Monte Carlo searching

Matthew Bedder

Background

Previously

Conventional tree searching techniques using heuristics
(e.g. depth-bounded Minimax)

Problem

Some games don’t have (useful) heuristics

Idea

Monte Carlo Evaluations

Monte Carlo Evaluations

Monte Carlo Evaluations (MCEs)

Idea

- We (generally) know the rewards at terminal states
- If we perform random actions until some terminal state we will get some rough estimation of the expected reward
- If we repeat this a bunch of times this estimation will improve

Monte Carlo Evaluation (MCEs) example

You are dropped off at defined location in an unknown city. You want to know if there are many coffee shops in the city.

Plan

- Wander in a random direction for five minutes, counting the coffee shops you see
- Return to the start point, repeat the process

You are then moved to a different location in another city. Can you estimate if this city has a greater or lesser number of coffee shops?

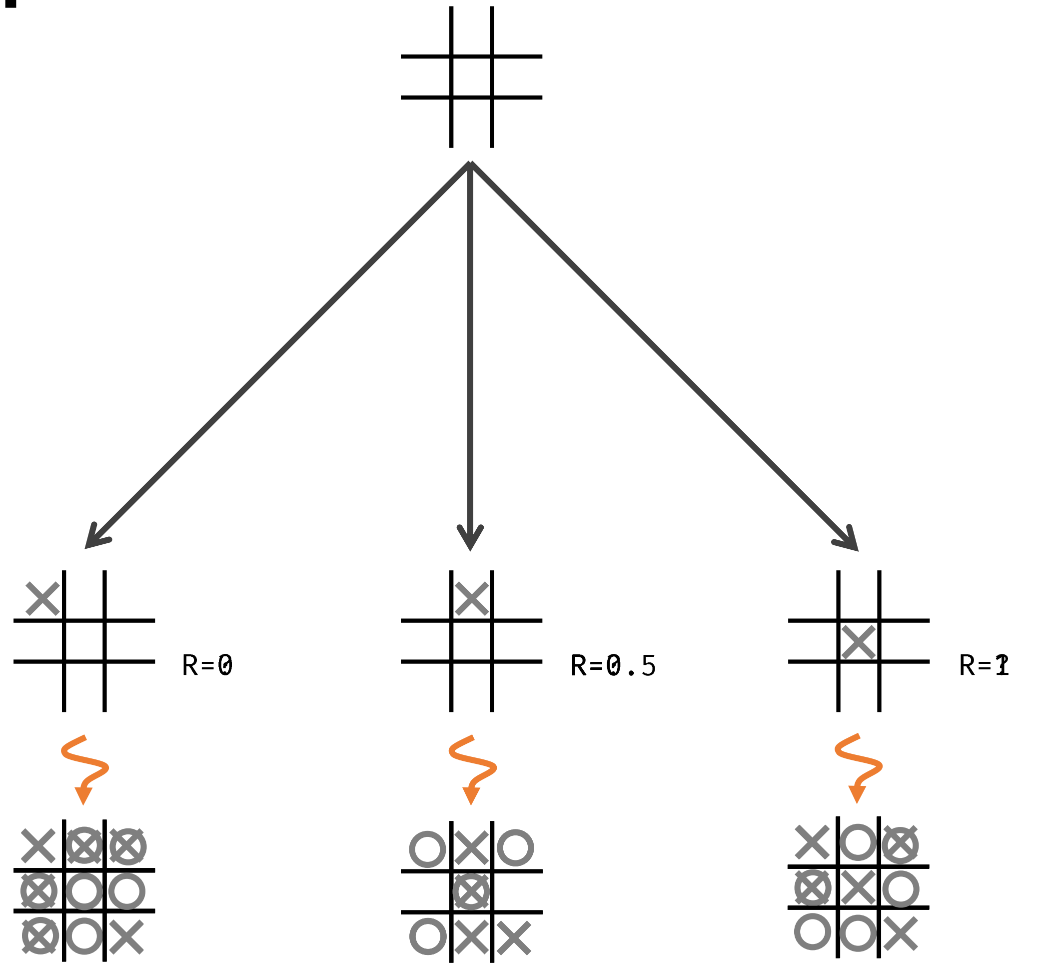
Monte Carlo Search

Flat Monte Carlo Search

We can use Monte Carlo Evaluations in a simple way to select which action to perform in a particular game state

Flat Monte Carlo Search

```
LOOP
   select next action a
   s' = apply(s, a)
   r = MCE(s')
   update estimate reward Ra
UNTIL stopping criteria
perform action a that maximises Ra
```



Flat Monte Carlo Search (MCS)

We can use Monte Carlo Evaluations in a simple way to select which action to perform in a particular game state

This approach is <u>aheuristic</u> and <u>anytime</u>

An even amount of time is dedicated to each action, so we might waste time on actions we know are bad

ASIDE: Multi Armed Bandit

You're sat in front of **n** slot machines with different reward structures. How do you play the game to maximise your long-term reward?

Need to balance exploration and exploitation

- Explore to try to work out the rewards of each machine
- Exploit the machine that you think gives the highest reward

Upper Confidence Bound (UCB1) selects arm j as follows

$$\arg\max_j \left(\bar{x}_j + \sqrt{\frac{2 \ln n}{n_j}} \right)$$

Monte Carlo Search with UCB (MCS+UCB)

We can treat selecting actions in Monte Carlo Search as an instance of the MAB problem

Selecting which child state to perform an MCE on using UCB can increase performance somewhat

This still only considers one move ahead, so results still aren't ideal

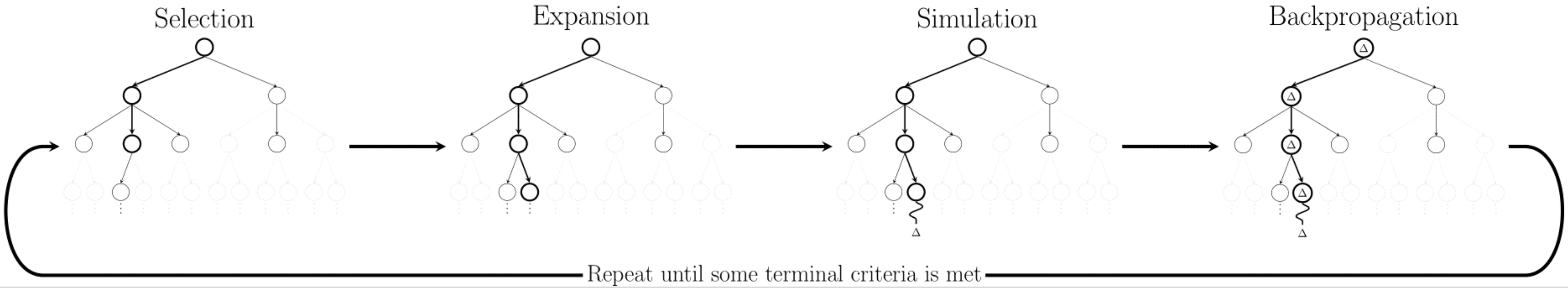
Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS)

Similar to MCS, but we iteratively build up the game tree

Consists of repeatedly applying four actions

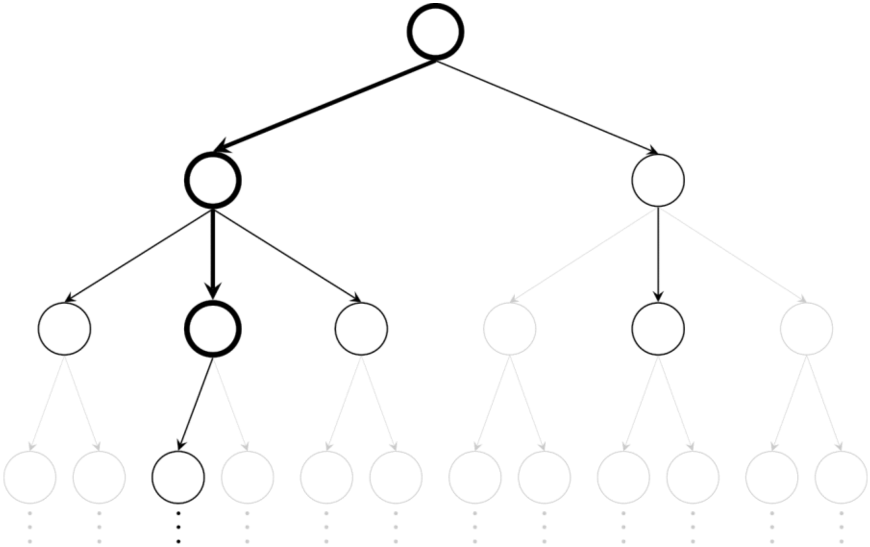
- Select part of the tree we're interested in
- Expand the tree by adding a new node
- Simulate the game using a Monte Carlo Evaluation
- Backpropagate the result of the MCE to update node statistics



MCTS – Selection

Starting at the root node, apply some selection criteria until we reach either a **terminal node** or one that is **not fully expanded**

Upper Confidence Bounds for Trees (UCT) is most often used



MCTS – Selection

Usually uses Upper Confidence Bounds for Trees (UCT)

- A simple change on UCT that uses a constant C to trade-off exploration and exploitation

$$\arg\max_j \left(\bar{x}_j + 2C\sqrt{\frac{2\ln n}{n_j}} \right)$$

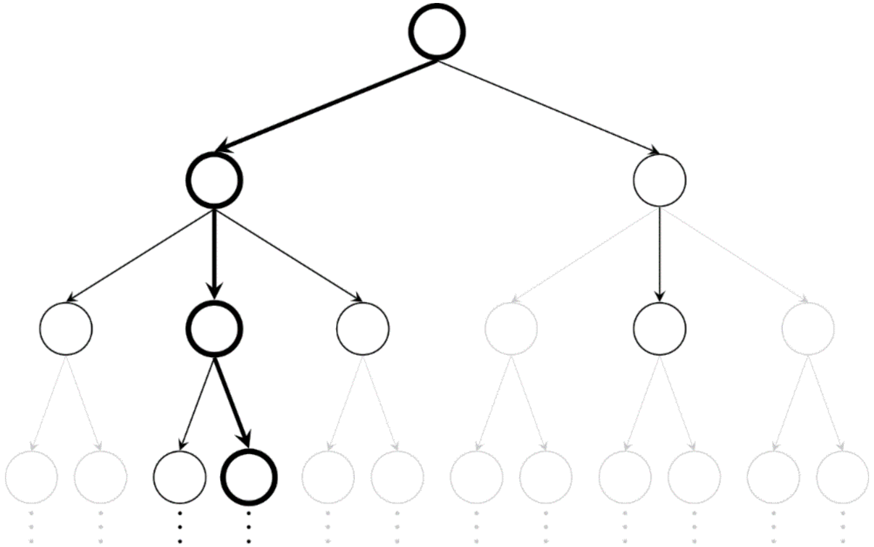
- Good values for C vary based on the game and reward structure

MCTS breaks the MAB assumption, but results for UCT are still good!

MCTS – Expansion

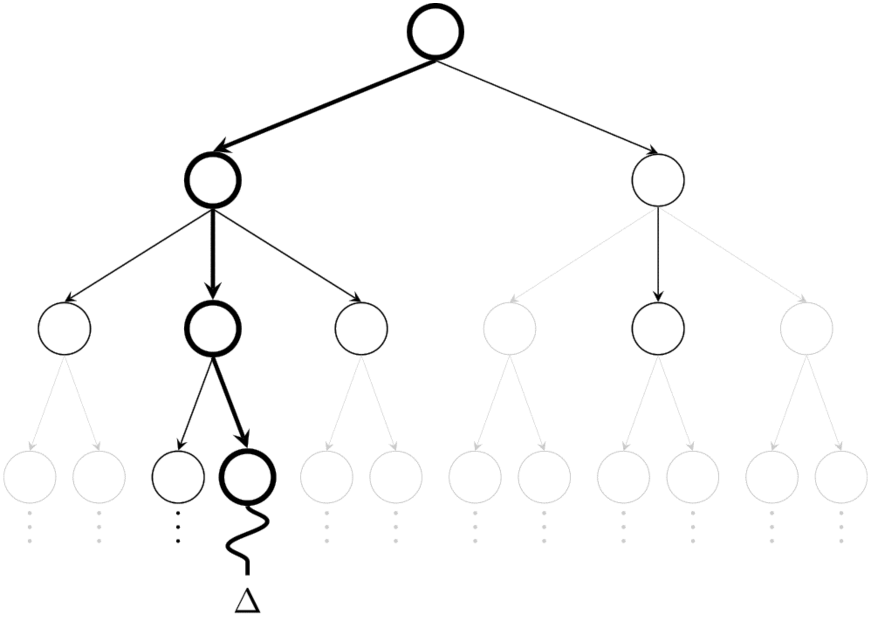
Pretty simple – add a new node onto the tree

Some approaches add multiple nodes, but this generally is worse



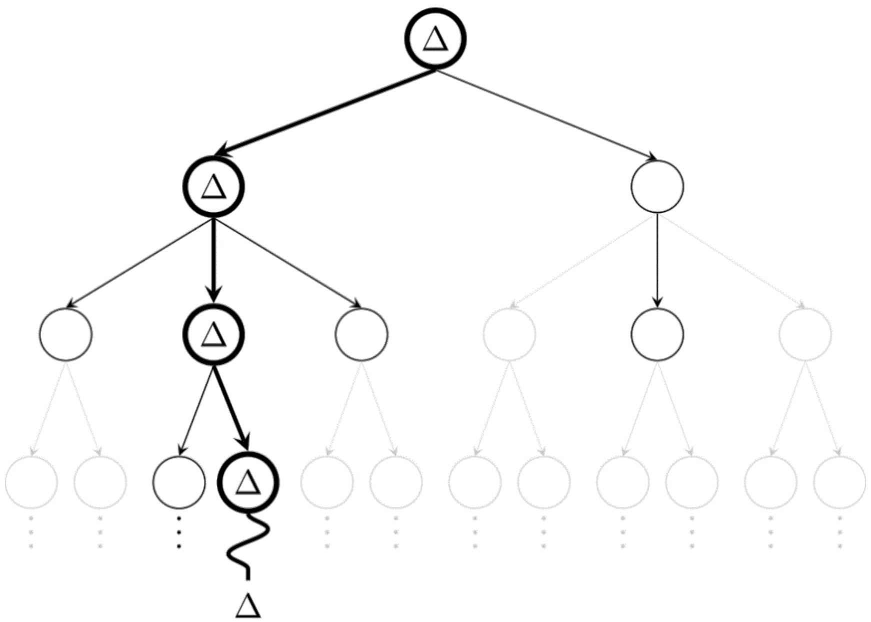
MCTS – Simulation

Also pretty simple – perform a Monte Carlo Evaluation



MCTS – Backpropagation

We update the new node and its ancestors with the results of the MCE



MCTS – Tree structure

Usually MCTS tree nodes need to store:

- The (per-player) mean reward of rollouts passing through it
- The number of rollouts passing through it
- The player who should make a move in that game state

Different extensions to MCTS sometimes store different values

Chance events are sometimes modelled as nodes

MCTS – Results

Builds a (potentially highly) asymmetric game tree

Still <u>aheuristic</u> and <u>anytime</u>

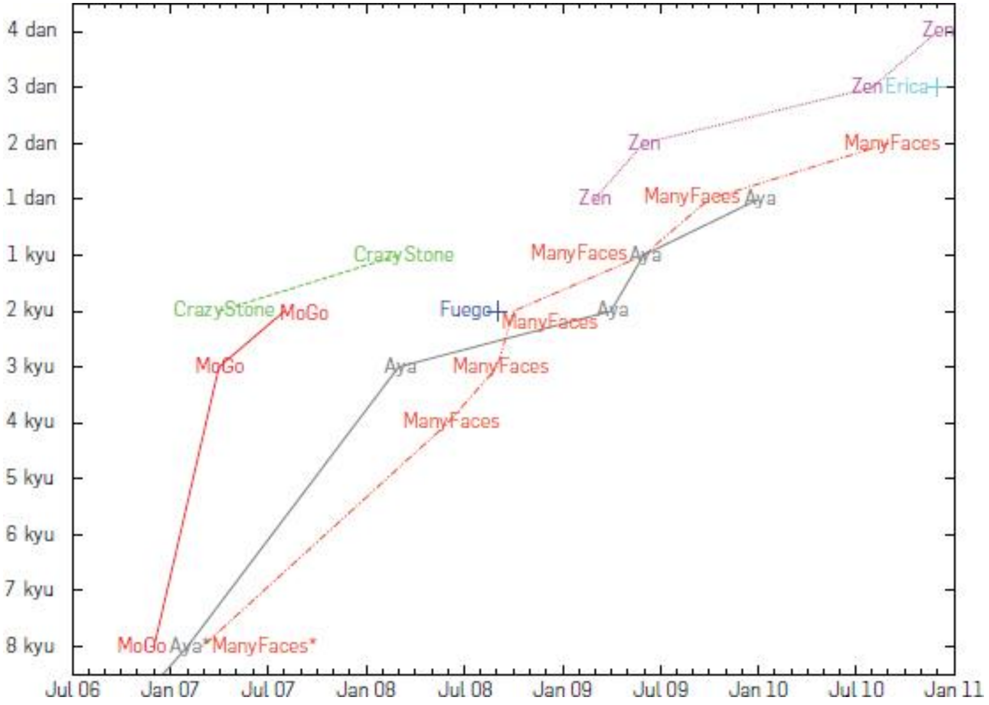
Question

- How should we chose actions from the generated tree?
- Try picking the node with the best expected return, or the most visits

MCTS – Performance

Very good over Go

commons.wikimedia.org



cacm.acm.org

MCTS – Performance

Good over some unconventional games

Best known agents for Physical Travelling Salesman Problem

- Video with move visualisation
- Video in real-time

A (highly optimised) MCTS agent won the 2014 GVGAI Competition

- Example games
- (A really simple MCTS implementation came 3rd of 18 submissions)

MCTS – Performance

Performs poorly over games like Chess

Why?

- Full-depth naïve MCEs on Chess can be very long
- Chess contains many <u>trap states</u> that are easily identified by heuristics but would require many simulations to identify

MCTS – Extensions

- Pruning the game tree
- Seeding new nodes with prior knowledge
- Transposition tables or Q-value structures
- Depth-limited Monte Carlo Evaluations using heuristics
- Replacing MCEs with simple policies
- Methods for dealing with partial information
  - Information Set MCTS (ISMCTS) in particular!

"A Survey of Monte Carlo Tree Search Methods" – Browne *et al.*

Conclusions

Monte Carlo techniques are potentially very interesting and powerful approaches

…especially for domains without useful heuristics

Monte Carlo techniques aren't the be-all and end-all

No free lunch!

MCTS is a very active area of research right now, so there's a lot of new work appearing all the time

Good resources

A website with some good descriptions, simple implementations

http://mcts.ai

A set of (Windows) MCTS demos you can play around with

http://bit.ly/mctsdemo

"A Survey of Monte Carlo Tree Search Methods" – Browne *et al.*

http://ccg.doc.gold.ac.uk/papers/browne_tciaig12_1.pdf