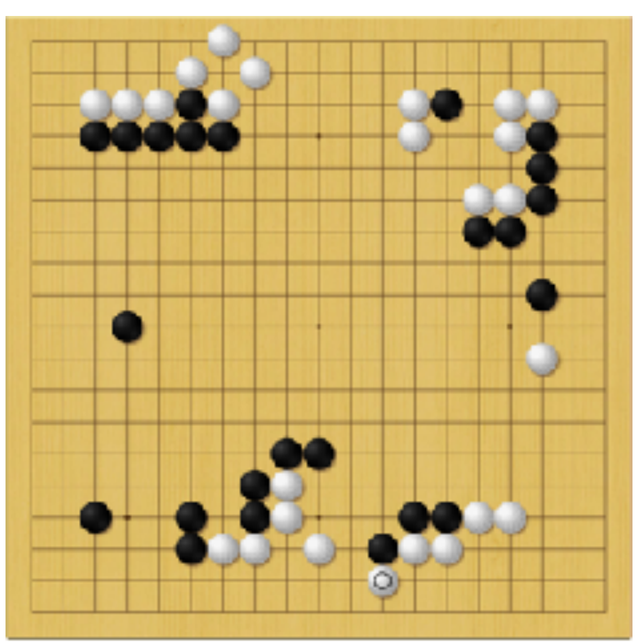
Contents

- 3+1 Pillars of Heuristic Search
- Monte Carlo Tree Search
- Learning and using Knowledge
- Deep neural nets and AlphaGo

Decision-Making



Source: http://cs231n.github.io/assets/classify.png



- One-shot decision making
  - Example - image classification
  - Analyze image, tell what's in it
- Sequential decision-making
  - Need to look at possible futures in order to make a good decision *now*

Heuristic Search

- State space (e.g. game position; location of robot and obstacles; state of Rubik's cube)
- Actions (e.g. play on C3; move 50cm North; turn left)
- Start state and goal
- Heuristic evaluation function - estimate distance of a state to goal

Three plus one Pillars of Modern Heuristic Search

- Search algorithm
- Evaluation function, heuristic
- Simulation
- We have had search+evaluation for decades (alphabeta, A*, greedy best-first search,…)
- *Combining* all three is relatively new -
- Machine learning is key

Alphabeta Search

- Classic algorithm for games
  - Search + evaluation, no simulation
- Minimax principle
  - My turn: choose best move
  - Opponent's turn: they choose move that's worst for me

αβ Successes (I)

- Solved games -
  proven value of starting position
  - checkers (Schaeffer et al 2007)
  - Nine men's morris (Gasser 1994)
  - Gomoku (5 in a row) (Allis 1990)
  - Awari, 5x5 Go, 5x5 Amazons,.....

αβ Successes (2)

- Not solved, but super-human strength:
  - chess (Deep Blue team, 1996)
  - Othello (Buro 1996)
  - shogi (Japanese chess, around 2013?)
  - xiangqi (Chinese chess, around 2013?)

αβ Failures

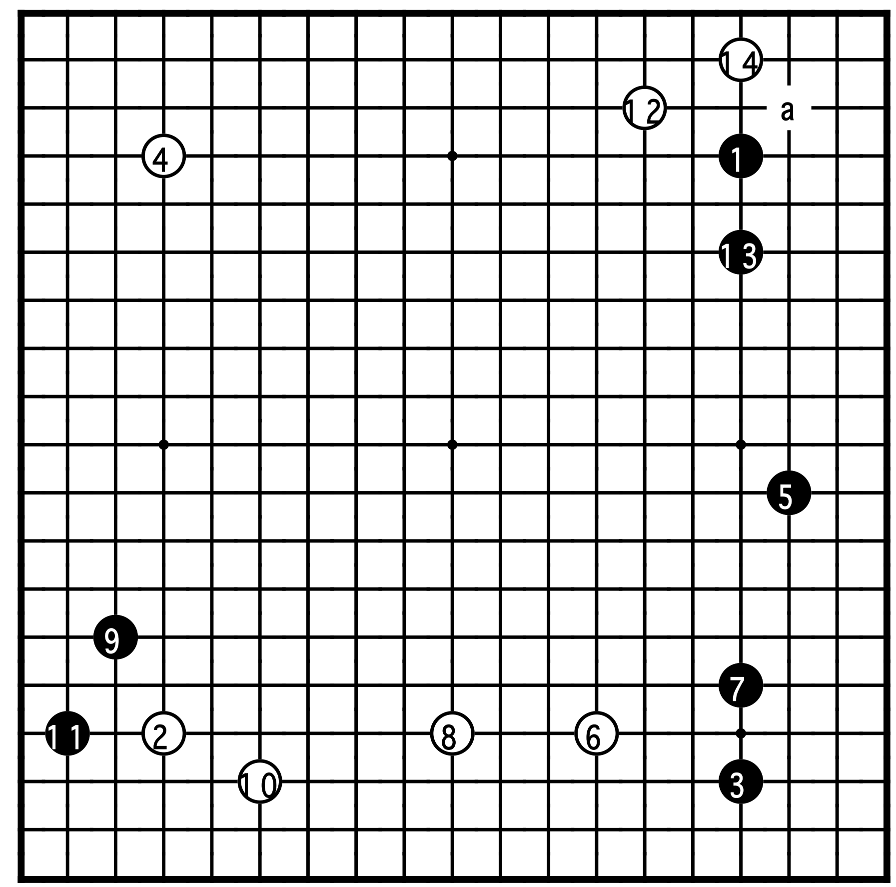
- Go
- General Game Playing (GGP)
- Why fail?
  - Focus on Go here

Go

- Classic Asian board game
- Simple rules, complex strategy
- Played by millions
- Hundreds of top experts - professional players
- Until recently, computers much weaker than humans

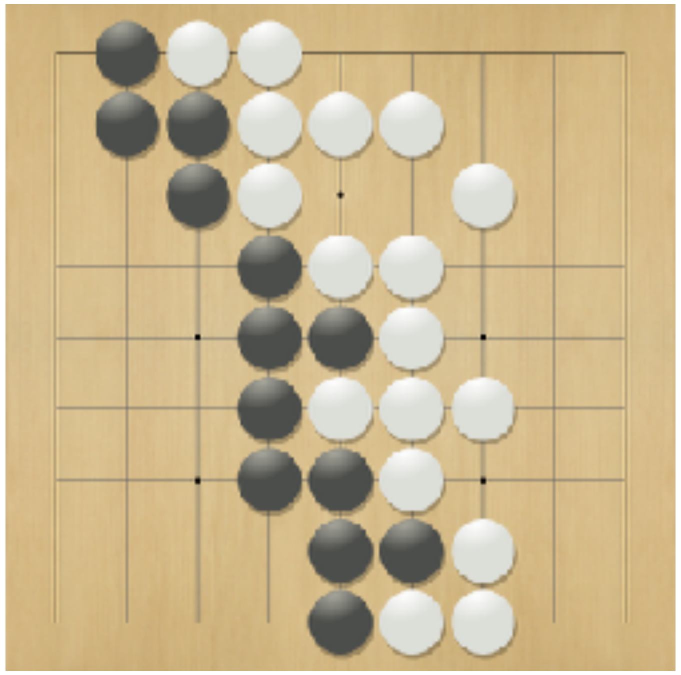
Go Rules

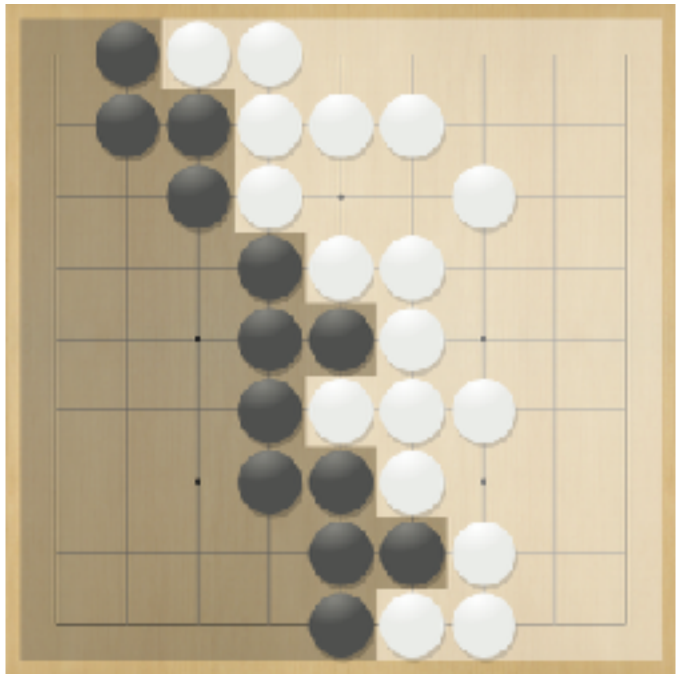
- Start: empty board
- Goal: surround
  - Empty points
  - Opponent (capture)
- Win: control more than half the board



End of Game

- End: both players pass
- *Territory* - intersections surrounded by one player
- The player with more (stones+territory) wins the game
- *Komi*: adjustment for first player advantage (e.g. 7.5 points)





Why does αβ Fail in Go?

- Huge state space,
  depth and width of game tree
  - 250 moves on average
  - game length > 250 moves average
- Until very recently:
  no good **evaluation function**

Monte Carlo Methods

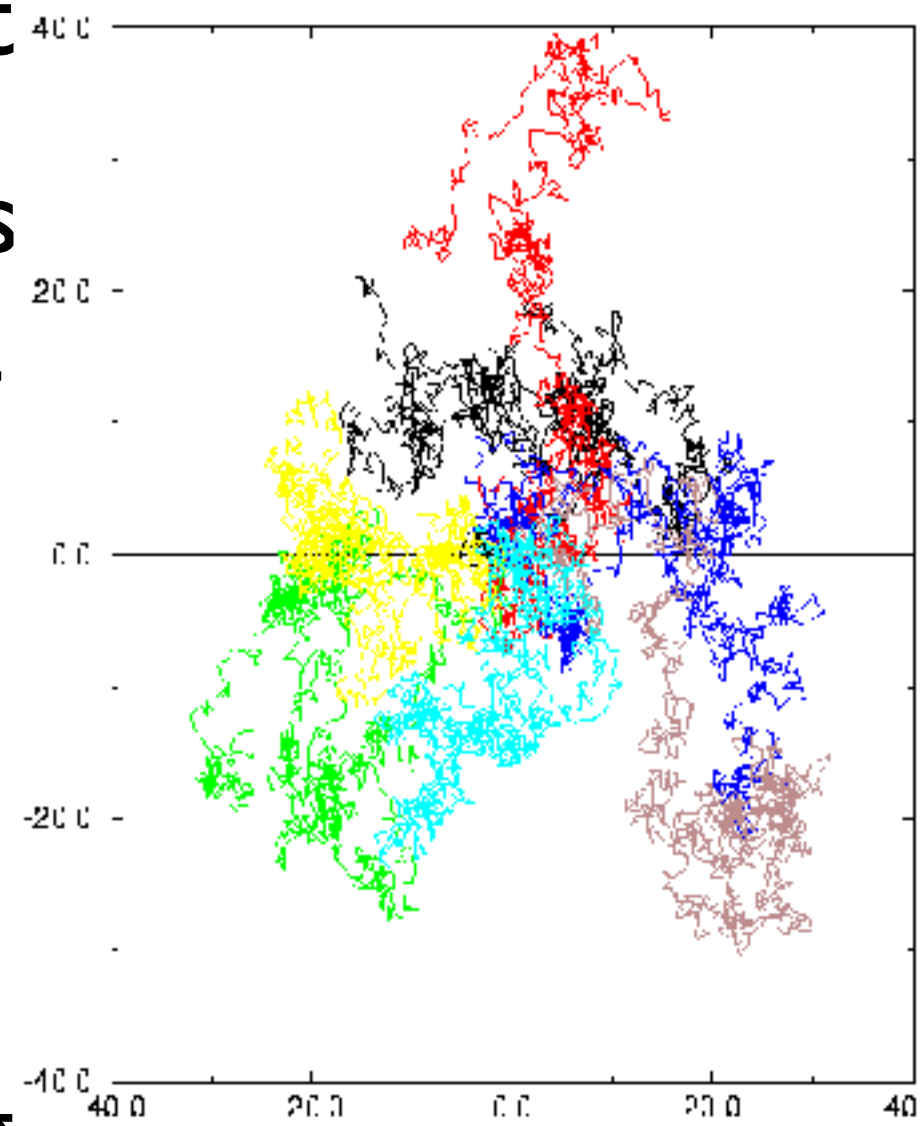
- Popular in the last 10 years
- Hugely successful in many applications
  - Backgammon (Tesauro) early example
  - Go (many)
  - Amazons, Havannah, Lines of Action, ...
  - Planning, energy management, mathematical optimization, solve MDP,...

Monte Carlo Simulation

- No evaluation function? No problem!
- Simulate rest of game using random moves (easy)
- Score the game at the end (easy)
- Use that as evaluation (hmm, but...)

The GIGO Principle

- **G**arbage **i**n, **g**arbage **o**ut
- Even the best algorithms do not work if the input data is bad
- Making random moves sounds pretty bad...
- How can we gain any information from playing them?



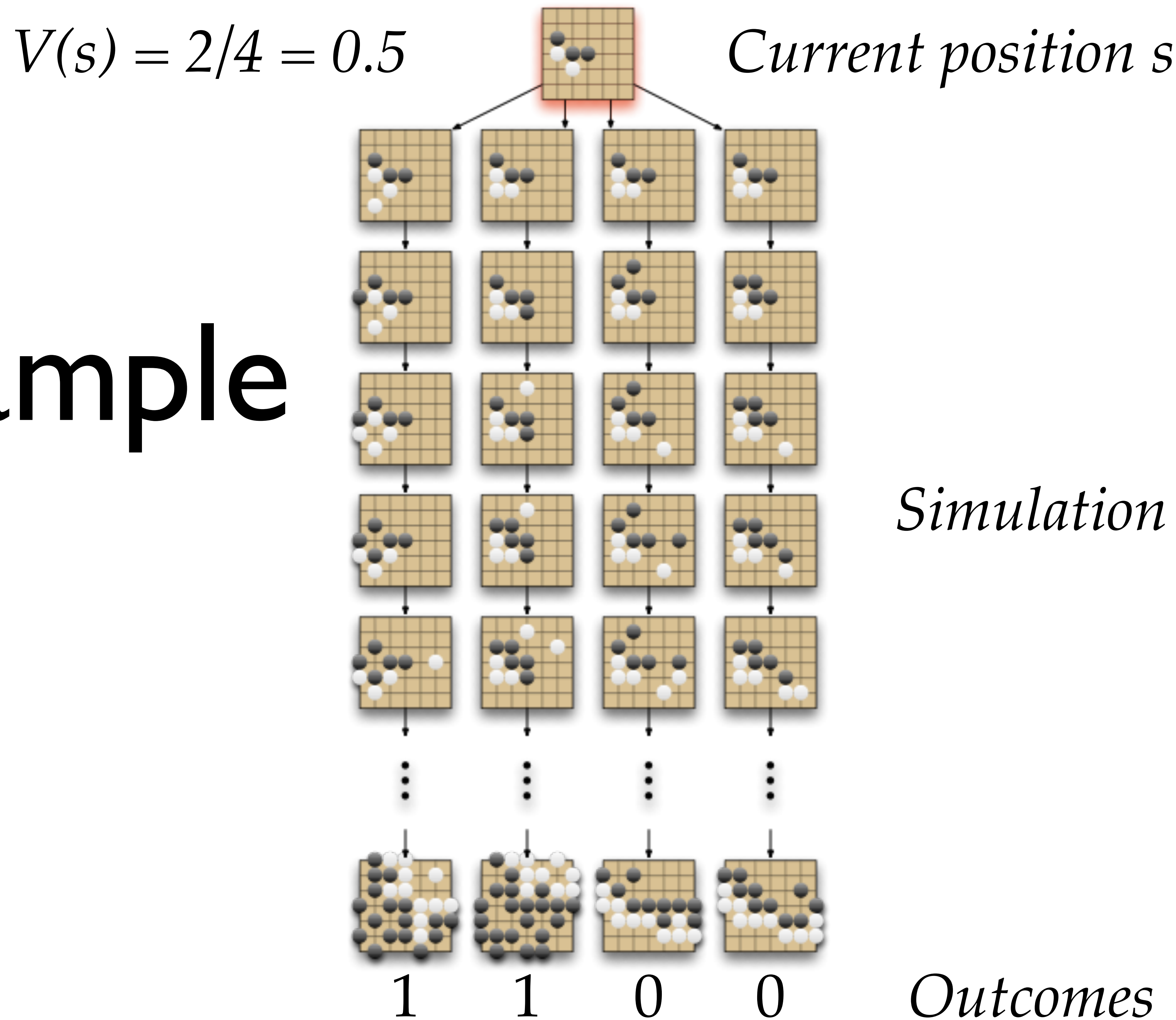
Well, it Works!

- For some games, anyway
- Even random moves often *preserve some difference* between a good position and a bad one
- The rest is (mostly) statistics…

Basic “Flat” Monte Carlo Search Algorithm

1. Play lots of random games starting with each possible move
2. Keep winning statistics for each move
3. Play move with best winning percentage

Example



How to Improve?

1. Better-than-random simulations
2. Add game tree (as in $\alpha\beta$)
3. Add knowledge as bias in the game tree
4. AlphaGo

1. Better Simulations

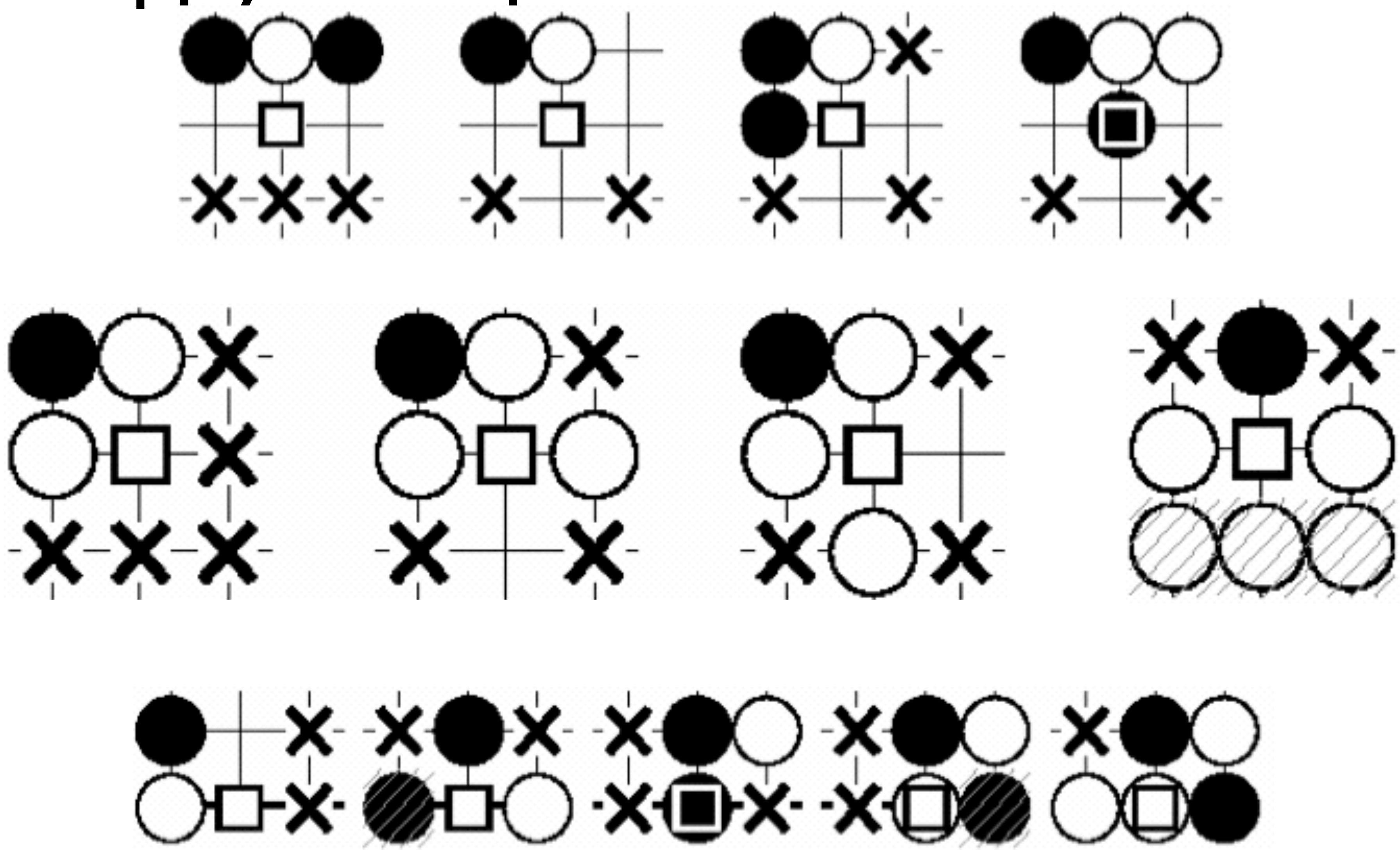
- Goal: strong correlation between initial position and result of simulation
- Try to preserve wins and losses
- How?

Use Knowledge in Simulations

- MoGo-style patterns
- Tactical rules
- Machine learning using features and feature weights

MoGo-Style Patterns

- 3x3 or 2x3 patterns
- Apply as response near last move



Building a better Randomized Policy

- Use rules, patterns
  to set *probabilities* for each legal move
- Learn probabilities
  - From human games
  - From self-play

2. Add Game Tree

- First idea:
  - Use $\alpha\beta$
  - Use simulations directly as an evaluation function for
- This fails:
  - Too much noise
  - Too slow

Monte Carlo Tree Search

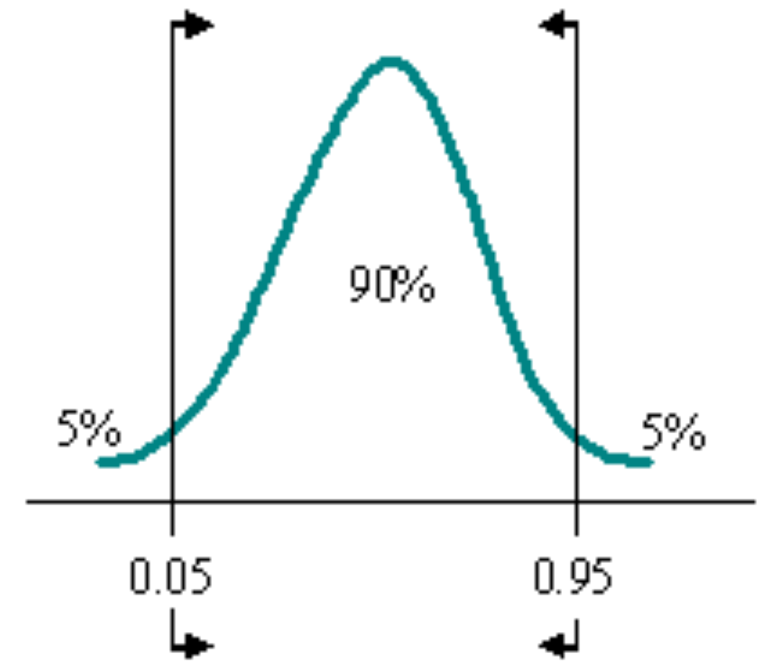
- Idea: use results of simulations to guide growth of the game tree
- **Exploitation**: focus on promising moves
- **Exploration**: focus on moves where uncertainty about evaluation is high
- Two contradictory goals?

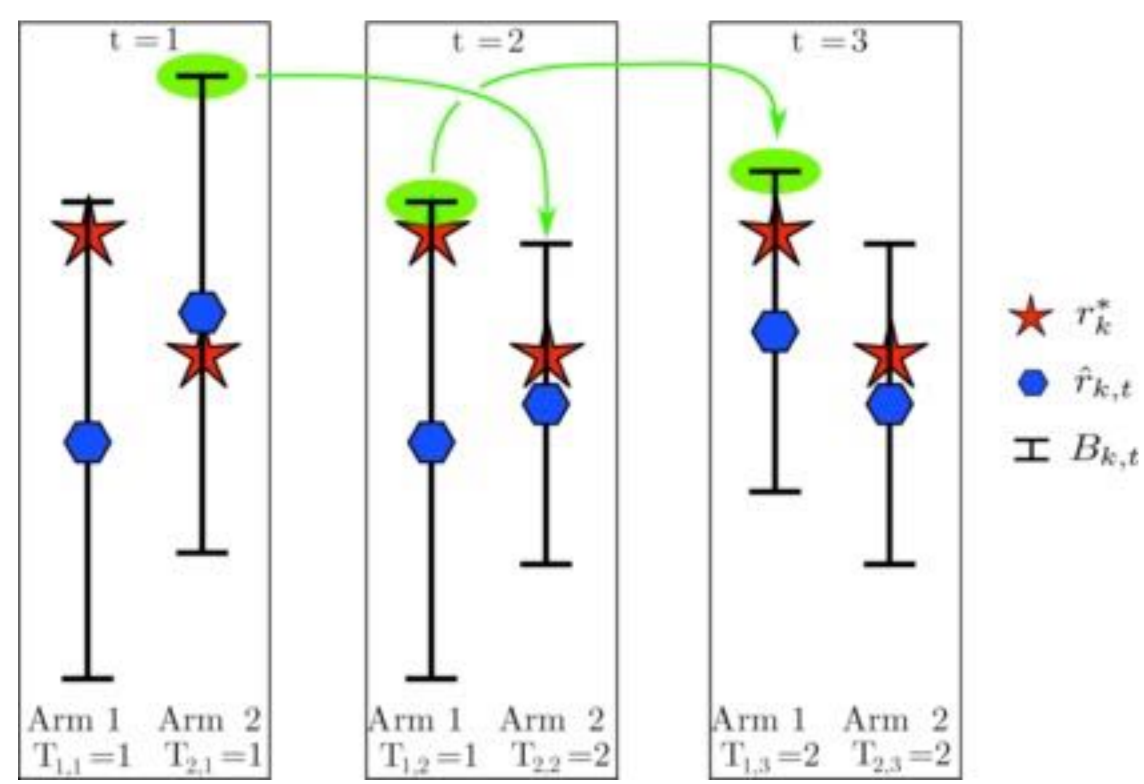
UCB Formula

- Multi-armed bandits (slot machines in Casino)
  - Which bandit has best payoff?
  - Explore all arms, but:
    - Play promising arms more often
  - Minimize *regret* from playing poor arms

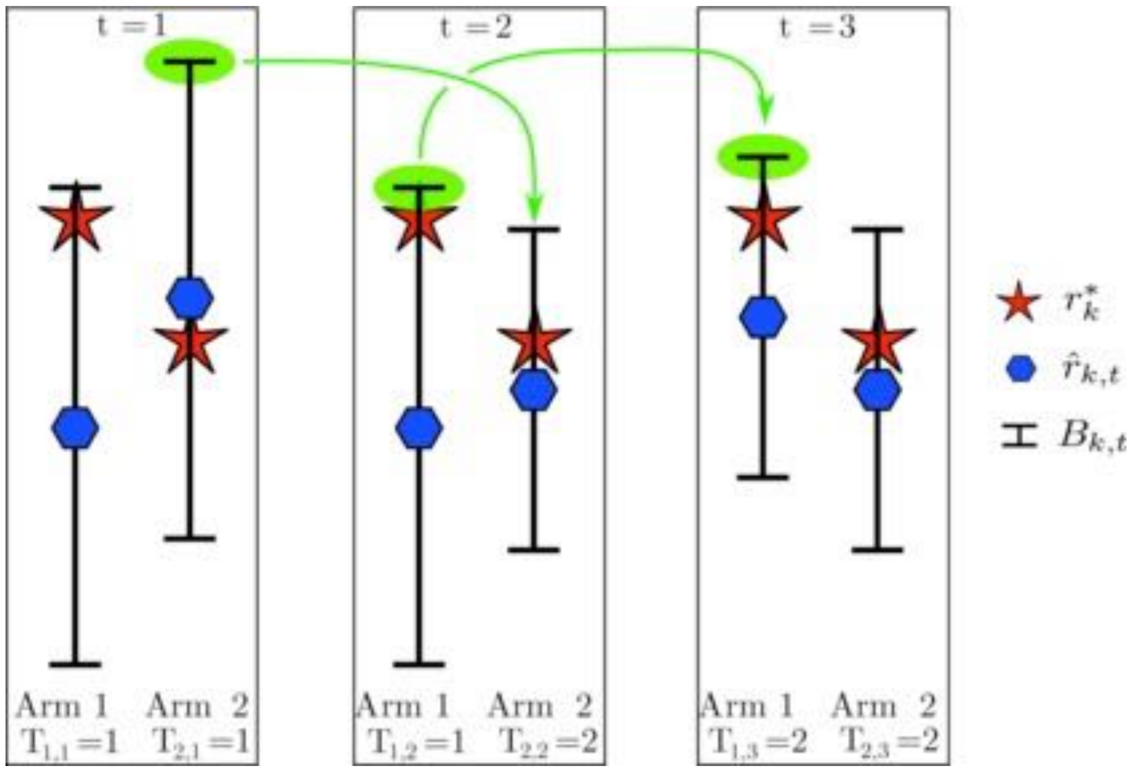
Some Statistics

- Take random samples from fixed probability distribution
- With many trials, average outcome will converge to the expected outcome
- Confidence bounds: true value is probably within these bounds





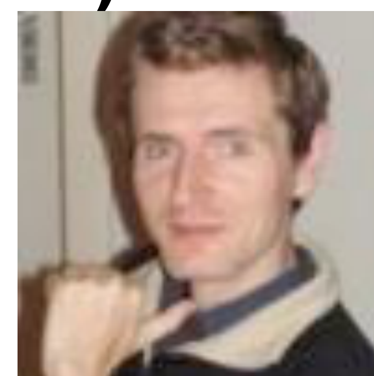
UCB Idea



- UCB = Upper confidence bound
- Take next sample for the arm for which UCB is highest
- Principle:
  optimism in the face of uncertainty

UCT Algorithm

- Kocsis and Szepesvari (2006)
- Apply UCB in each node of a game tree
- Which node to expand next?
    - Start at root (current state)
    - While in tree, choose child *n* that maximizes:



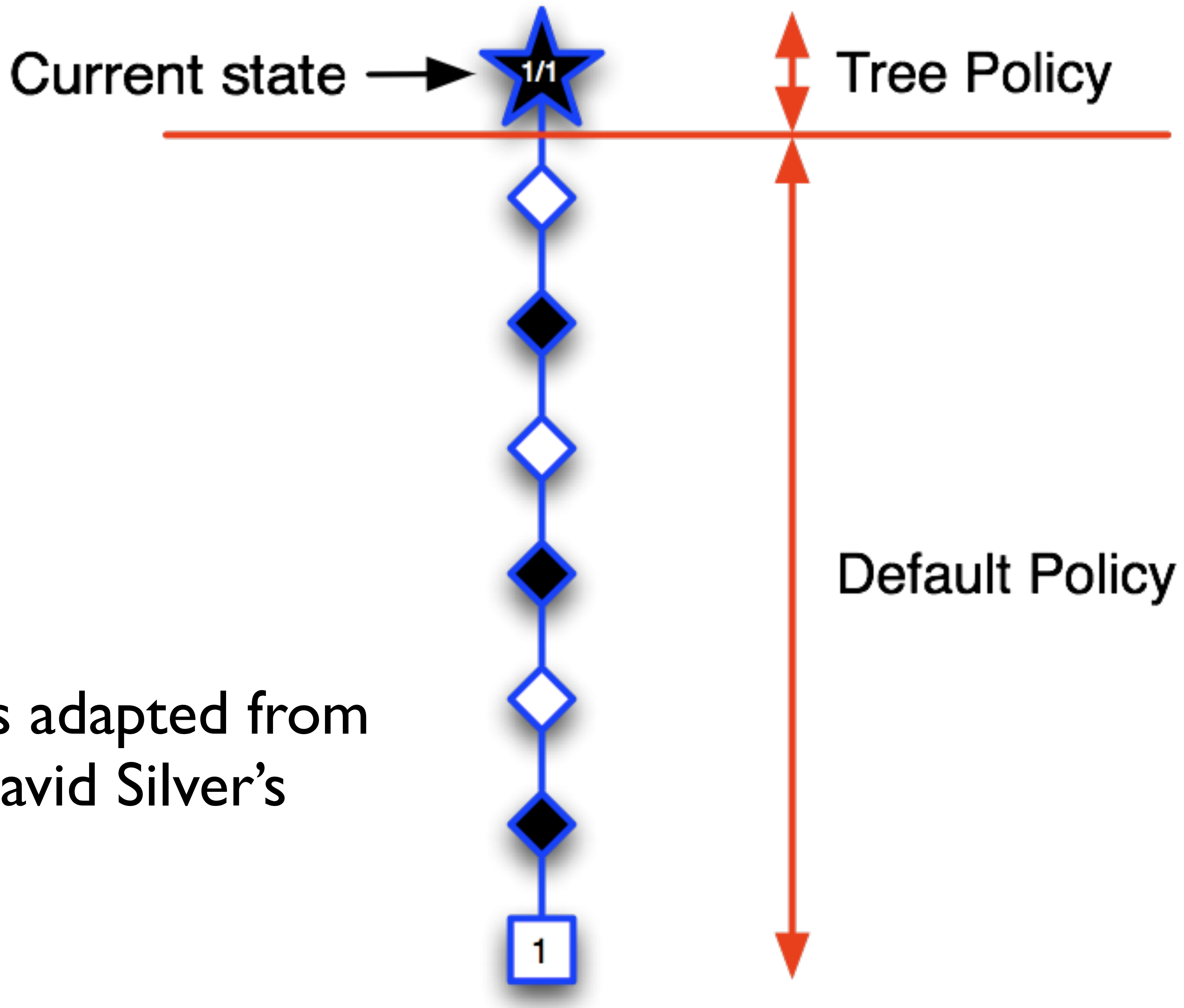
UCTValue(*parent*, *n*) =

winrate(*n*)
+ C * sqrt(ln(*parent.visits*)/*n.visits*)

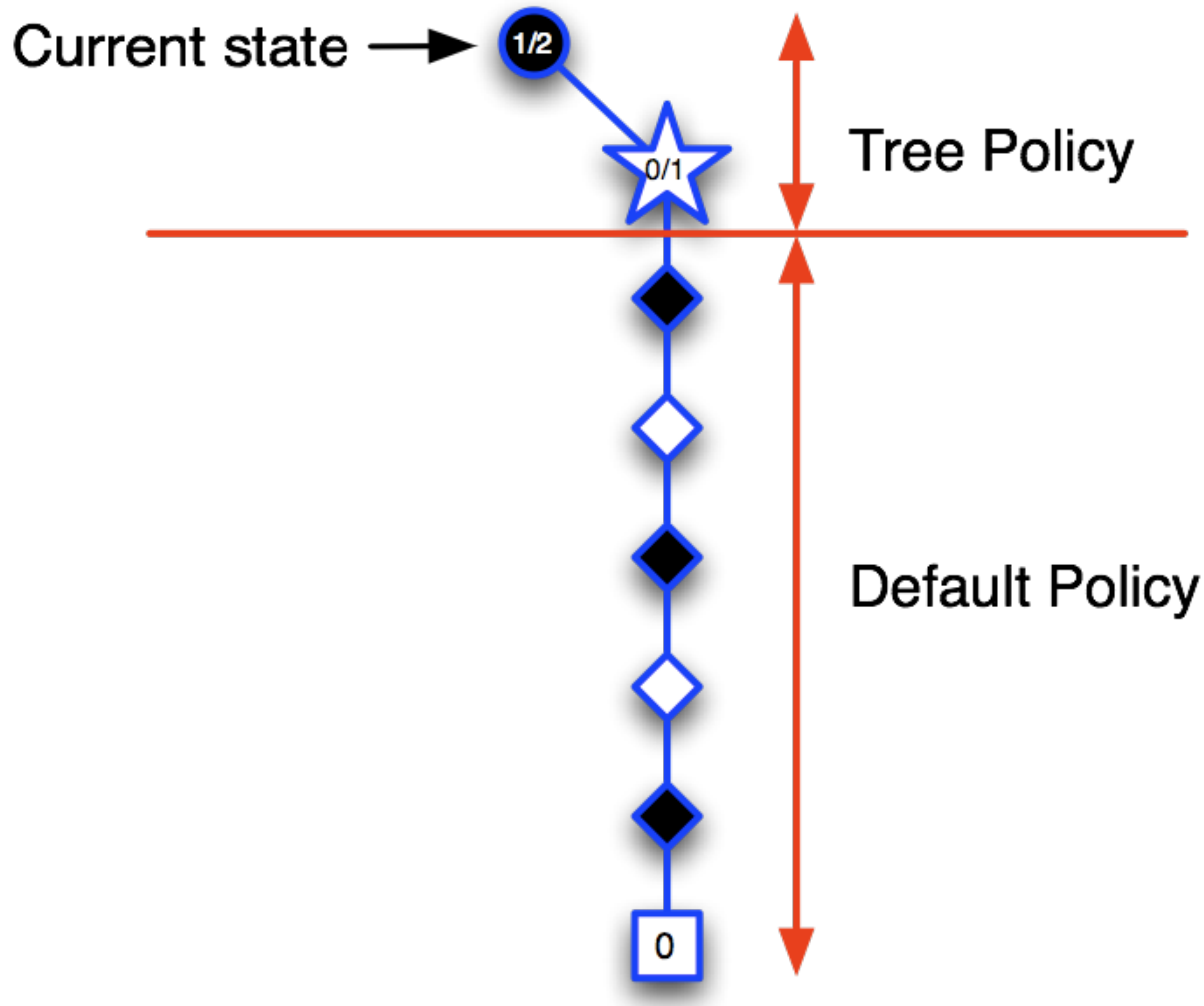
UCTValue(*parent*, *n*) =

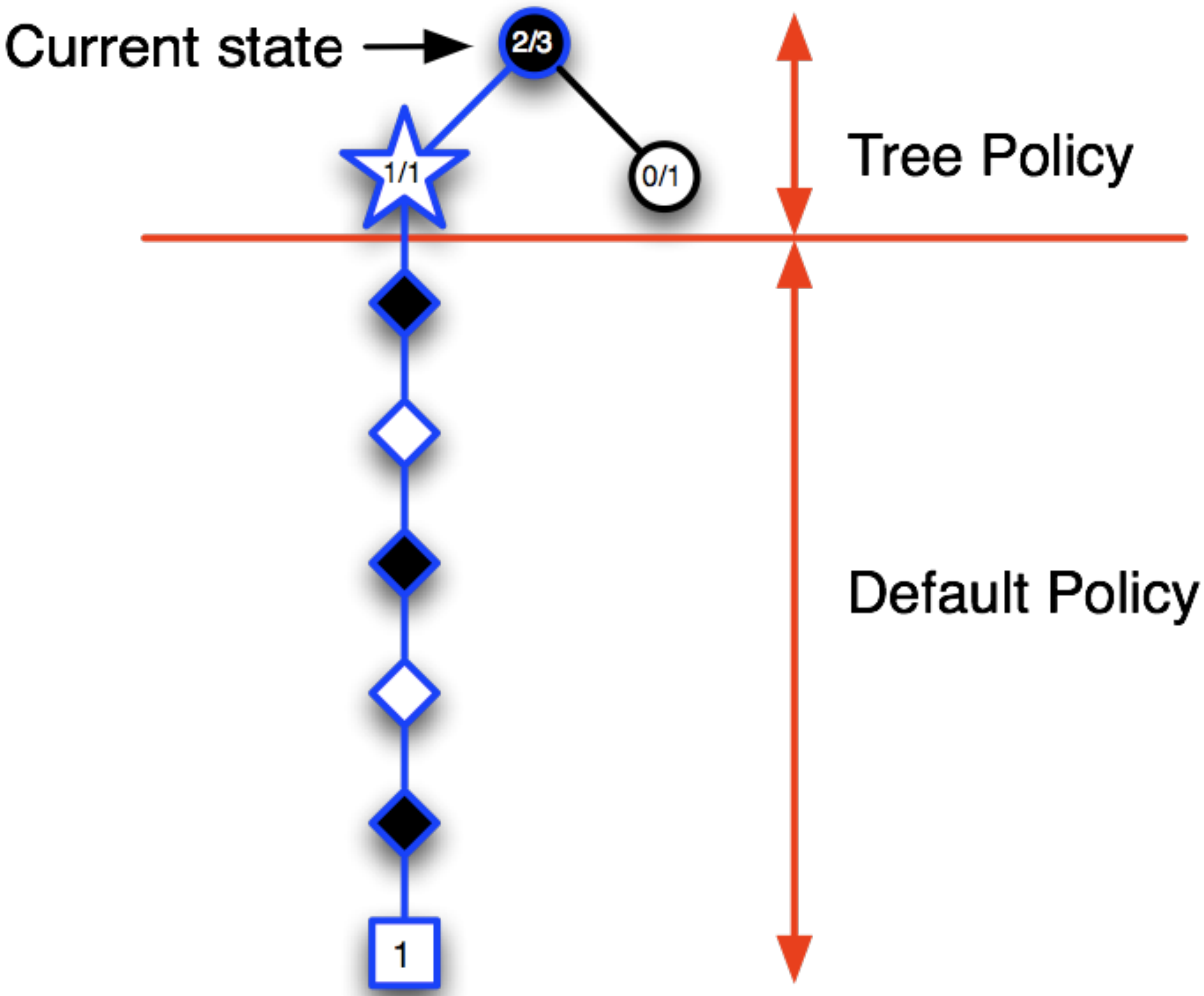
winrate(*n*) + C * sqrt(ln(*parent.visits*)/*n.visits*)

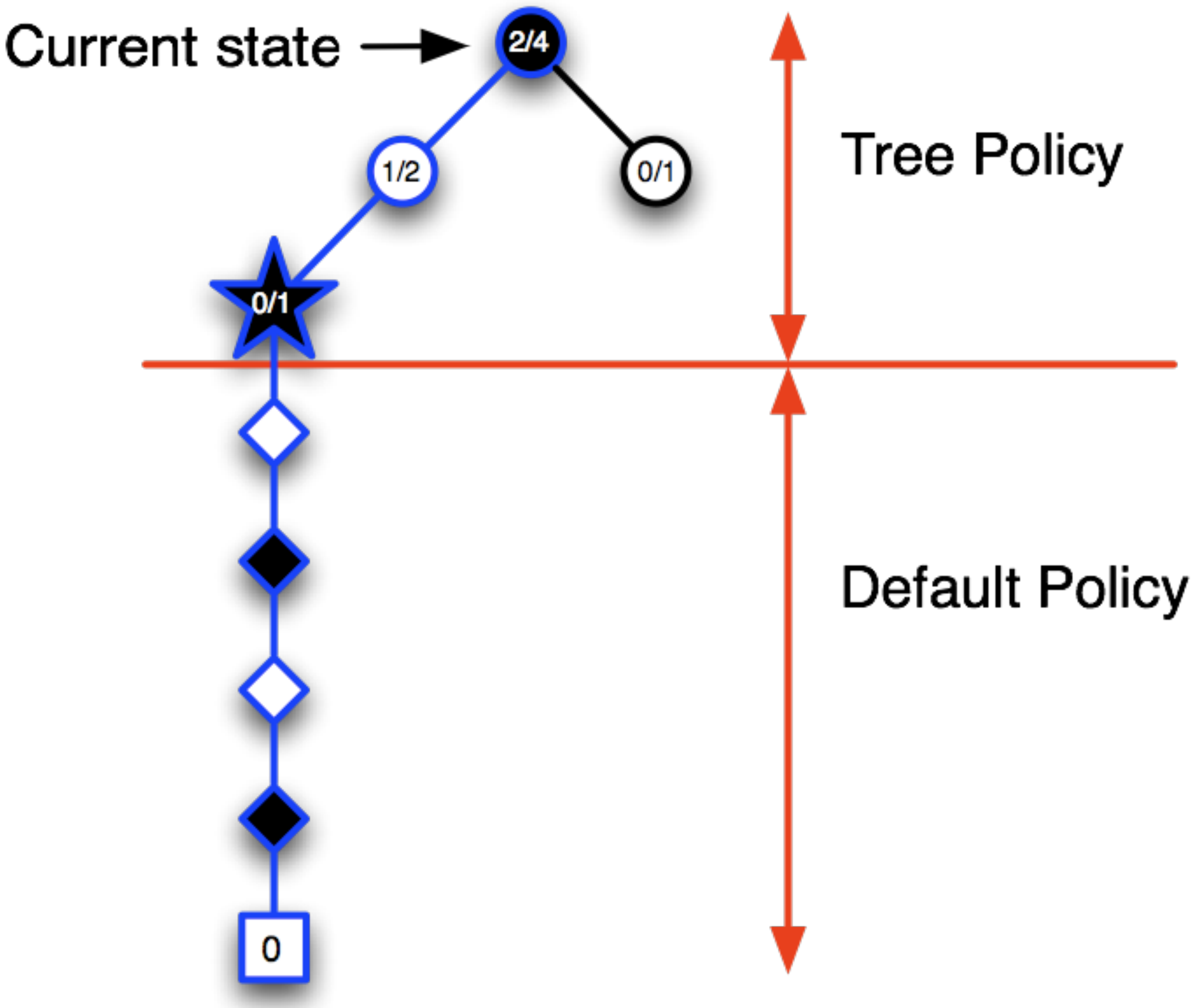
- winrate(*n*) .. *exploitation* term - average success of *n* so far
- 1/*n.visits* .. part of *exploration* term - explore nodes with very few visits - reduce uncertainty
- ln(*parent.visits*) .. part of *exploration* term - explore all nodes at least a little bit
- C .. *exploration constant* - how important is exploration relative to exploitation?



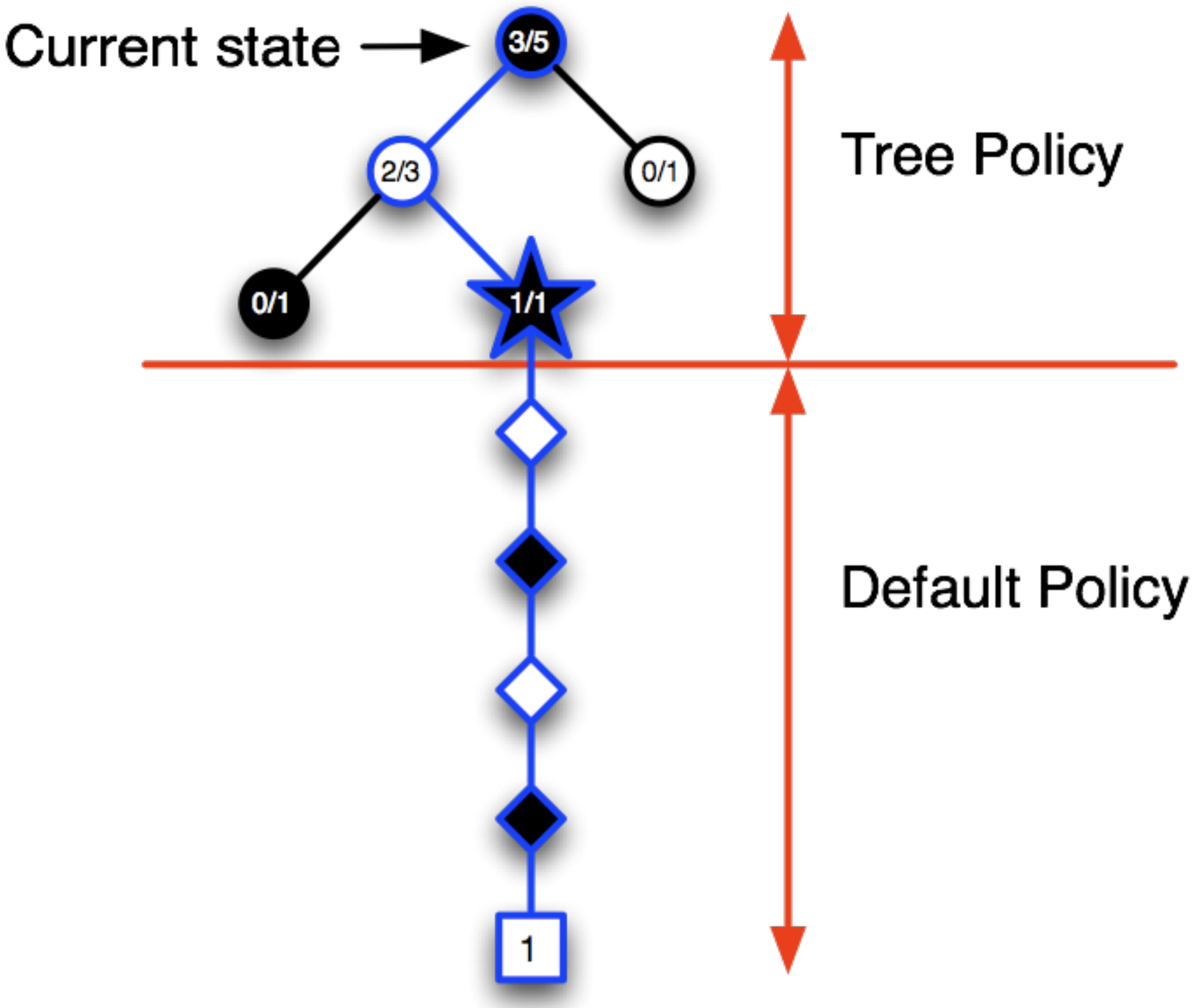
Slides adapted from
David Silver's


Current state
1/2
0/1
Tree Policy
Default Policy
0


Current state
2/3
1/1
0/1
Tree Policy
Default Policy
1



Current state
2/4
1/2
0/1
0/1
0
Tree Policy
Default Policy


Current state
3/5
2/3
0/1
0/1
1/1
1
Tree Policy
Default Policy

Summary - Monte Carlo Tree Search

- Amazingly successful in games and in probabilistic planning (PROST system)
  - Top in Backgammon, Go, General Game Playing, Hex, Amazons, Lines of Action, Havannah,...
  - Similar methods work in multiplayer games (e.g. card games), planning, puzzles, energy resource allocation,...

MCTS Comments

- Very successful in practice
- Scales OK to parallel machines
- Why and how does it work?
  - Still poorly understood
- Some limitations (see next slide)

Adding Machine-Learned Knowledge to MCTS

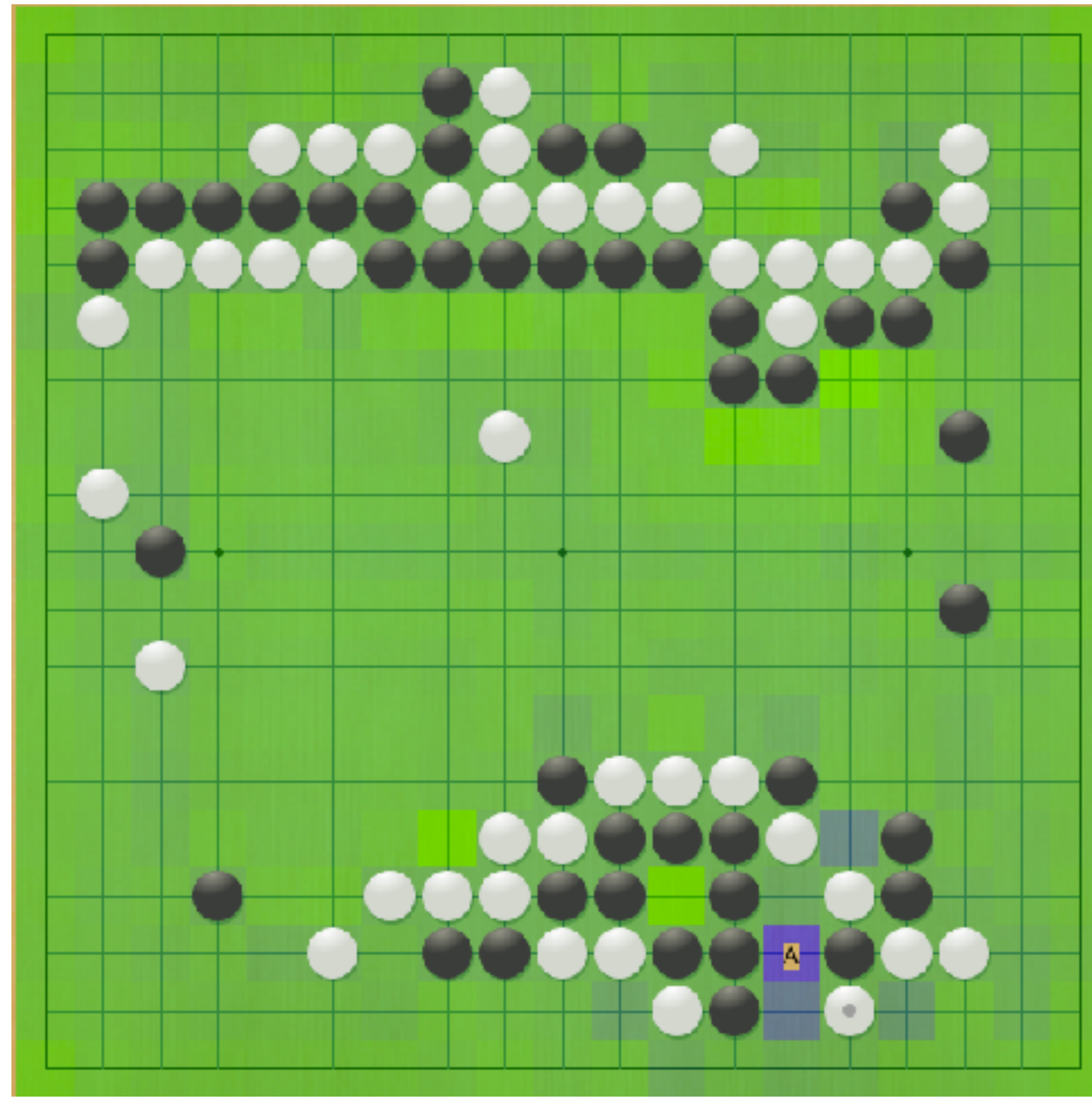
- Game-specific knowledge can overcome limitations
- Two case studies
  - Learning with simple features
  - Deep convolutional neural nets and AlphaGo

Why Learn Knowledge?

- In Go, usually only a small number of good moves
- Human masters strongly prune almost all other moves - and it works!
- It takes time for noisy simulations to rediscover these bad moves every time
- So - let’s learn it.

Example of Knowledge

- Learned move values
  Blue = good
  Green = bad
- Use as initial bias
  in the MCTS tree
  (in-tree, not in playouts)
- Search will *initially* focus
  on *probably* good moves
- Search can still discover
  other moves later



Simple Knowledge

- Fast machine-learned evaluation function
- Supervised learning from master games
- Simple features express quality of moves
- Algorithms learn weights for individual features, and combinations of features
- Training goal: move prediction
  - what did the master play?

Simple Knowledge Examples

- Properties of a candidate move
- Help to predict whether that move is good
- Examples:
    - location on board
    - local context, e.g. 3x3 pattern
    - capture/escape with stones, “ladder”
    - liberties, cut/connect, eye,...

How to Learn Features?

- Standard approach in MCTS (Coulom):
- Each feature has a weight
- If a move has several features, then:
  move value is the product (or sum)
  of the feature weights
- Improvement: take *interactions* of features
  into account (Wistuba, Xiao)

Learning Example

- Professional game records
- about 40.000 games from badukmovies.com
- about 10 Million positions, 2.5 billion move candidates
- Label all moves in all positions in all games with their features
- Each feature has a unique ID number

Example of Labeled Candidate Moves for One Position

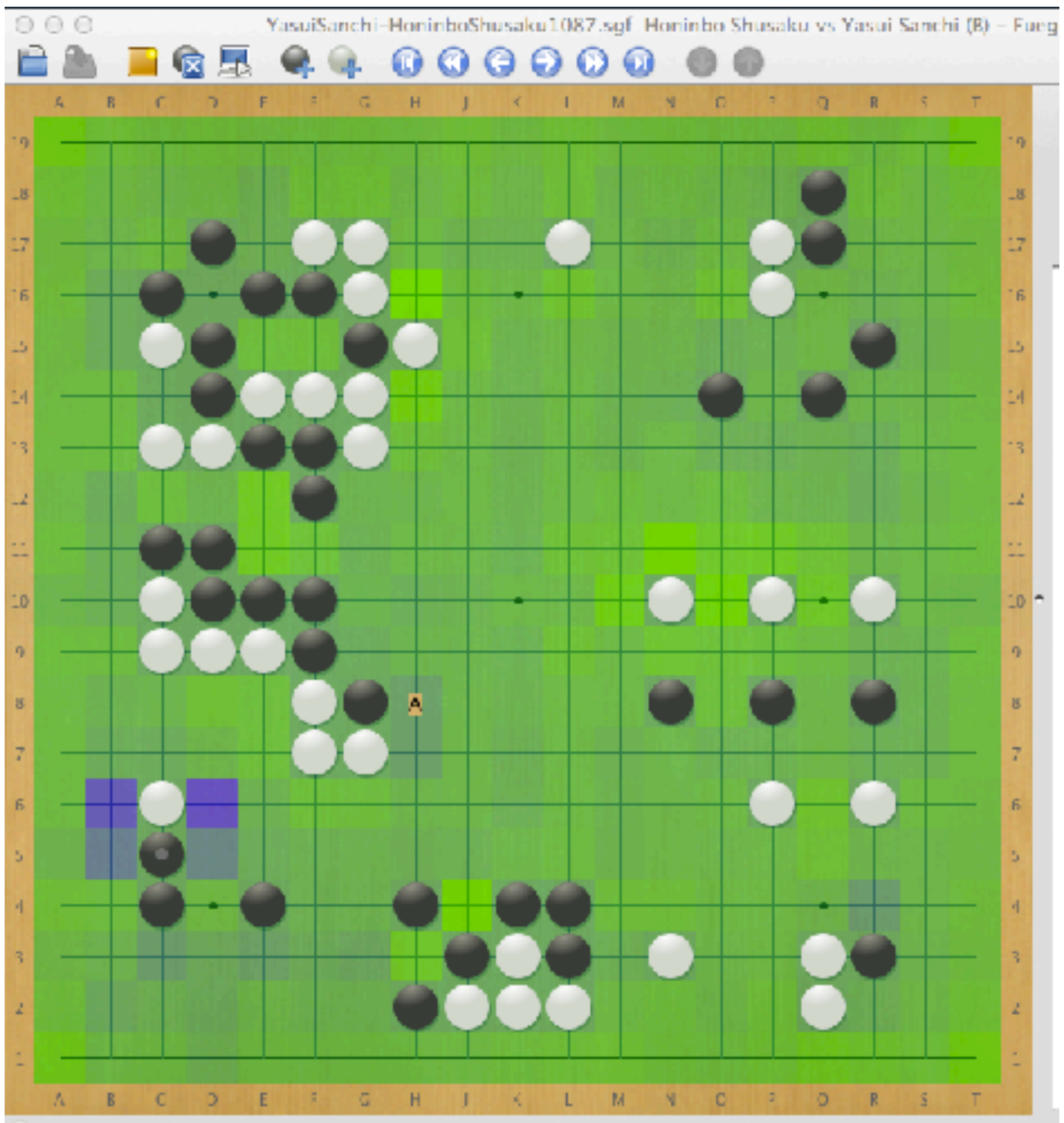
.....
0 16 21 80 85 117 122 136 1122
0 21 41 81 85 117 122 124 1127
0 21 40 82 85 117 122 1125
0 21 39 81 85 117 122 1134
0 21 38 80 85 117 122 1134
0 21 37 79 85 117 122 1134
0 21 36 78 85 117 122 1134
0 21 41 73 85 117 122 123 142
0 0
1 10 18 22 77 85 117 122 128 1883

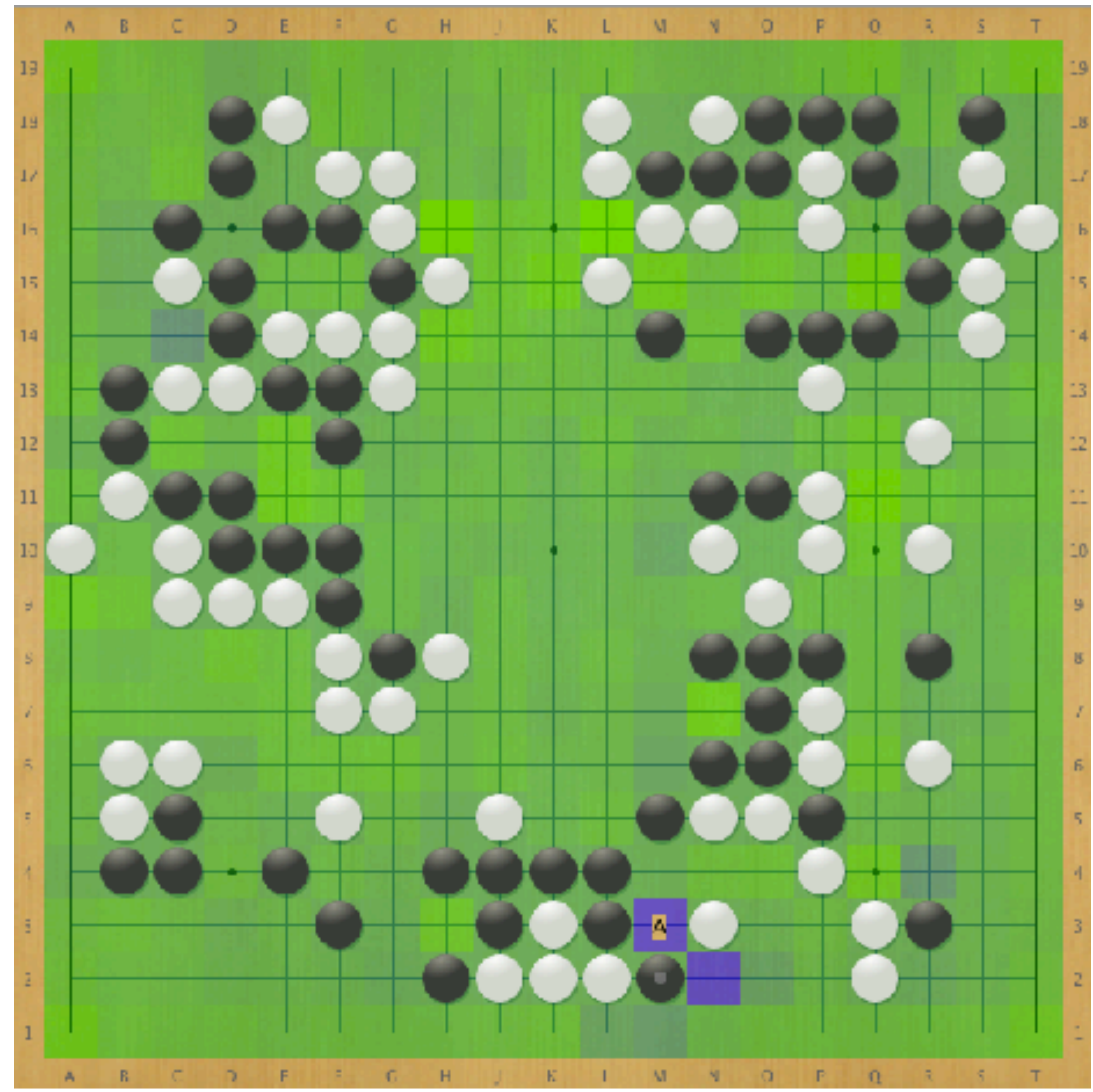
0 .. move not played
1 .. move played
16, 21, ... feature IDs

Training

- Total data: about 65GB
- Learn *model:* values for all features using *stochastic gradient descent*
- Use a *validation* set to check progress
  - 5-10% of data, kept separate
- Iterate over data until 3x no improvement
- Keep the model that does best on validation set
- Best result: about 39% move prediction

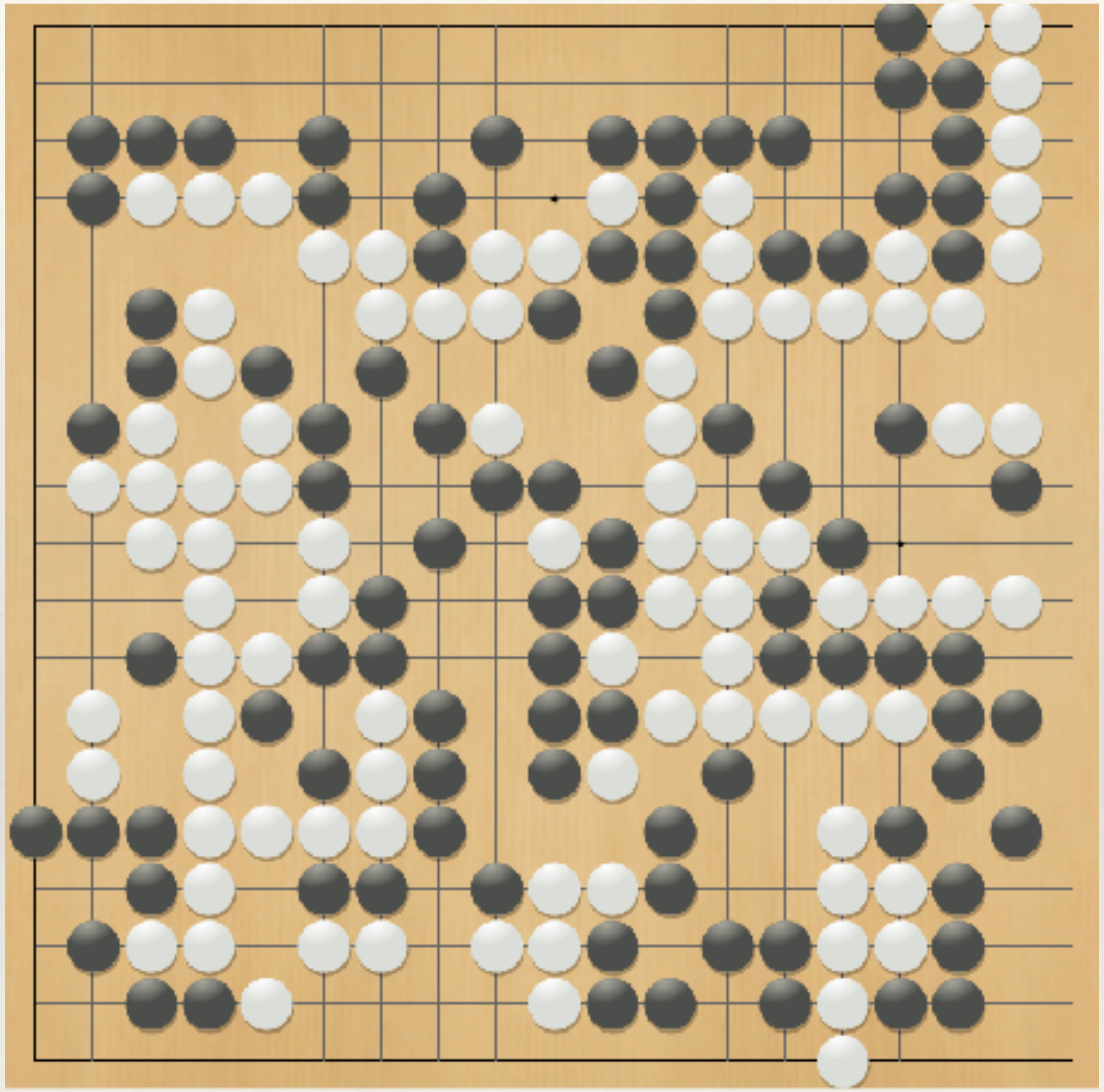
Examples


YasuiSanchi-HoninboShusaku1087.sgf Honinbo Shusaku vs Yasui Sanchi (B) - Fueg
Features Evaluate Board



Computer Go Before AlphaGo

- Summary of state of the art before AlphaGo:
- Search - quite strong
- Simulations - OK, but hard to improve
- **Knowledge**
  - Good for move selection
  - **Considered hopeless for position evaluation**



Who is better here?

Neural Networks (1)

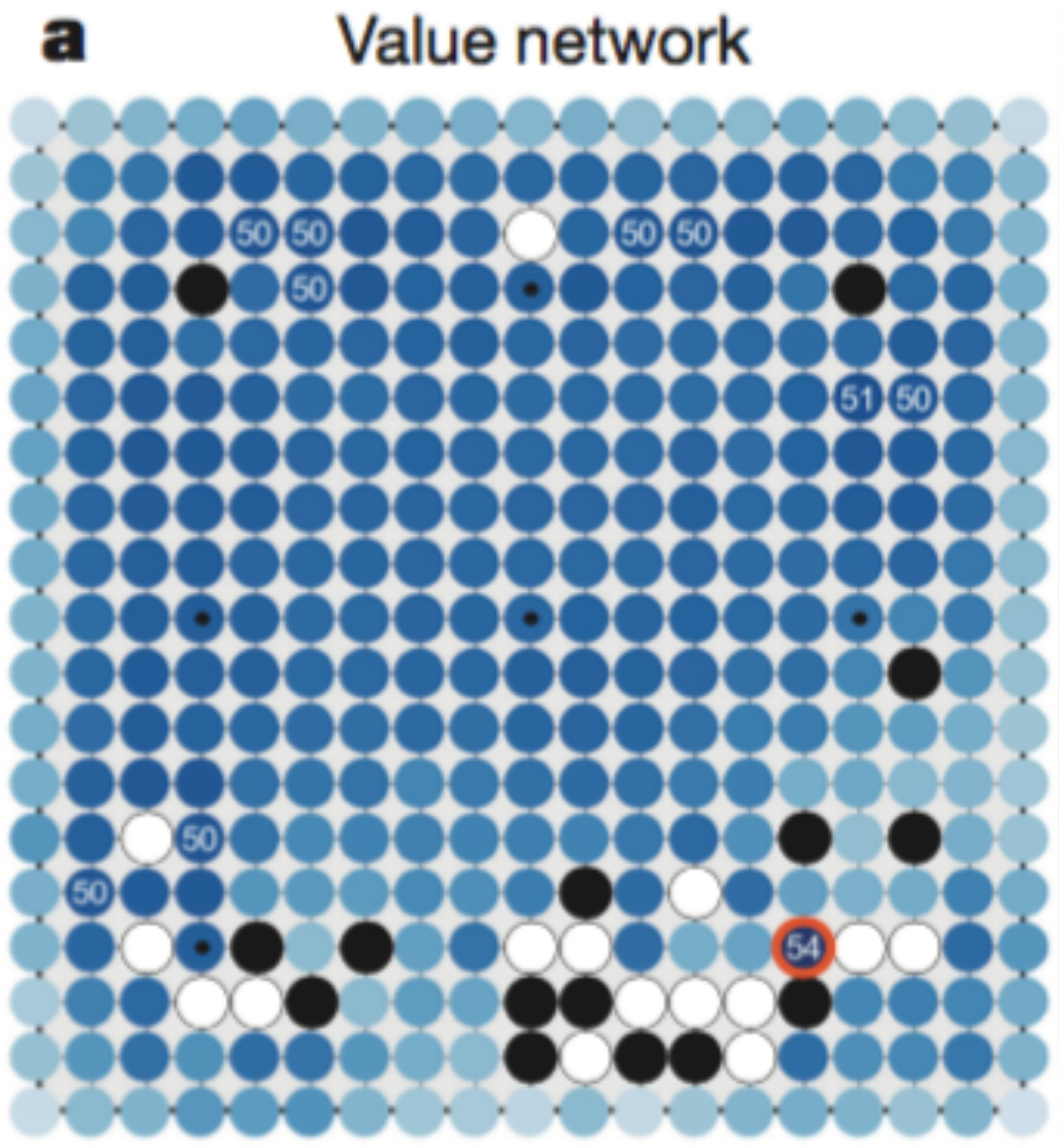
- Deep convolutional neural networks (DCNN)
- Large, multilayer networks
  - None of the limitations of simple features
  - Learn complex relations on the board
- Originally trained by supervised learning
- 2015: Human-level move prediction (57%)

Neural Networks (2)

- AlphaGo (2016)
- Start with supervised learning for DCNN
- Improve move selection by self-play and reinforcement learning (RL)
- Learned value network for evaluation
- Integrate networks in MCTS
- Beat top human Go player 4-1 in match

Value Network (2016)

- Given a Go position
- Computes probability of winning
- Static evaluation function
- Trained from millions of Go positions labeled with self-play game result (win, loss)
- Trains a deep neural network



AlphaGo Zero (2017)

- Learn Go without human knowledge
- Train by RL, only from self play
- Start with random play, continuously update neural net
- Train a single net for both policy and value

AlphaGo Zero Details

- Policy net is trained by running MCTS (!)
  - Move selection frequency mapped to probability
- MCTS: no more simulations!!!
  - Only in-tree phase
  - Evaluate leaf node by value net
  - Update value net from result at end of game
- Becomes stronger than previous AlphaGo

AlphaGo Zero Comments

- Architecture is a lot more elegant
- Strong integration of learning and MCTS
  - MCTS used to define the learning target for policy
  - MCTS uses thelearned net at every step
- Requires massive, Google-scale resources to train

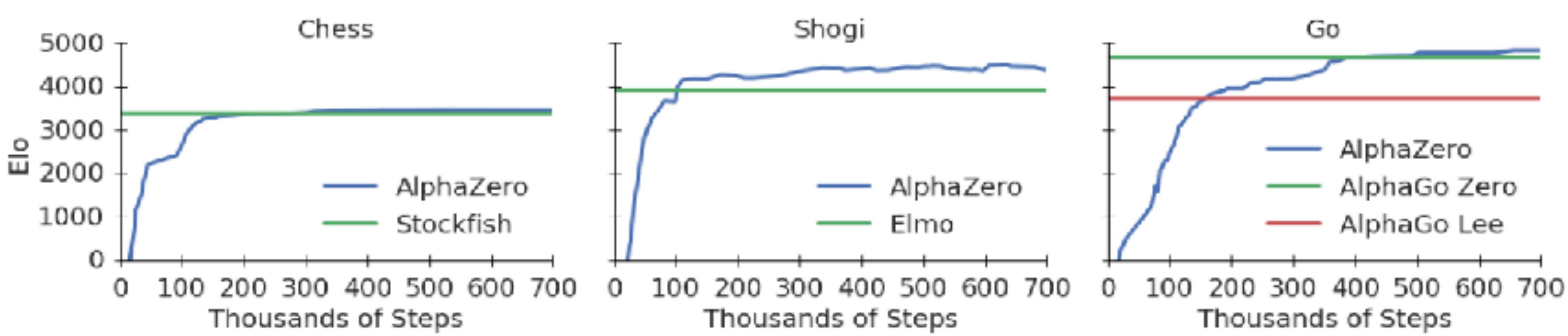
Alpha Zero

- Just published on arxiv, Dec 5, 2017
- Apply AlphaGo Zero approach to chess, shogi (Japanese chess)
  - Remove Go-specific training details
  - Simplify training procedure for network
- Learns to beat top chess, shogi programs
- Requires massive, Google-scale resources to train

Alpha Zero Results

Game	White	Black	Win	Draw	Loss
Chess	*AlphaZero*	*Stockfish*	25	25	0
	Stockfish	*AlphaZero*	3	47	0
Shogi	*AlphaZero*	*Elmo*	43	2	5
	Elmo	*AlphaZero*	47	0	3
Go	*AlphaZero*	*AG0 3-day*	31	–	19
	AG0 3-day	*AlphaZero*	29	–	21

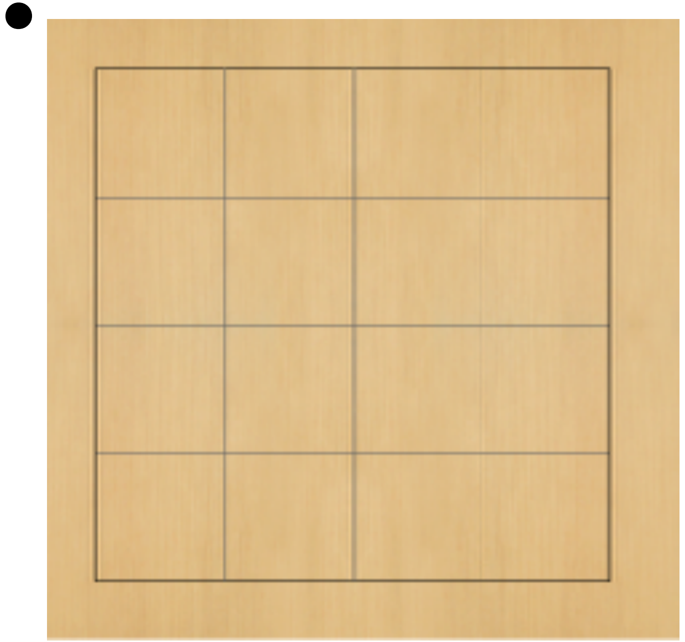
Table 1: Tournament evaluation of *AlphaZero* in chess, shogi, and Go, as games won, drawn or lost from *AlphaZero*'s perspective, in 100 game matches against *Stockfish*, *Elmo*, and the previously published *AlphaGo Zero* after 3 days of training. Each program was given 1 minute of thinking time per move.



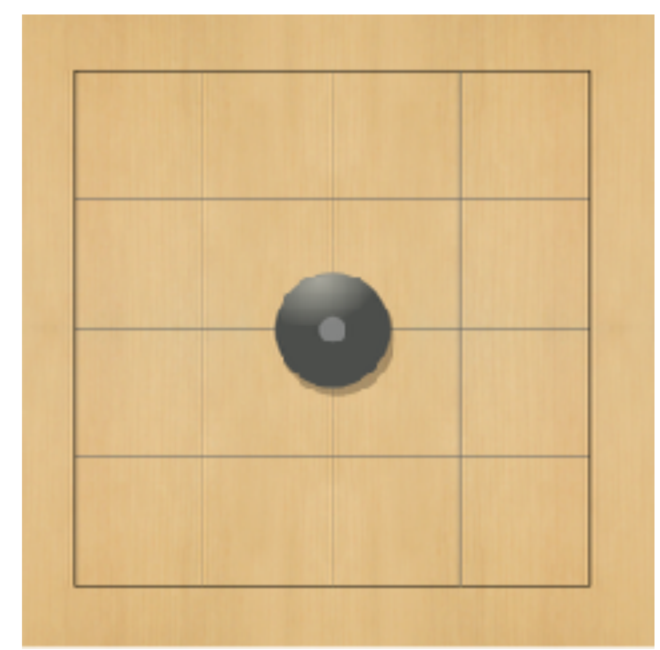
Where do we Go from Here?

- Which problems can we use this for?
- The methods are quite general,
  not game-specific
- We need an internal **model** of the problem in order to learn from self play
- Can we use similar approaches when we have lots of data to define an approaximate model?

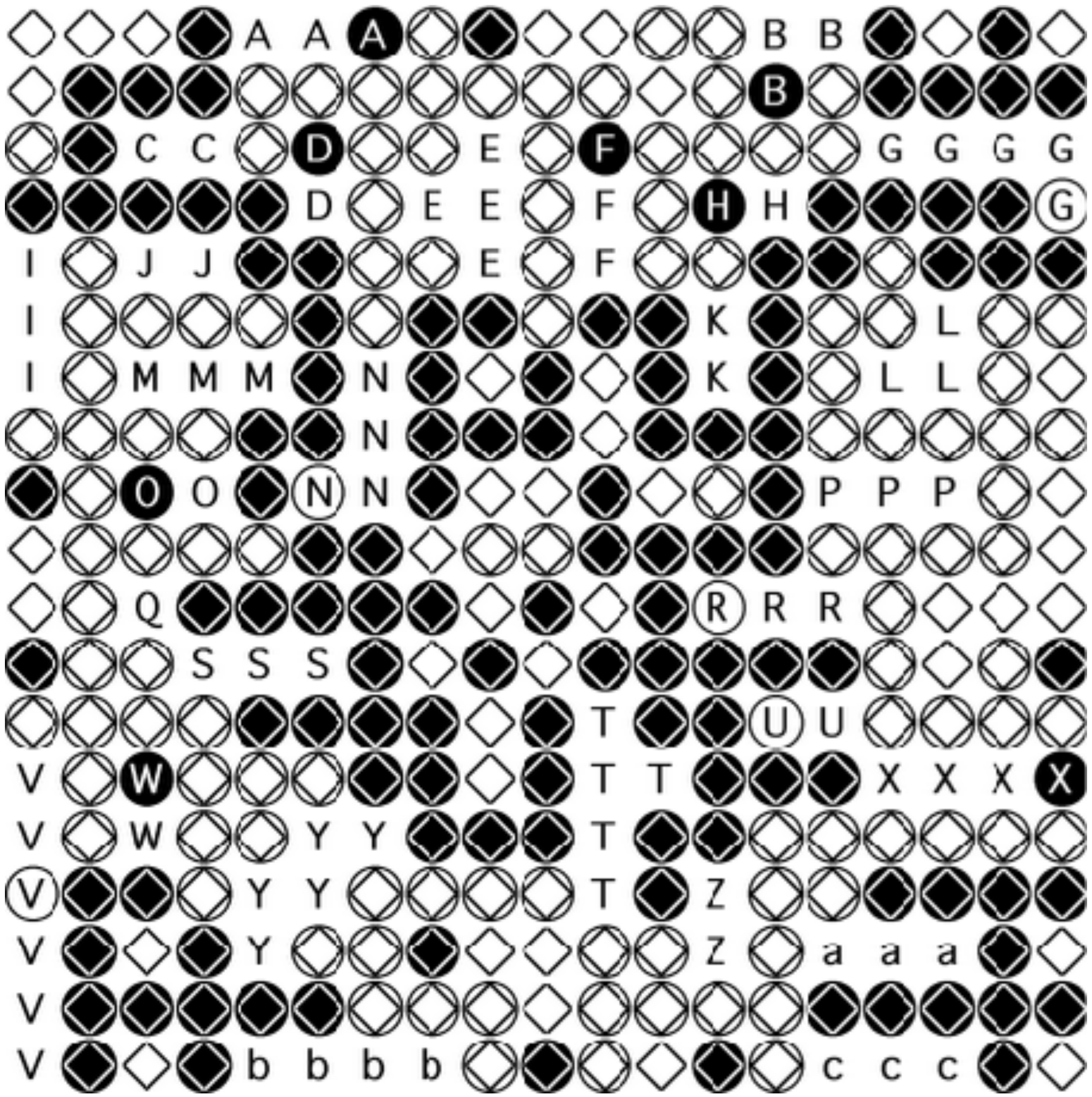
Is the Game of Go Solved Now?

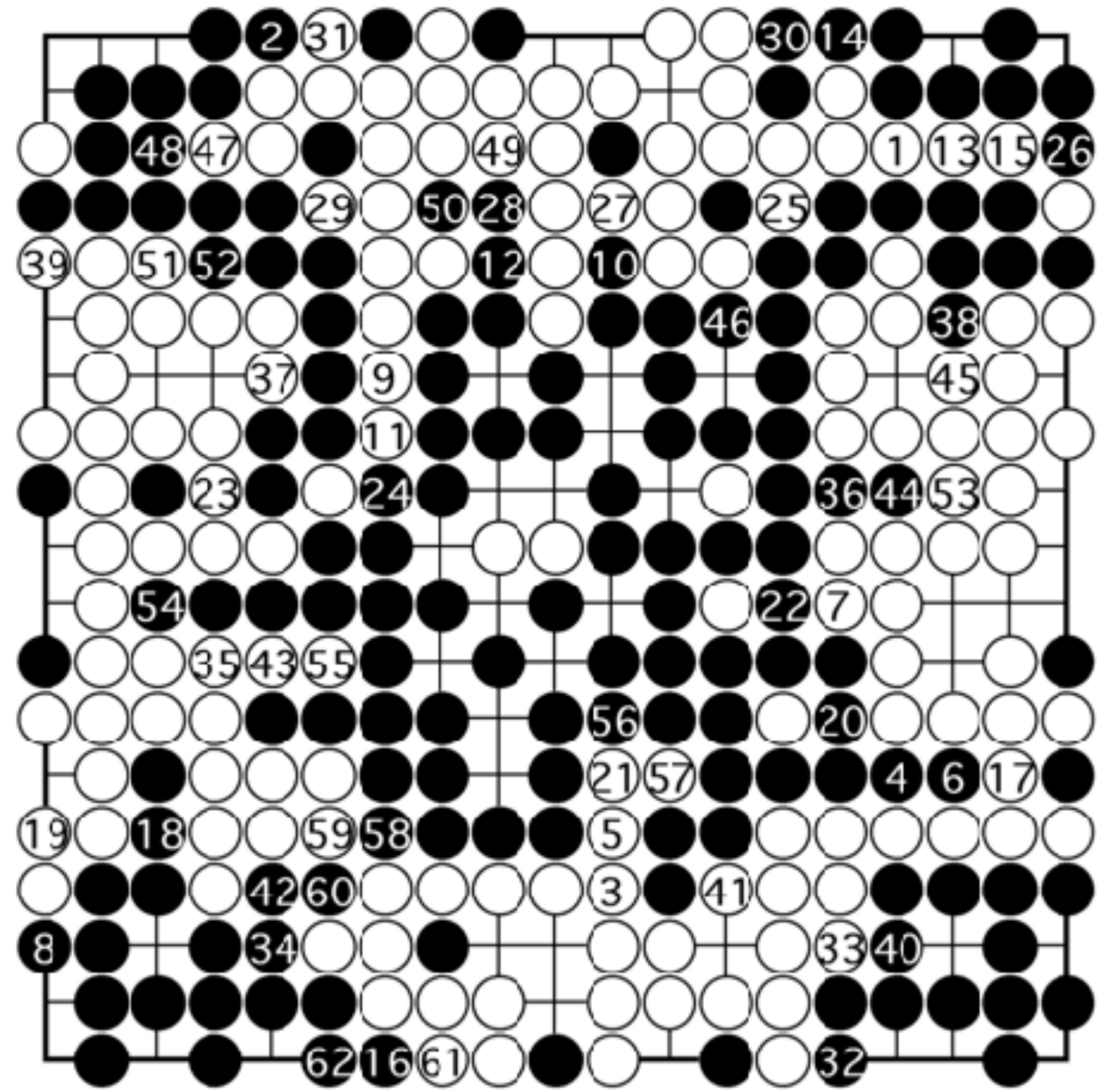


- **No!**
- AlphaGo is incredibly strong…
- But it is all heuristics
- AlphaGo still makes mistakes
- 5x5, 5x6 Go are solved
- Can play some full-board 19x19 puzzles perfectly using combinatorial game theory



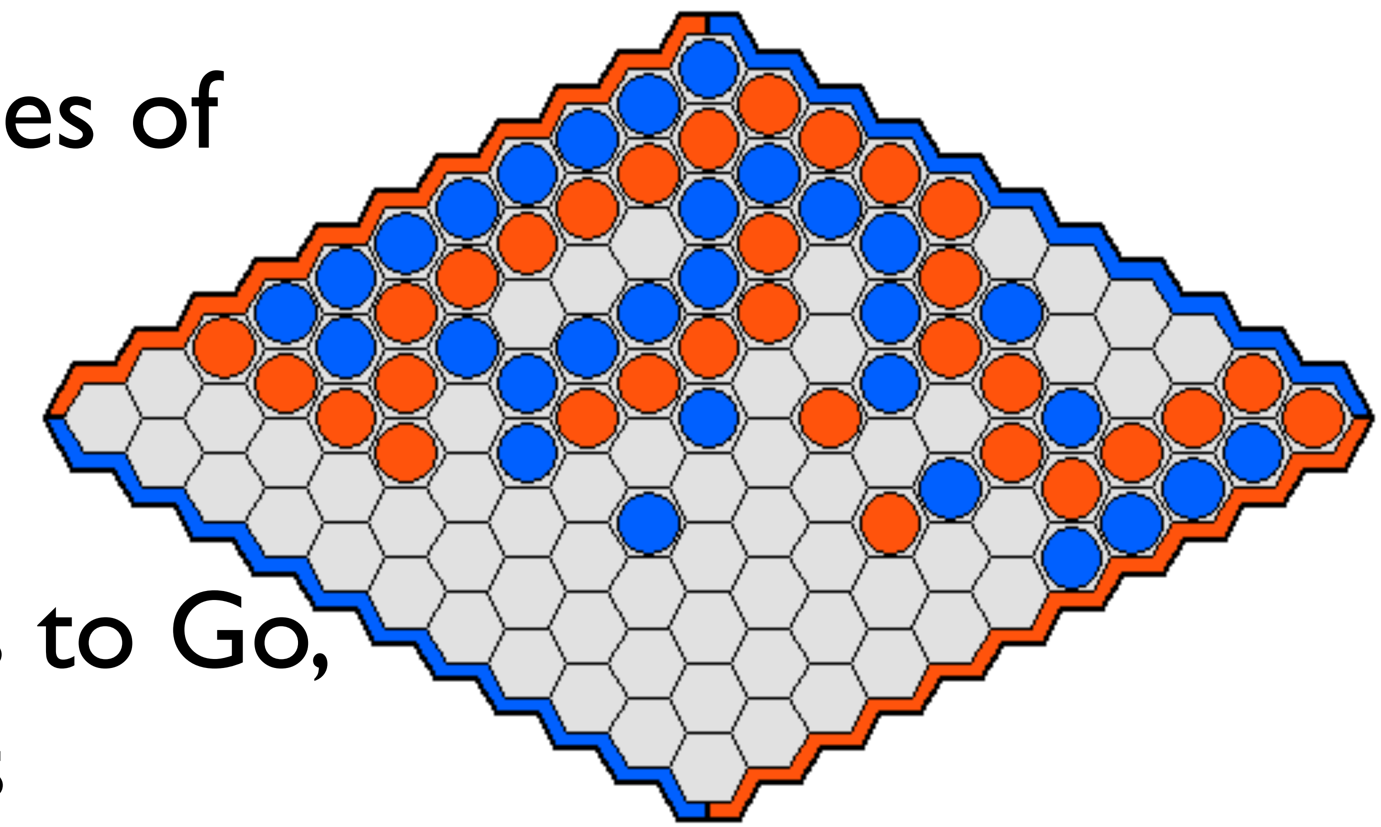
Solving Go Endgame Puzzles





Game of Hex

- Connect two sides of your own color
- No draws
- Some similarities to Go, some differences
- Very hard game of pure strategy



Red won

Image: https://ilk.uvt.nl/icga/games/hex/hex0m.gif

MoHex (1)

- **MoHex**: world's strongest Hex program
  - Developed by Ryan Hayward's group in Alberta
  - Open source
  - Won last four Computer Olympiads

MoHex (2)

Game-specific enhancements:

- Hard pruning - provably bad or inferior moves
- Very strong exact endgame solver - uses an search algorithm called depth-first proof-number search
- See https://webdocs.cs.ualberta.ca/~hayward/hex/

Learn more about modern heuristic search, MCTS and AlphaGo

- Course Cmput 496
- Search, Knowledge and Simulations
- From the basics to AlphaGo
- Second run starting Winter 2018
- Low math content, focus on concepts and code examples

Summary (1)

- Monte Carlo methods revolutionized heuristic search in games and planning
- Modern algorithms use all three: search, knowledge and simulation Except Alpha Zero…
- Machine learning to improve knowledge, e.g. feature learning, deep neural nets

Summary (2)

- Alpha Zero combines all these methods effectively - superhuman strength in Go, chess, shogi
- MCTS: Many *very* successful applications, still not well understood in general
- Newest development: tightly integrate search and deep learning
- Future challenge: extend to exact solutions?