Trade-offs in Sampling-based Adversarial Planning

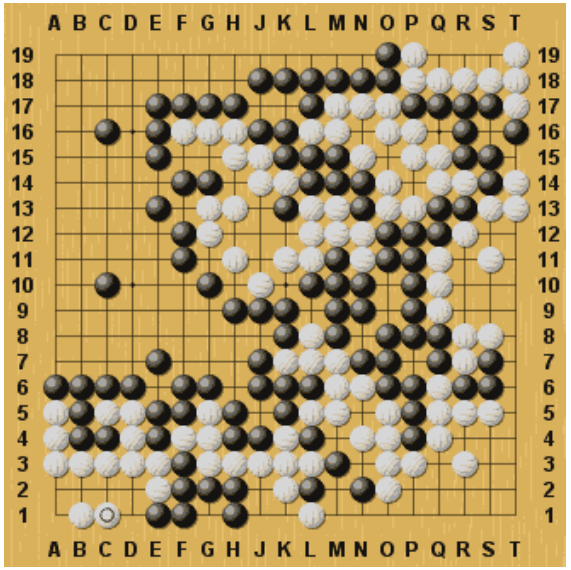
Raghuram Ramanujan

Bart Selman

Cornell University

Upper Confidence bounds for Trees (UCT)

- UCT has revolutionized computer game-playing in recent years

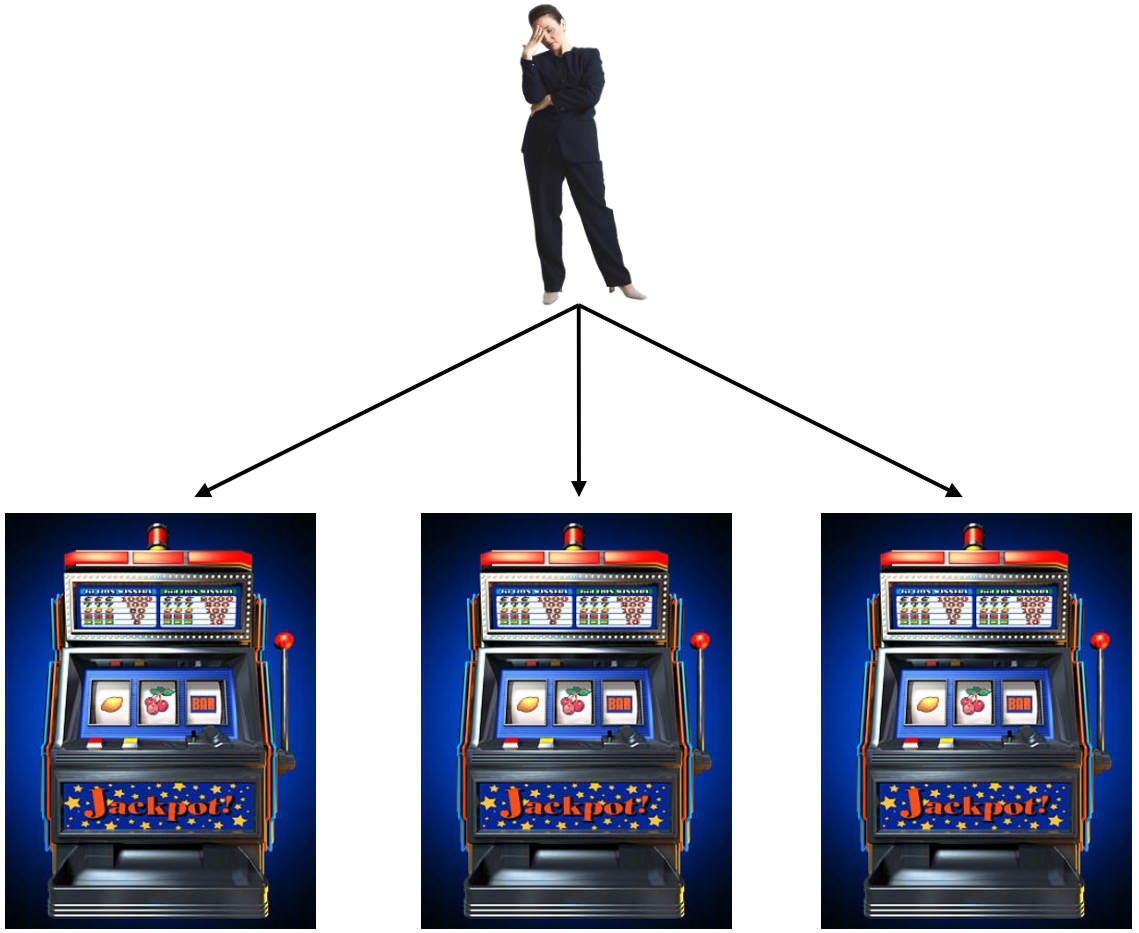


Understanding UCT

There are large gaps in our understanding of UCT

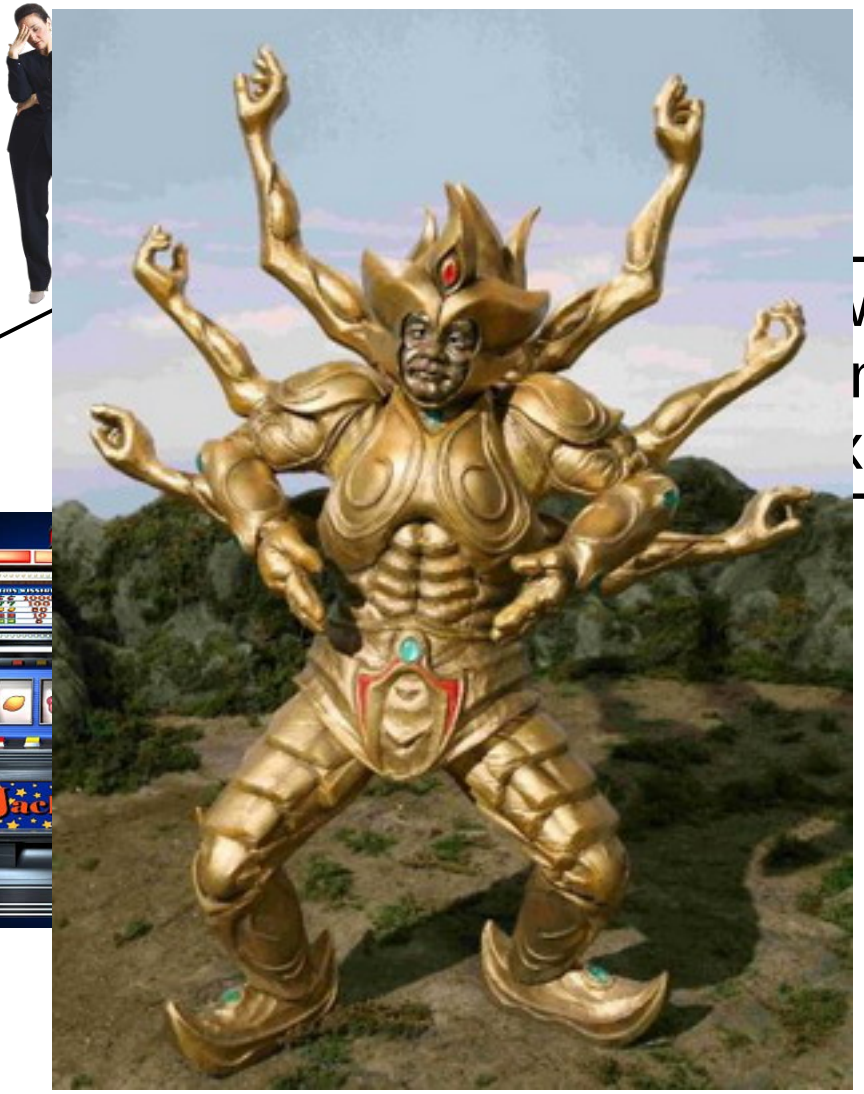
Our work focuses on gaining better insights into UCT by studying its behavior in search spaces where comparisons to Minimax search are feasible

The Multi-Armed Bandit Problem



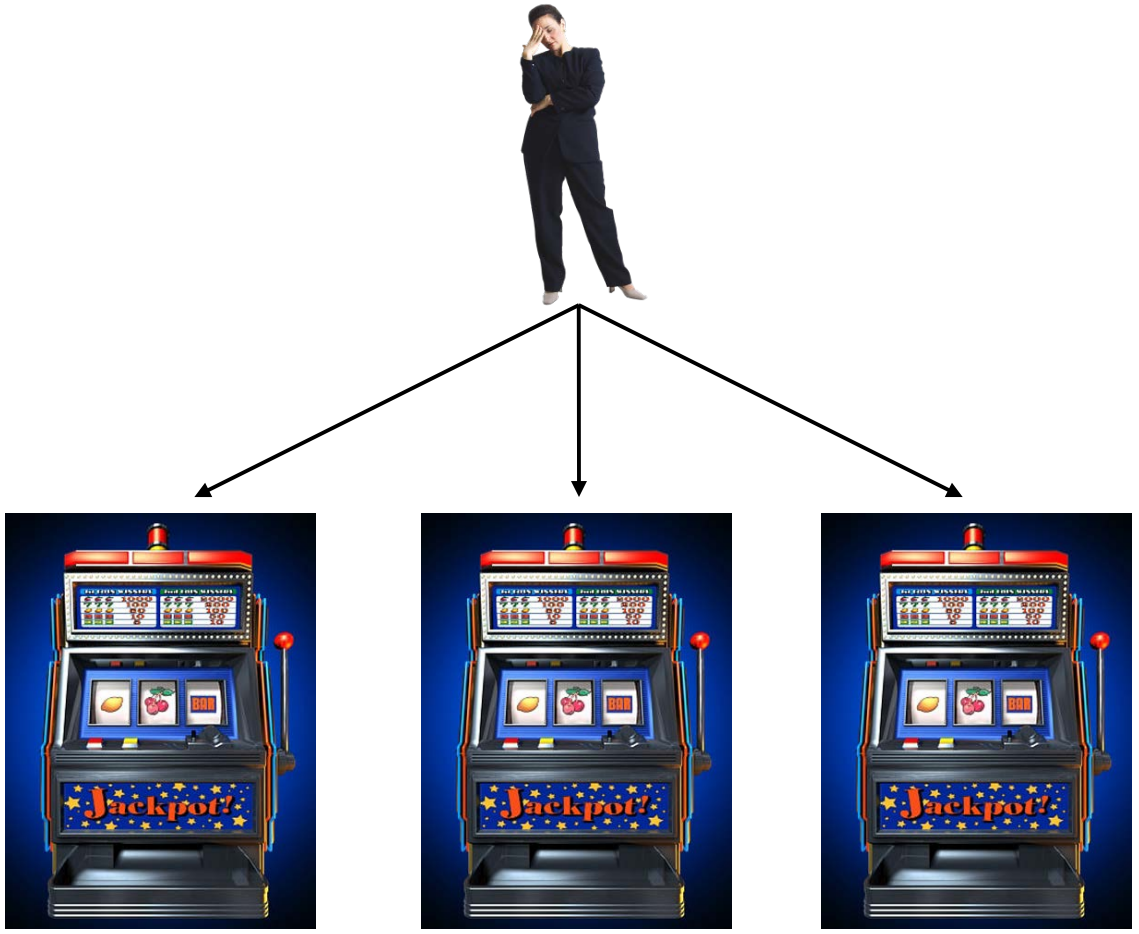
How should one play the machines to maximize the payoff?

The Multi-Armed Bandit Problem



v should one play
machines to
ximize the payoff?

The UCB1 Bandit Algorithm

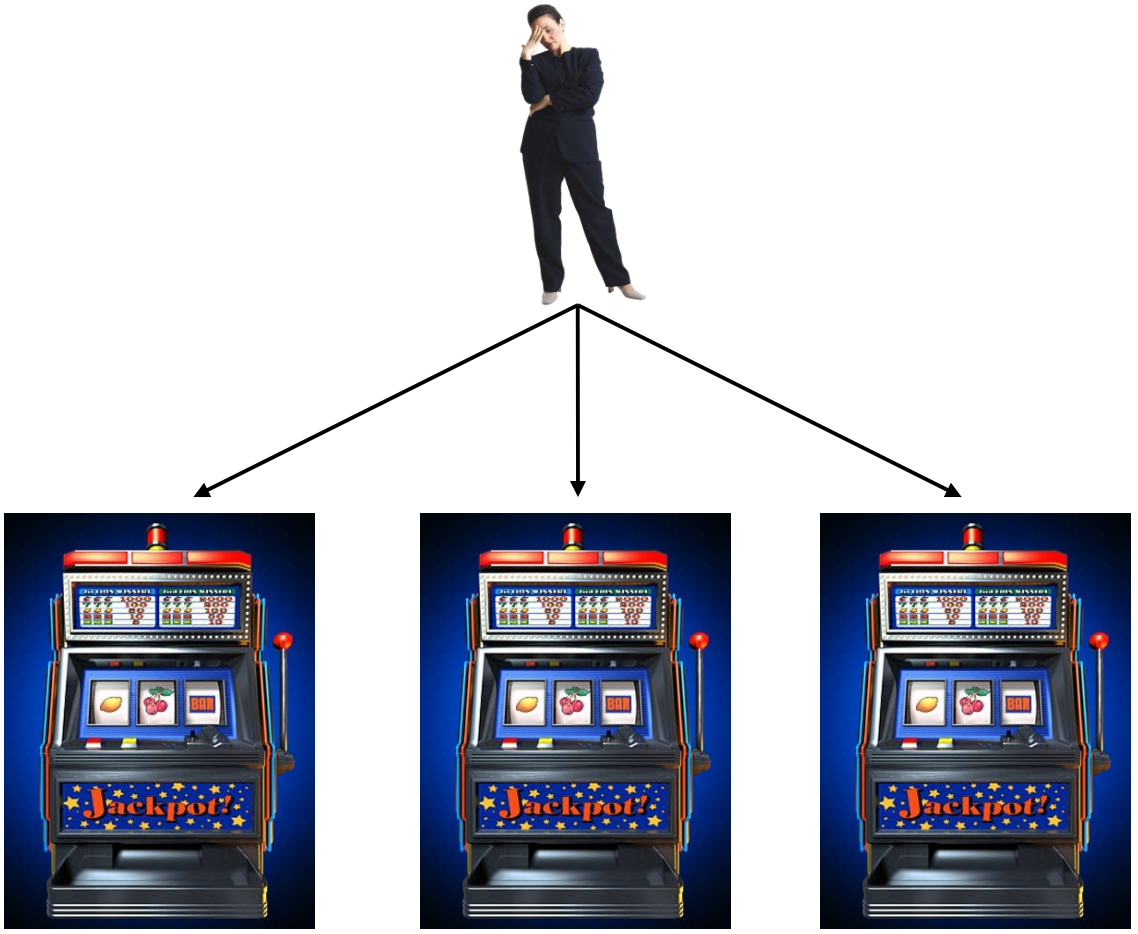


Pick the machine that maximizes:

$$Q(k) + c \cdot \sqrt{\frac{\log t}{n(k)}}$$

- $Q(k)$: observed mean payoff of machine k
- $n(k)$: number of plays of machine k
- t : number of trials

The UCB1 Bandit Algorithm



Pick the machine that maximizes:

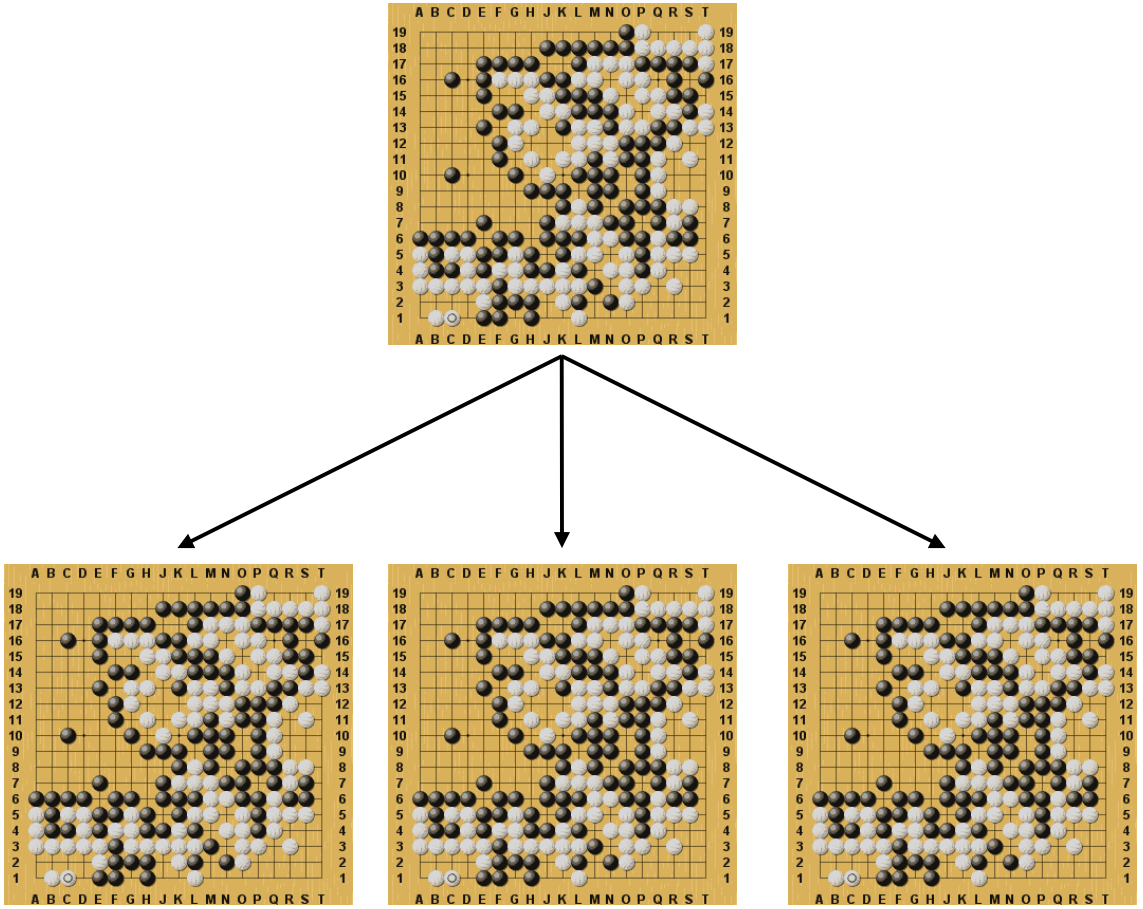
$$Q(k) + c \cdot \sqrt{\frac{\log t}{n(k)}}$$

Exploitation

Exploration

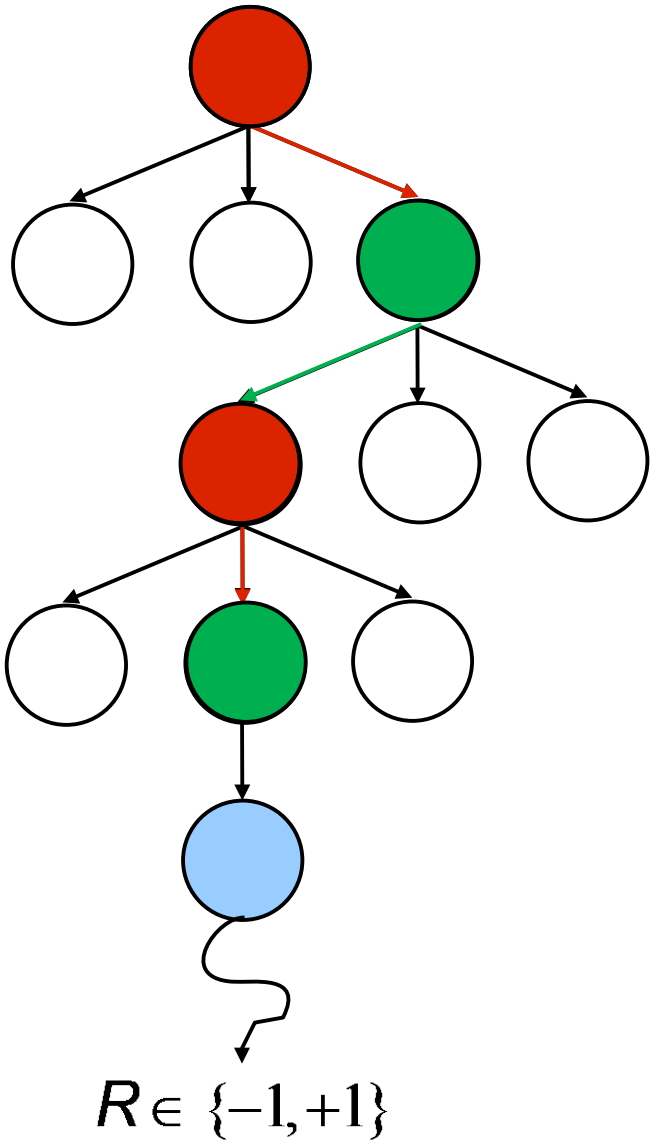
- $Q(k)$: observed mean payoff of machine k
- $n(k)$: number of plays of machine k
- t : number of trials

From Bandits to Tree Search



Key Idea: Treat every node in the game tree as a multi-armed bandit

The UCT Algorithm



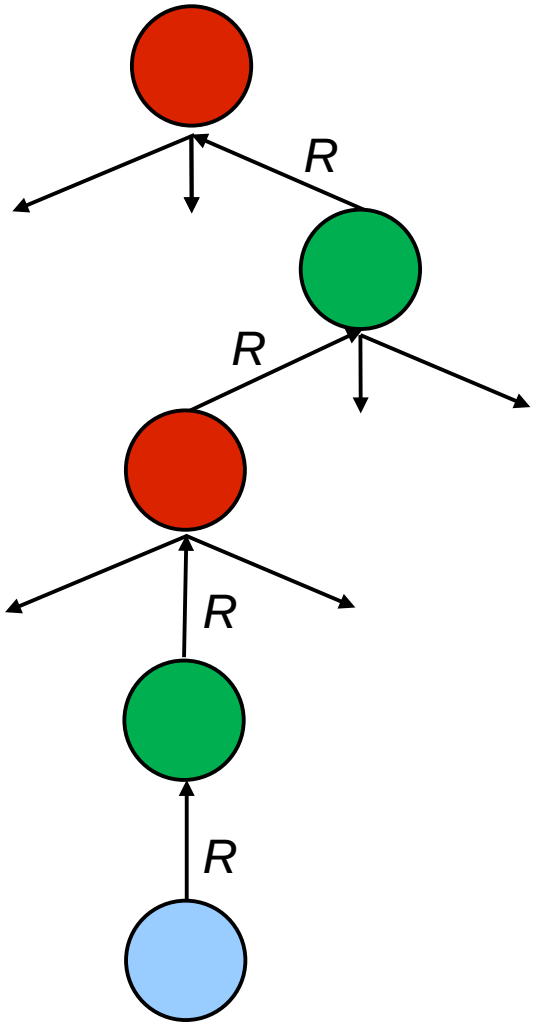
Descend the tree from the root node by applying the UCB1 node selection policy

At the opponent's nodes, a symmetric lower confidence bound is minimized

At a leaf node, a new child is created

A random playout is performed to estimate the utility *R* of this state

The UCT Algorithm



An averaging backup is used to update the value estimates of all nodes on the path from the root to the new node

$$n(s) \leftarrow n(s) + 1$$

Visit count update

$$Q(s) \leftarrow Q(s) + \frac{R - Q(s)}{n(s)}$$

$$Q(s) \leftarrow \frac{(n(s)-1) \cdot Q(s) + R}{n(s)}$$

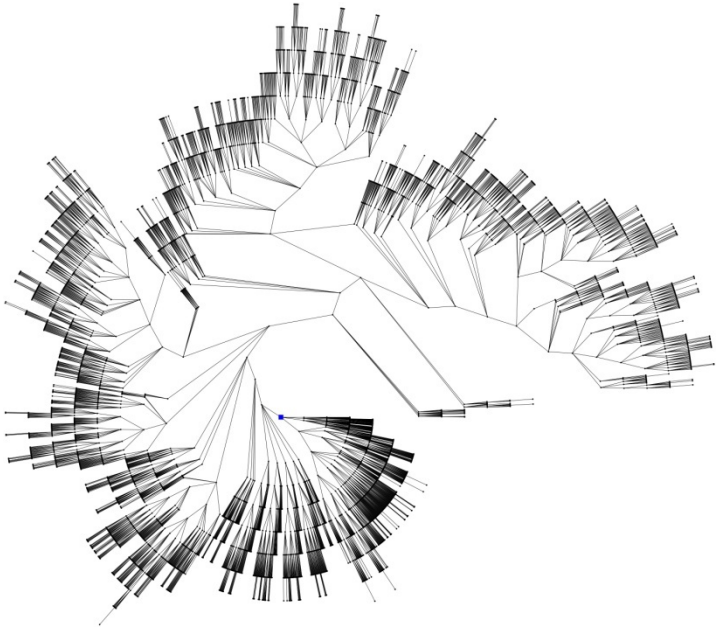
State utility update

UCT in Action

Minimax in Action

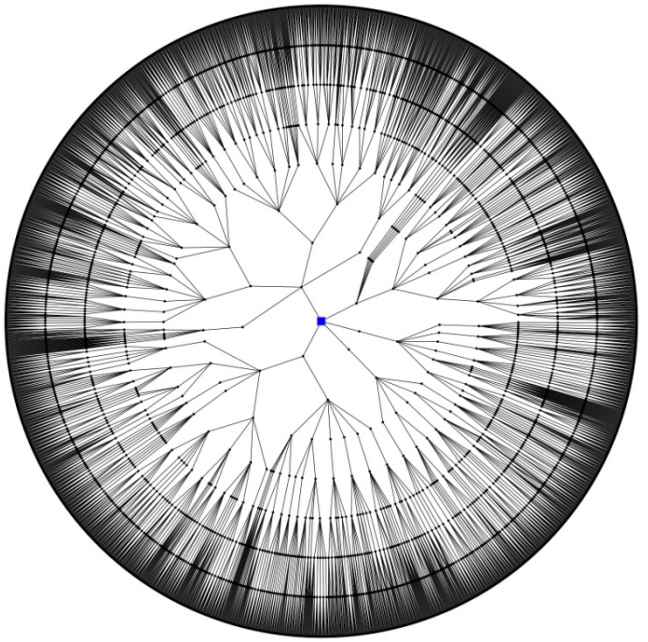
UCT versus Minimax

UCT Tree



- Asymmetric tree
- Best-performing method for Go

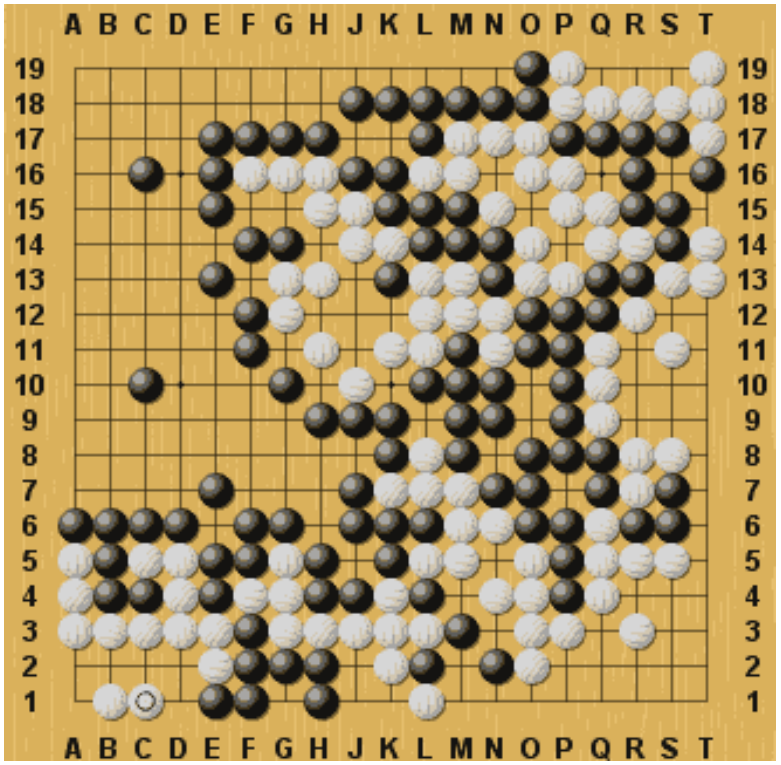
Minimax Tree



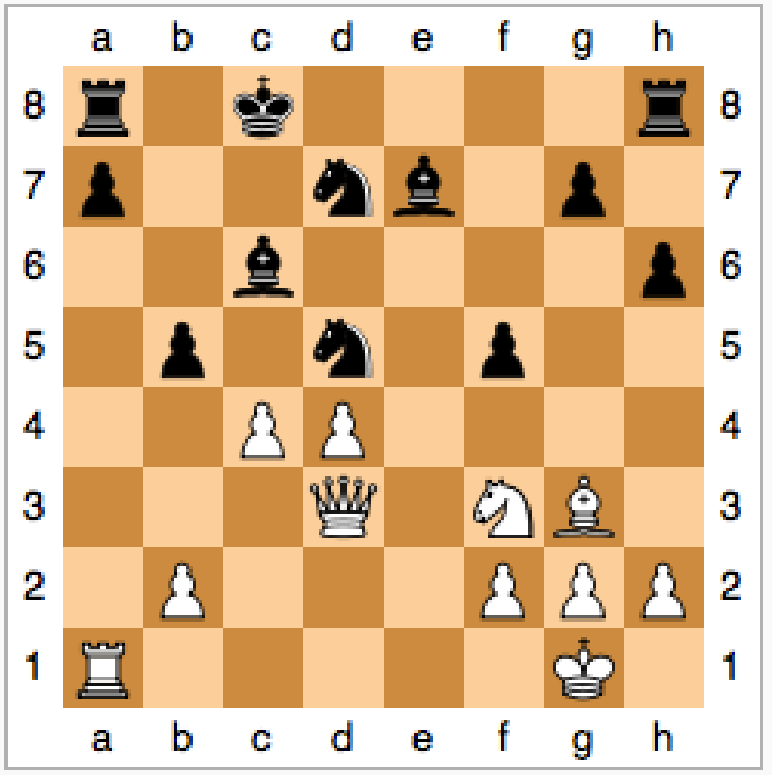
- Complete tree up to some depth bound *k*
- Best-performing method for Chess

UCT versus Minimax

Minimax is too weak at Go

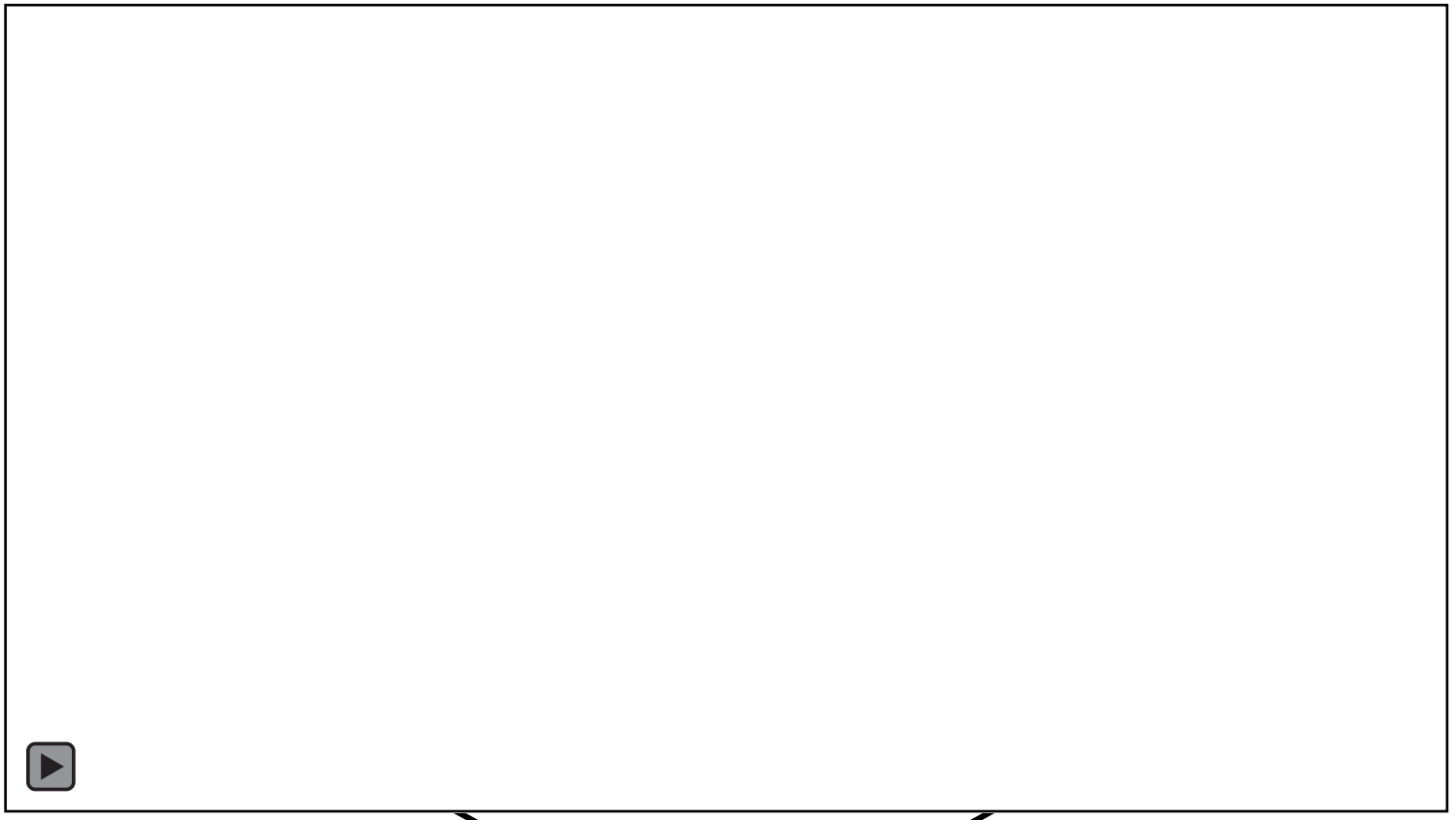


UCT is too weak at Chess



We need a domain where *both* UCT and Minimax search produce good players, with minimal enhancement

Mancala



A move = sow stones in counter clockwise fashion

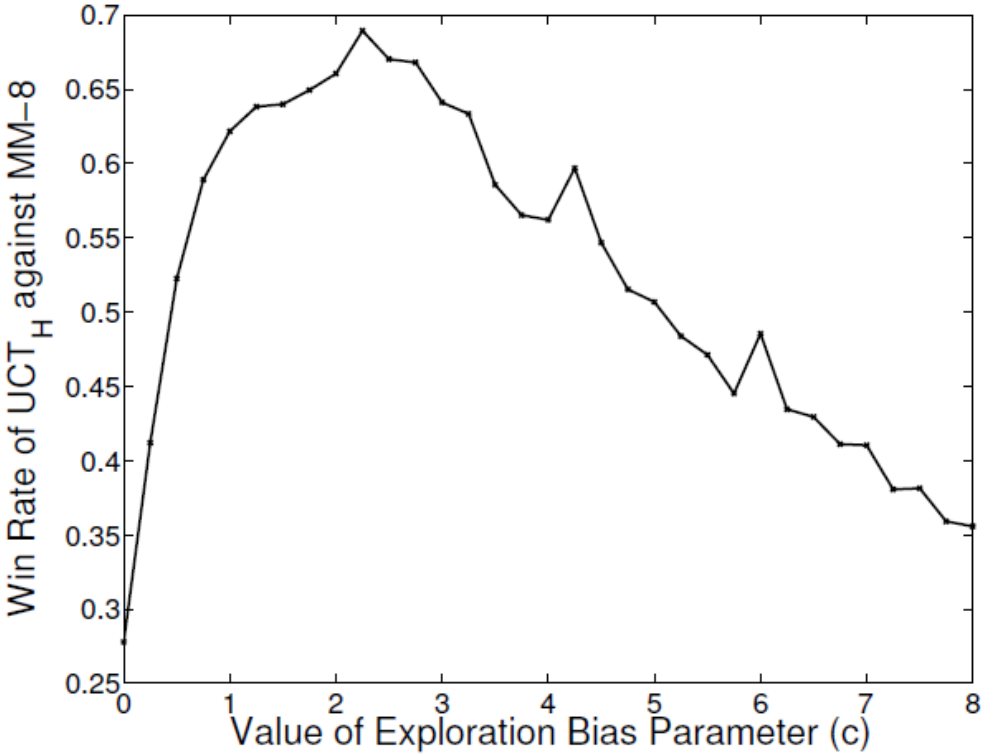
Stores

UCT in Mancala

- We examine three key trade-offs in UCT while designing a winning Mancala player
  - Complete versus selective search
  - More sampling versus more nodes
  - Averaging back-ups versus minimax back-ups
- We deploy the winning UCT agent in a novel partial game setting to understand *how* it wins

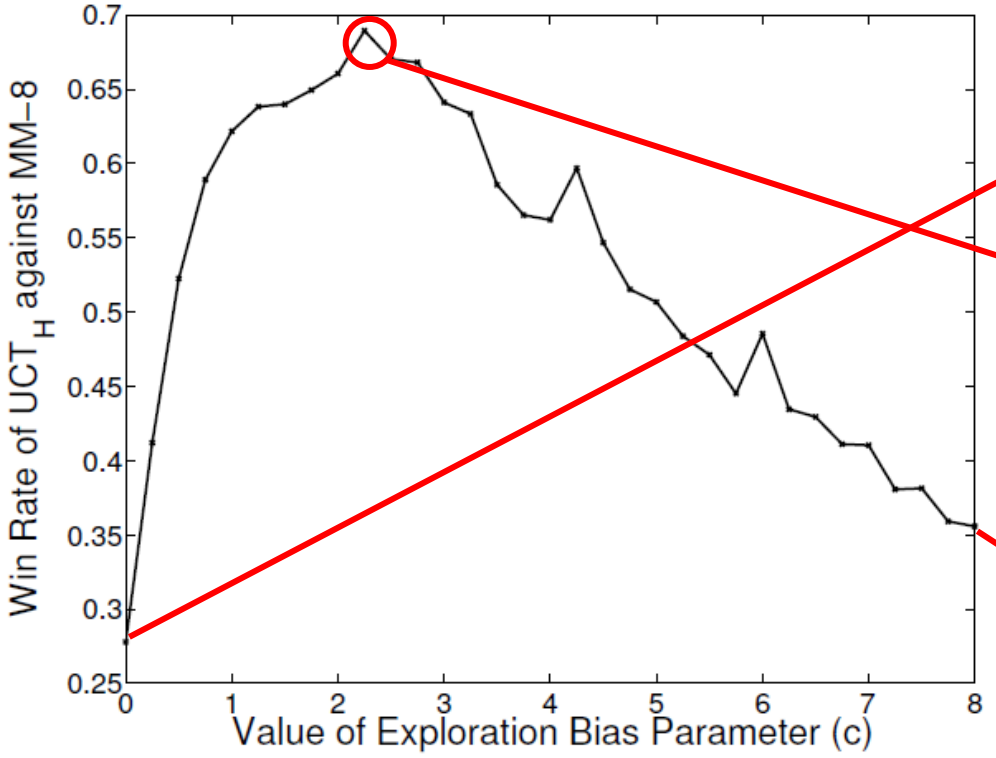
Main Finding: *Different parts of the search space may favor different search strategies*

Complete versus Selective Search



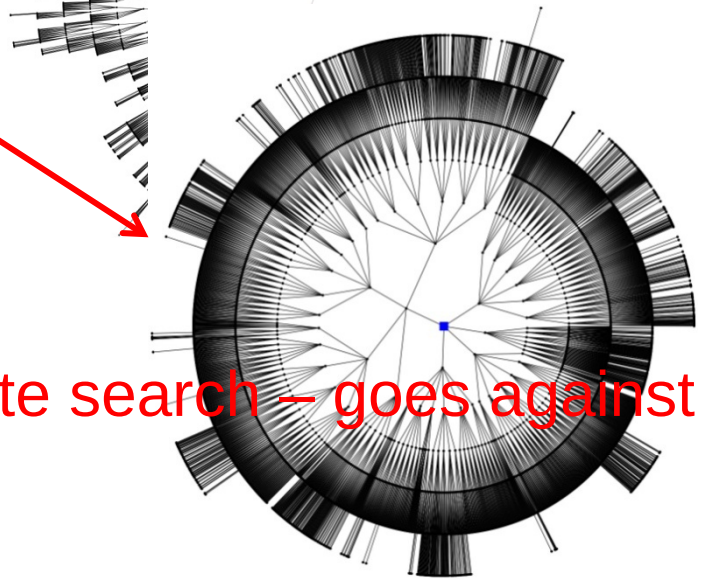
$$Q(k) + c\sqrt{\frac{\log t}{n(k)}}$$

Complete versus Selective Search



Optimal setting for *c*

Vary *c* and measure the win-rate of UCT against a standardized Minimax player

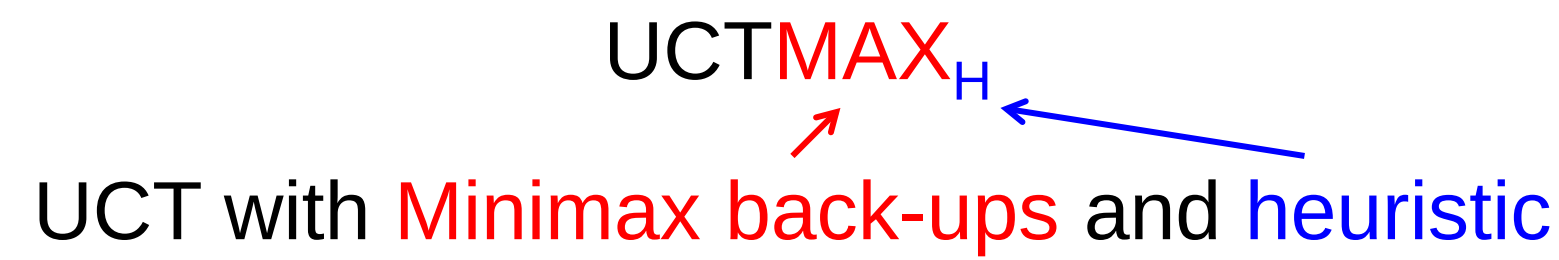


Selective search *can* outperform complete search – goes against experience in Chess

Other Trade-offs in UCT

- Sampling versus Tree Size
  - Larger trees + fewer playouts per node > smaller trees + more playouts per node
  - Playouts yield some information, but quality of feedback is limited
- Averaging versus Minimaxing back-ups
  - We have a good heuristic for this domain – use it instead!
  - Minimaxing back-up is better when using a quality node evaluation function

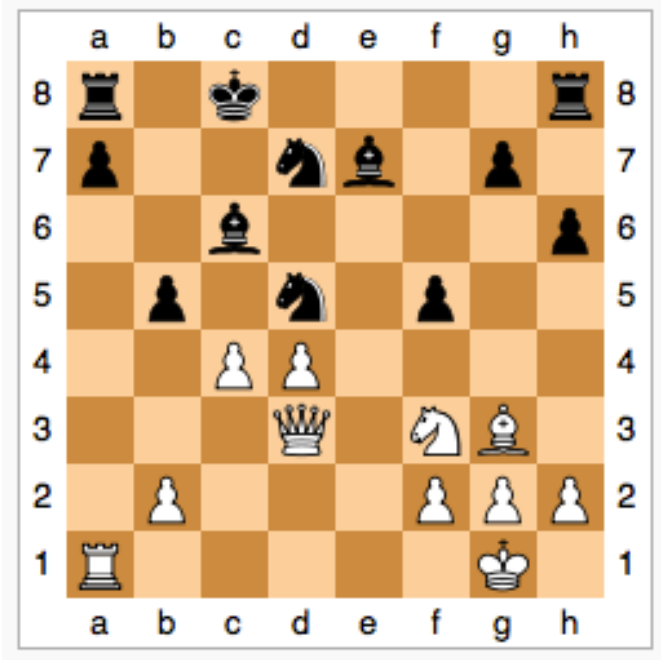
UCTMAX$_H$ versus Minimax



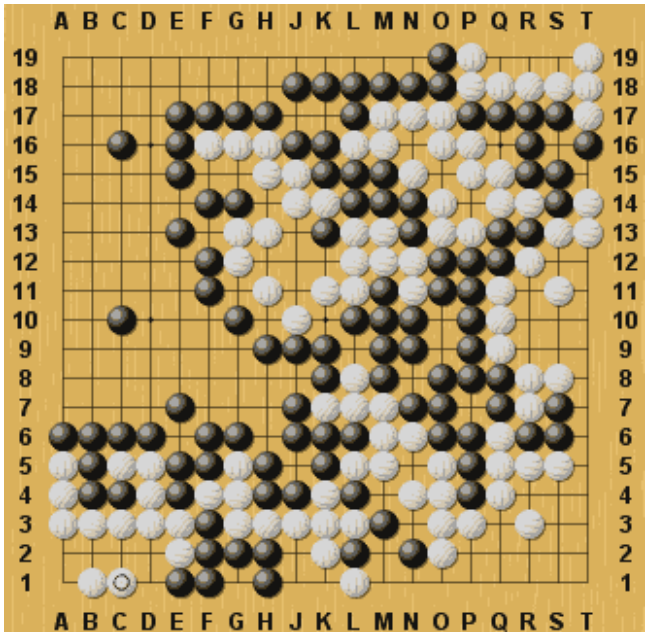
- UCTMAX$_H$ defeats Minimax searches up to depth 12 in Mancala
  - Minimax uses the same heuristic
  - UCTMAX$_H$ is only allowed to expand as many nodes as Minimax
- Can we gain some insight into where UCT derives it's advantage?

Background: Trap States

Trap state: Position where a mistake can result in a quick loss



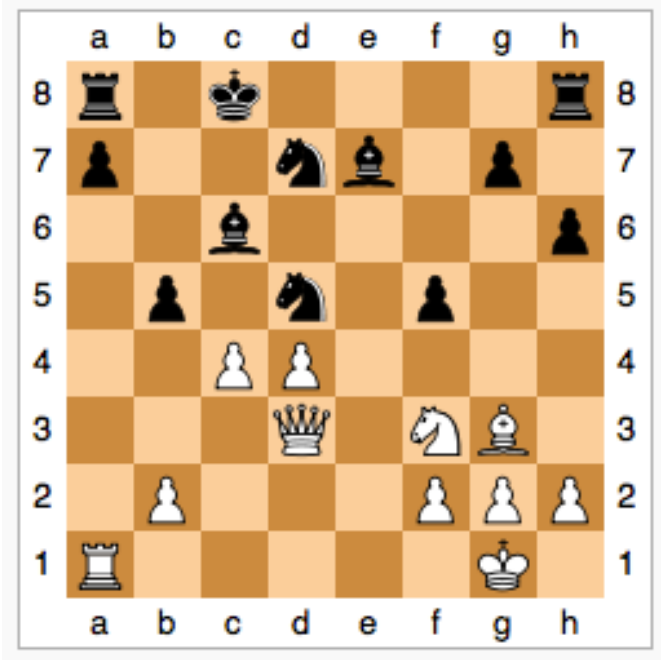
They are sprinkled throughout the Chess search space

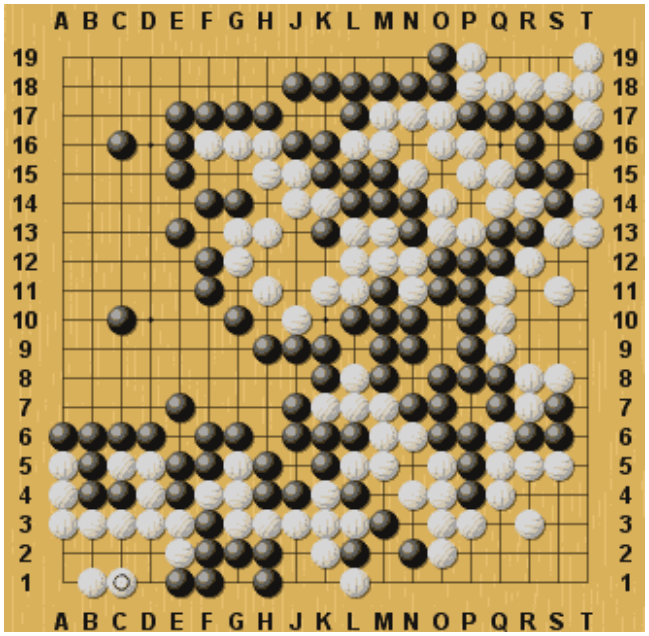


Appear only near the end-game in Go

Background: Trap States

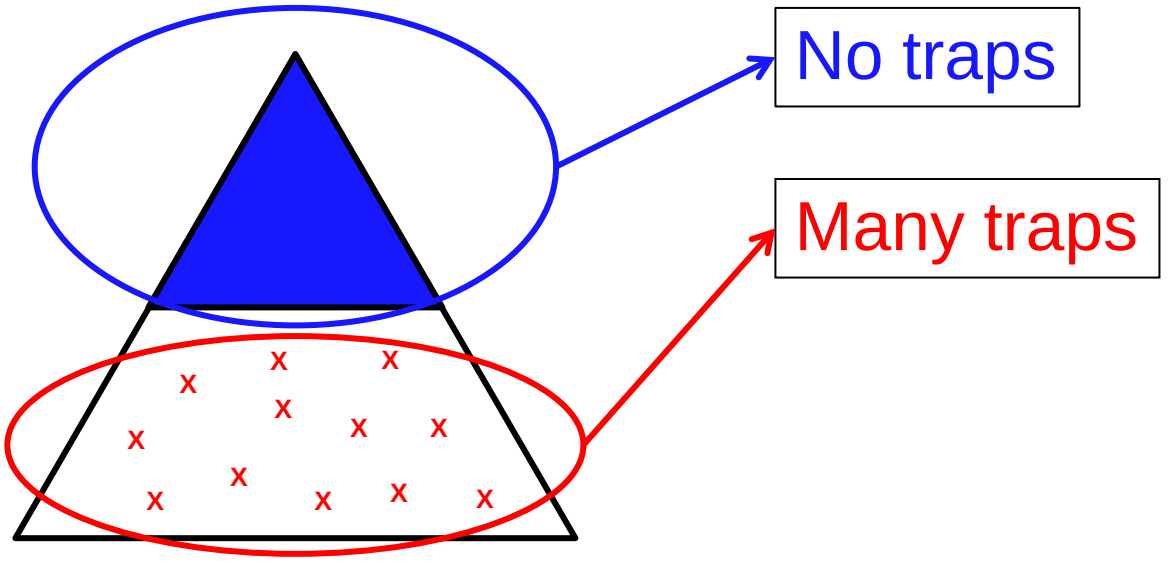
Trap state: Position where a mistake can result in a quick loss





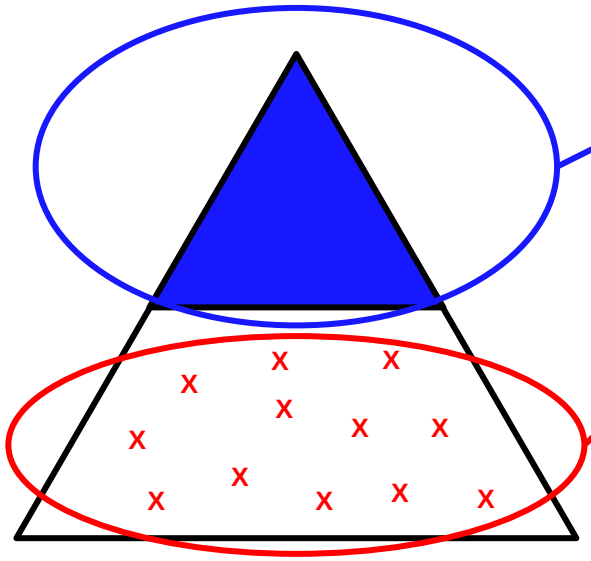
Traps were shown to play a role in the poor performance of UCT in Chess

Traps in Mancala



Does UCT perform better in the absence of traps?

‘Partial’ Games of Mancala

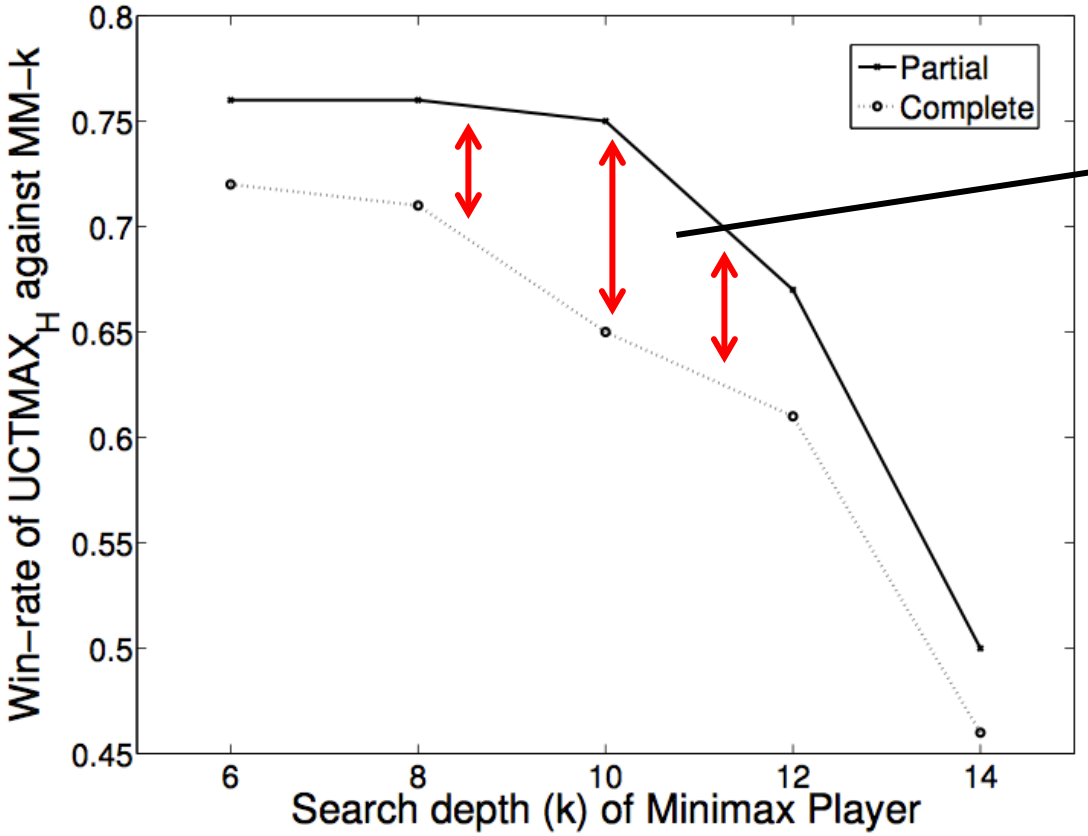


Play UCTMAX$_H$ against Minimax

Complete the game with two identical players (MM-16)

Does UCT perform better in the absence of traps?

$UCTMAX_H$ versus Minimax



UCT is better when it is making decisions only in parts of the space without traps

Different parts of the search space may favor different search methods

Conclusions

- The exploration-exploitation balancing performed by UCT *is* valuable
- Minimax back-ups in UCT can help when we have a good heuristic
- Hybrid search strategies, combining UCT and Minimax, can outperform either on it's own