Coherent Inference on Optimal Play in Game Trees

inference in exponentially big trees, in linear time

Philipp Hennig, David Stern, Thore Graepel

AISTATS 2010
Chia Laguna, Sardinia

Overview

the main idea:

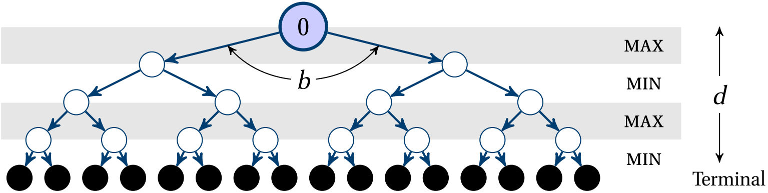
Games evolve smoothly. This insight allows approximate inference on optimal play, using data from nonoptimal play.

You might also like this talk if you are interested in …

- Reinforcement Learning
- Probabilistic Optimization
- Graphical Models
- Bandit Problems
- Tree Search
- Approximate Inference

Exact Game Tree Search is Hard

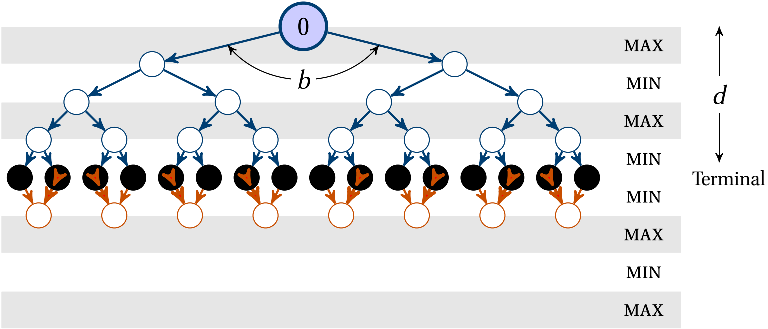
games are AND/OR Trees



- real game trees are very big (*Go* has $\sim 10^{400}$ nodes)

Exact Game Tree Search is Hard

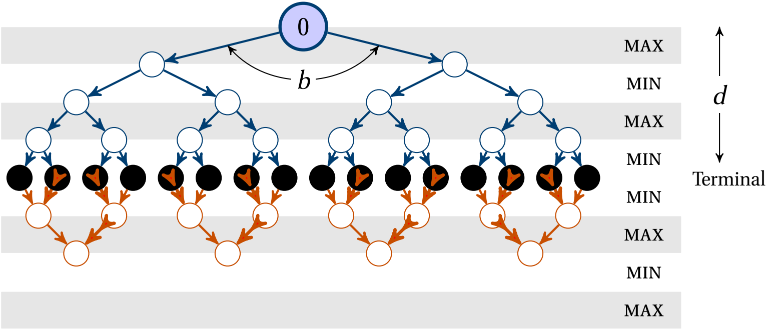
games are AND/OR Trees



- real game trees are very big (*Go* has $\sim 10^{400}$ nodes)
- a min/max (AND/OR) optimization problem
- solving exact is $\mathcal{O}(b^d)$ (with good heuristics, $\mathcal{O}(b^{d/2})$ at best)

Exact Game Tree Search is Hard

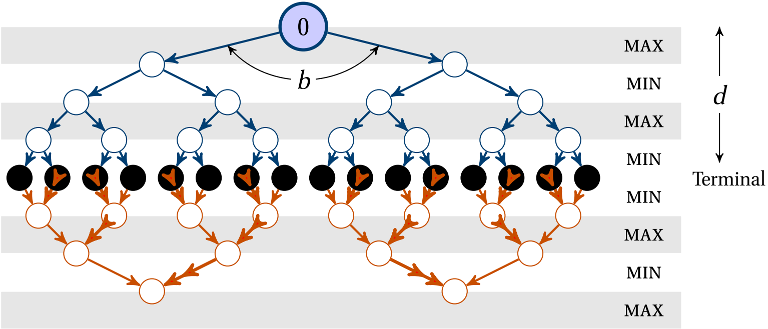
games are AND/OR Trees



- real game trees are very big (*Go* has $\sim 10^{400}$ nodes)
- a min/max (AND/OR) optimization problem
- solving exact is $\mathcal{O}(b^d)$ (with good heuristics, $\mathcal{O}(b^{d/2})$ at best)

Exact Game Tree Search is Hard

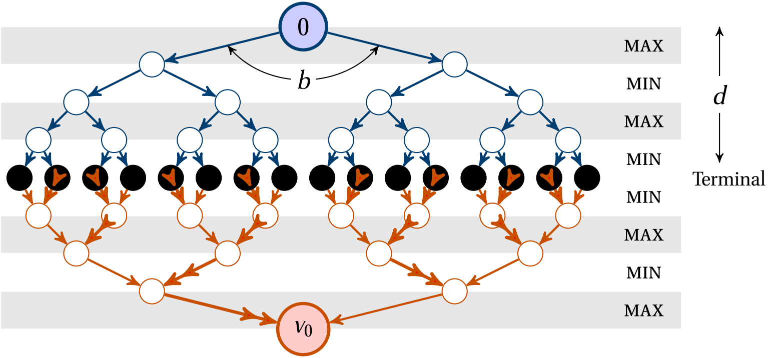
games are AND/OR Trees



- real game trees are very big (*Go* has $\sim 10^{400}$ nodes)
- a min/max (AND/OR) optimization problem
- solving exact is $\mathcal{O}(b^d)$ (with good heuristics, $\mathcal{O}(b^{d/2})$ at best)

Exact Game Tree Search is Hard

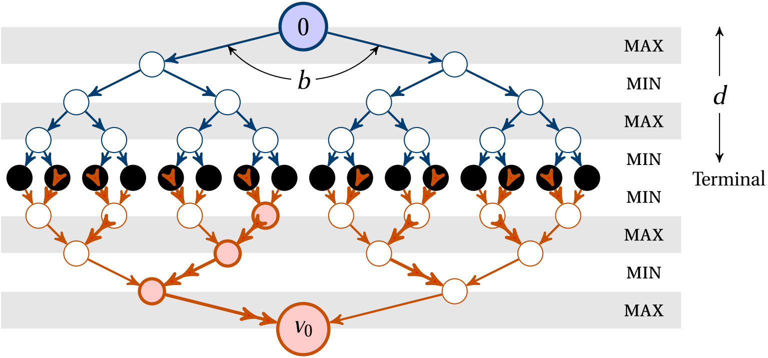
games are AND/OR Trees



- real game trees are very big (*Go* has $\sim 10^{400}$ nodes)
- a min/max (AND/OR) optimization problem
- solving exact is $\mathcal{O}(b^d)$ (with good heuristics, $\mathcal{O}(b^{d/2})$ at best)

Exact Game Tree Search is Hard

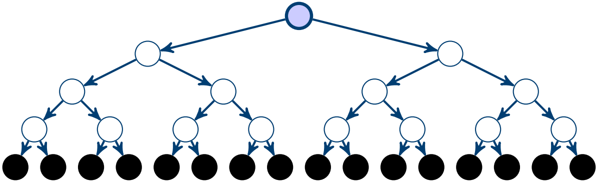
games are AND/OR Trees



- real game trees are very big (*Go* has $\sim 10^{400}$ nodes)
- a min/max (AND/OR) optimization problem
- solving exact is $\mathcal{O}(b^d)$ (with good heuristics, $\mathcal{O}(b^{d/2})$ at best)

State of the Art: Monte Carlo Tree Search

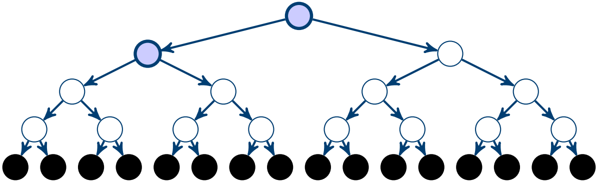
stochastically explore tree, focus on promising parts



- assymetric part of tree stored below the root
- repeated descents through the stored part, roll-outs at end
- performance depends crucially on descent policy
- one such policy: Upper Confidence Bound for Trees (UCT) [Kocsis and Szepesvári, 2006]
- works surprisingly well on Go [Gelly & Silver, 2008]

State of the Art: Monte Carlo Tree Search

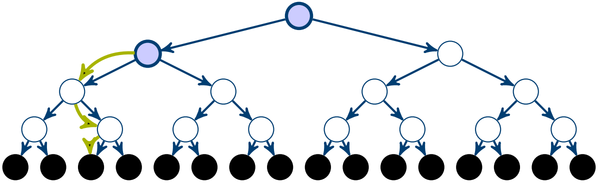
stochastically explore tree, focus on promising parts



- assymetric part of tree stored below the root
- repeated descents through the stored part, roll-outs at end
- performance depends crucially on descent policy
- one such policy: Upper Confidence Bound for Trees (UCT) [Kocsis and Szepesvári, 2006]
- works surprisingly well on Go [Gelly & Silver, 2008]

State of the Art: Monte Carlo Tree Search

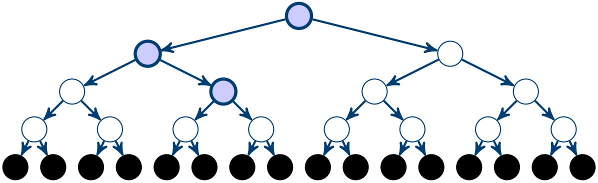
stochastically explore tree, focus on promising parts



- assymetric part of tree stored below the root
- repeated descents through the stored part, roll-outs at end
- performance depends crucially on descent policy
- one such policy: Upper Confidence Bound for Trees (UCT) [Kocsis and Szepesvári, 2006]
- works surprisingly well on Go [Gelly & Silver, 2008]

State of the Art: Monte Carlo Tree Search

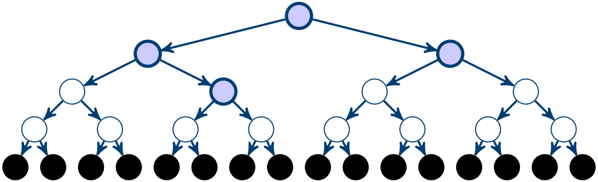
stochastically explore tree, focus on promising parts



- assymetric part of tree stored below the root
- repeated descents through the stored part, roll-outs at end
- performance depends crucially on descent policy
- one such policy: Upper Confidence Bound for Trees (UCT) [Kocsis and Szepesvári, 2006]
- works surprisingly well on Go [Gelly & Silver, 2008]

State of the Art: Monte Carlo Tree Search

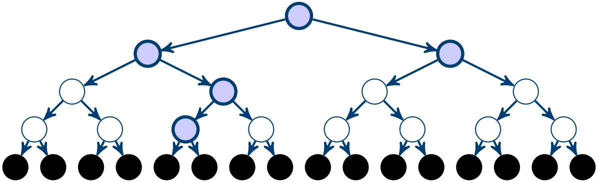
stochastically explore tree, focus on promising parts



- assymetric part of tree stored below the root
- repeated descents through the stored part, roll-outs at end
- performance depends crucially on descent policy
- one such policy: Upper Confidence Bound for Trees (UCT) [Kocsis and Szepesvári, 2006]
- works surprisingly well on Go [Gelly & Silver, 2008]

State of the Art: Monte Carlo Tree Search

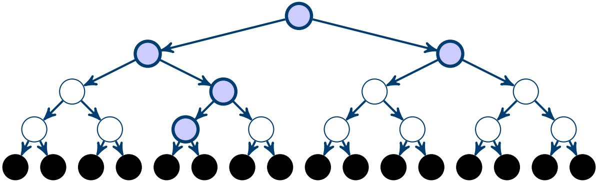
stochastically explore tree, focus on promising parts



- assymetric part of tree stored below the root
- repeated descents through the stored part, roll-outs at end
- performance depends crucially on descent policy
- one such policy: Upper Confidence Bound for Trees (UCT) [Kocsis and Szepesvári, 2006]
- works surprisingly well on Go [Gelly & Silver, 2008]

State of the Art: Monte Carlo Tree Search

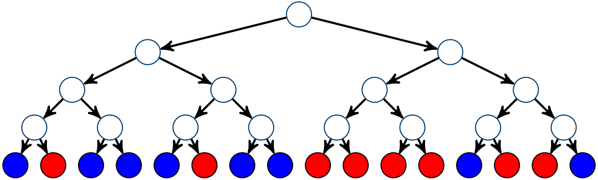
stochastically explore tree, focus on promising parts



- assymetric part of tree stored below the root
- repeated descents through the stored part, roll-outs at end
- performance depends crucially on descent policy
- one such policy: Upper Confidence Bound for Trees (UCT) [Kocsis and Szepesvári, 2006]
- works surprisingly well on Go [Gelly & Silver, 2008]
- **why?**

Games Have Structure

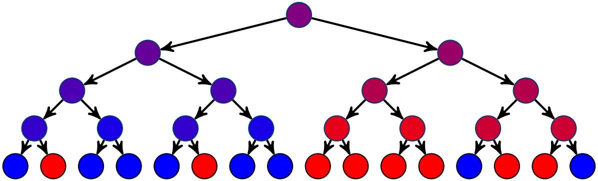
wins and losses are clustered among leaves



- otherwise MC tree search *could* not work [Pearl, 1985]
- implies a latent score g for non-terminal nodes
- each roll-out contains information about entire tree
- vanilla MC tree search passes information upwards only

Games Have Structure

wins and losses are clustered among leaves



- otherwise MC tree search *could* not work [Pearl, 1985]
- implies a latent score g for non-terminal nodes
- each roll-out contains information about entire tree
- vanilla MC tree search passes information upwards only

Building a Bayesian Tree Searcher

rest of this talk

- define generative model for latent scores g
- establish model match with Go
- approximate inference on g
- approximate inference on optimal values v

A Generative Model for On-Policy and Off-Policy Values

random play follows Brownian random walks

- let $G = \{g_i\}$ the values under the *roll-out policy*

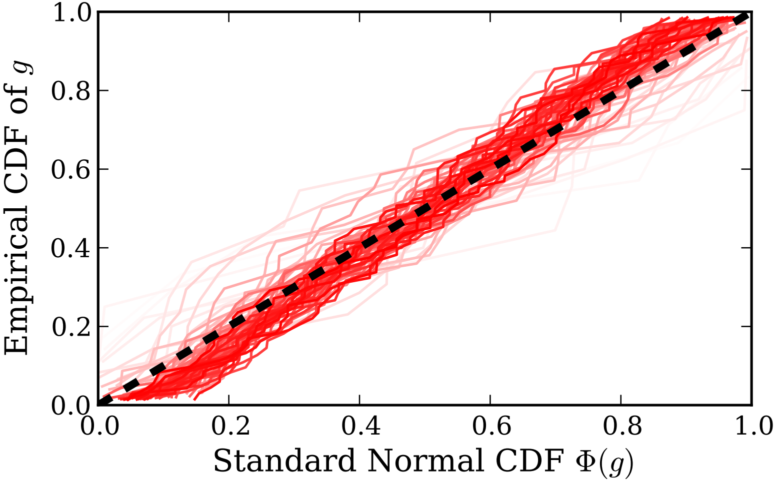
$$p(g_0) = \mathcal{N}(g_0; \mu_0, \sigma_0^2)$$
$$p(g_i | g_{\text{parent}(i)}) = \mathcal{N}(g_i; g_{\text{parent}(i)}, 1)$$

- let $V = \{v_i\}$ the values under *optimal play*

$$v_i = \begin{cases} g_i & \text{if } i \text{ is terminal} \\ \max_{\text{children}(i)}\{v_c\} & \text{if MAX plays at } i \\ \min_{\text{children}(i)}\{v_c\} & \text{if MIN plays at } i \end{cases}$$

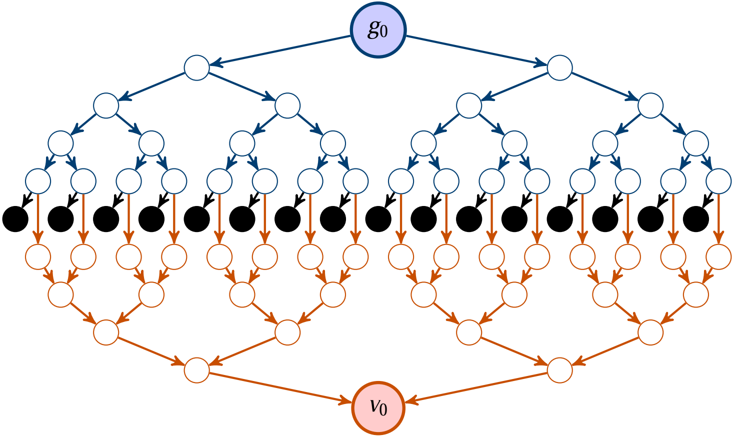
Model Matches Well With Go

empirical distributions of g values relative parents



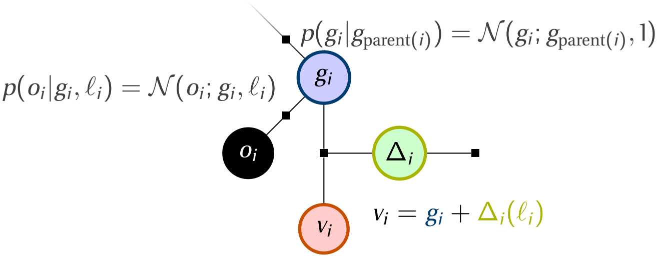
How to Obtain Marginals Over Optimal Values

generative model of optimal values



How to Obtain Marginals Over Optimal Values

inference separates into *inductive* and *deductive* parts



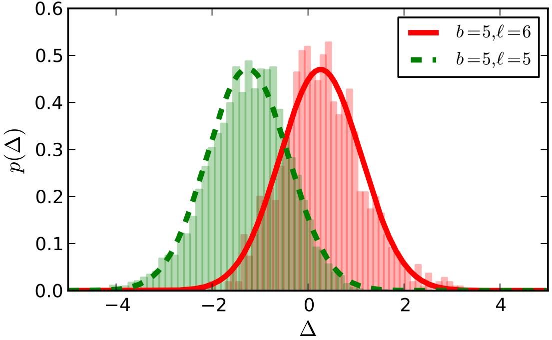
- inference separates into deductive and inductive parts

Inductive Inference by Approximate Density Evolution

Expectation Propagation yields recursive relationship on Δ

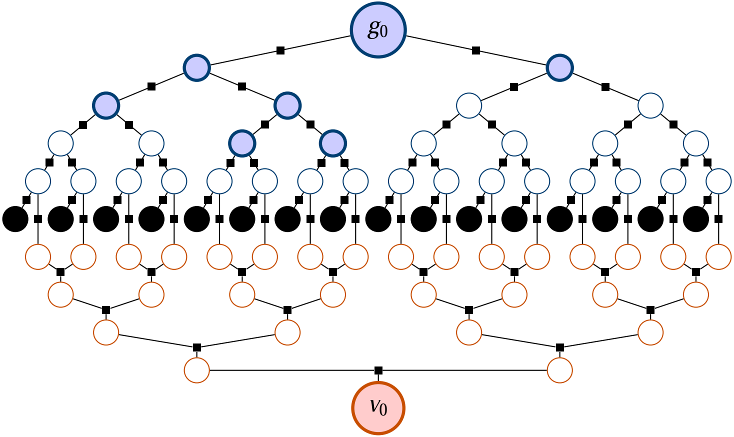
$$\Delta_i(\ell_i, b) = \begin{cases} 0 & \text{if } i \text{ terminal} \\ \max_{\text{children } j(i)}\{\Delta_j(\ell_i - 1, b) + \xi_j\} & \text{if } i \mod 2 = 0 \\ \min_{\text{children } j(i)}\{\Delta_j(\ell_i - 1, b) + \xi_j\} & \text{if } i \mod 2 = 1 \end{cases}$$

where $\xi_j \sim \mathcal{N}(0, 1)$



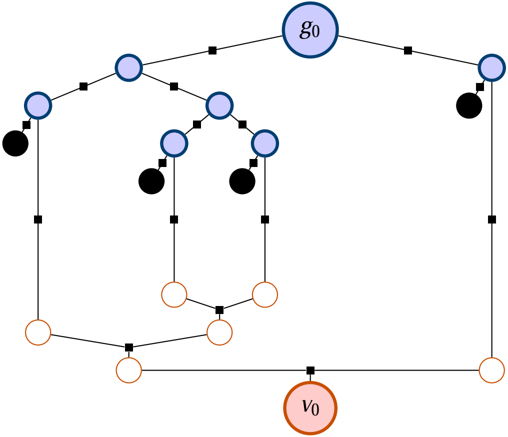
How to Obtain Marginals Over Optimal Values

factor graph representation



How to Obtain Marginals Over Optimal Values

replace unobserved parts of the tree with a probabilistic model



Notes

- an instance of Bayesian off-policy reinforcement learning
- can use any policy for stored tree (e.g. UCT) and roll-outs
- inference separates into deductive and inductive parts
- can evaluate belief over optimal value of any node in the tree
- Expectation Propagation keeps inference tractable
- marginals are coherent throughout the tree (i.e. can be used to train global evaluation function)

Conclusion

- game trees have structure
- this allows (approximate) inference on the solution of the min/max optimization problem
- see paper #113 for details