Reinforcement Learning in Classic Board Games

David Silver

Plan

I. State of the Art

II. Minimax search

III. Reinforcement learning and minimax search

IV. Case Studies: Alpha-Beta Search

V. Case Studies: Monte-Carlo search

VI. Conclusion: Similarities and Lessons

Part I:

State of the Art

Why Classic Board Games?

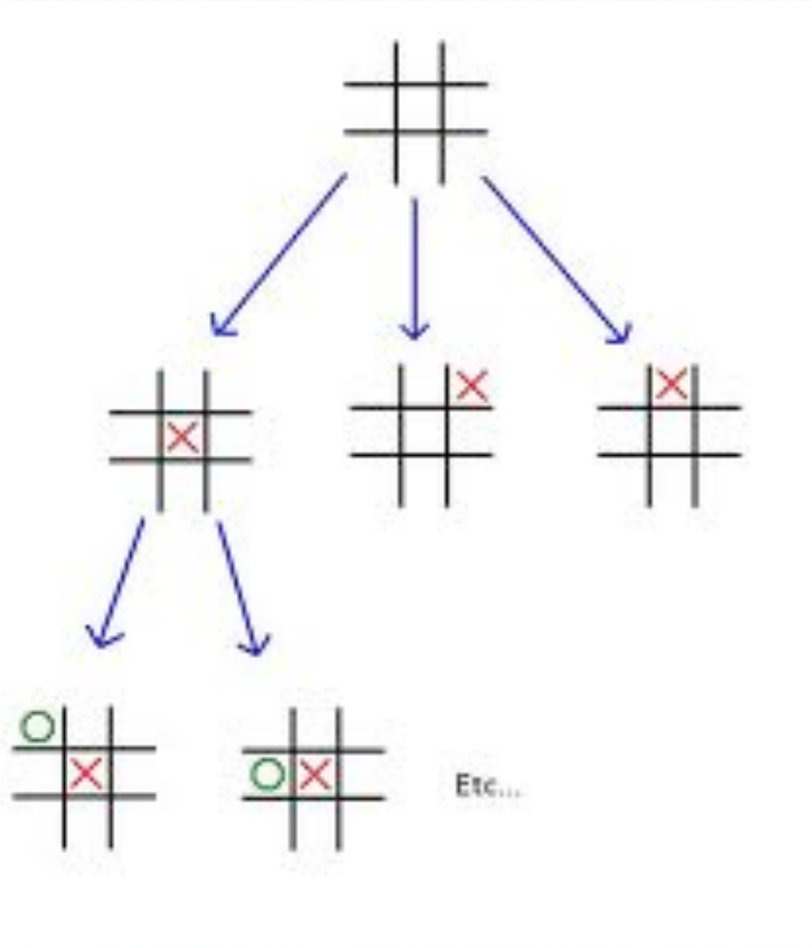
- Simple rules, deep concepts
- Studied for hundreds (or thousands) of years
- Meaningful IQ test - e.g. Elo rating in chess
- *Drosophila* of artificial intelligence
- Microcosms encapsulating real-world issues
- **Games are fun!**

AI in Games: State of the Art

	Level of Play	First Program at this Level
Checkers	**PERFECT**	*Chinook*
Chess	**WORLD CHAMPION**	*Deep Blue*
Othello	**WORLD CHAMPION**	*Logistello*
Backgammon	**WORLD CHAMPION**	*TD-Gammon*
Scrabble	**WORLD CHAMPION**	*Maven*
9x9 Go	**GRANDMASTER**	*MoGo*
19x19 Go	**MASTER**	*Zen*

Search in Games

- These success rely heavily on game-tree search
- The idea is to use *lookahead* to evaluate positions effectively



Reinforcement Learning in Games

- High performance has been achieved in many games
  - Training by self-play
  - Learning an effective strategy
  - Based only on feedback of win or loss
- This paradigm is called *reinforcement learning*

RL and Search

- RL and search have produced remarkable successes
- We will see examples of:
  - Reinforcement learning *from* search
  - Reinforcement learning *for* search
  - Reinforcement learning *as* search

RL in Games: State of the Art

	Level of Play	First Program at this Level
Checkers	**PERFECT**	*Chinook*
Chess	**INTERNATIONAL MASTER**	*Knightcap / Meep*
Othello	**WORLD CHAMPION**	*Logistello*
Backgammon	**WORLD CHAMPION**	*TD-Gammon*
Scrabble	**WORLD CHAMPION**	*Maven*
9x9 Go	**GRANDMASTER**	*MoGo*
19x19 Go	**MASTER**	*Zen*

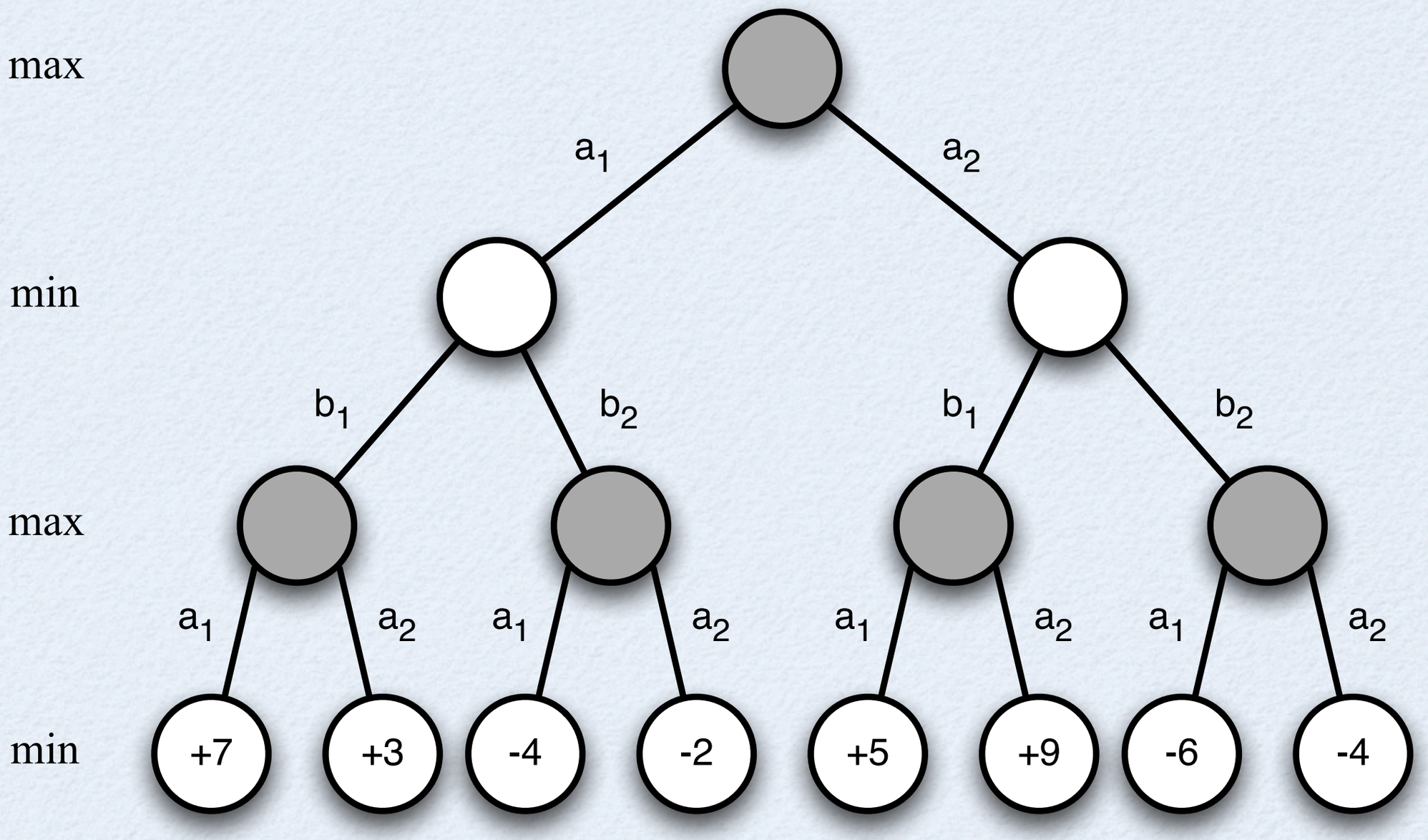
Part II

Minimax Search

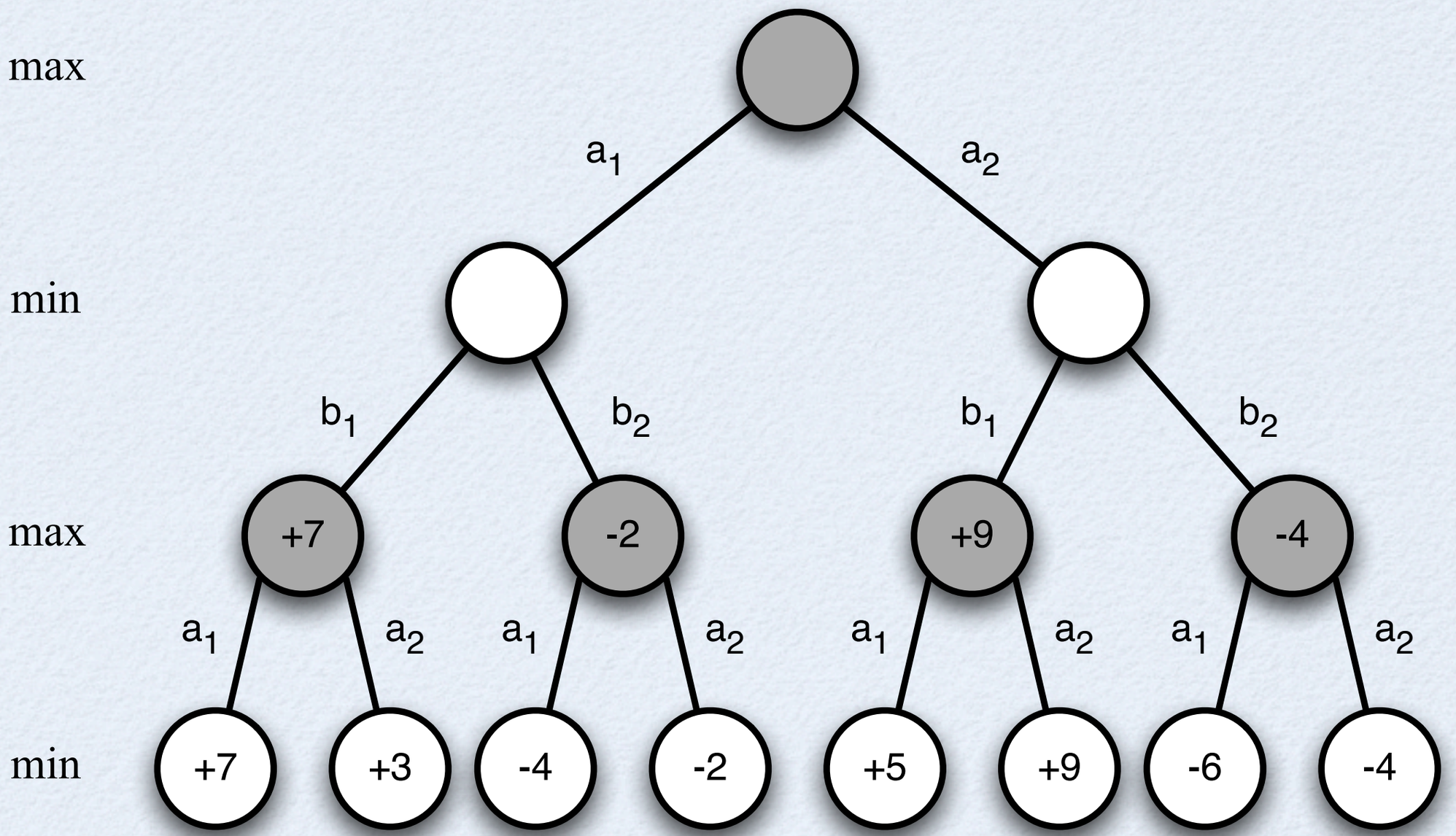
Minimax Search

- First minimax search algorithm
- Claude Shannon, father of information theory
- “Programming a Computer for Playing Chess”
- Ran on paper!

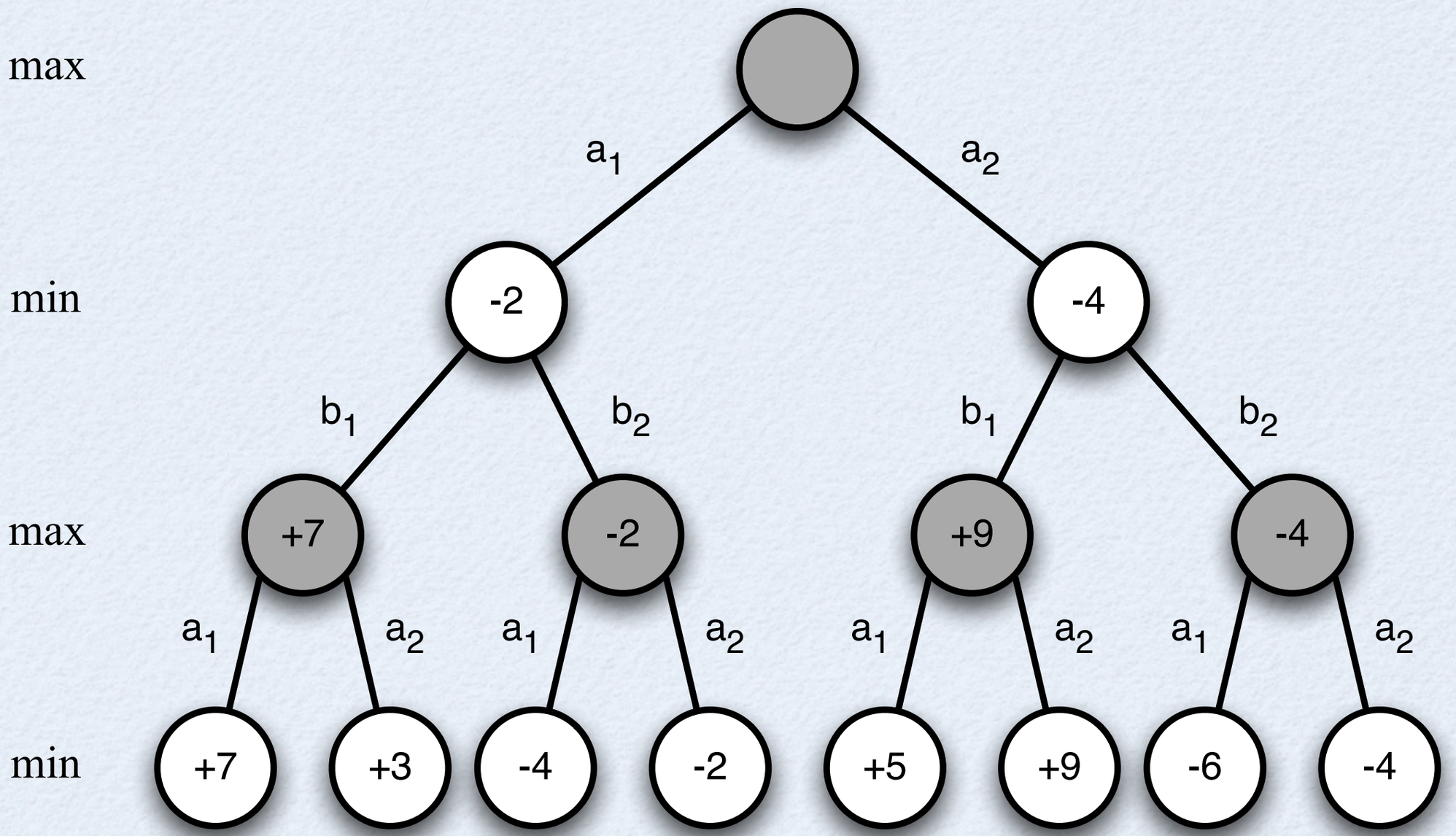
Minimax Search



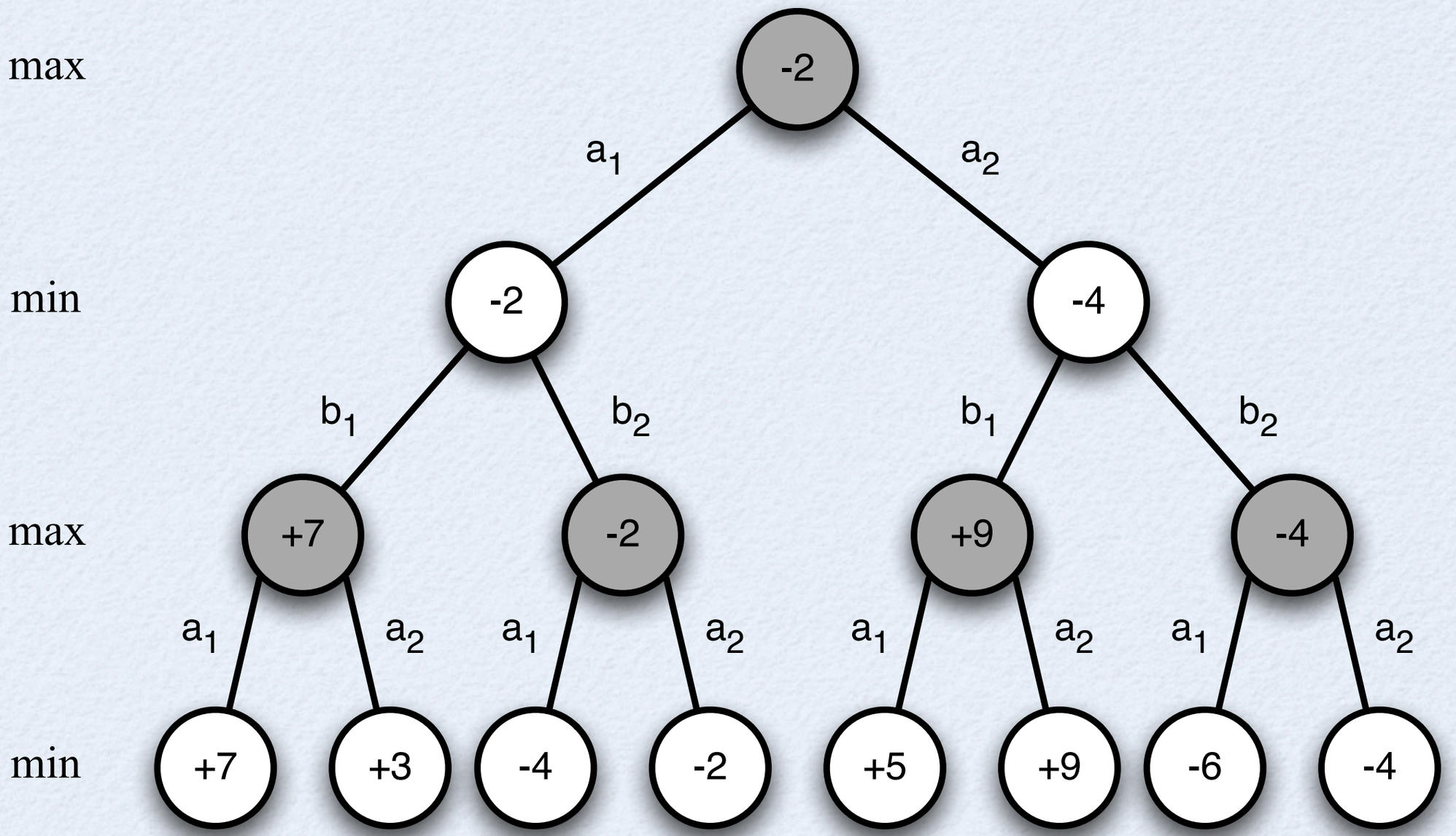
Minimax Search



Minimax Search

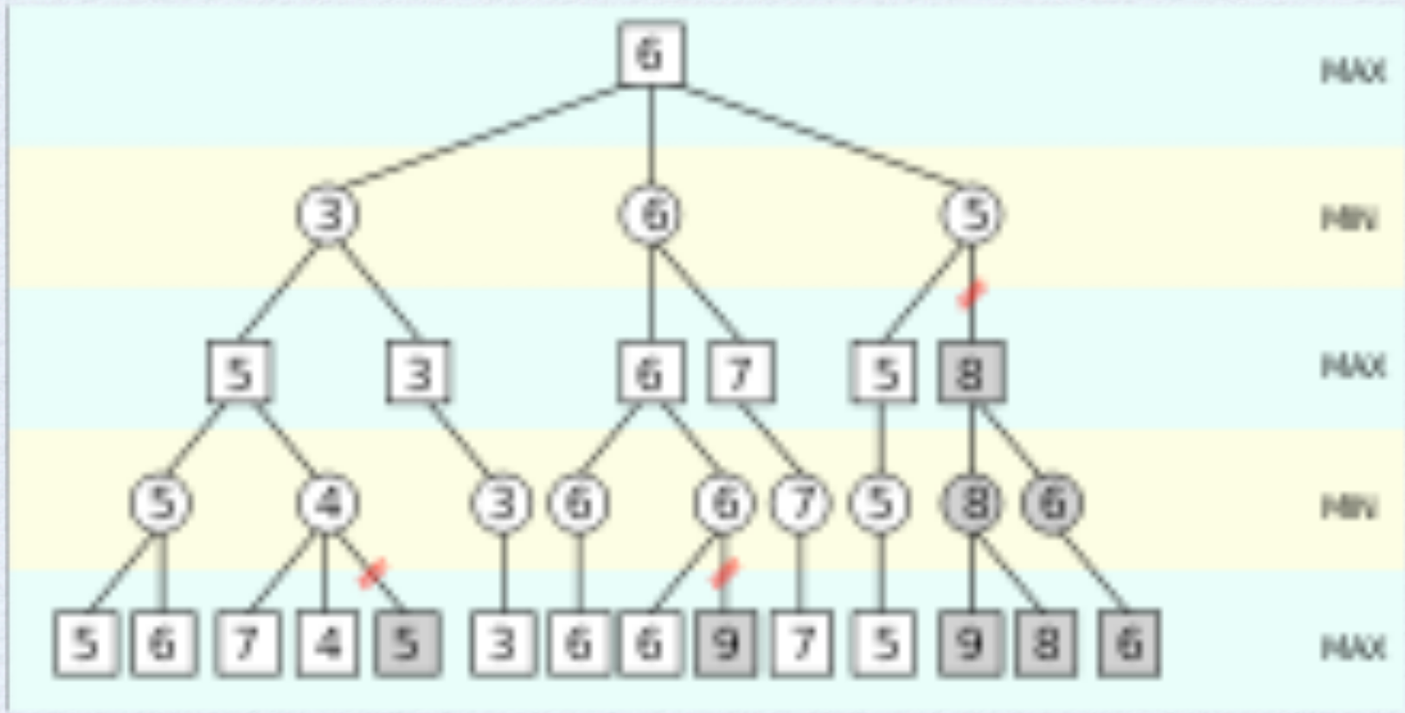


Minimax Search



Alpha-Beta Search

- Depth-first search
- Branch and bound pruning
- Maintain window [α, β]
- Prune any node outside window
- Doubles the search depth with optimal ordering



Alpha-Beta Search Enhancements

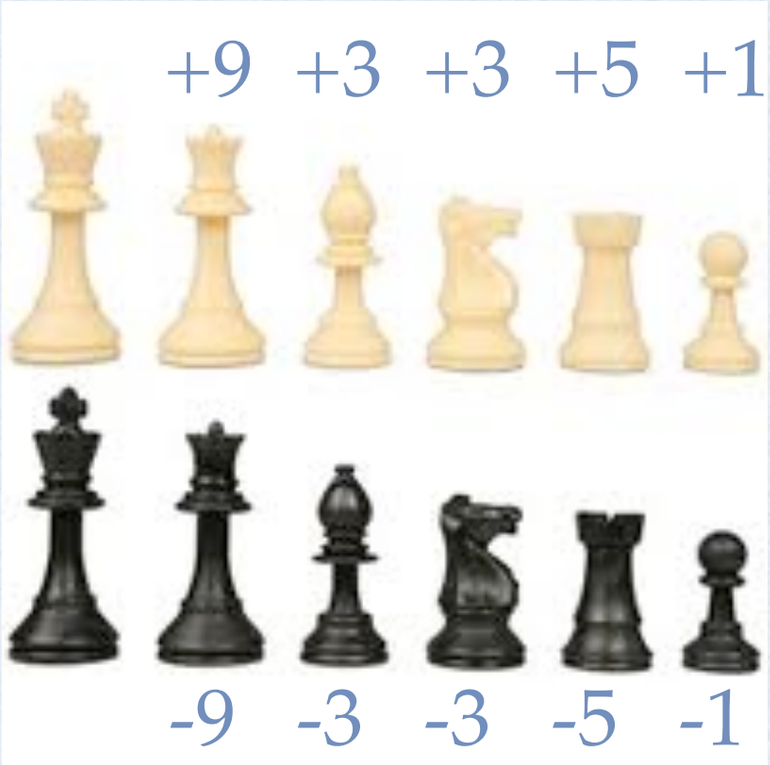
- Transposition tables
- Move ordering, iterative deepening
- Aspiration windows
- Forward pruning

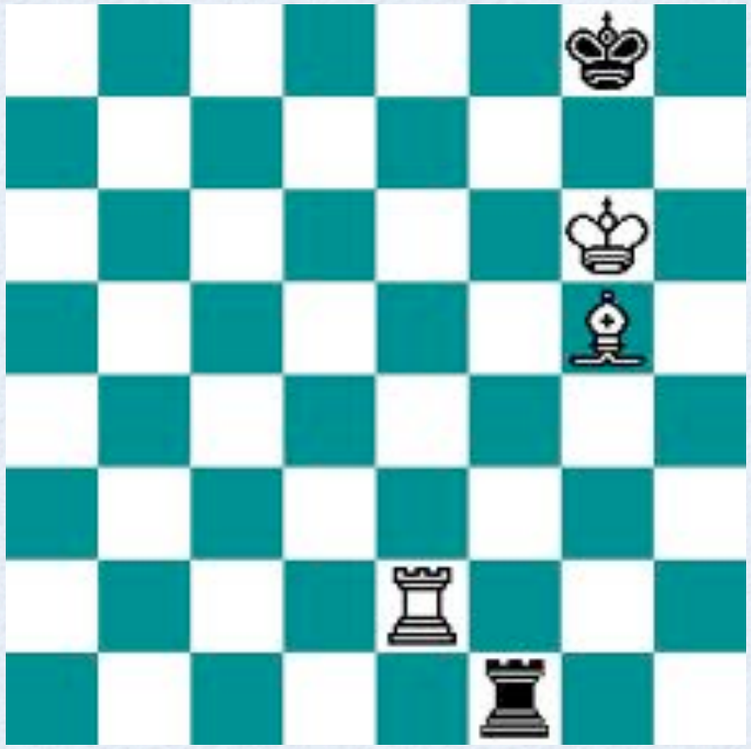
Evaluation Function

- Impractical to search to the end of the game
- *Evaluation function* estimates value of leaf nodes
- Minimax search performed with respect to leaf values

How to Evaluate a Chess Position

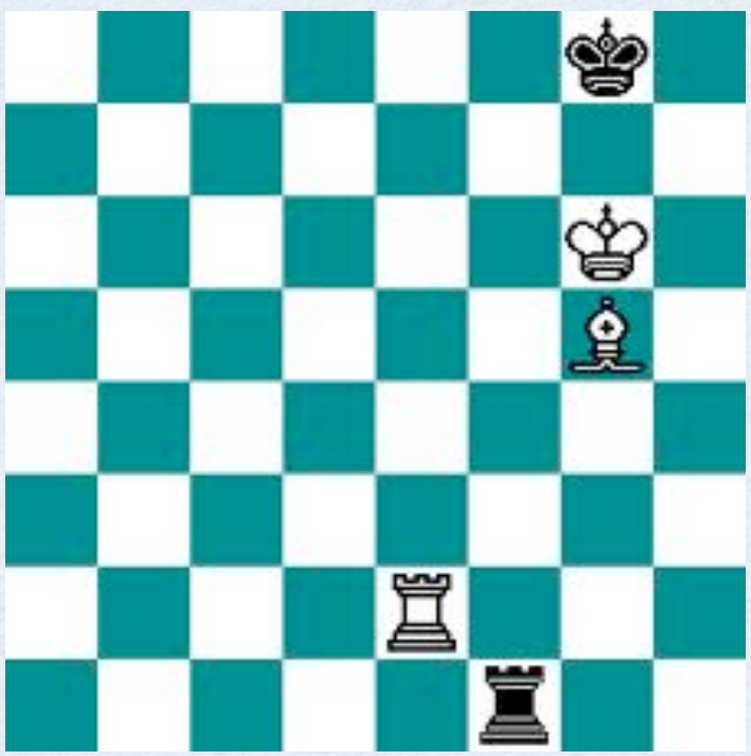
- Beginners evaluate position by giving each piece a value





- Sum up values of pieces in position

Binary-Linear Evaluation Function

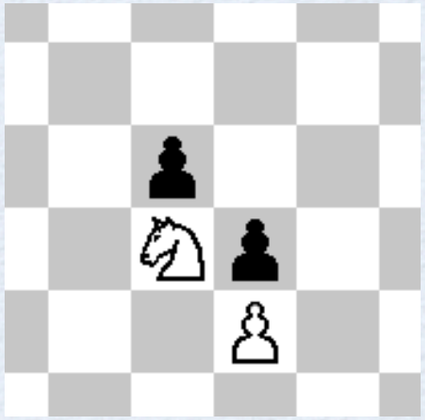


$$V(s) = \phi(s) \cdot \theta = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix} \cdot \begin{bmatrix} +5 \\ +3 \\ +1 \\ -5 \\ -3 \\ -1 \\ \vdots \end{bmatrix}$$

$V(s) = +5 +3 -5 = +3$

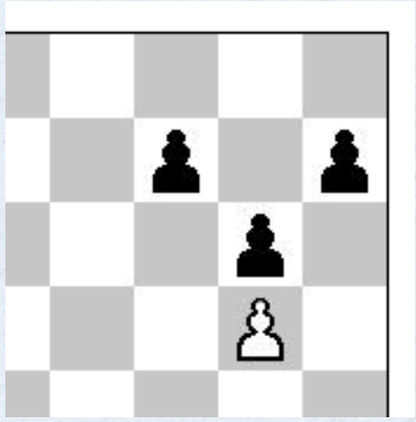
Sophisticated Evaluation Function

- Experts evaluate position using sophisticated features
  - Knight on outpost



+0.5

  - Weak kingside pawn structure



+0.7

- e.g. Deep Blue evaluates position using 8000 features

Generalised Linear Evaluation

- Linear evaluation function sufficient for move selection
- But inappropriate for learning
- Reward: +1 for win, 0 for loss

$$V(s) = \mathbb{E}[r|s] = \mathbb{P}(black\ win|s)$$

$$V(s) = \sigma(\phi(s)^\top \theta)$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Knowledge Acquisition Bottleneck

- Humans may evaluate positions based on many complex features, but:
  - Hard to extract human features, $\phi(s)$
  - Hard to quantify human weights, θ
  - Hand-tuning weights is laborious and error-prone
- How can we defeat knowledge-acquisition bottleneck?

Part III

RL and Minimax Search

Binary Linear TD(0) in Games

After every move t

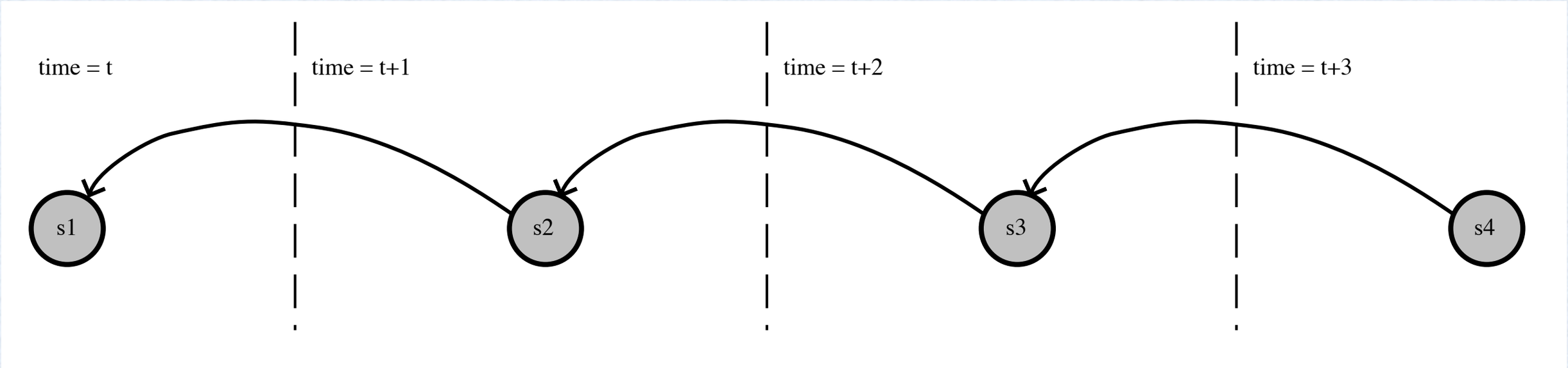
$$\delta_t = V(s_{t+1}) - V(s_t)$$

For all j s.t. $\phi_j(s_t) = 1$

$$\theta_j \leftarrow \theta_j + \alpha\delta$$

Standard *TD(0)*

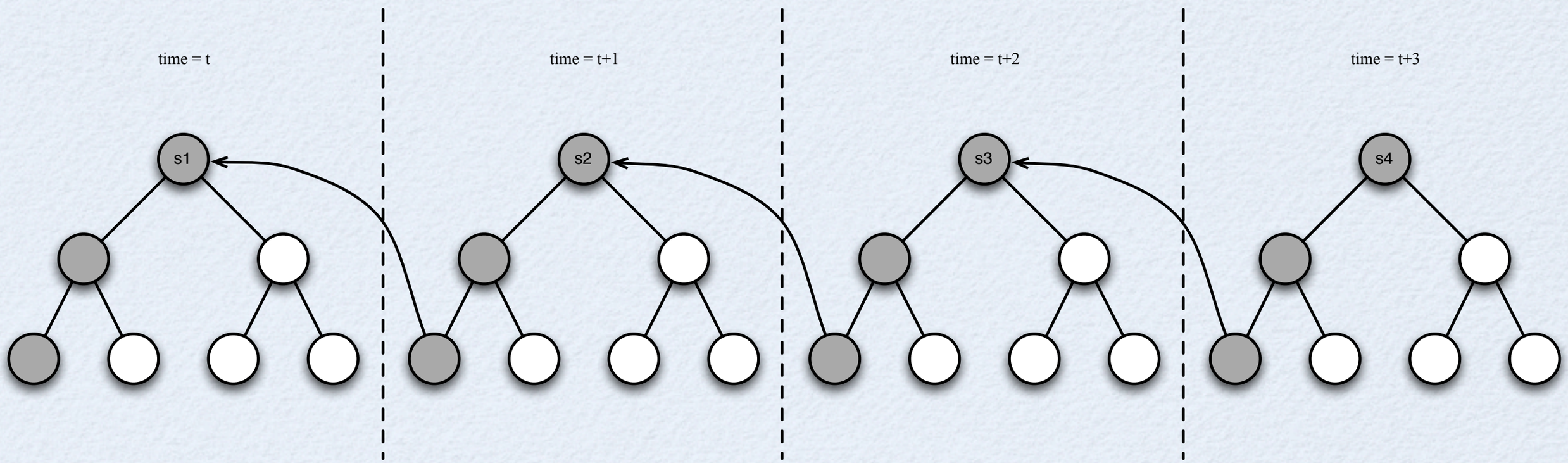
$$V(s_t) \leftarrow V(s_{t+1})$$



Performs poorly in chess and checkers

TD(Root): Learning from Search

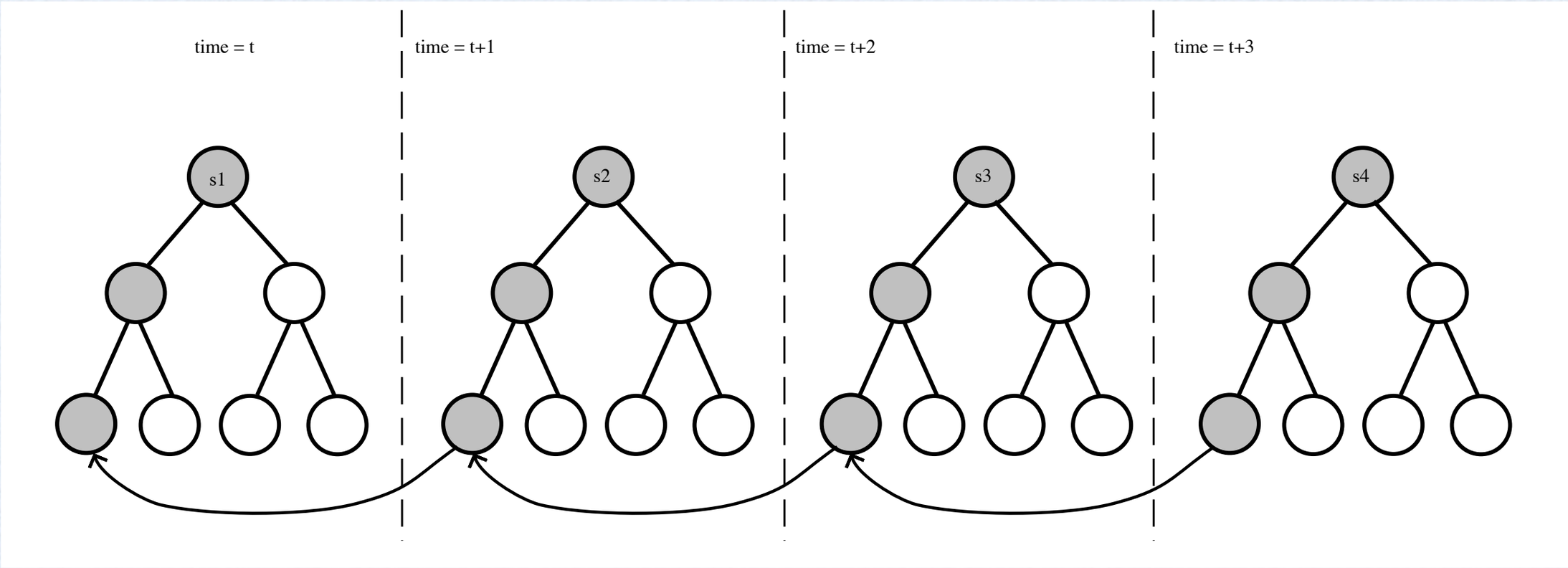
$$V(s_t) \leftarrow V^*(s_{t+1})$$



First strong program in checkers (*Samuel 1959*)

TD(Leaf): Learning to Search

$$V^*(s_t) \leftarrow V^*(s_{t+1})$$

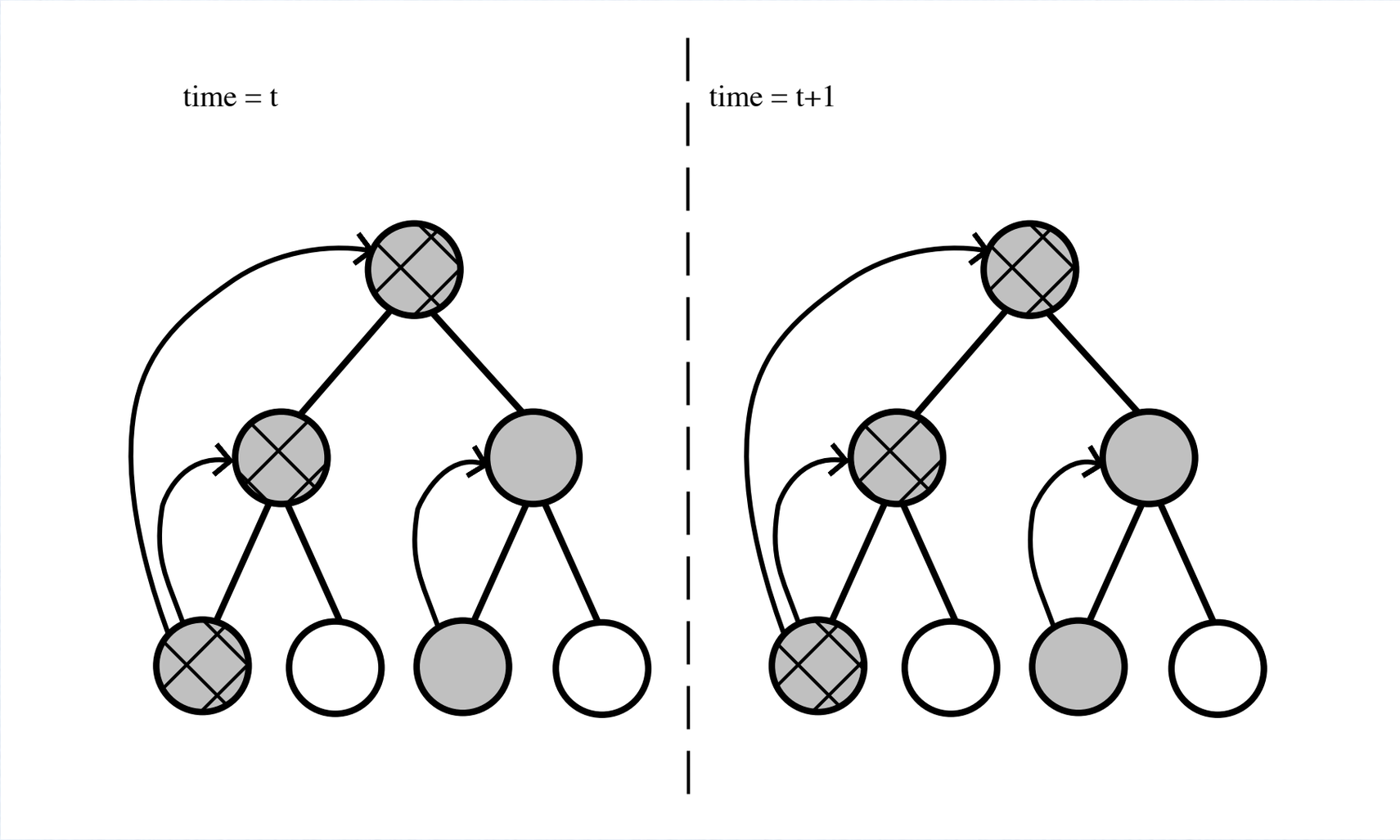


Master-level chess with caveats (*KnightCap)*

World-champion level checkers *(Chinook)*

TreeStrap: Learning from All Searches

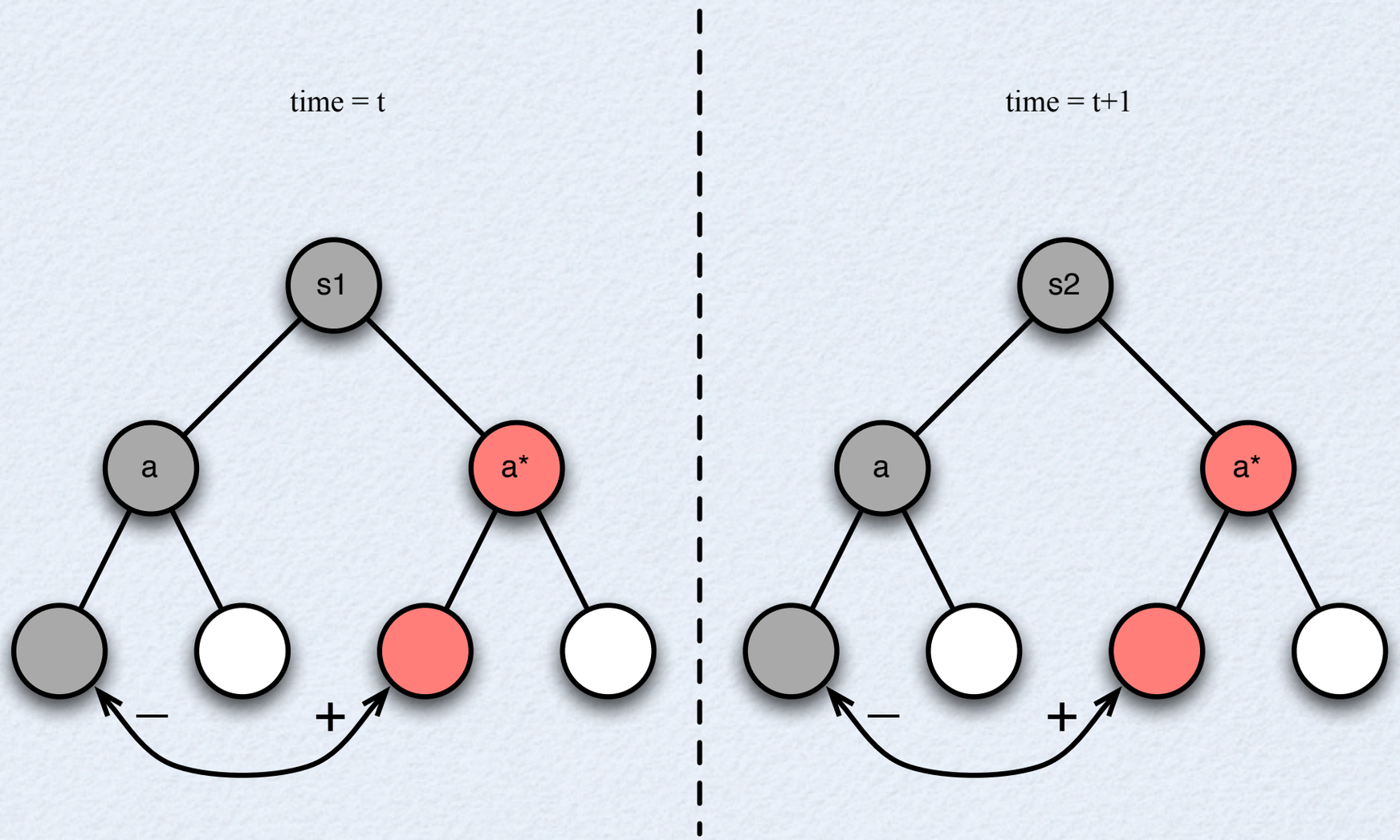
$$V(s) \leftarrow V^*(s) \qquad \forall s \in tree$$



Master-level chess without caveats (*Meep)*

Comparison Training (Supervised)

$$V^*(s \circ a) \leftrightarrow V^*(s \circ a^*)$$



Partial world-champion level chess (*Deep Blue)*

Grandmaster-level shogi *(Bonanza)*

Part IV

Case Studies: Alpha-Beta Search

Chess

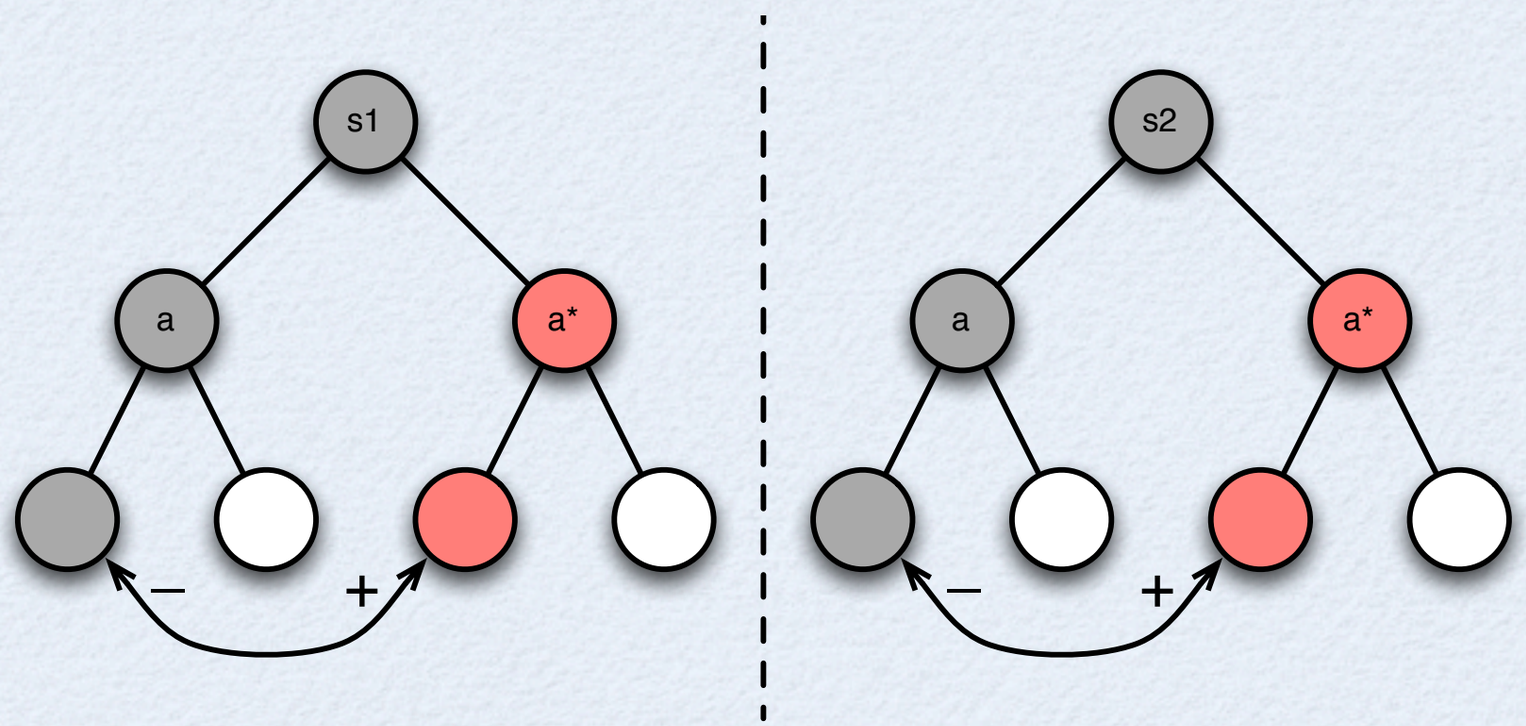
Deep Blue, KnightCap, Meep

Chess

- ~10^{47} unique positions
- ~35 branches
- ~80 moves long
- ~10^{123} complexity

Chess Evaluation in Deep Blue

- Deep Blue used 8000 handcrafted chess features
- Binary-Linear evaluation function
- Weights initially hand-tuned by human experts
- Some weights fine-tuned by comparison training



Deep Blue

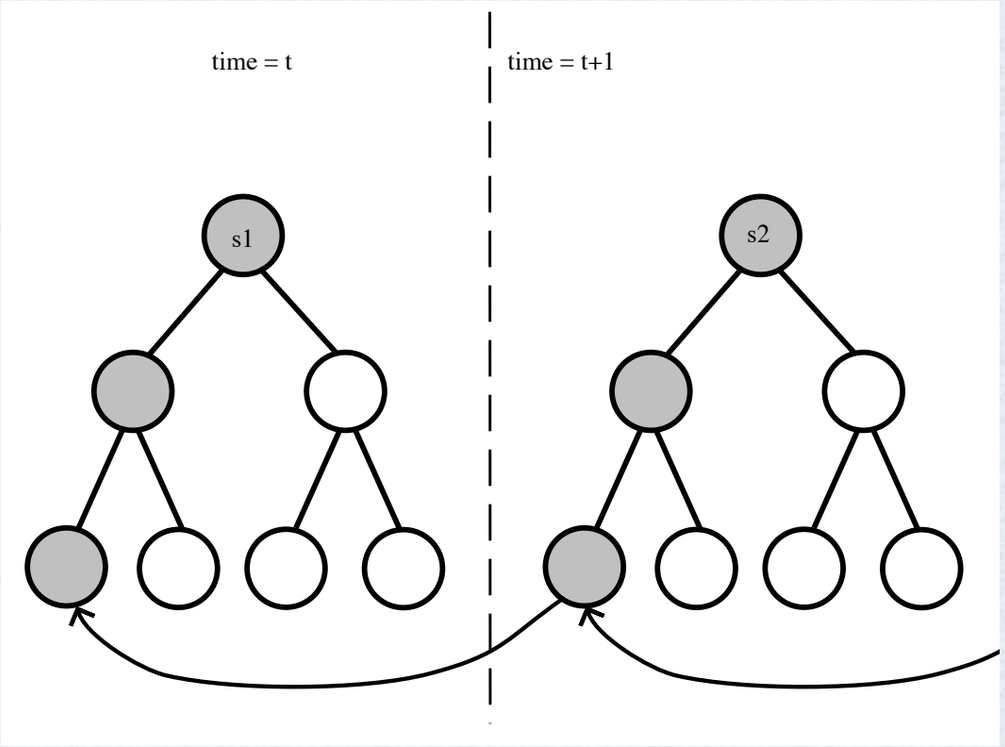
- High performance parallel alpha-beta search
- 480 special-purpose VLSI chess processors
- Searches 200 million positions/second
- Looks ahead 16-40 ply

Deep Blue vs. Kasparov

- Defeated human champion Garry Kasparov 4-2 (1997)
- Kasparov made uncharacteristic errors
- Most watched event in internet history

TD Leaf in Chess: *KnightCap*

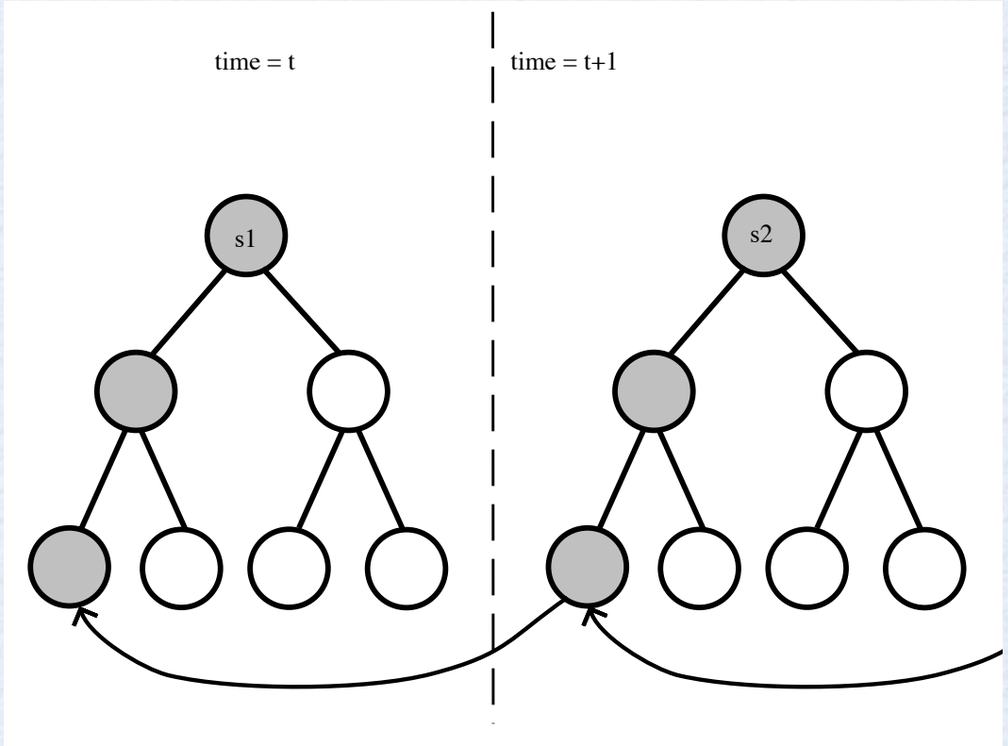
- *Knightcap* trained against expert opponent
- Starting from standard piece values only
- Learnt weights using *TD(leaf)*
- Achieved master level play



$$V^*(s_t) \leftarrow V^*(s_{t+1})$$

TD Leaf in Chess: *KnightCap*

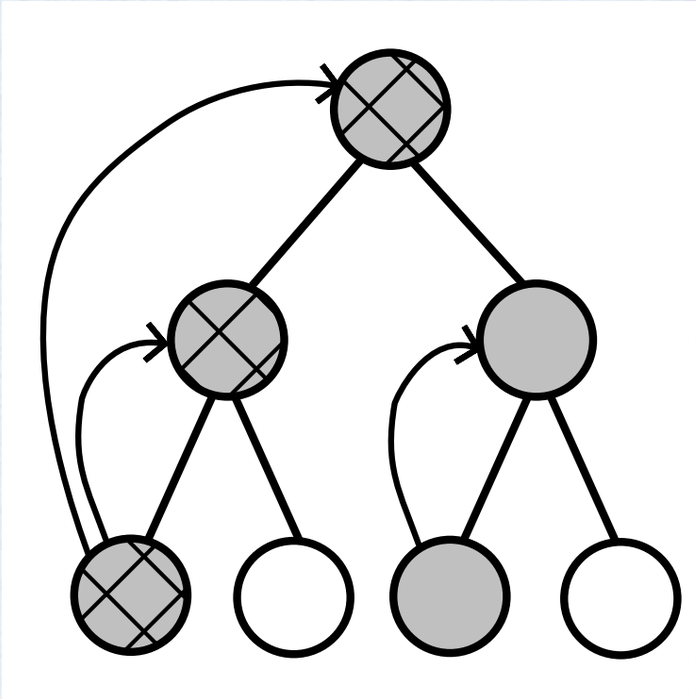
- TD-leaf is a *mixed* backup
  - Full-width search
  - Sample-based moves
- Was not effective in self-play
- Was not effective without starting from good weights



$$V^*(s_t) \leftarrow V^*(s_{t+1})$$

TreeStrap in Chess: *Meep*

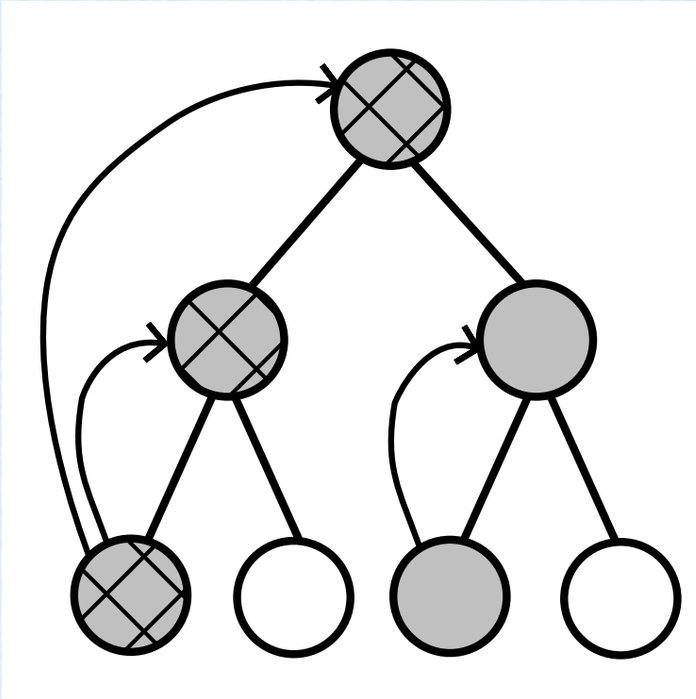
- Binary linear evaluation function with 2000 features
- Starting from random initial weights (no prior knowledge)
- Weights adjusted by *TreeStrap*
- Won 13/15 vs. international masters



$$V(s) \leftarrow V^*(s)$$

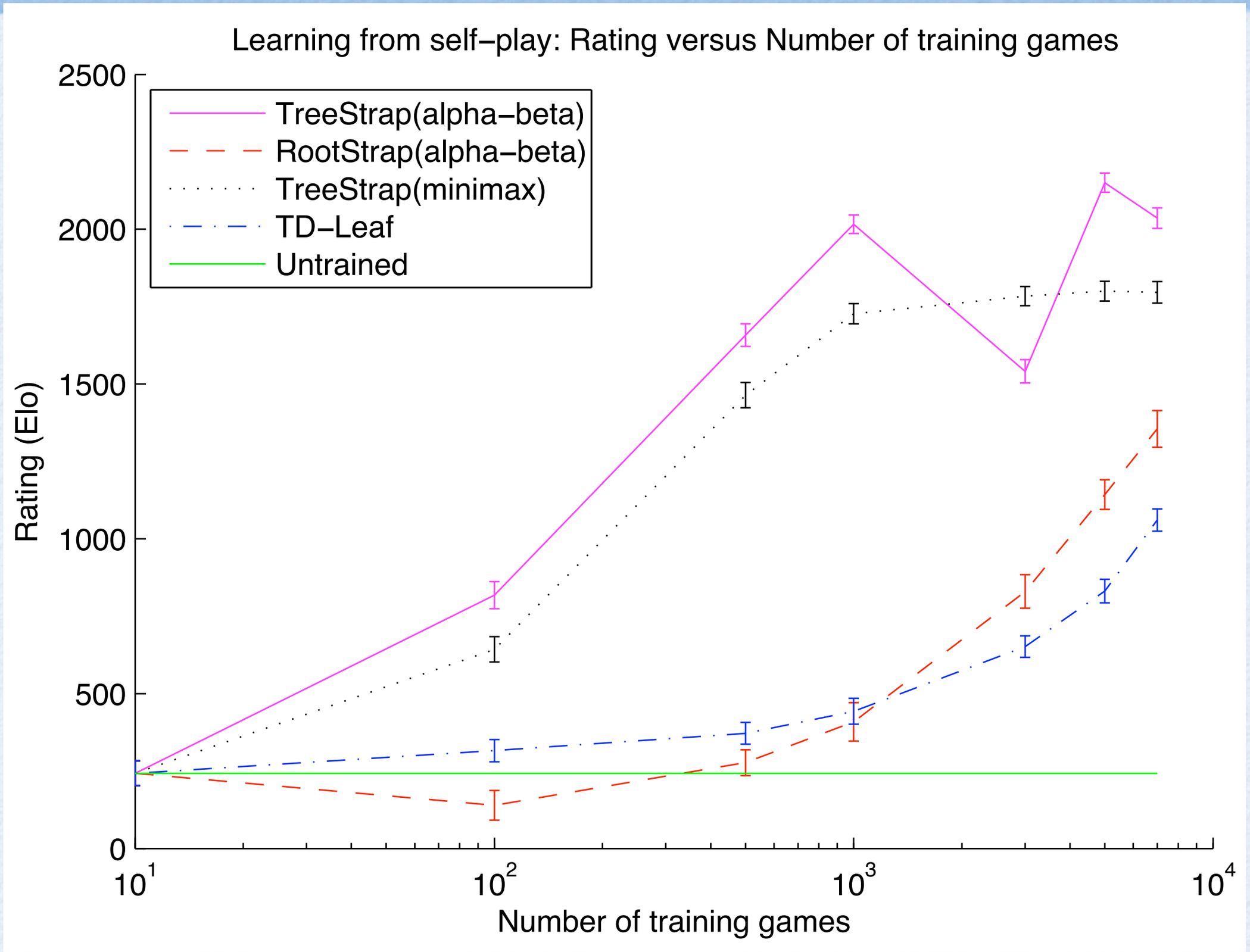
RL in Chess: *Meep*

- *TreeStrap* uses full-width backups
  - Not a *mixed* backup
  - Trains on positions seen in search
  - Uses search data more efficiently
- Effective in self-play
- Effective from poor initial weights



$$V(s) \leftarrow V^*(s)$$

Meep Algorithm Comparison

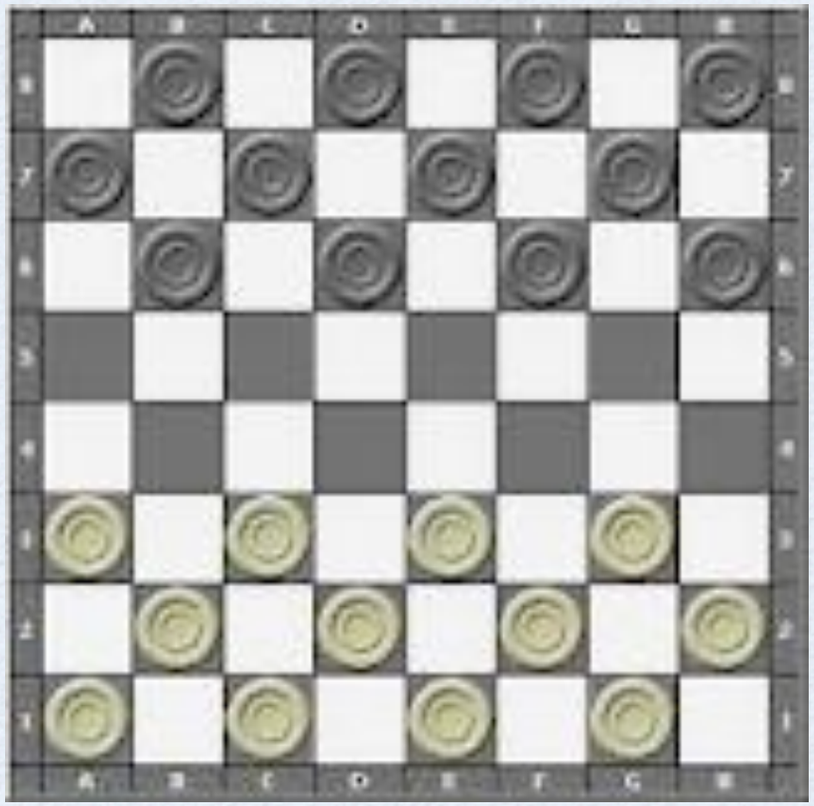


Checkers

Chinook

Checkers

- ~10^{20} unique positions
- ~3 legal moves
- ~70 moves long
- ~10^{31} complexity



Chinook

- High Performance Alpha-Beta Search
- Binary-Linear Evaluation function
- 21 knowledge-based features (position, mobility, ...)
- x4 phases of the game
- Fixed material weights

Chinook's Endgame Tablebase

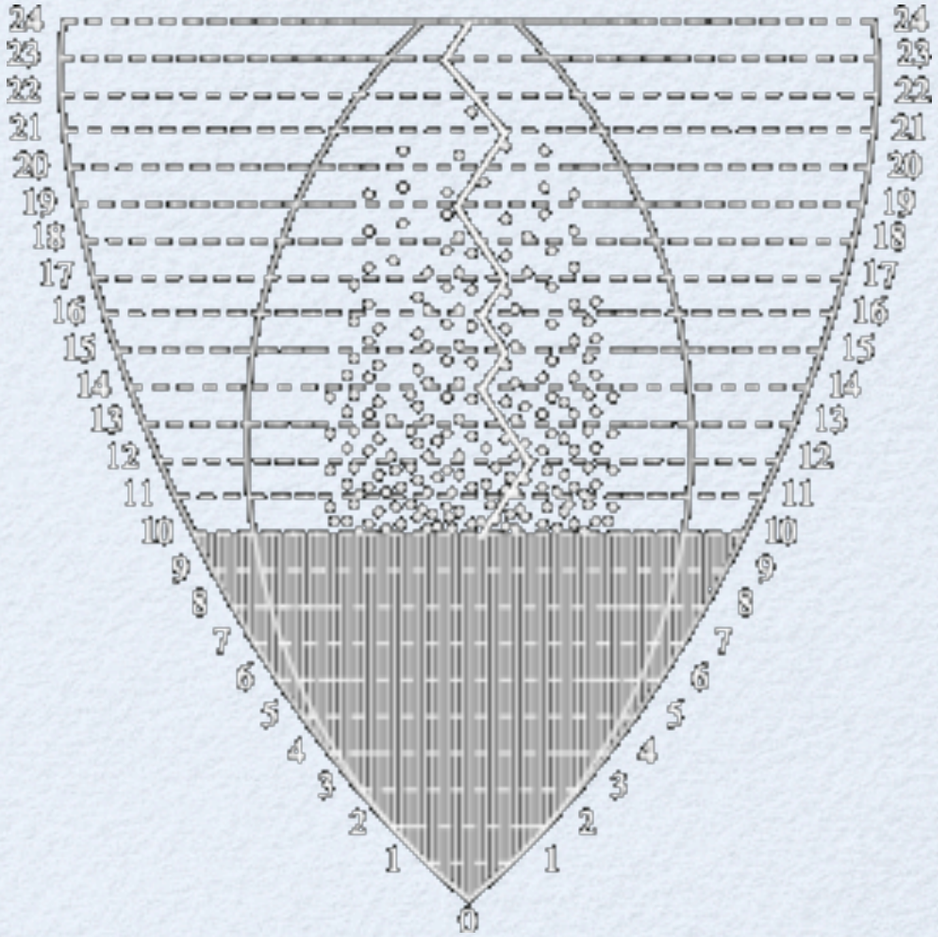
- *Retrograde analysis*: search backward from won positions
- Store all winning positions in lookup tables
- Can play *perfectly* from last *n* checkers

Chinook vs. Marion Tinsley

- Marion Tinsley only lost 7 games in his 45 year career
- Lost 2 games to Chinook in world championship 1994
- Tinsley then withdrew for health reasons
- First computer world champion?
- Debatable

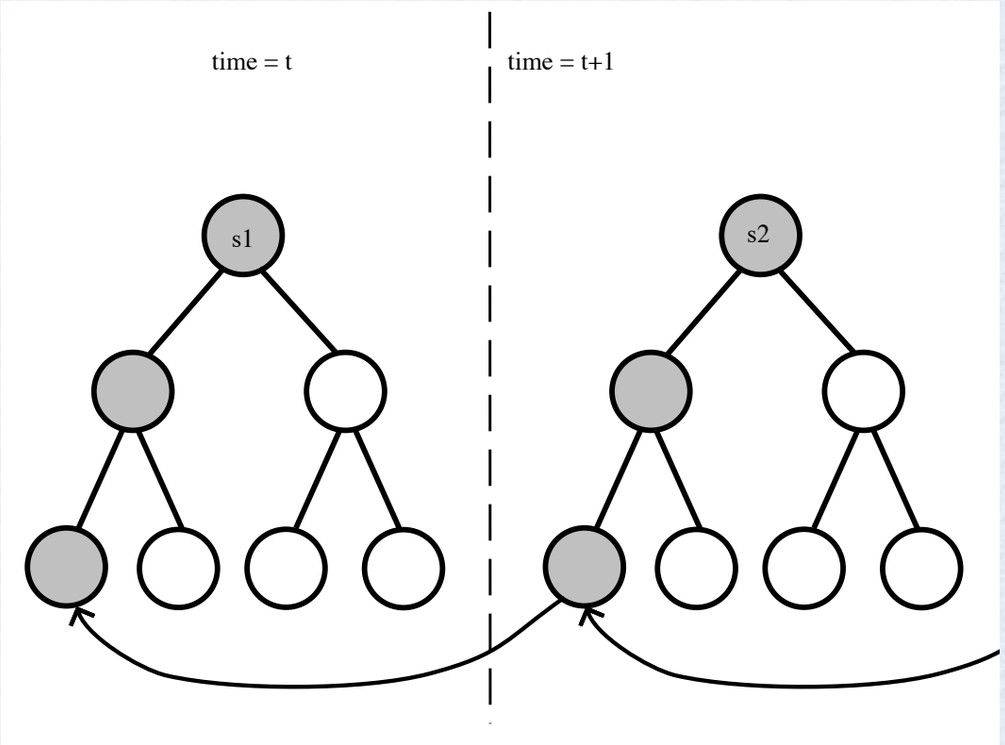
Chinook vs. God

- Schaeffer *solved* checkers in 2007
- Alpha-beta search + endgame tables from start to end of game
- Perfect intelligence in this domain
- Chinook can draw vs. God



RL in Checkers

- Chinook originally used hand-tuned weights
- Later version was trained by self-play
- Using *TD(leaf)* to adjust weights
- Learnt weights outperformed hand-tuned weights



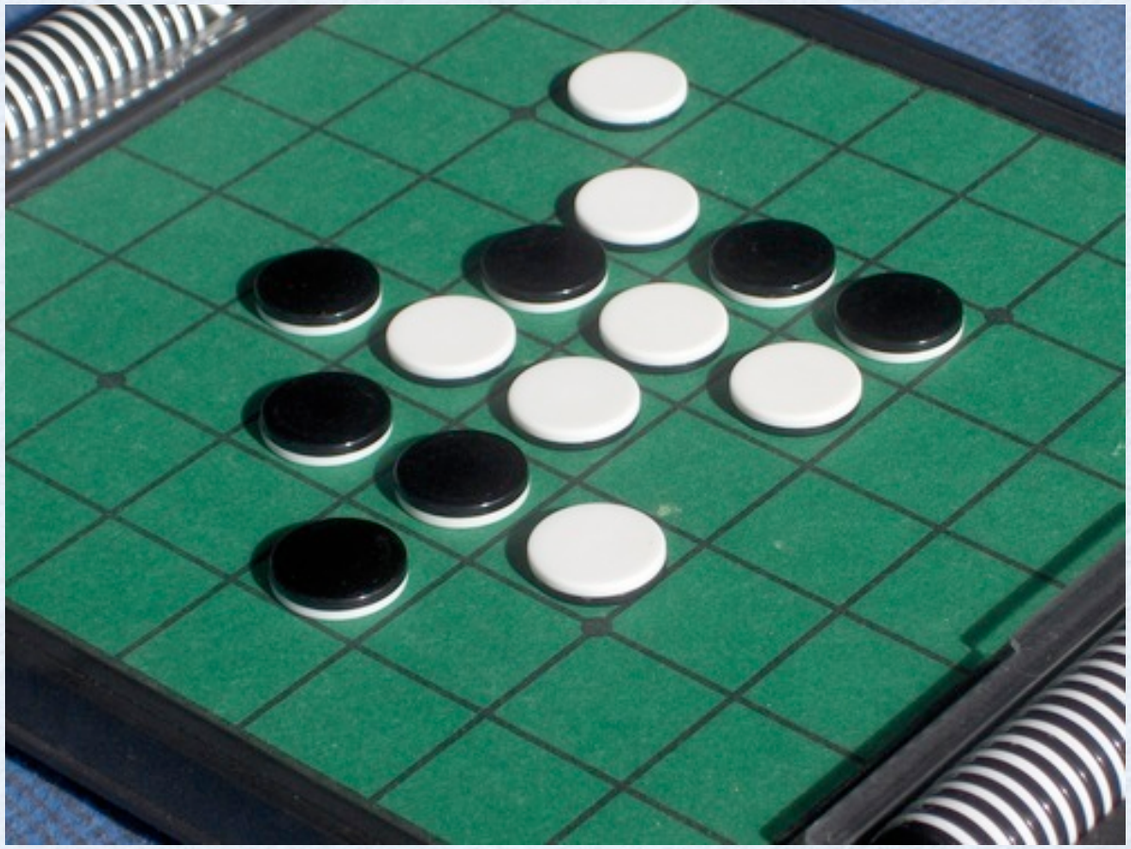
$$V^*(s_t) \leftarrow V^*(s_{t+1})$$

Othello

Logistello

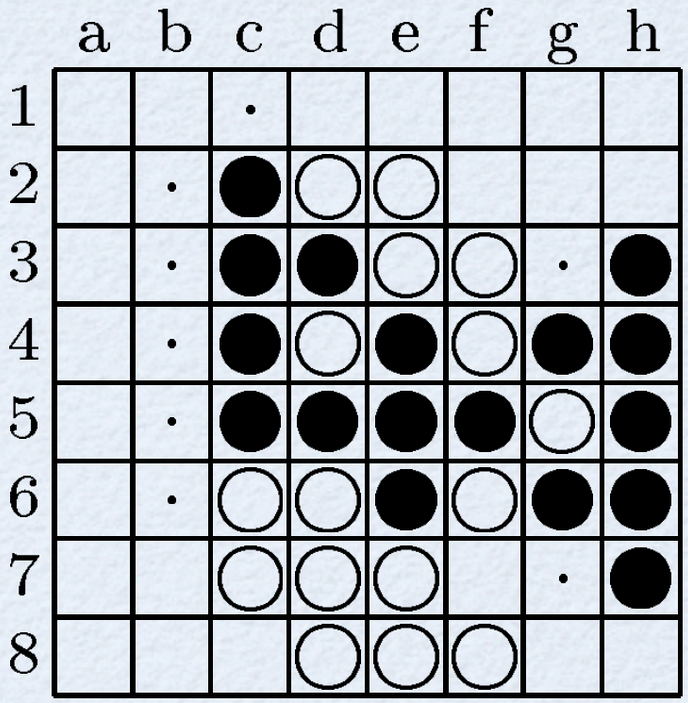
Othello

- ~10^{28} unique positions
- ~7 branches
- ~60 moves long
- ~10^{31} complexity

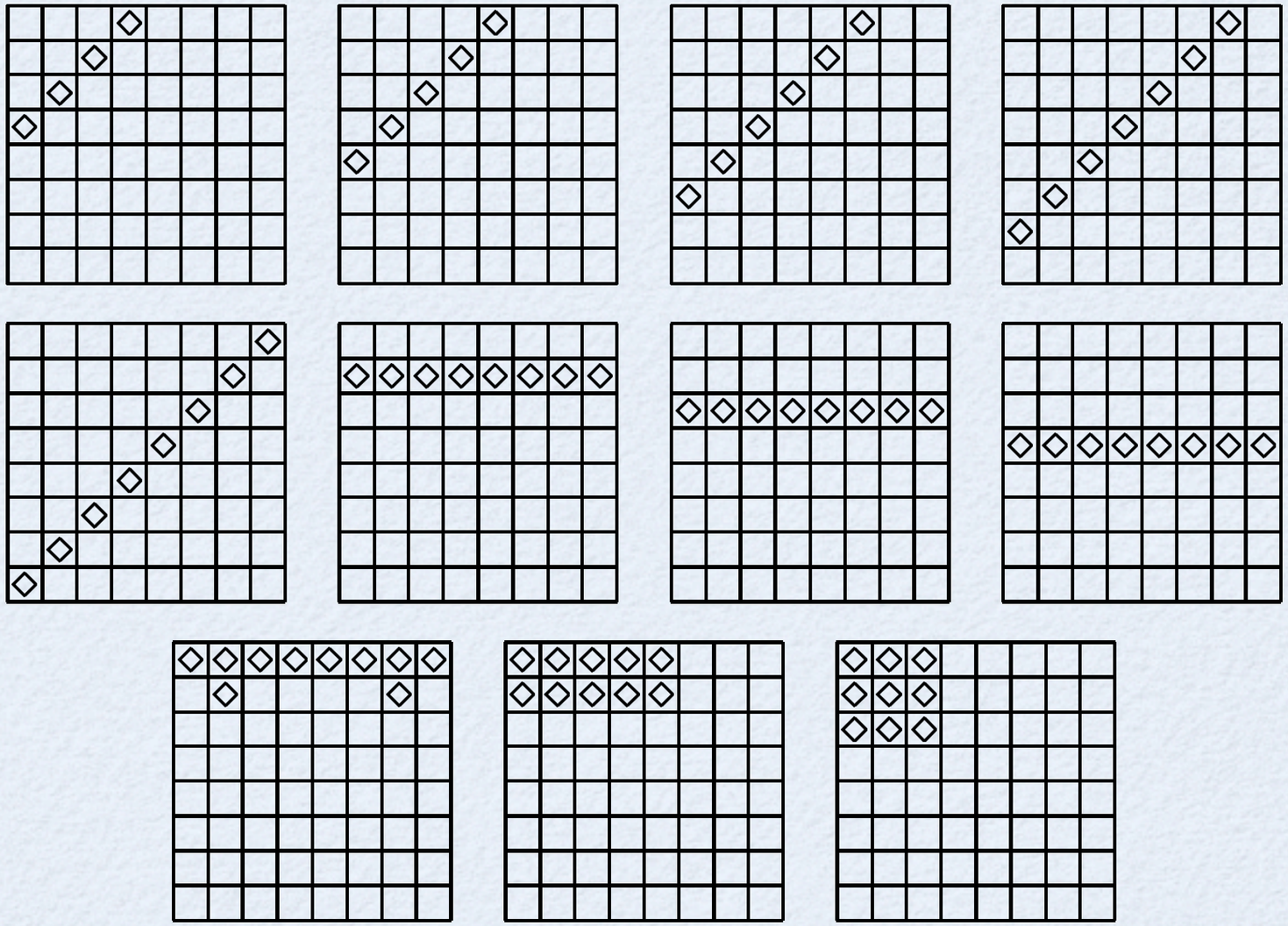


Logistello Features

- Logistello created its own features
- Start with raw input features, e.g. "black stone at C1"
- Construct new features by conjunction/disjunction
- Created 1.5 million features in different configurations



Configuration Templates



RL in Logistello

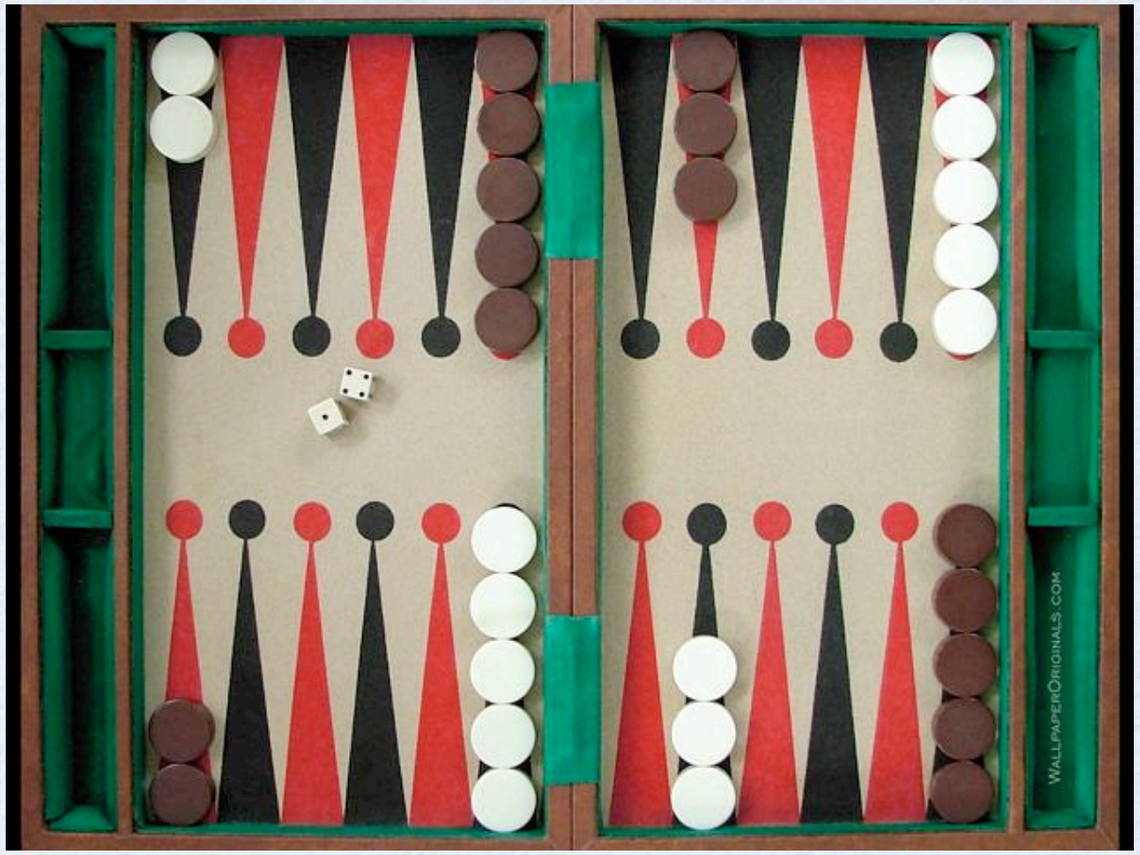
- Generalised linear evaluation function (logistic)
- Learnt weights by batch Monte-Carlo
  - Generate batch of games by self-play
  - Evaluate current policy from batch of games
  - Improve policy
- Defeated World Champion Takeshi Murukami 6-0

Backgammon

TD-Gammon

Backgammon

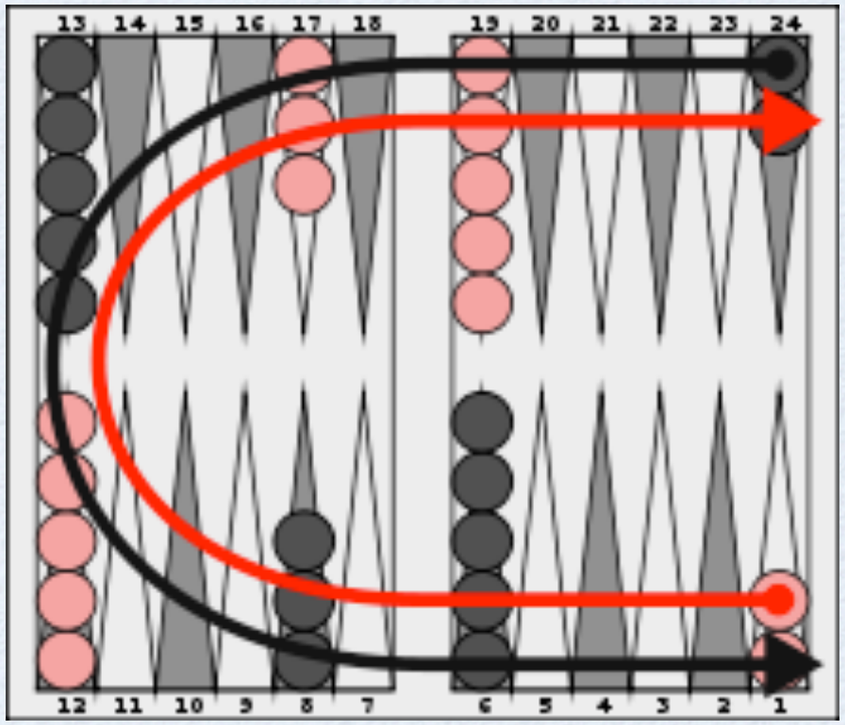
- ~10^{20} unique positions
- ~400 branches
- ~60 moves long
- ~10^{144} complexity



- Stochastic (dice rolls)

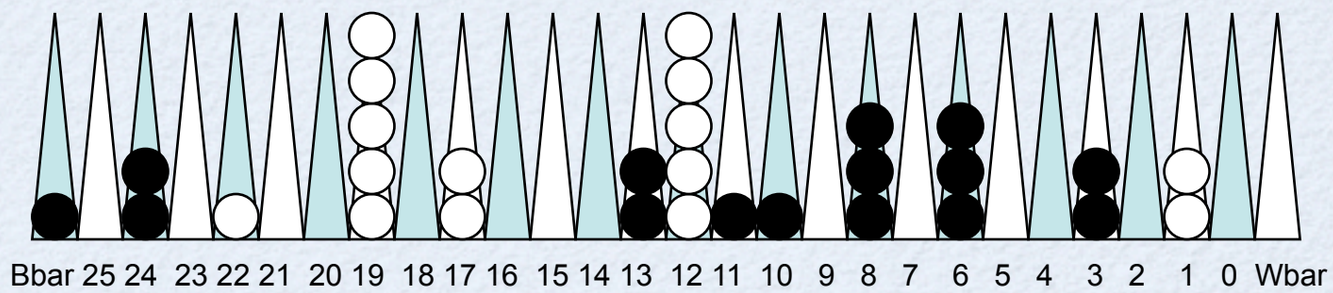
Backgammon

- Objective: advance all checkers off end of board
- Move up to two checkers according to dice roll
- Doubles count twice
- Can hit single opposing checker - sent back to start
- Can't hit two or more checkers

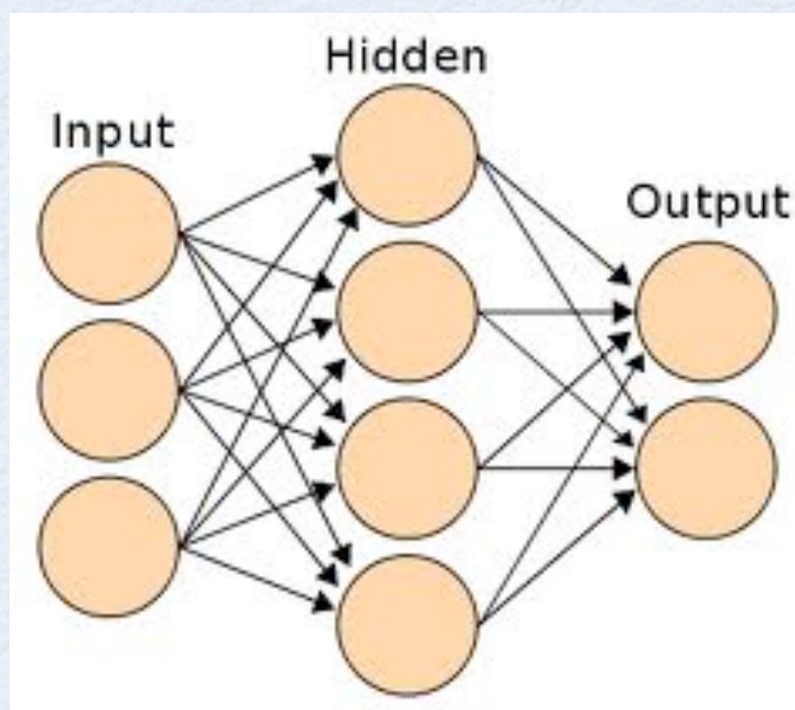


TD Gammon Architecture

Binary features $\phi(s)$



Binary features $\phi(s)$



Evaluation $V(s)$

TD Gammon Training

- Initialise with random weights
- Train by games of self-play
- Using *non-linear* temporal-difference learning
- Algorithm always converged in practice

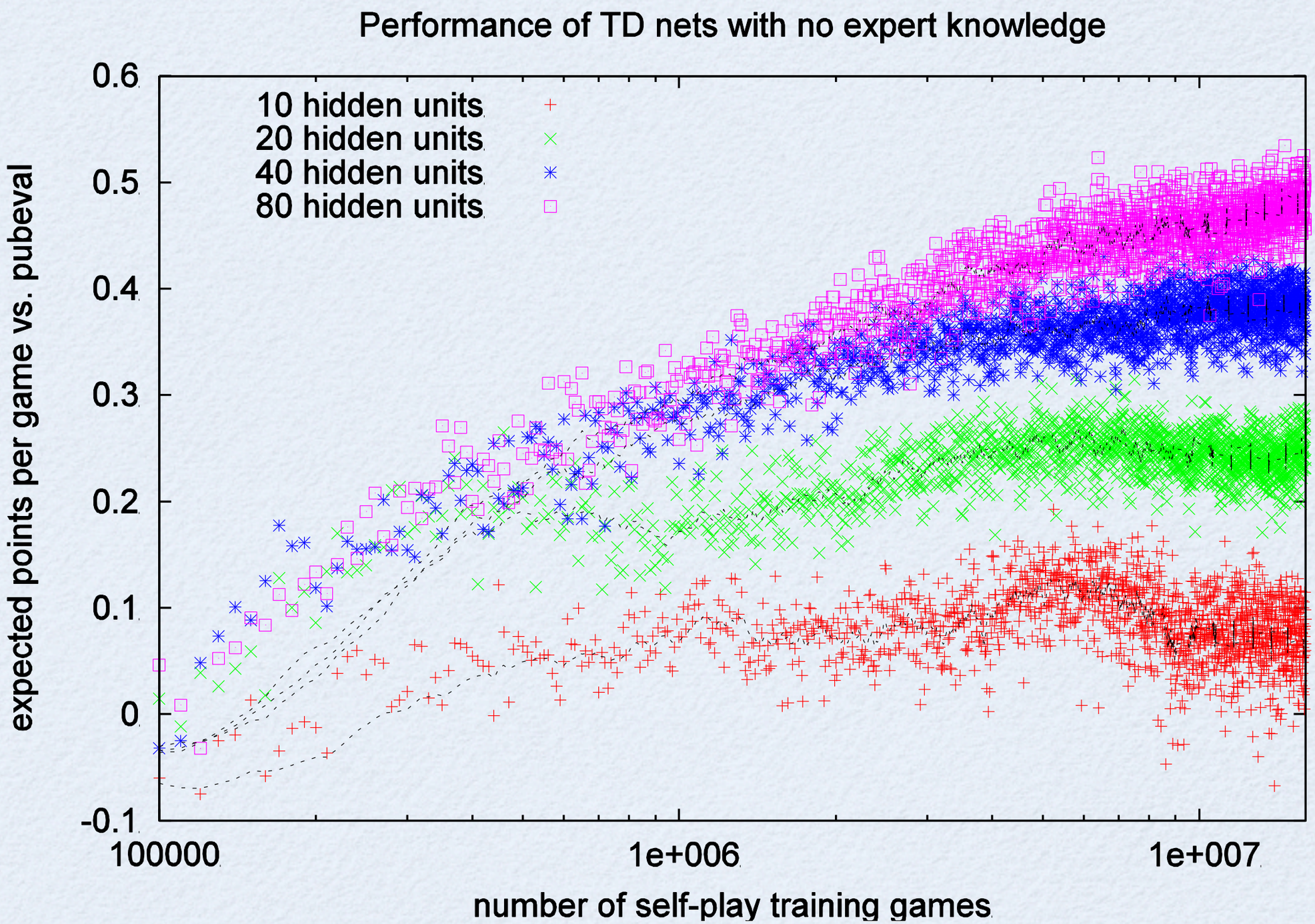
$$\delta_t = V(s_{t+1}) - V(s_t)$$

$$\theta_{t+1} = \theta_t + \alpha \delta_t \nabla_\theta V(s_t)$$

Performance of TD Gammon

- Zero expert knowledge => strong intermediate play
- Hand-crafted features => advanced level of play (1991)
- 2-ply search => strong master play (1993)
- 3-ply search => superhuman play (1998)
- Defeated world champion Luigi Villa 7-1 (1992)

New TD Gammon Results



Part V:

Case Studies: Monte-Carlo Search

Scrabble

Maven

Scrabble

- ~700 branches
- ~25 moves long
- ~10^{70} complexity

- Stochastic (letters drawn randomly from bag)
- Imperfect information (unseen opponent rack)

Strategy in Scrabble

- Move generation is easy with a perfect dictionary
- Strategy is hard: which letters to leave on rack?
- Need to know *potential* value of remaining letters

Rack Evaluation in Maven

- Maven evaluates moves by *score* + *V(rack)*
- Binary-linear evaluation function of rack leave
- Using one, two and three letter features

```
Q??????    QU?????    III????
```

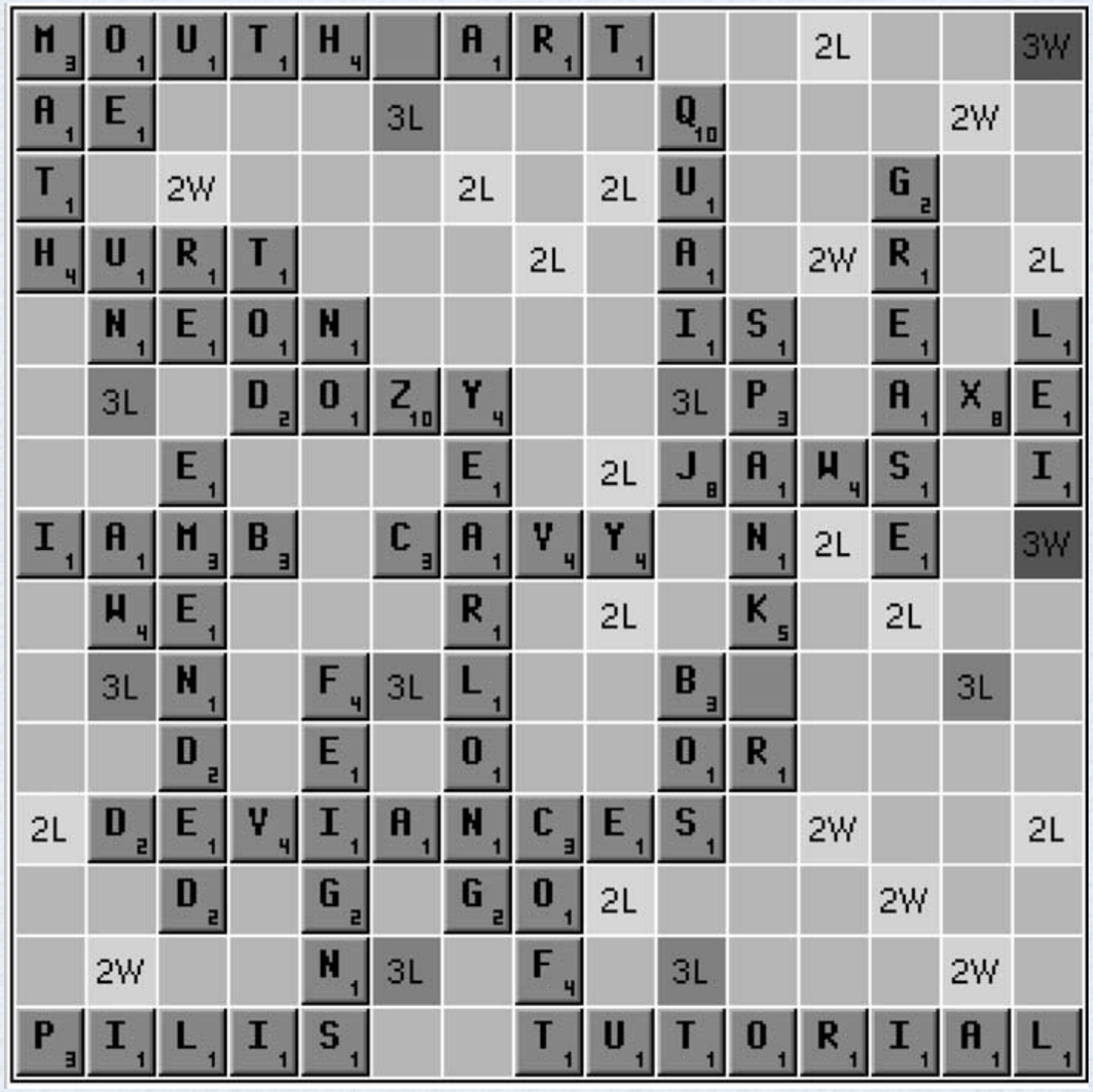
- Trained by games of self-play
- Weights learnt by batch Monte-Carlo, a.k.a. TD(1)

Truncated Monte-Carlo Search

- For each legal move:
  - *Roll-out* move by imagining *n* steps of self-play
  - Evaluate resulting position by *score* + *V(rack)*
  - Score move by average evaluation in rollouts
- Select and play highest scoring move

Super-Human Scrabble

- Maven beats world champion Adam Logan 9-5
- Here Maven predicts endgame to finish with MOUTHPART
- Analysis showed Maven had error rate of 3 points per game



Go

MoGo

The Game of Go

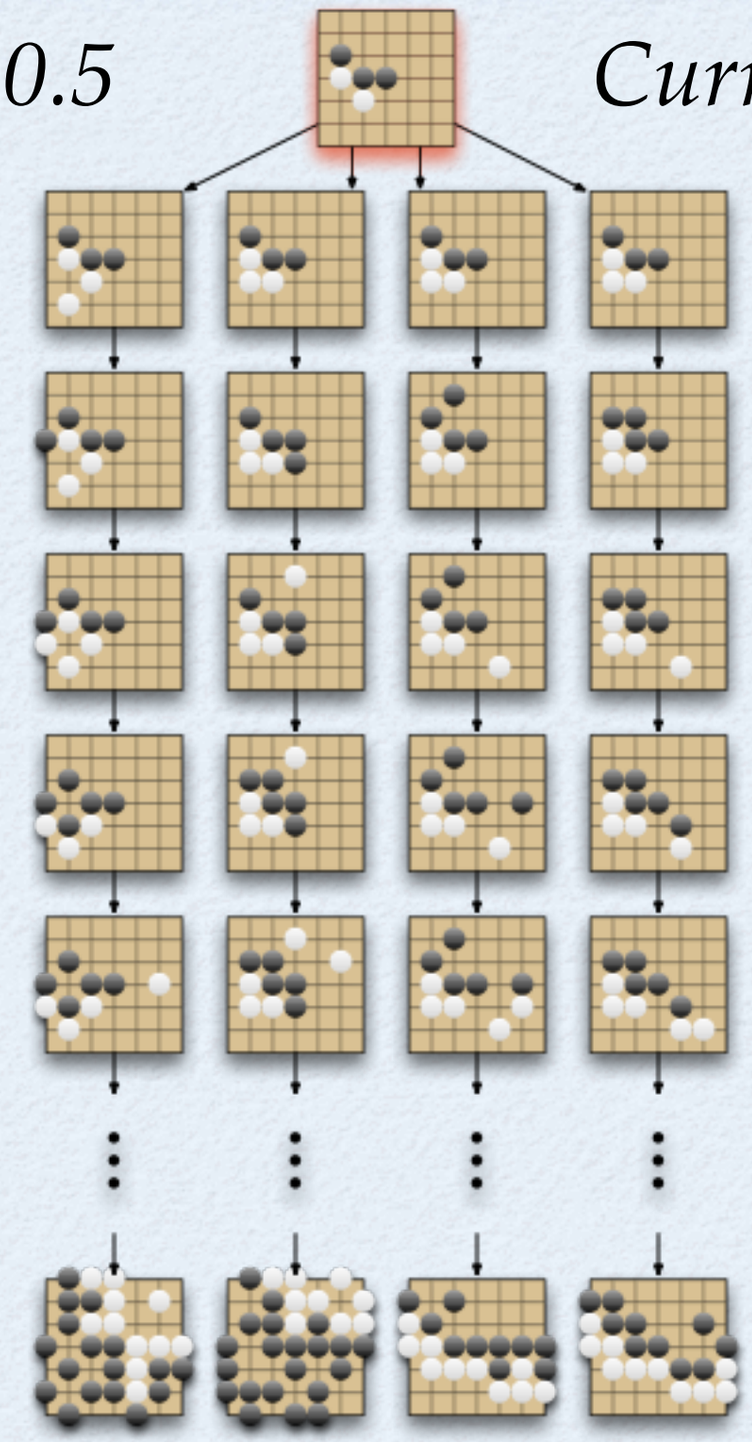
- ~10^{170} unique positions
- ~200 moves long
- ~200 branching factor
- ~10^{360} complexity

The Story of Go

- The ancient oriental game of Go is 2000 years old
- Considered to be the hardest classic board game
- Considered to be a grand challenge task for AI *(e.g. John McCarthy)*
- *Traditional approaches to game-tree search have failed in Go*

Monte-Carlo Simulation

$V(s) = 2/4 = 0.5$



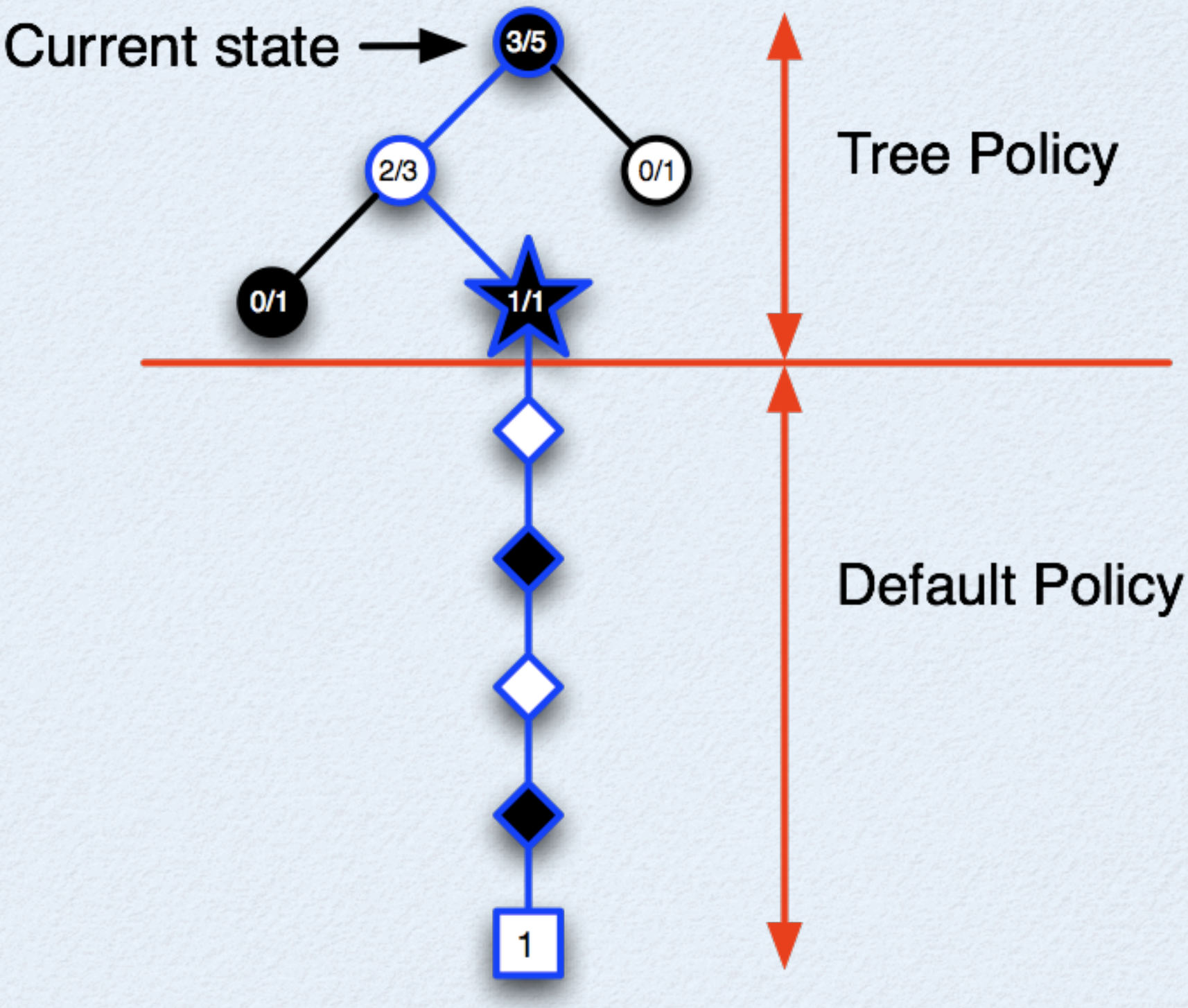
Current position s

Simulation

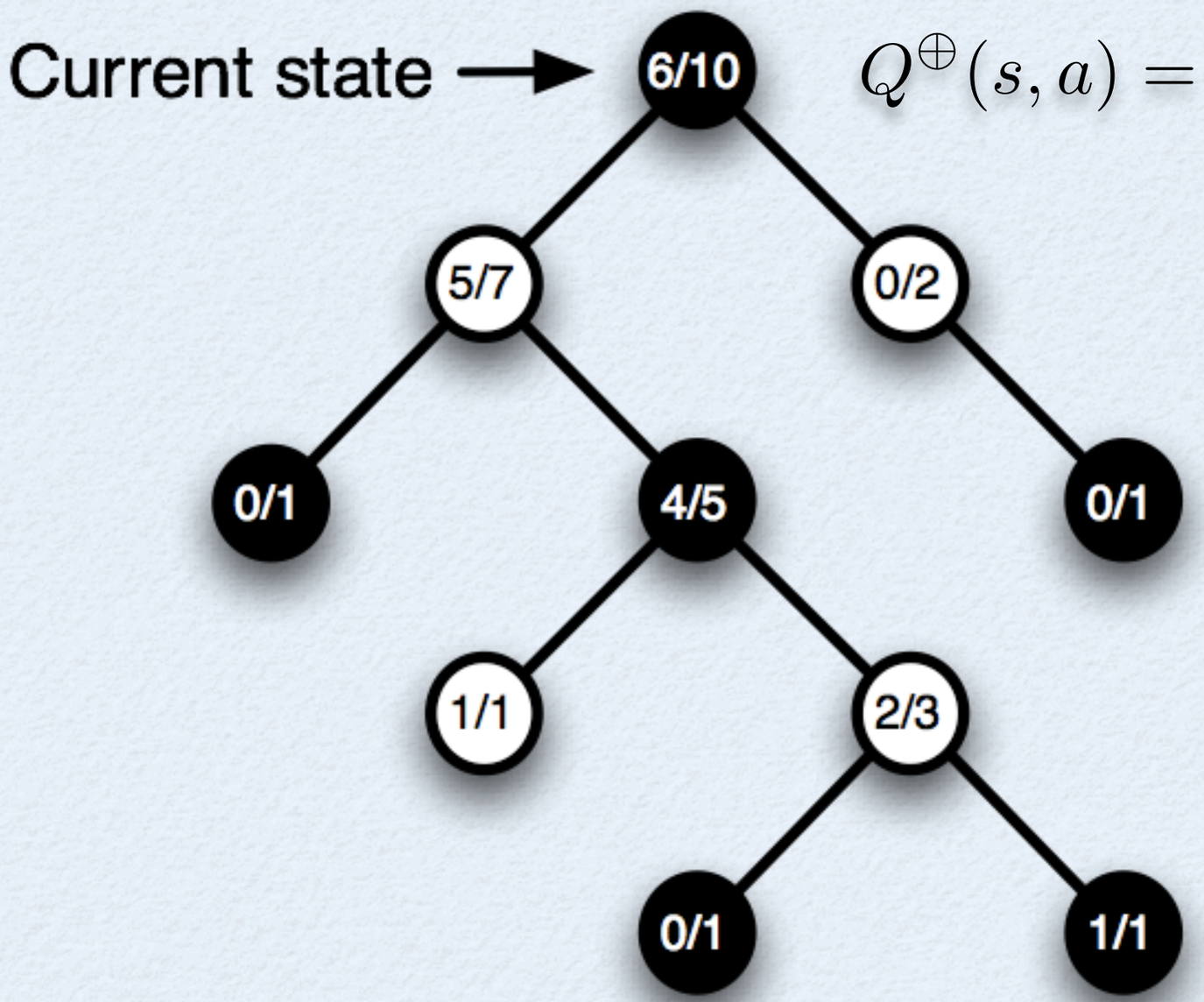
1 1 0 0

Outcomes

Monte-Carlo Tree Search

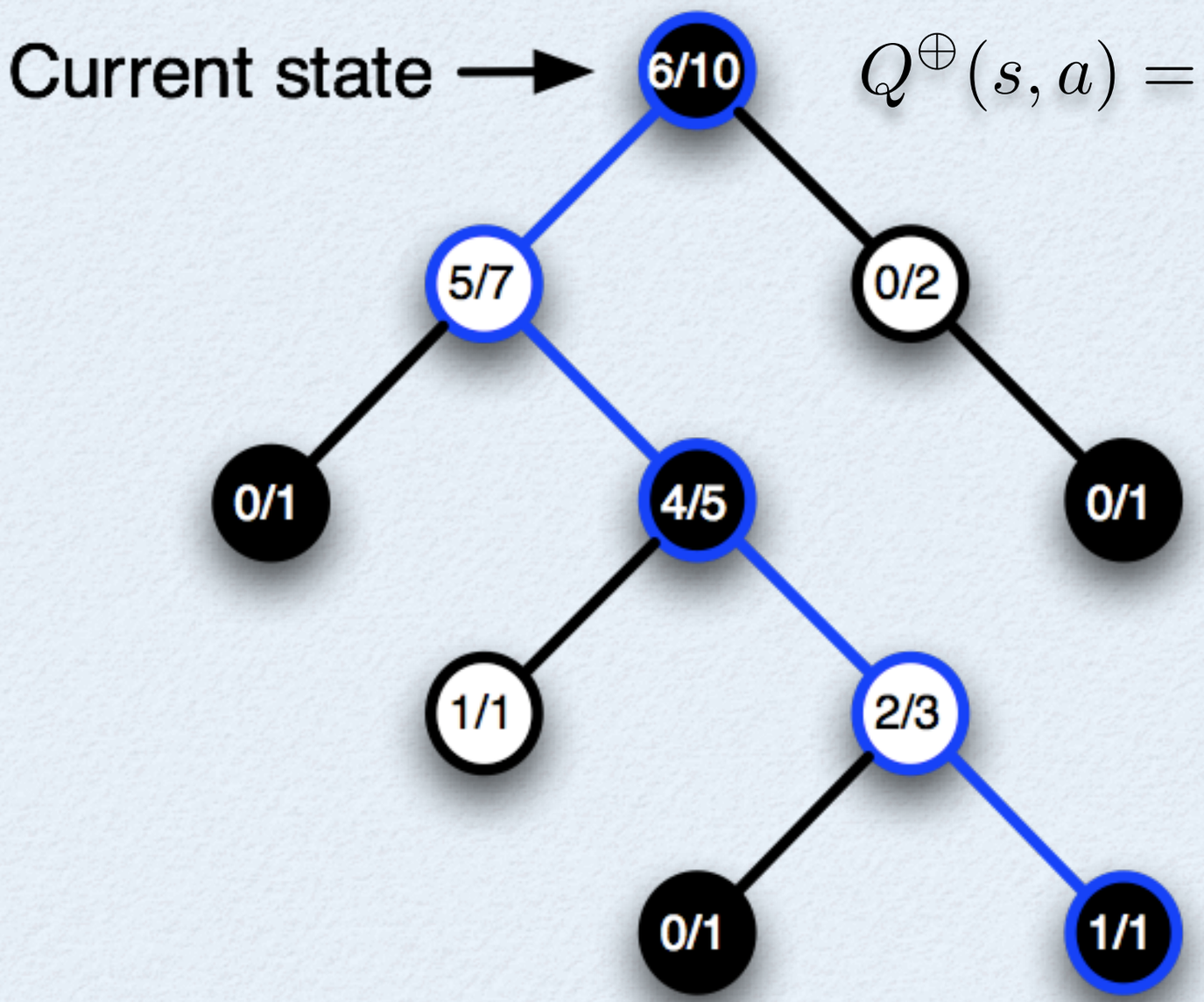


UCT (Upper Confidence Trees)



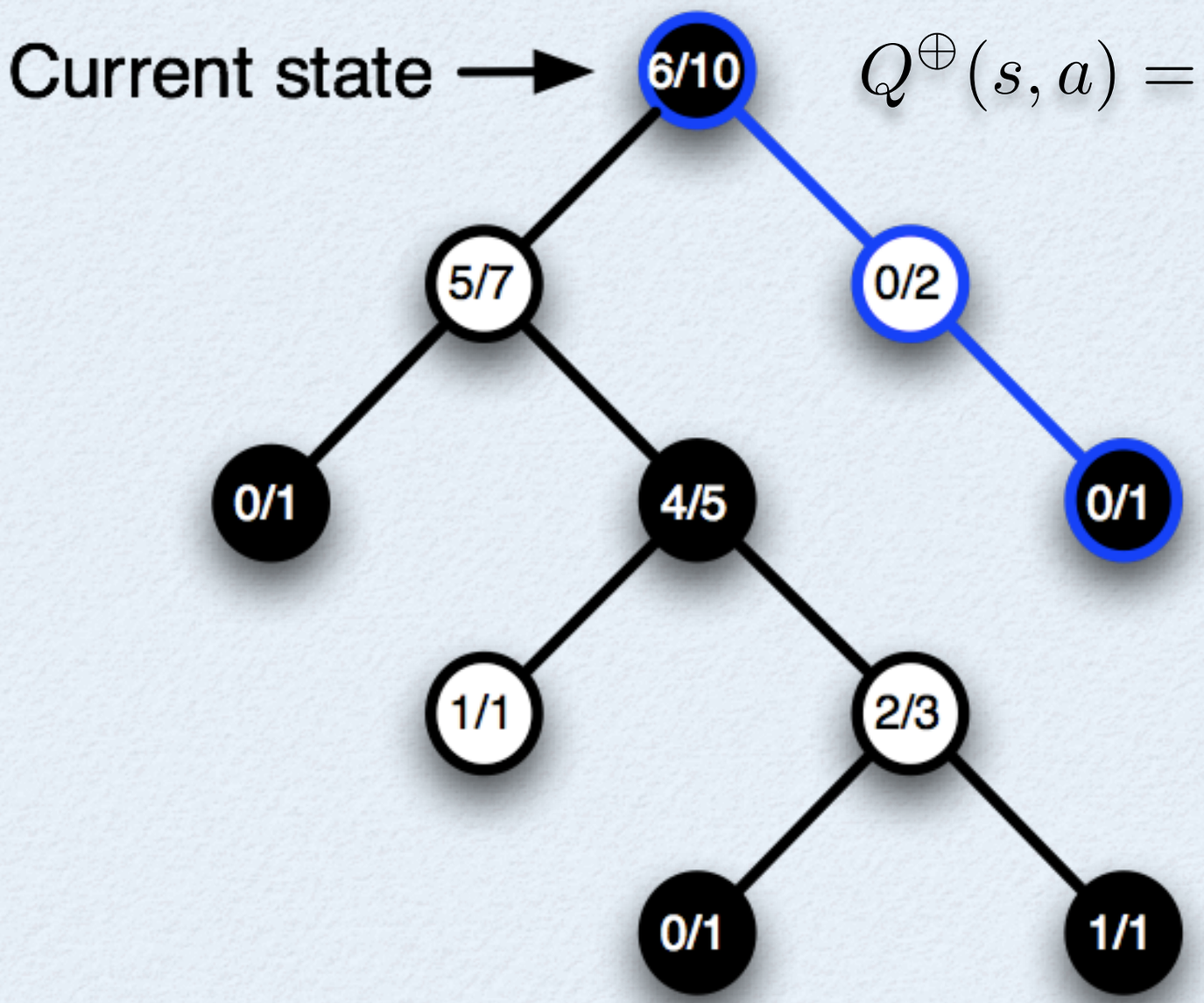
$$Q^{\oplus}(s,a) = Q(s,a) + c\sqrt{\frac{\log n(s)}{n(s,a)}}$$

Exploitation



$$Q^{\oplus}(s,a) = \boxed{Q(s,a)} + c\sqrt{\frac{\log n(s)}{n(s,a)}}$$

Exploration



$$Q^{\oplus}(s,a) = Q(s,a) + \boxed{c\sqrt{\frac{\log n(s)}{n(s,a)}}}$$

Advantages of MCTS

- Highly selective best-first search
- Evaluates positions dynamically by self-play simulation
- Removes requirement for evaluation function at leaves
- Converges on true minimax game tree
- Computationally efficient, anytime, parallelisable

MoGo

- Generalise value of same move in similar positions
- Use a heuristic function to initialise leaf nodes

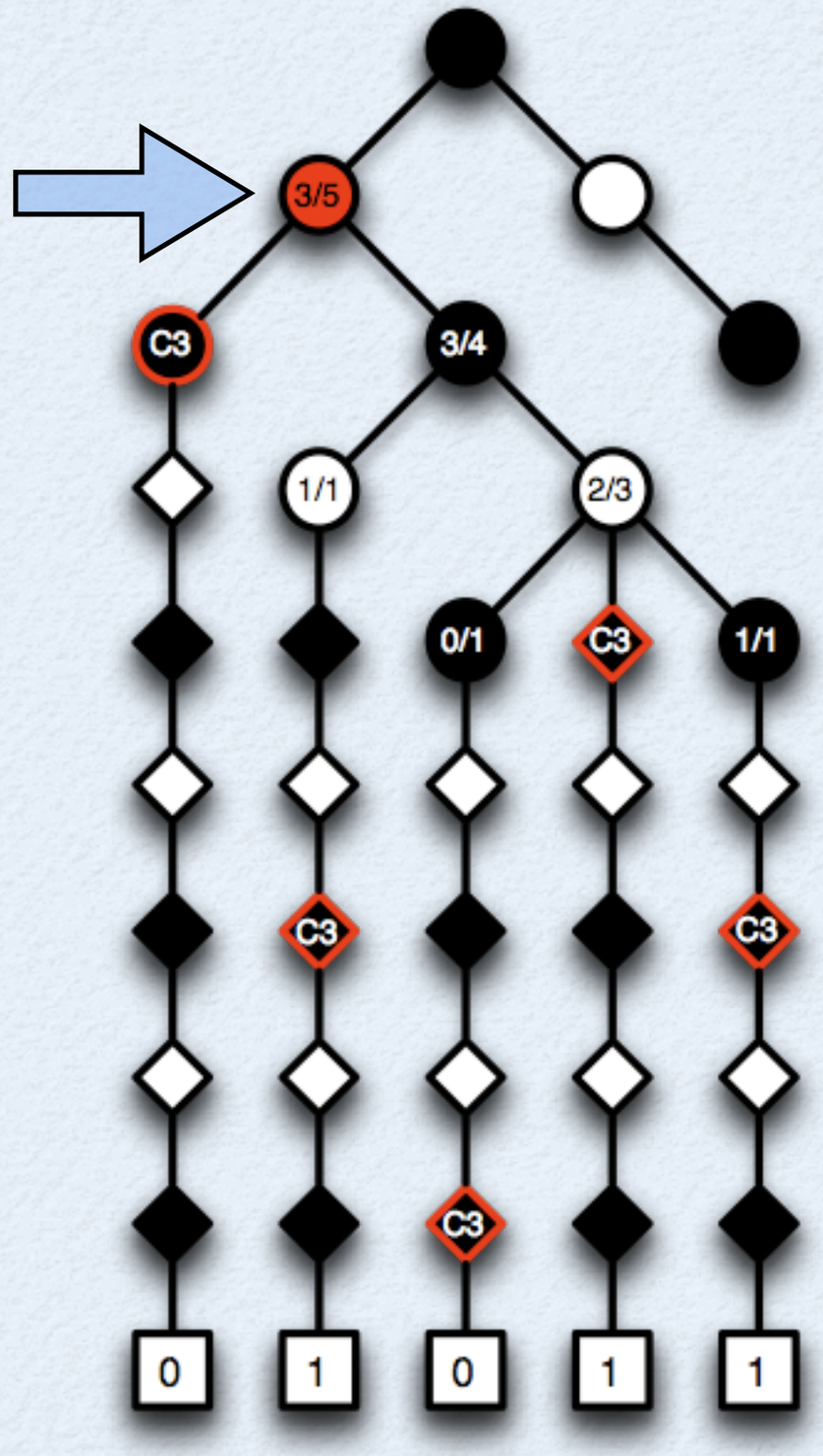
(Gelly and Silver, 2007)

Rapid Action Value Estimate (RAVE)

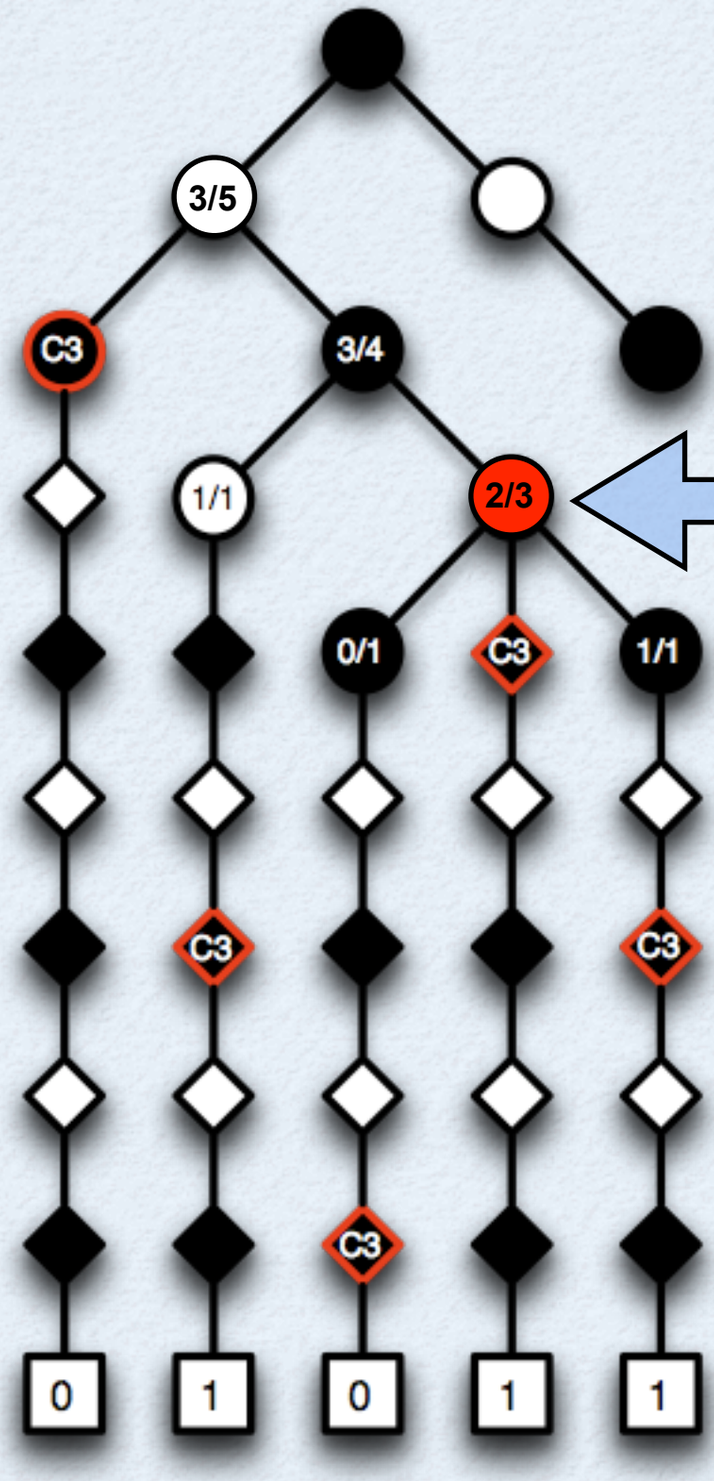
- Assume that the value of move is the same
- Regardless of when move is played

Rapid Action Value Estimate (RAVE)

MC value of C3 = 0/1
RAVE value of C3 = 3/5



Rapid Action Value Estimate (RAVE)

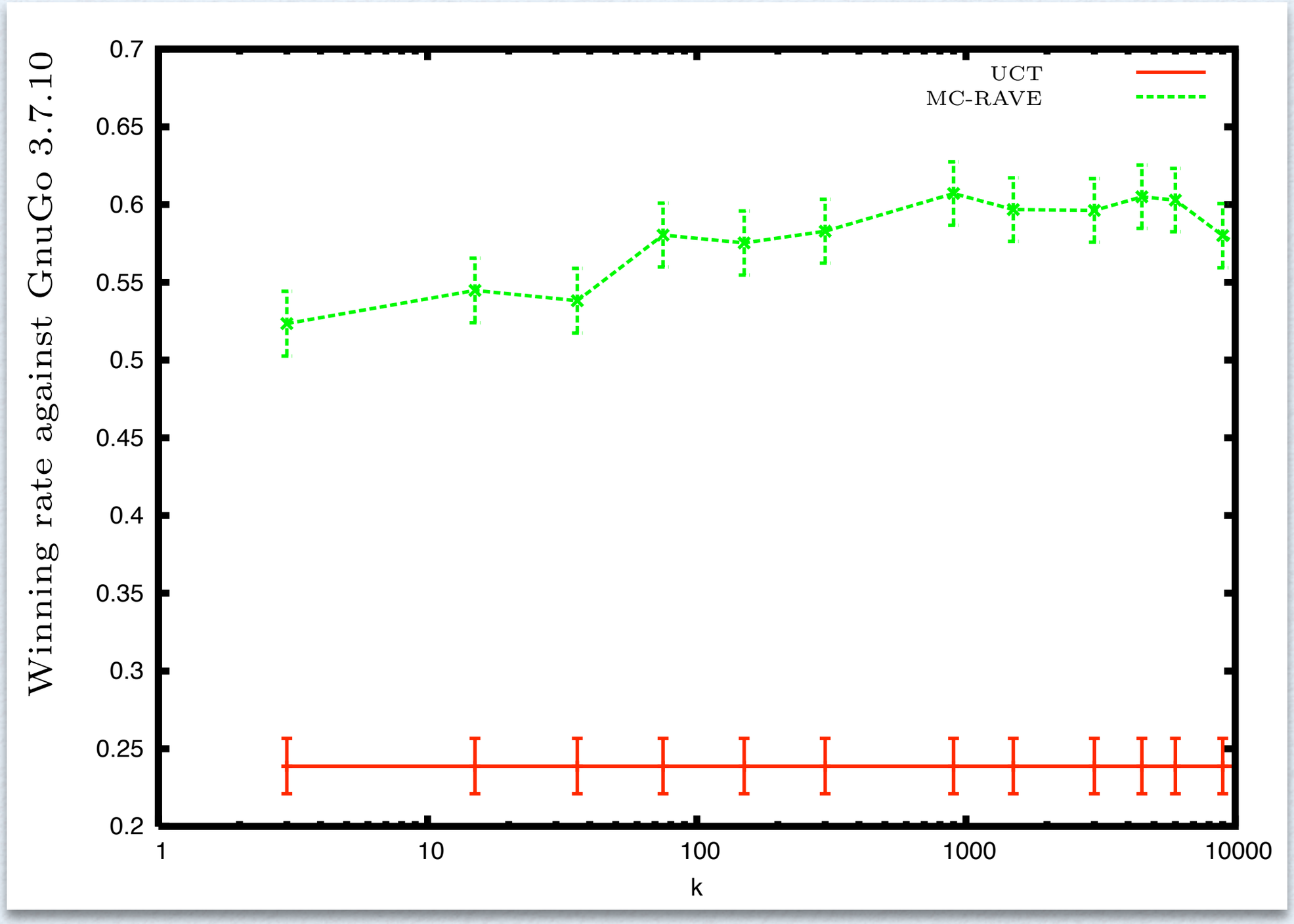


MC value of C3 = 1/1
RAVE value of C3 = 2/3

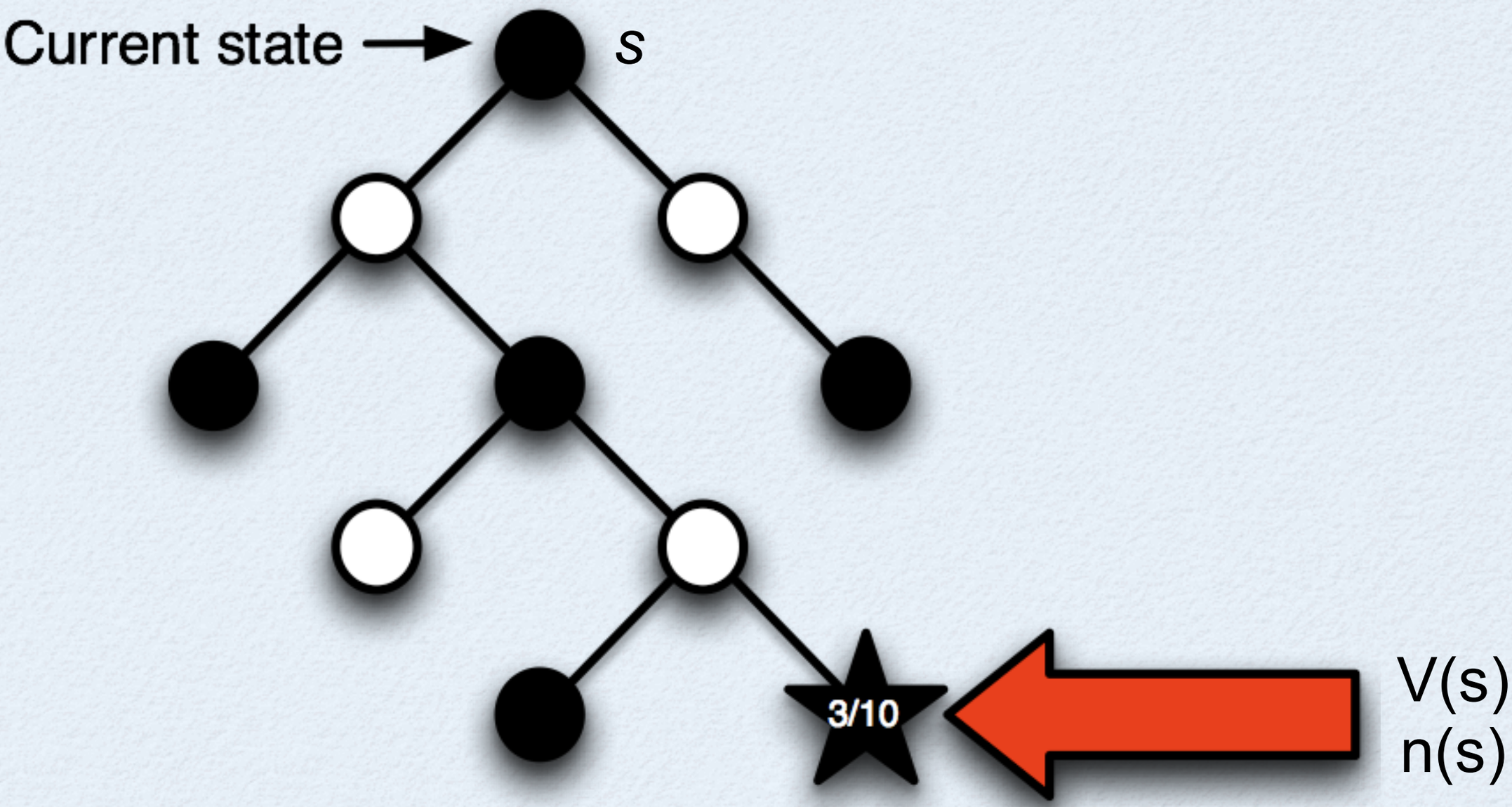
MC-RAVE

- Monte-Carlo value is unbiased but has high variance
- RAVE value is biased but has low variance
- Combine both values so as to minimise MSE

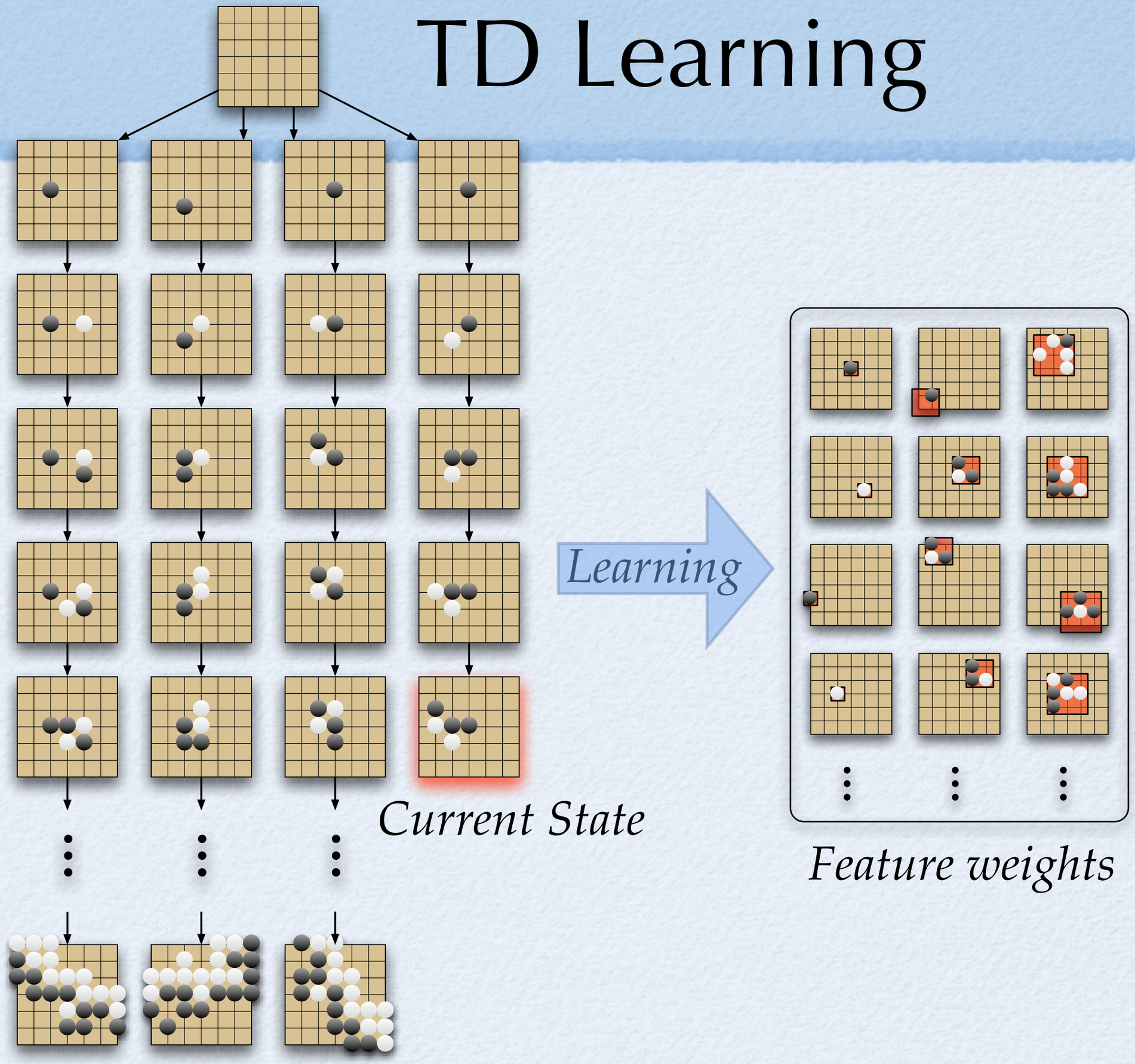
MC-RAVE in *MoGo*



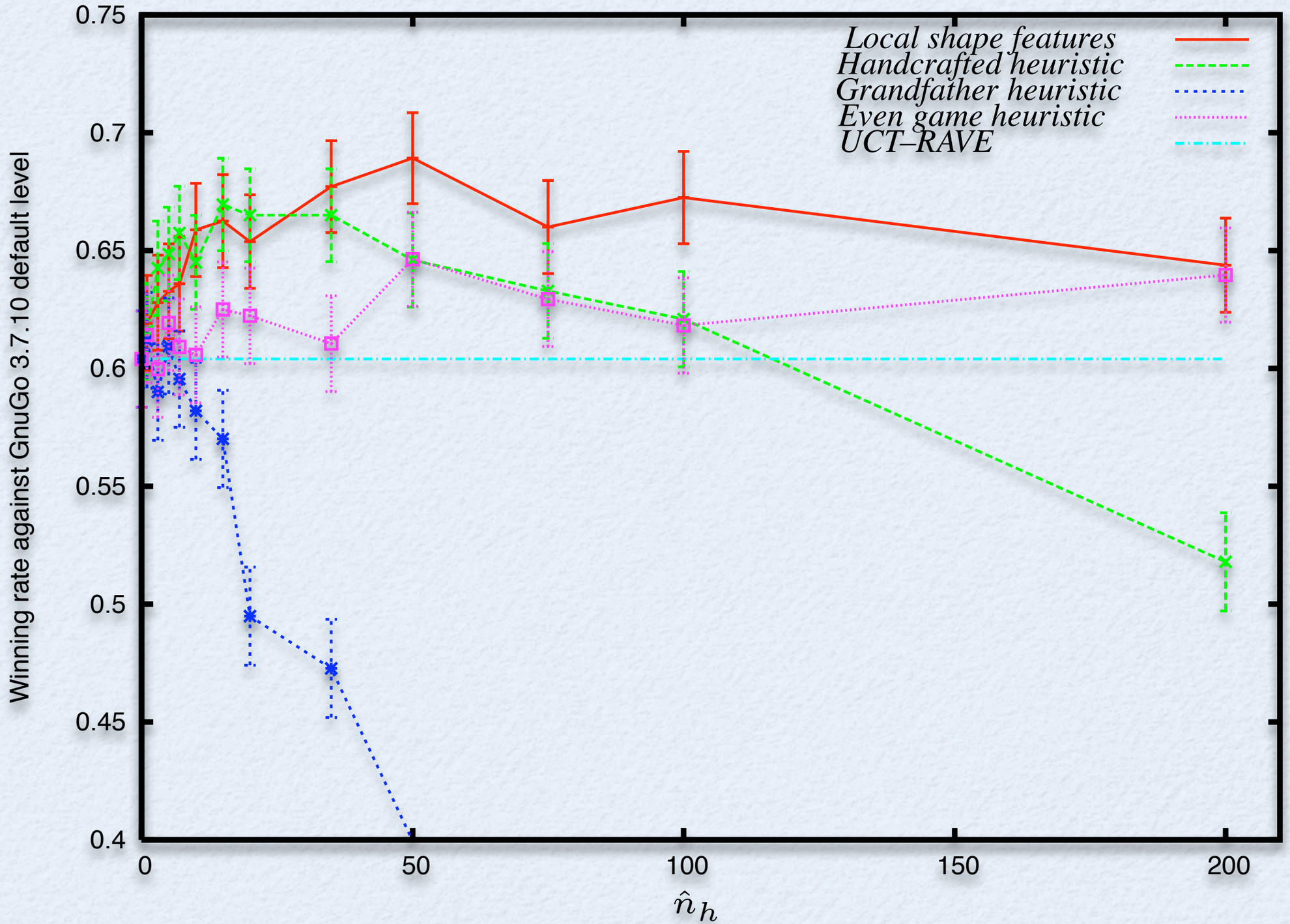
Heuristic MCTS



TD Learning



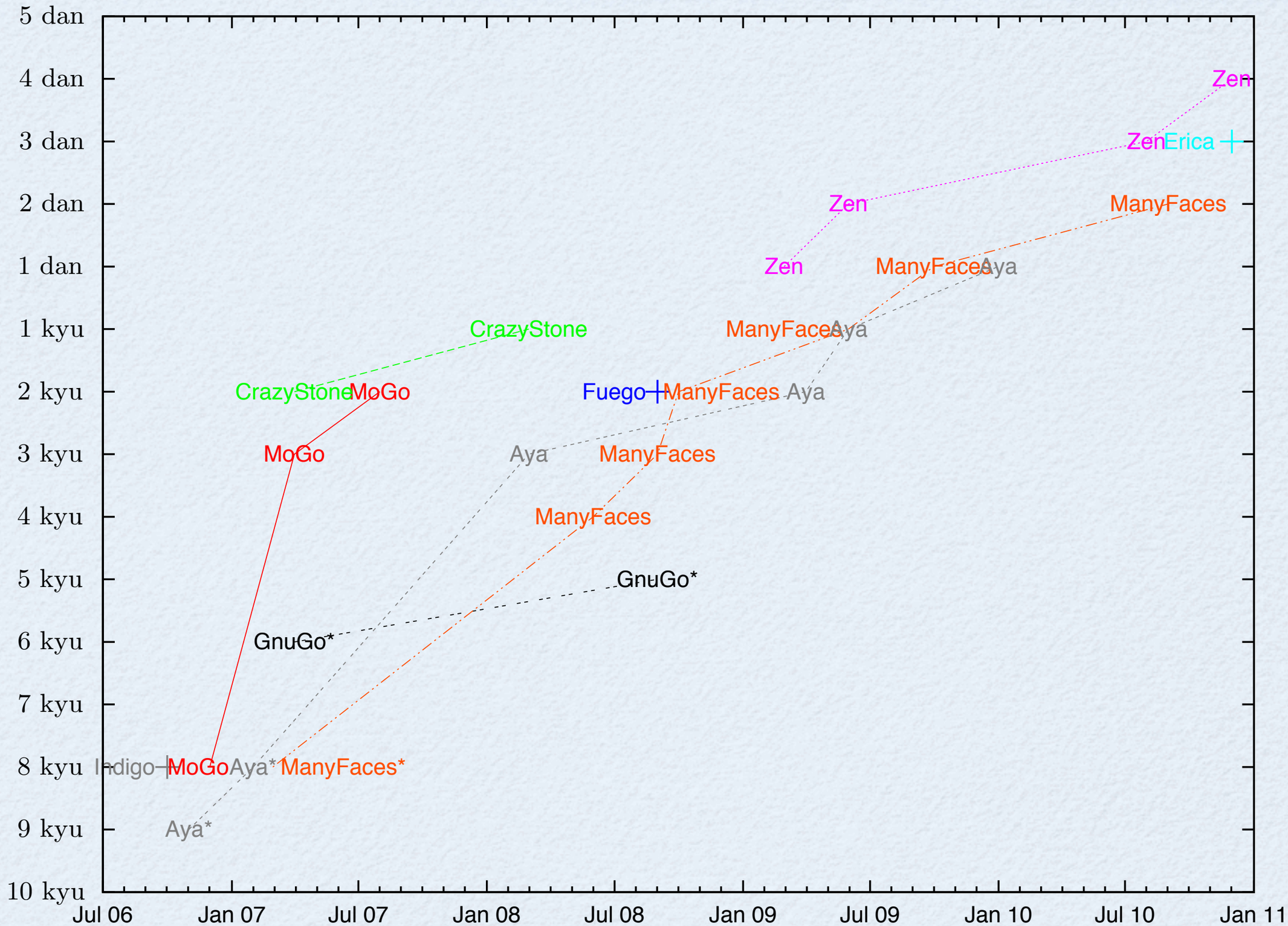
Heuristic MCTS in *MoGo*



MoGo (2007)

- MoGo = heuristic MCTS + MC-RAVE + handcrafted default policy
  - 99% winning rate against best traditional programs
  - Highest rating on 9x9 and 19x19 Computer Go Server
  - Gold medal at Computer Go Olympiad
  - First victory against 9-dan professional player (9x9)

Progress in 19x19 Computer Go



Conclusion

Similarities and Lessons

A Recipe for Success

	Input features	*Value function*	*Learning Algorithm*	*Training Regime*	*Search algorithm*
Chess *Meep*	Binary *Pieces, pawns, ...*	Linear	TD(tree)	Self-Play / Expert	Alpha-beta
Checkers *Chinook*	Binary *Pieces, ...*	Linear	TD(leaf)	Self-Play / Expert	Alpha-beta
Othello *Logistello*	Binary *Disc configs*	Linear	TD(1)	Self-Play	Alpha-beta
Backgammon *TD Gammon*	Binary *Num checkers*	Non-linear	TD(λ)	Self-Play	Monte-Carlo search
Scrabble *Maven*	Binary *Letters on rack*	Linear	TD(1)	Self-Play	Monte-Carlo search
Go *RLGO*	Binary *Stone configs*	Linear	TD(λ)	Self-Play	TD search

A Simple Equation

Binary features

+ Linear evaluation function

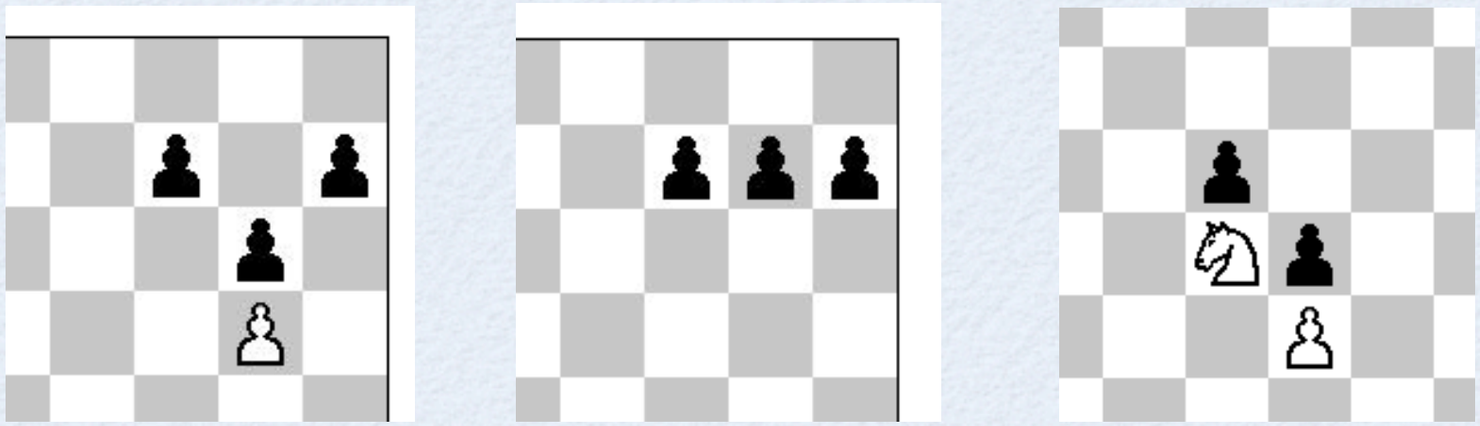
+ Temporal-difference learning

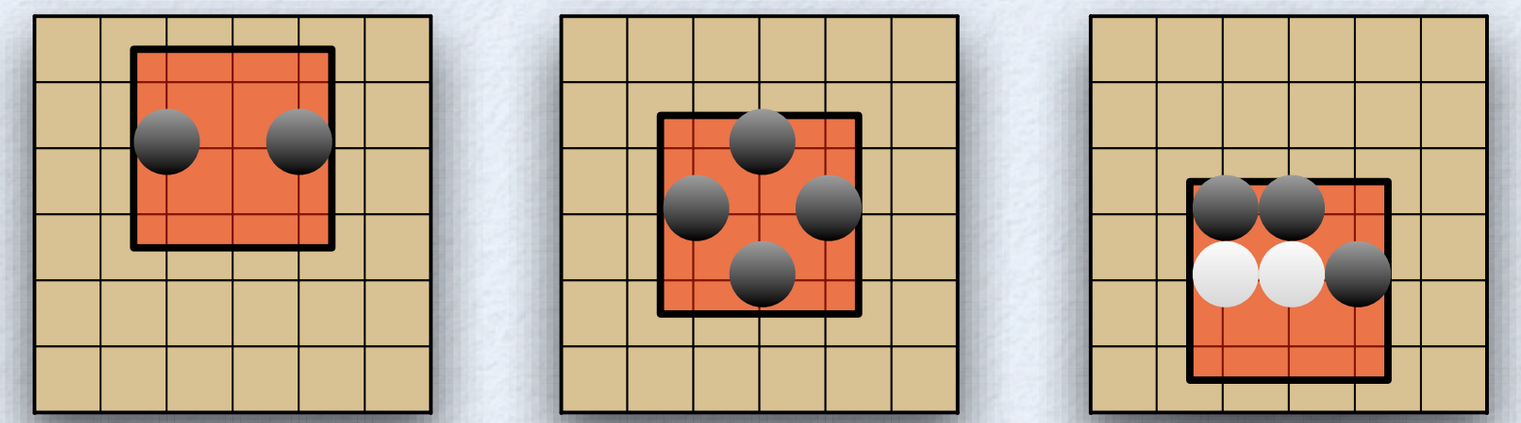
+ Self-play training

+ High-performance search

= World championship play

i) Many Binary Features





Q?????? QU????? III????

$$\phi(s) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

ii) Binary Linear Evaluation

$$V(s) = \sum_j \phi_j(s)\theta_j = \sum_{j|\phi_j(s)=1} \theta_j$$

- One update can adjust the value of many states
- Can compute *millions* of features (10 million in Shogi)
- Computationally efficient
- Can approximate any function
- Good theoretical properties (convergence, efficiency, ...)
- **Easy to understand and debug**

iii) Temporal-Difference Learning

- Much lower variance than Monte-Carlo
- At the cost of increased bias
- **Usually achieves much better performance**

After every move t

$$\delta_t = V(s_{t+1}) - V(s_t)$$

For all j s.t. $\phi_j(s_t) = 1$

$$\theta_j \leftarrow \theta_j + \alpha\delta$$

iv) Self-Play Training

- How can we overcome knowledge acquisition bottleneck?
- How can we *exceed* the performance of human experts?
- How do we find training partner of appropriate level?
- **Train by self-play**

v) High-Performance Search

- Alpha-beta search effective with low branching factors
  - Exhaustive lookahead
  - Highly optimised over 30+ years
- Naive Monte-Carlo simulation works better in highly branching games (e.g. Backgammon, Scrabble)
- Monte-Carlo tree search most fruitful for very challenging domains (Go, Hex, Amazons, ...)

The Importance of Search

- In many games, direct evaluation is **naive**
- **Most** of the performance comes from deep search
  - e.g. in chess, each additional ply gives approx +100 Elo
- Search = knowledge
- Have we learnt this lesson?

The End