A Unified Framework for Learning and Search

David Silver, Rich Sutton, Martin Müller, Sylvain Gelly

Sample Based Learning

Reinforcement Learning

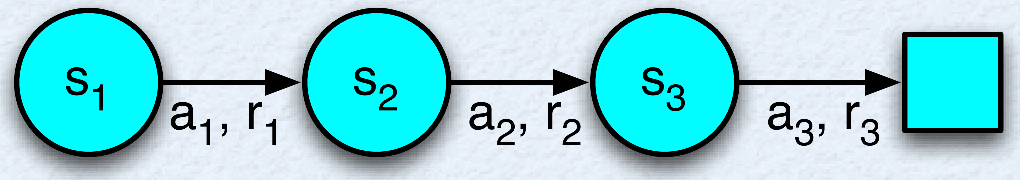
- Sequential decision making problems
  - Riding a bicycle
  - Flying a helicopter
  - Navigating a maze
  - Playing a game

Reinforcement Learning

- Every time-step t the agent
  - Receives a state s_t
  - Selects an action a_t
  - Receives a scalar reward r_t

Reinforcement Learning

- How can the agent maximise its future reward?
- Given *experience* of the world?

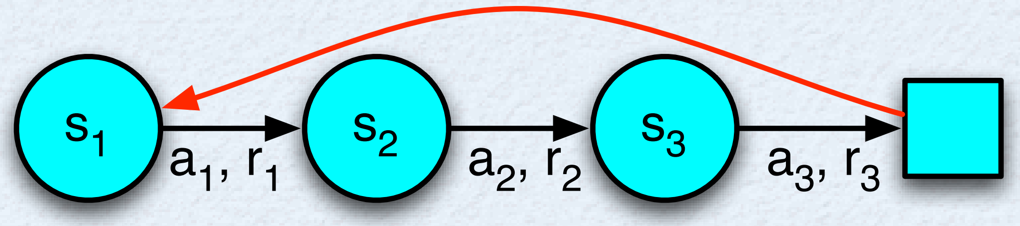


Value Functions

- All efficient reinforcement learning methods use a *value function* as an intermediate step
- The return R_t is the total reward received from time t onwards
- The state value function $V(s)$ is the expected return from state s
- The action value function $Q(s,a)$ is the expected return from state s and action a

Monte-Carlo Evaluation

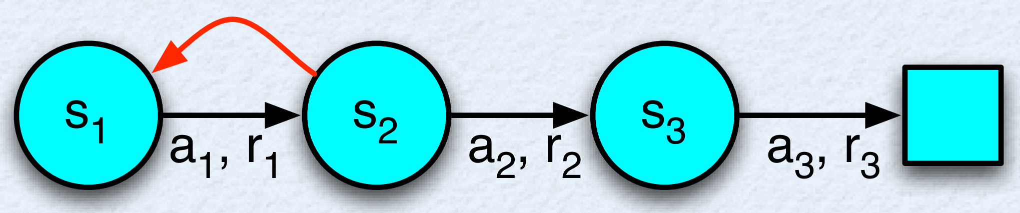
- Value function is estimated by the empirical average return



- $V(s_t) = V(s_t) + 1/N(s_t)\ [R_t - V(s_t)]$

TD(λ)

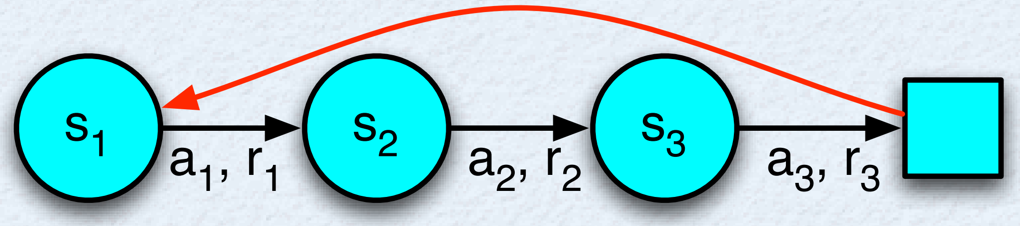
- Value function is updated from future estimates
- Temporal difference parameter λ controls timescale
- $\lambda=0$: value function is updated from successor state



- $V(s_t) = V(s_t) + \alpha[r + V(s_{t+1}) - V(s_t)]$

TD(λ)

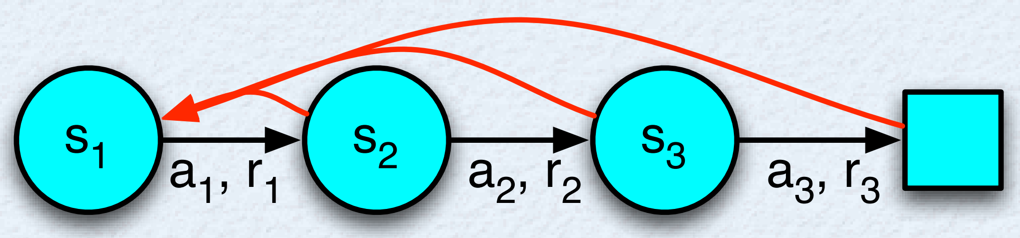
- Value function is updated from future estimates
- Temporal difference parameter λ controls timescale
- $\lambda=1$: value function is updated from actual return



- $V(s_t) = V(s_t) + \alpha[R_t - V(s_t)]$

TD(λ)

- Value function is updated from future estimates
- Temporal difference parameter λ controls timescale
- General λ: value function updated from future values



- $V(s_t) = V(s_t) + \alpha[R^{\lambda} - V(s_t)]$

Sarsa(λ)

- TD(λ) is used to evaluate the current policy
- Policy is updated to be ε-greedy with respect to action value function
- Exploration parameter ε ensures all states and actions are visited
- Converges on optimal policy

Linear Sarsa(λ)

- Approximate value function by a linear combination of features $\phi(s,a)$ and parameters θ, $Q(s,a) = \theta^T\phi(s,a)$
- Tabular Sarsa is a special case

Sample Based Search

Sample Based Search

- Sample based learning applied to simulated experience
- Experience is simulated using a *transition model* and *reward model*
- Complete episodes are simulated from current state
- Learning is specialised to the distribution of states encountered from the current position

Self-Play

- For two-player, zero sum games
- Self-play provides a model of the environment
- Assumption: opponent plays as we do
- Sample-based search with table lookup and self-play converges on the minimax solution

Sample-Based Search Algorithms

- Monte-Carlo simulation
- Monte-Carlo Tree Search
- UCT
- UCT-RAVE

Advantages

- Evaluation is grounded in experience
- Experience is sampled selectively
- Can use state abstraction
- Accuracy of evaluation increases over time
- High performance, anytime algorithms
- Relatively easy to parallelise

A Unified Framework

Dyna

- Combines learning with search
- Experience is simulated using a model
- Value function is updated from real experience
- Value function is also updated from simulated experience
- Sarsa(λ) update rule

Dyna-2

- Permanent memory updated from real experience
- Transient memory updated from simulated experience
- Linear Sarsa(λ) update rule
- Value function combines both memories
- Transient memory is reset at start of new episode

Choices

- Model: self-play, ...
- Features: table lookup, local shape features, RAVE, ...
- Exploration policy: UCB, ϵ-greedy, ...
- Bootstrapping: Monte-Carlo, TD
- Learning rate: 1/N, constant, ...

Choices: UCT

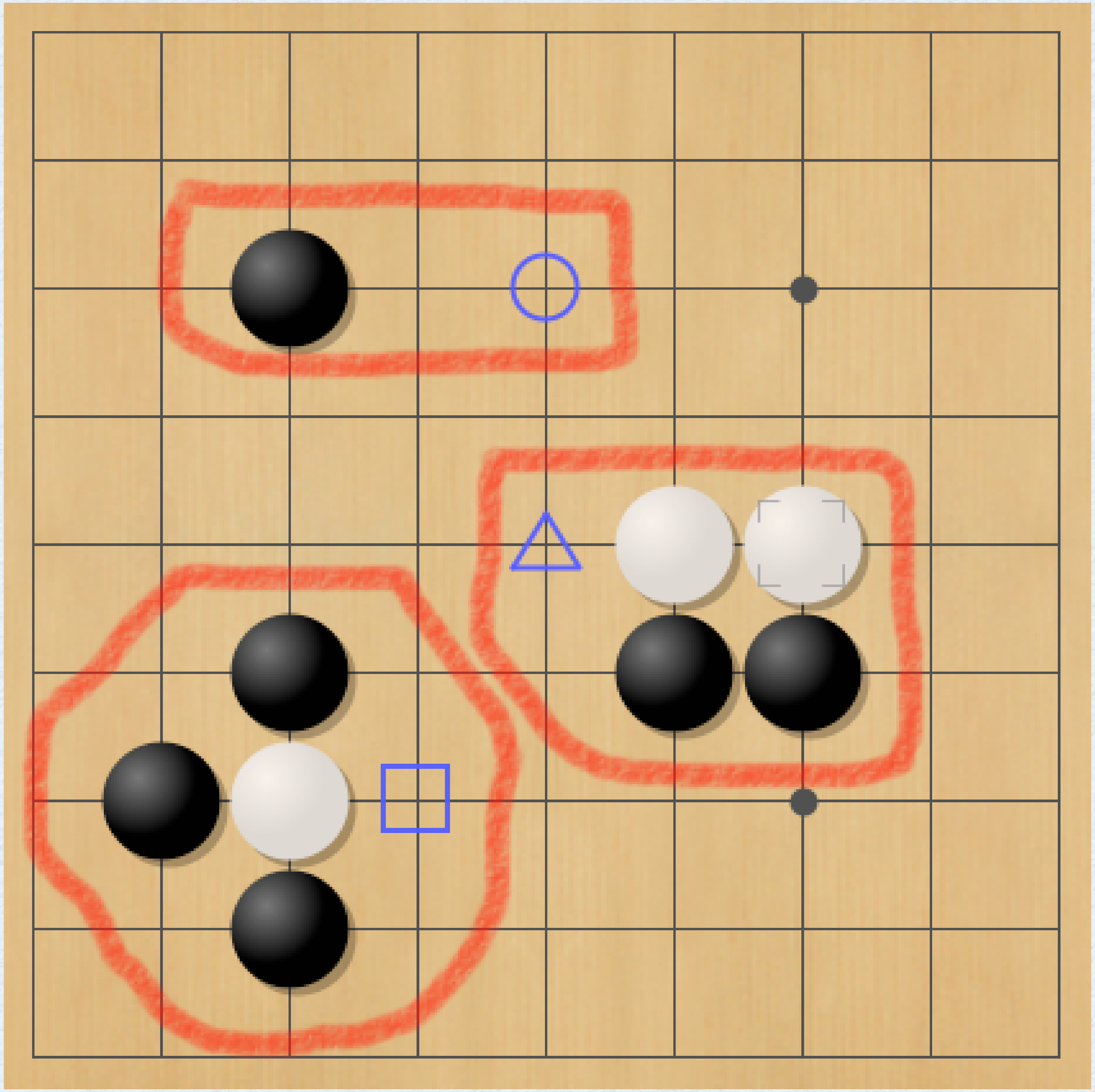
- Model: self-play
- Features: table lookup, local shape features, RAVE, ...
- Exploration policy: UCB, ϵ-greedy, ...
- Bootstrapping: Monte-Carlo, TD
- Learning rate: 1/N, constant, ...

Choices: RLGO

- Model: self-play
- Features: table lookup, local shape features, RAVE, ...
- Exploration policy: UCB, ϵ-greedy, ...
- Bootstrapping: Monte-Carlo, TD
- Learning rate: 1/N, constant, ...

Local Shape Features

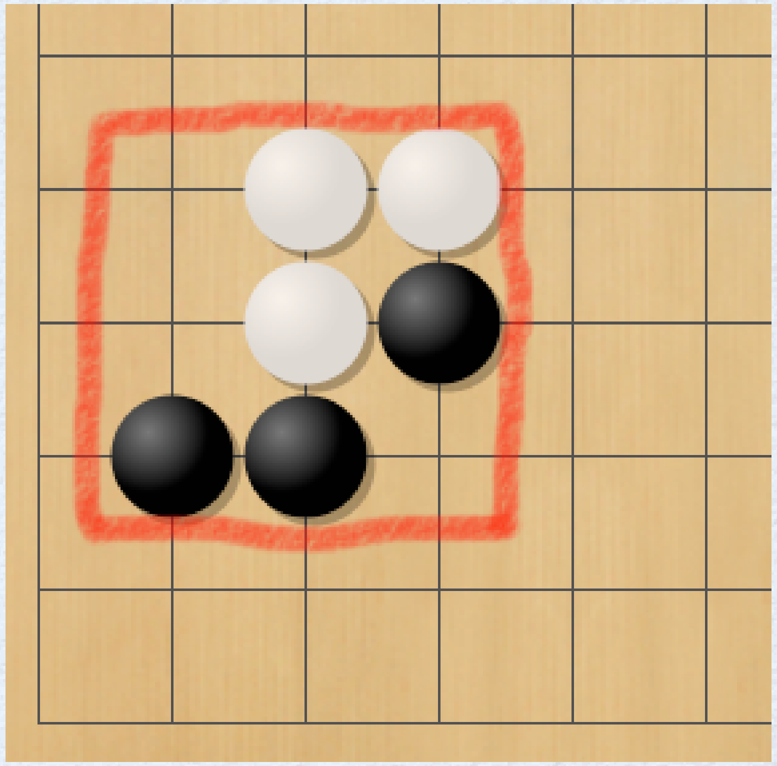
Go Proverbs



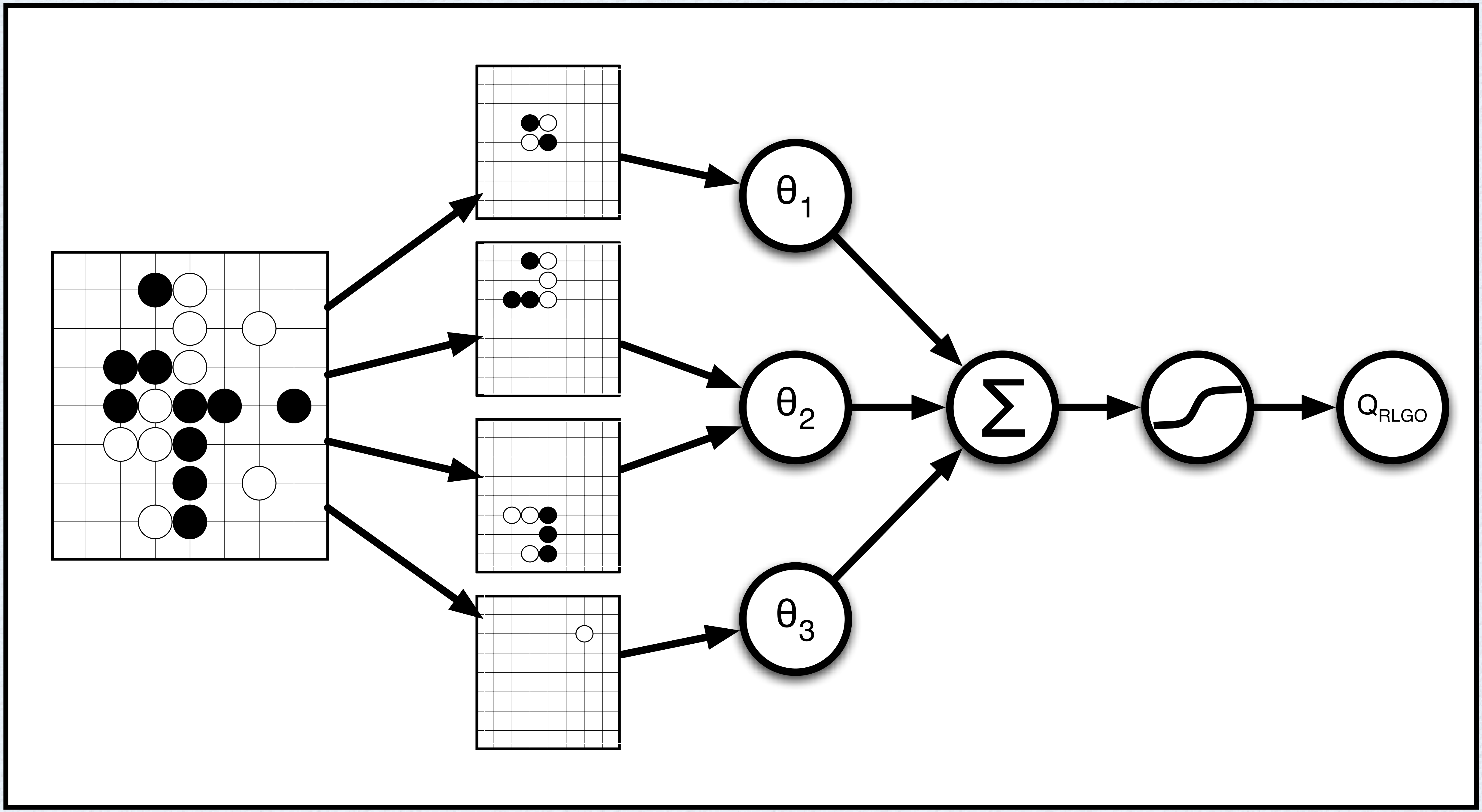
- The one point jump is never wrong
- Hane at the head of two stones
- Ponnuki is worth 30 points

Local Shape Features

- A *local shape* is a square on the board specifying a configuration of stones
- All possible configurations are used from 1x1 through to 3x3
- ~1 million binary features



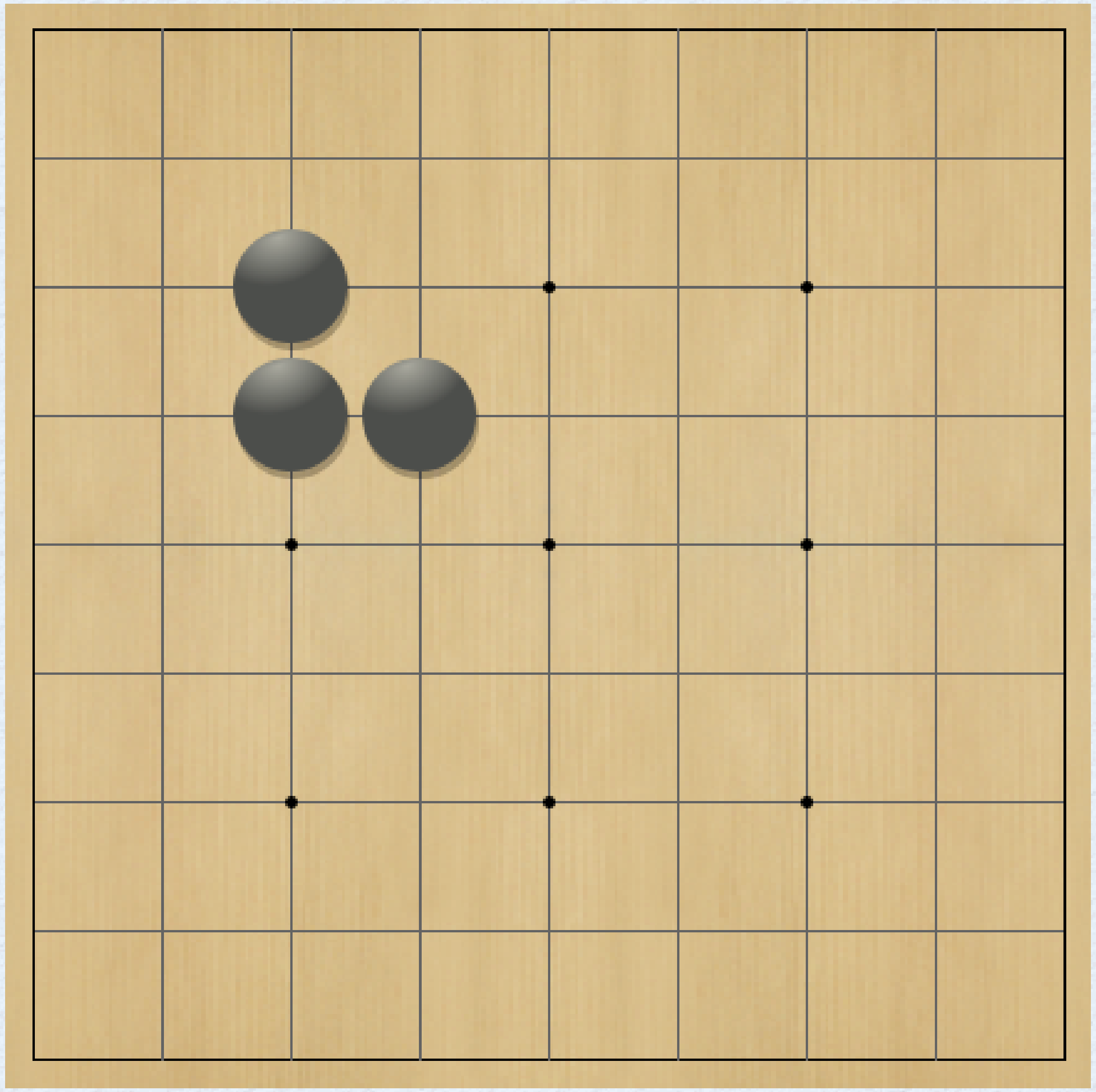
RLGO Value Function



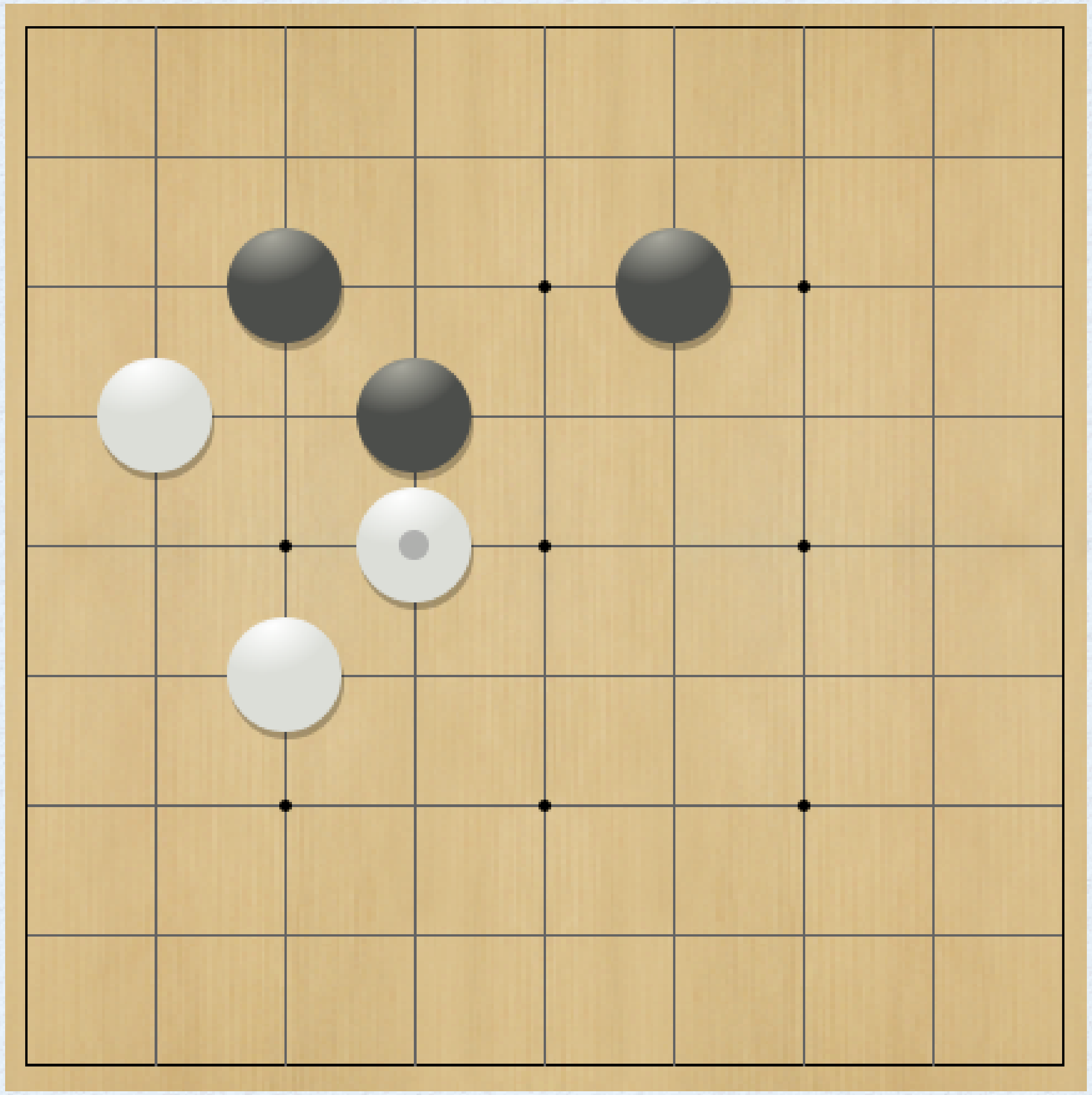
Two Memories

- Local shape features
- Two sets of parameters
- Learning memory: general shape
- Search memory: contextual shape

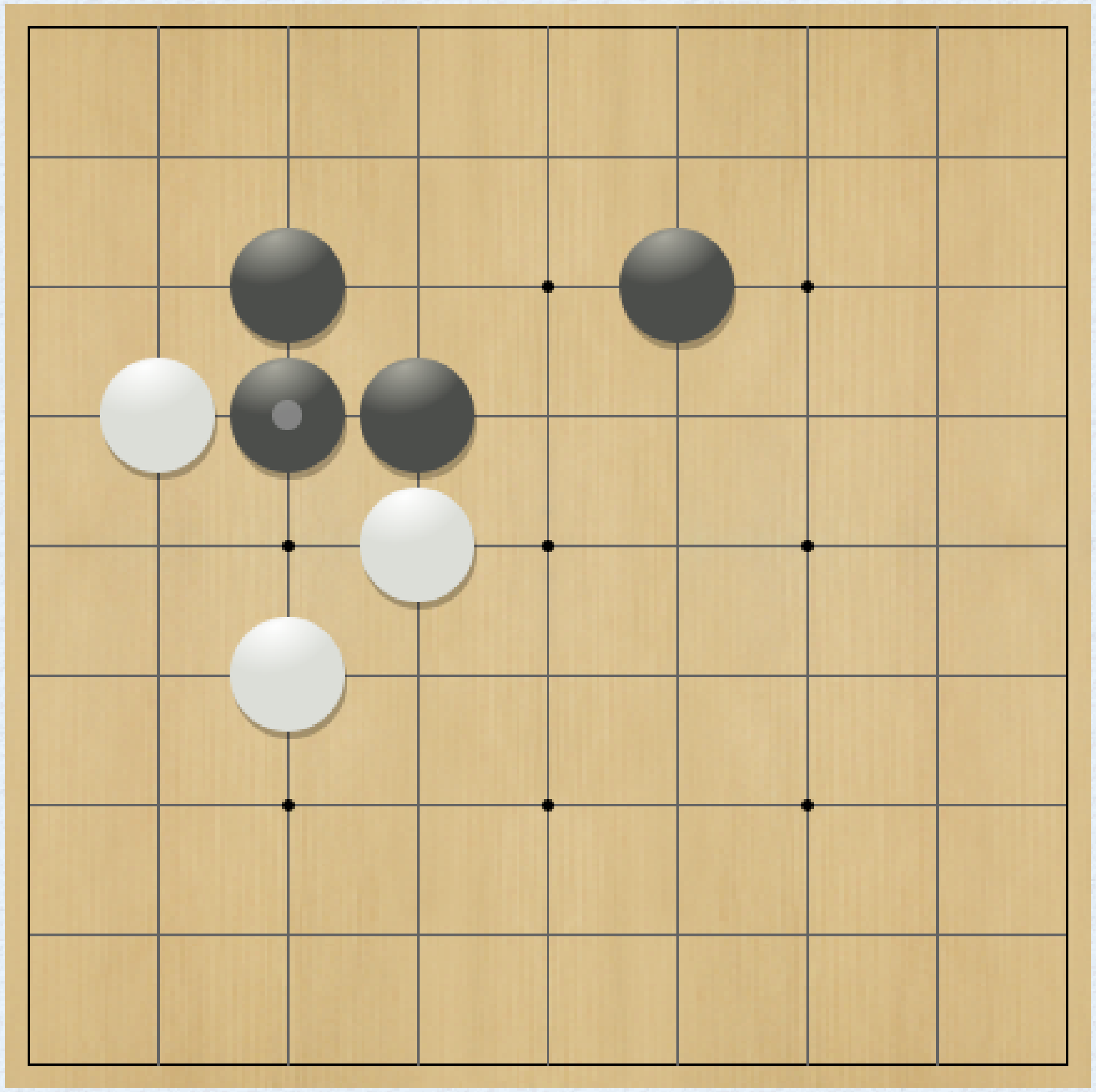
Empty Triangle: Bad Shape



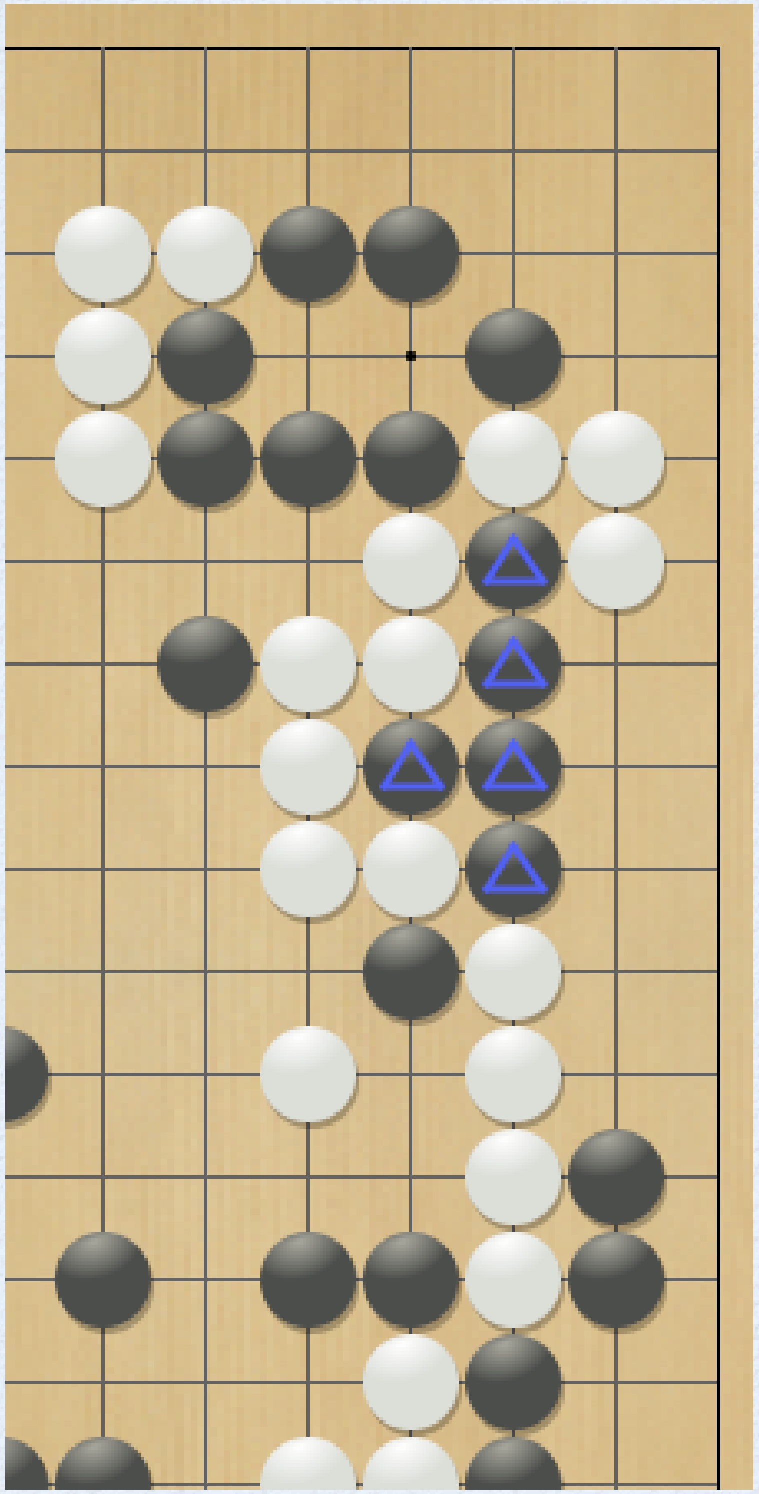
Empty Triangle: Guzumi



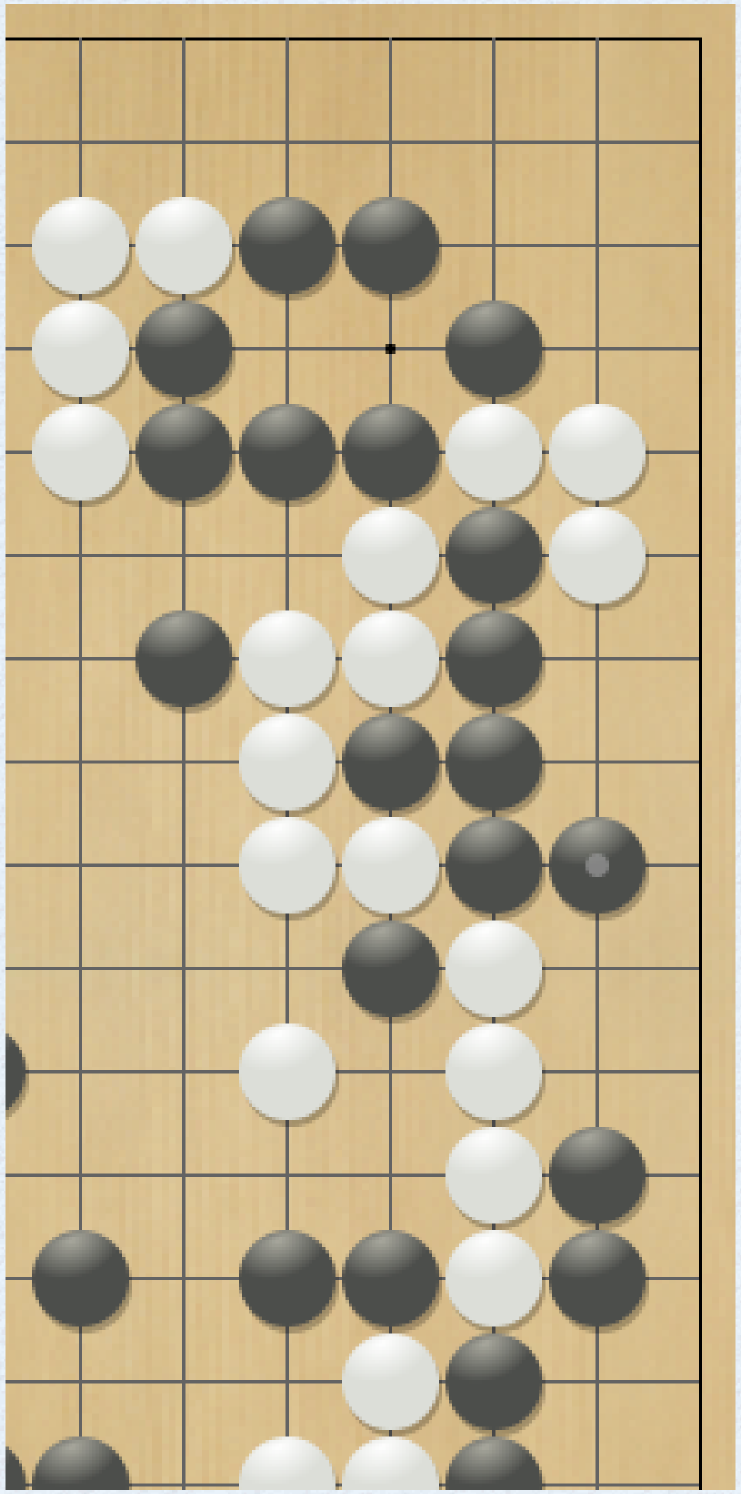
Empty Triangle: Guzumi



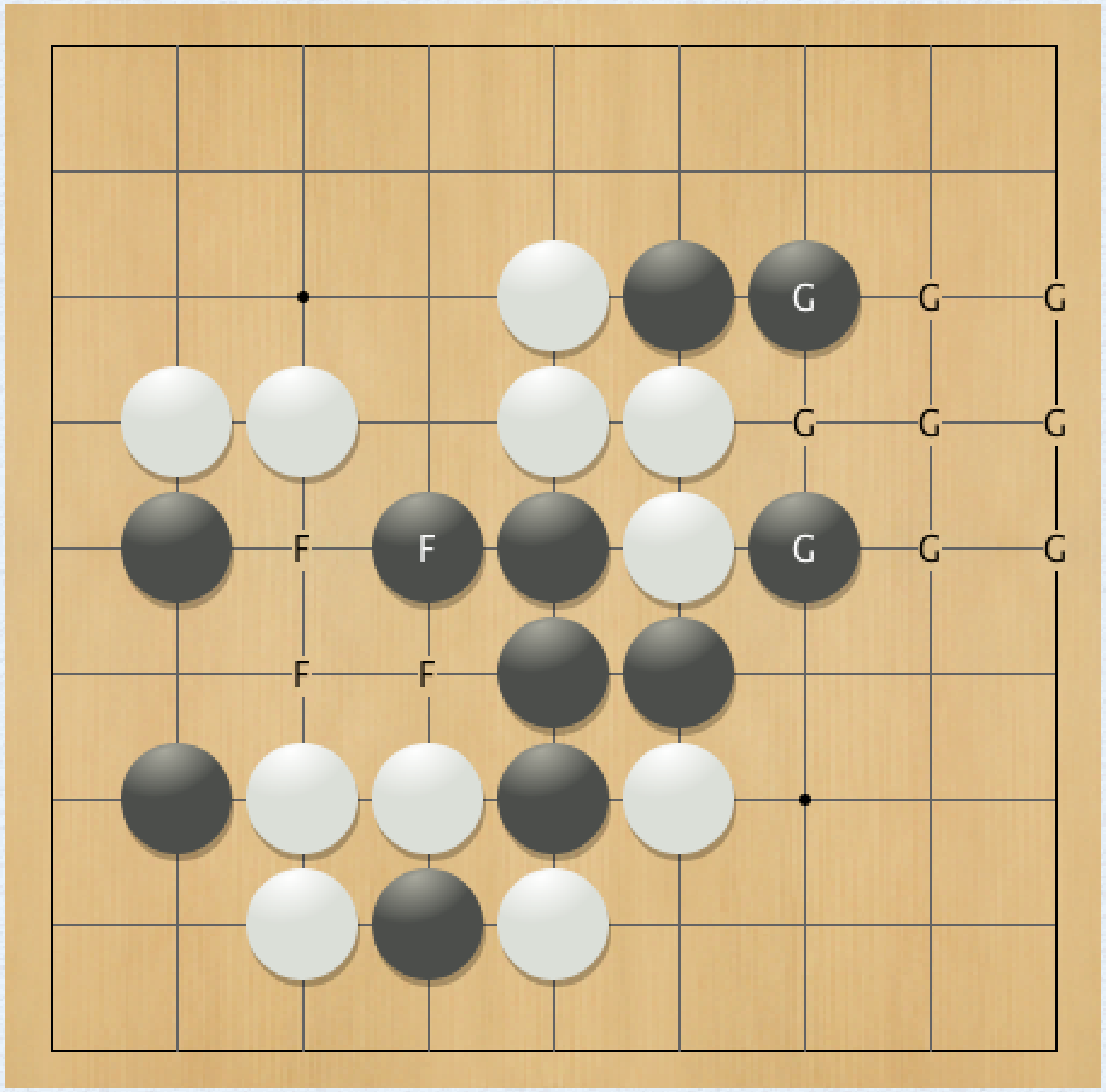
Empty Triangle: “Brilliant”



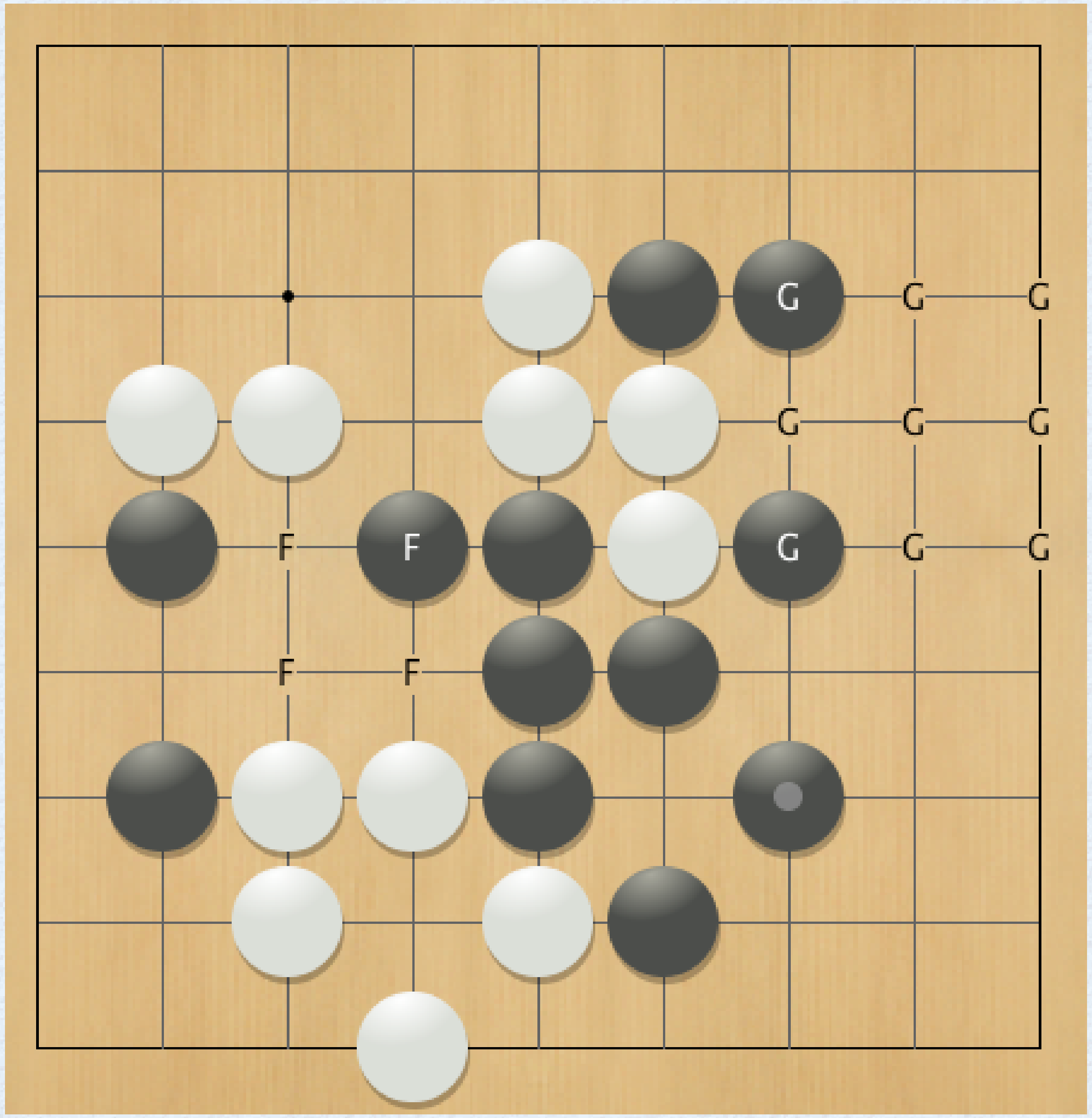
Empty Triangle: “Brilliant”



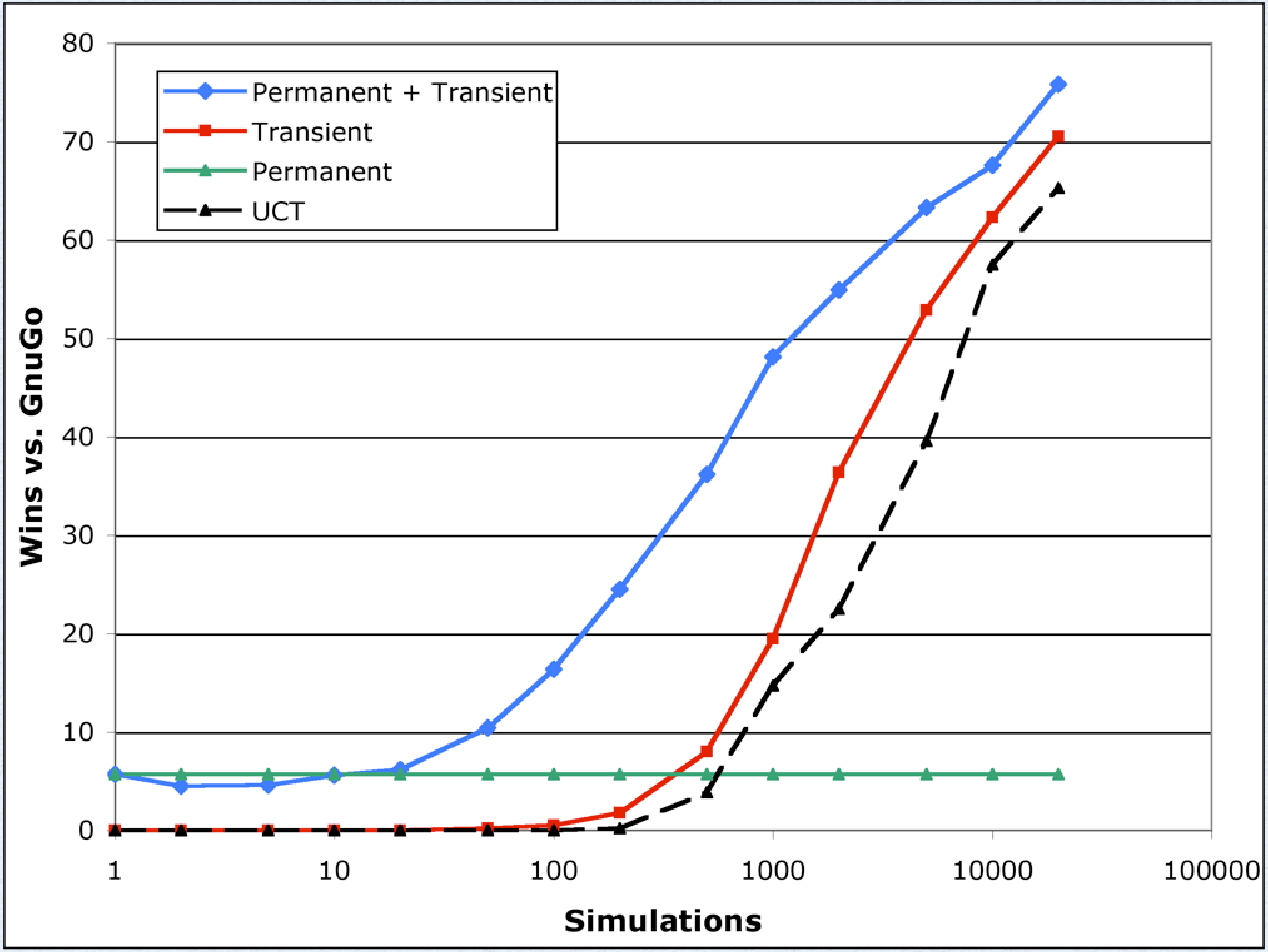
Generalisation in Search



Generalisation in Search



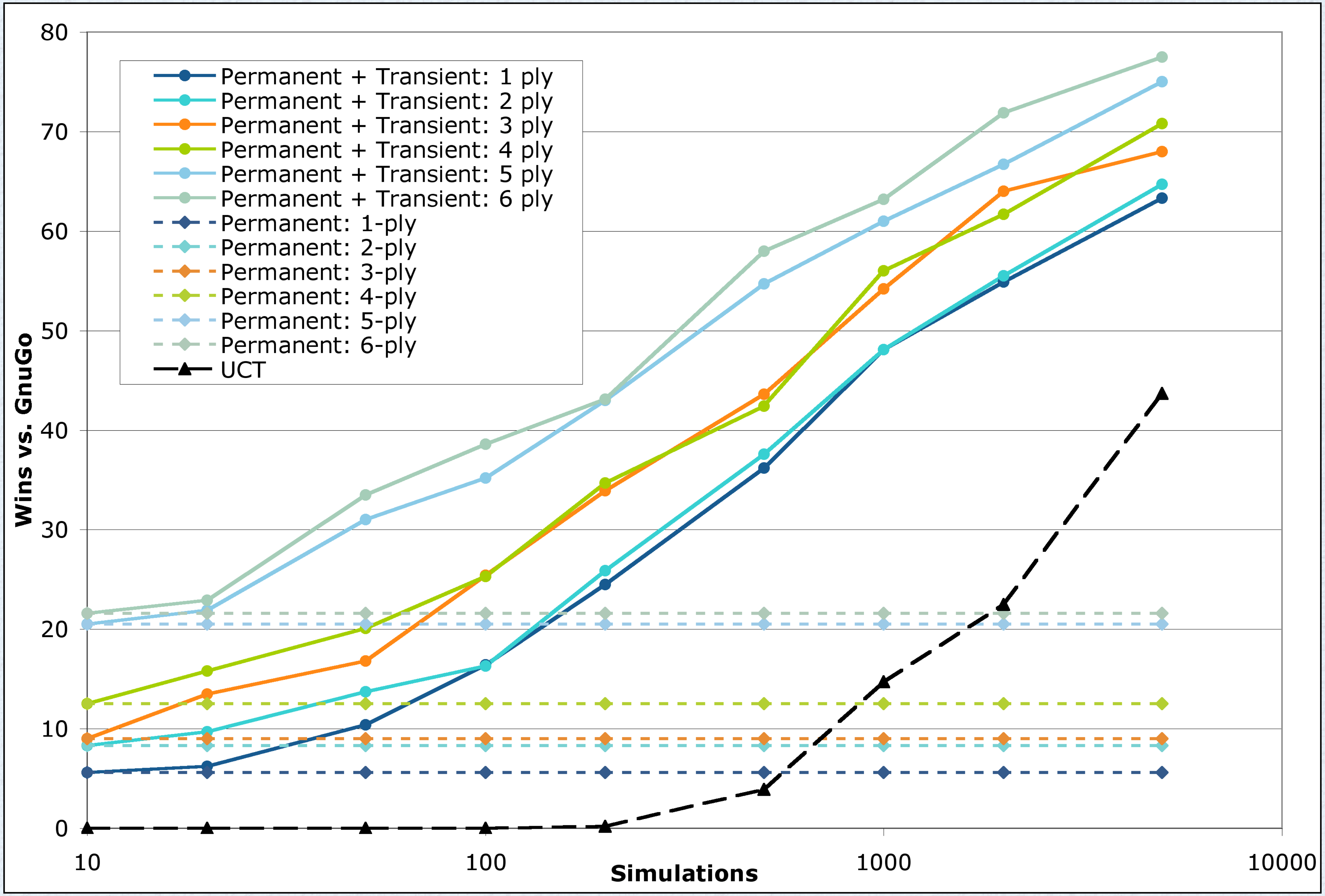
Results for Dyna-2



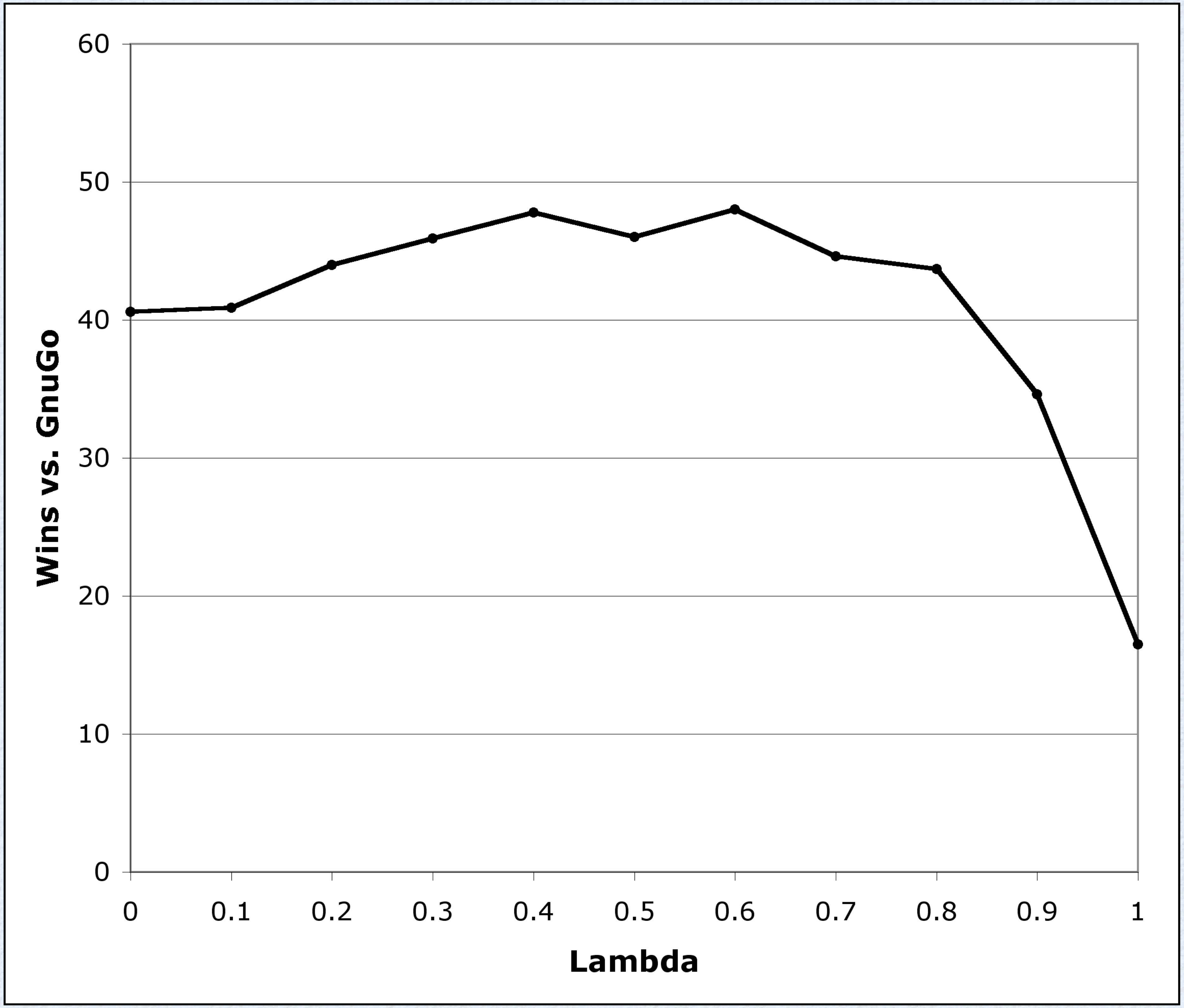
Dyna-2 with Alpha-Beta

- Dyna-2 value function provides an evaluation function
- Adapted online to the current context
- Why not use traditional alpha-beta search
- Using permanent + transient memory as evaluation functions?

Dyna-2 + Alpha-Beta



Bootstrapping



9x9 Go programs

Program	*Learning*	*Search*	*Elo*
Magog	Supervised learning	Global alpha-beta search	~1700
GnuGo	(Handcrafted)	Local alpha-beta search	~1800
NeuroGo	Temporal difference learning	Global alpha-beta search	~1800
RLGO	Temporal difference learning	Abstract search	~2100
MoGo	Temporal difference learning	UCT-RAVE	~2500
CrazyStone, GreenPeep	Supervised learning	UCT	~2500

Summary

- Generalisation during sample based search outperforms UCT
- Combining learning and search combines either method alone
- Dyna-2 provides a principled architecture for learning and search with generalisation
- Heuristic UCT and UCT-RAVE are special cases

Questions?