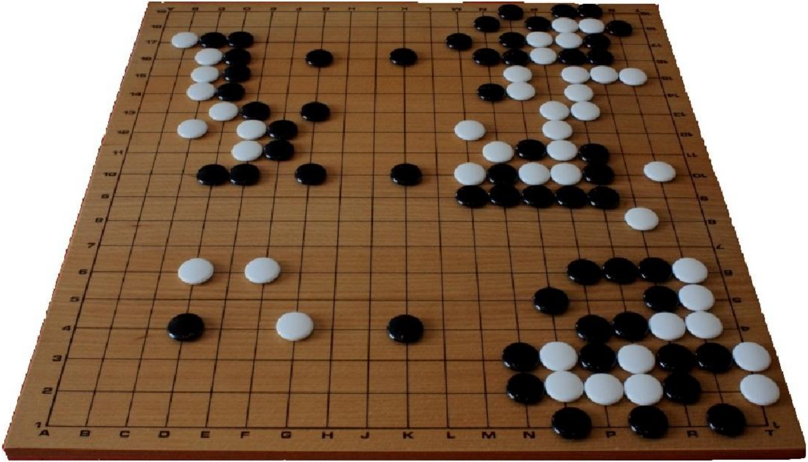
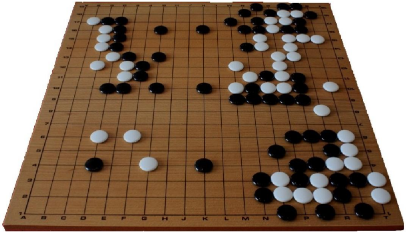
This talk is about sequential decision making

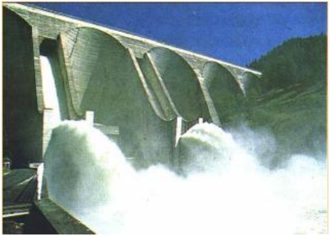
- Reinforcement learning:
  First learn the optimal policy; then apply it
- Monte-Carlo Tree Search:
  Any-time algorithm: learn the next move; play it; iterate.

MCTS: computer-Go as explanatory example



Not just a game: same approaches apply to optimal energy policy

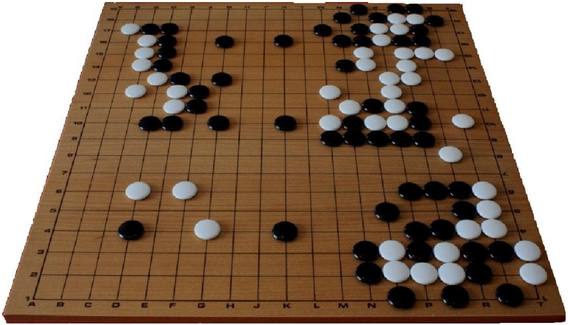


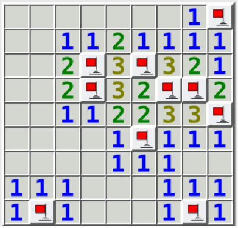


MCTS for computer-Go and MineSweeper

Go: deterministic transitions

MineSweeper: probabilistic transitions

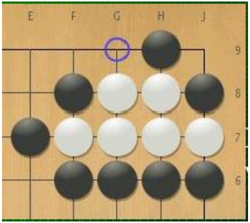




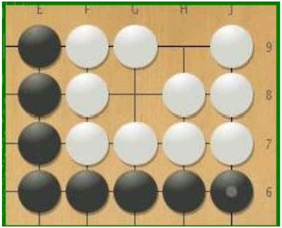
The game of Go in one slide

Rules

- Each player puts a stone on the goban, black first
- Each stone remains on the goban, except:



group w/o degree freedom is killed



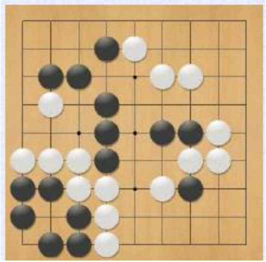
a group with two eyes can't be killed

- The goal is to control the max. territory

Go as a sequential decision problem

Features

- Size of the state space 2.10^{170}
- Size of the action space 200
- No good evaluation function
- Local and global features (symmetries, freedom, ...)
- A move might make a difference some dozen plies later



Setting

- State space $\mathcal{S}$
- Action space $\mathcal{A}$
- Known transition model: $p(s, a, s')$
- Reward on final states: win or lose

Baseline strategies do not apply:

- Cannot grow the full tree
- Cannot safely cut branches
- Cannot be greedy

Monte-Carlo Tree Search

- An any-time algorithm
- Iteratively and asymmetrically growing a search tree

most promising subtrees are more explored and developed

Overview

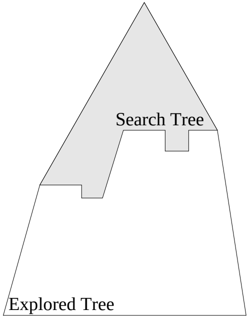
Overview

Monte-Carlo Tree Search

Kocsis Szepesvári, 06

Gradually grow the search tree:

- Iterate Tree-Walk
  - Building Blocks
    - Select next action
      Bandit phase
    - Add a node
      Grow a leaf of the search tree
    - Select next action bis
      Random phase, roll-out
    - Compute instant reward
      Evaluate
    - Update information in visited nodes
      Propagate
- Returned solution:
  - Path visited most often

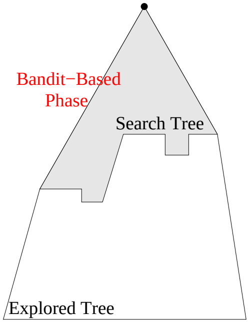


Monte-Carlo Tree Search

Kocsis Szepesvári, 06

Gradually grow the search tree:

- Iterate Tree-Walk
  - Building Blocks
    - Select next action
      Bandit phase
    - Add a node
      Grow a leaf of the search tree
    - Select next action bis
      Random phase, roll-out
    - Compute instant reward
      Evaluate
    - Update information in visited nodes
      Propagate
- Returned solution:
  - Path visited most often



Monte-Carlo Tree Search

Kocsis Szepesvári, 06

Gradually grow the search tree:

- Iterate Tree-Walk
  - Building Blocks
    - Select next action

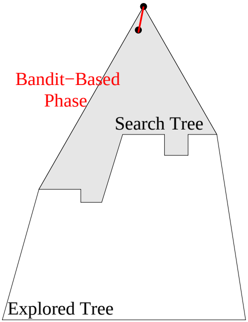
      Bandit phase
    - Add a node

      Grow a leaf of the search tree
    - Select next action bis

      Random phase, roll-out
    - Compute instant reward

      Evaluate
    - Update information in visited nodes

      Propagate
- Returned solution:
  - Path visited most often



Monte-Carlo Tree Search

Kocsis Szepesvári, 06

Gradually grow the search tree:

- Iterate Tree-Walk
  - Building Blocks
    - Select next action

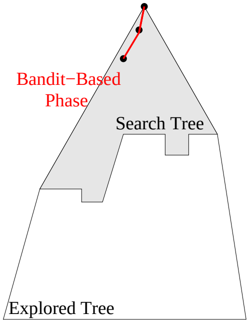
      Bandit phase
    - Add a node

      Grow a leaf of the search tree
    - Select next action bis

      Random phase, roll-out
    - Compute instant reward

      Evaluate
    - Update information in visited nodes

      Propagate
- Returned solution:
  - Path visited most often

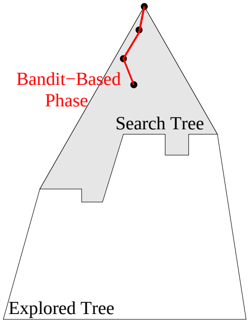


Monte-Carlo Tree Search

Kocsis Szepesvári, 06

Gradually grow the search tree:

- Iterate Tree-Walk
  - Building Blocks
    - Select next action
      Bandit phase
    - Add a node
      Grow a leaf of the search tree
    - Select next action bis
      Random phase, roll-out
    - Compute instant reward
      Evaluate
    - Update information in visited nodes
      Propagate
- Returned solution:
  - Path visited most often



Monte-Carlo Tree Search

Kocsis Szepesvári, 06

Gradually grow the search tree:

- Iterate Tree-Walk
  - Building Blocks
    - Select next action

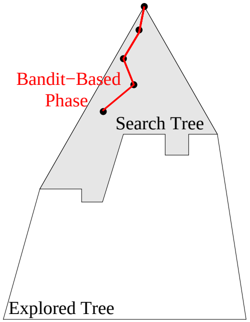
      Bandit phase
    - Add a node

      Grow a leaf of the search tree
    - Select next action bis

      Random phase, roll-out
    - Compute instant reward

      Evaluate
    - Update information in visited nodes

      Propagate
- Returned solution:
  - Path visited most often

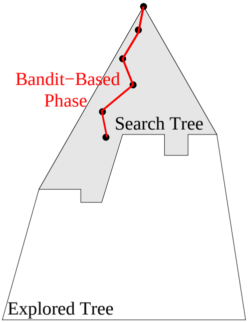


Monte-Carlo Tree Search

Kocsis Szepesvári, 06

Gradually grow the search tree:

- Iterate Tree-Walk
  - Building Blocks
    - Select next action
      Bandit phase
    - Add a node
      Grow a leaf of the search tree
    - Select next action bis
      Random phase, roll-out
    - Compute instant reward
      Evaluate
    - Update information in visited nodes
      Propagate
- Returned solution:
  - Path visited most often

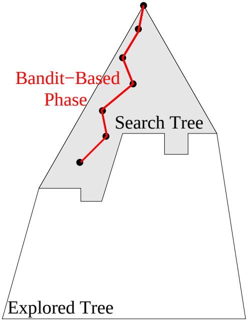


Monte-Carlo Tree Search

Kocsis Szepesvári, 06

Gradually grow the search tree:

- Iterate Tree-Walk
  - Building Blocks
    - Select next action
      Bandit phase
    - Add a node
      Grow a leaf of the search tree
    - Select next action bis
      Random phase, roll-out
    - Compute instant reward
      Evaluate
    - Update information in visited nodes
      Propagate
- Returned solution:
  - Path visited most often



Monte-Carlo Tree Search

Kocsis Szepesvári, 06

Gradually grow the search tree:

- Iterate Tree-Walk
  - Building Blocks
    - Select next action

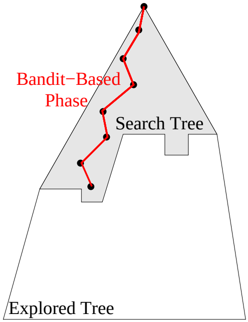
      Bandit phase
    - Add a node

      Grow a leaf of the search tree
    - Select next action bis

      Random phase, roll-out
    - Compute instant reward

      Evaluate
    - Update information in visited nodes

      Propagate
- Returned solution:
  - Path visited most often



Monte-Carlo Tree Search

Kocsis Szepesvári, 06

Gradually grow the search tree:

- Iterate Tree-Walk
  - Building Blocks
    - Select next action

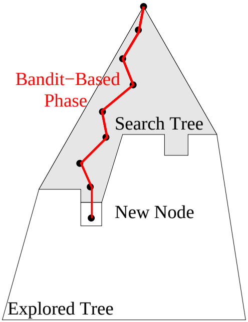
      Bandit phase
    - Add a node

      Grow a leaf of the search tree
    - Select next action bis

      Random phase, roll-out
    - Compute instant reward

      Evaluate
    - Update information in visited nodes

      Propagate
- Returned solution:
  - Path visited most often



Monte-Carlo Tree Search

Kocsis Szepesvári, 06

Gradually grow the search tree:

- Iterate Tree-Walk
  - Building Blocks
    - Select next action

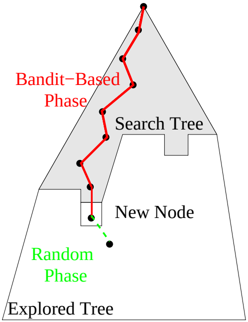
      Bandit phase
    - Add a node

      Grow a leaf of the search tree
    - Select next action bis

      Random phase, roll-out
    - Compute instant reward

      Evaluate
    - Update information in visited nodes

      Propagate
- Returned solution:
  - Path visited most often

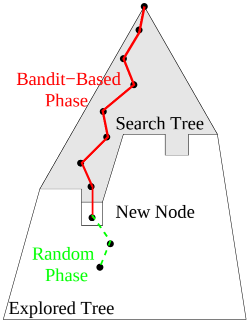


Monte-Carlo Tree Search

Kocsis Szepesvári, 06

Gradually grow the search tree:

- Iterate Tree-Walk
  - Building Blocks
    - Select next action
      Bandit phase
    - Add a node
      Grow a leaf of the search tree
    - Select next action bis
      Random phase, roll-out
    - Compute instant reward
      Evaluate
    - Update information in visited nodes
      Propagate
- Returned solution:
  - Path visited most often

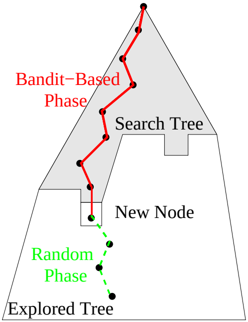


Monte-Carlo Tree Search

Kocsis Szepesvári, 06

Gradually grow the search tree:

- Iterate Tree-Walk
  - Building Blocks
    - Select next action
      Bandit phase
    - Add a node
      Grow a leaf of the search tree
    - Select next action bis
      Random phase, roll-out
    - Compute instant reward
      Evaluate
    - Update information in visited nodes
      Propagate
- Returned solution:
  - Path visited most often



Monte-Carlo Tree Search

Kocsis Szepesvári, 06

Gradually grow the search tree:

- Iterate Tree-Walk
  - Building Blocks
    - Select next action

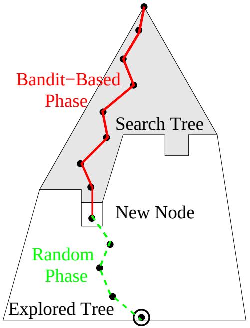
      Bandit phase
    - Add a node

      Grow a leaf of the search tree
    - Select next action bis

      Random phase, roll-out
    - Compute instant reward

      Evaluate
    - Update information in visited nodes

      Propagate
- Returned solution:
  - Path visited most often



MCTS Algorithm

Main
Input: number N of tree-walks
Initialize search tree $\mathcal{T} \leftarrow$ initial state
Loop: For $i = 1$ **to** N
  TreeWalk($\mathcal{T}$, initial state)
EndLoop
Return most visited child node of root node

MCTS Algorithm, ctd

Tree walk

Input: search tree $\mathcal{T}$, state s

Output: reward r

If s is not a leaf node

Select $a^* = \text{argmax } \{\hat{\mu}(s, a), tr(s, a) \in \mathcal{T}\}$

$r \leftarrow$ TreeWalk$(\mathcal{T}, tr(s, a^*))$

Else

$\mathcal{A}_s = \{$ admissible actions not yet visited in $s\}$

Select a^* in $\mathcal{A}_s$

Add $tr(s, a^*)$ as child node of s

$r \leftarrow$ RandomWalk$(tr(s, a^*))$

End If

Update n_s, n_{s,a^*} and $\hat{\mu}_{s,a^*}$

Return r

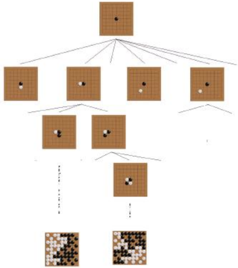
MCTS Algorithm, ctd

Random walk
Input: search tree $\mathcal{T}$, state u
Output: reward r

$\mathcal{A}_{rnd} \leftarrow \{\}$ // store the set of actions visited in the random phase
While u is not final state
  Uniformly select an admissible action a for u
  $\mathcal{A}_{rnd} \leftarrow \mathcal{A}_{rnd} \cup \{a\}$
  $u \leftarrow \text{tr}(u, a)$
EndWhile

$r = \textit{Evaluate}(u)$ //reward vector of the tree-walk
Return r

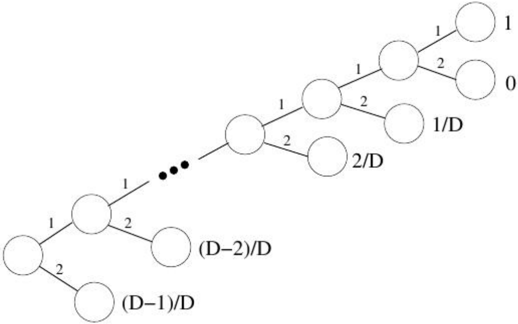
Monte-Carlo Tree Search



Properties of interest

- Consistency: Pr(finding optimal path) $\rightarrow$ 1 when the number of tree-walks go to infinity
- Speed of convergence; can be exponentially slow.

Coquelin Munos 07



Comparative results

2012	MoGoTW used for physiological measurements of human players	
2012	7 wins out of 12 games against professional players and 9 wins out of 12 games against 6D players	MoGoTW
2011	20 wins out of 20 games in 7x7 with minimal computer komi	MoGoTW
2011	First win against a pro (6D), H2, 13×13	MoGoTW
2011	First win against a pro (9P), H2.5, 13×13	MoGoTW
2011	First win against a pro in Blind Go, 9×9	MoGoTW
2010	Gold medal in TAAI, all categories 19×19, 13×13, 9×9	MoGoTW
2009	Win against a pro (5P), 9× 9 (black)	MoGo
2009	Win against a pro (5P), 9× 9 (black)	MoGoTW
2008	in against a pro (5P), 9× 9 (white)	MoGo
2007	Win against a pro (5P), 9× 9 (blitz)	MoGo
2009	Win against a pro (8P), 19× 19 H9	MoGo
2009	Win against a pro (1P), 19× 19 H6	MoGo
2008	Win against a pro (9P), 19× 19 H7	MoGo


Overview

Action selection as a Multi-Armed Bandit problem

Lai, Robbins 85

In a casino, one wants to maximize one's gains *while playing*.

Lifelong learning

Exploration vs Exploitation Dilemma

- Play the best arm so far ? Exploitation
- But there might exist better arms... Exploration

The multi-armed bandit (MAB) problem

- K arms
- Each arm gives reward 1 with probability μ_i, 0 otherwise
- Let $\mu^* = argmax\{\mu_1, \ldots \mu_K\}$, with $\Delta_i = \mu^* - \mu_i$
- In each time t, one selects an arm i^*_t and gets a reward r_t

$$\begin{array}{lcll} n_{i,t} & = & \sum_{u=1}^{t} \mathbb{1}_{i^*_u = i} & \textit{number of times } i \textit{ has been selected} \\ \hat{\mu}_{i,t} & = & \frac{1}{n_{i,t}} \sum_{i^*_u = i} r_u & \textit{average reward of arm } i \end{array}$$

Goal: Maximize $\sum_{u=1}^{t} r_u$

$\Leftrightarrow$

$$\text{Minimize Regret } (t) = \sum_{u=1}^{t} (\mu^* - r_u) = t\mu^* - \sum_{i=1}^{K} n_{i,t}\, \hat{\mu}_{i,t} \approx \sum_{i=1}^{K} n_{i,t} \Delta_i$$

The simplest approach: ϵ-greedy selection

At each time t,

- With probability $1 - \varepsilon$
  select the arm with best empirical reward

$$i_t^* = argmax\{\hat{\mu}_{1,t}, \ldots \hat{\mu}_{K,t}\}$$

- Otherwise, select i_t^* uniformly in $\{1 \ldots K\}$

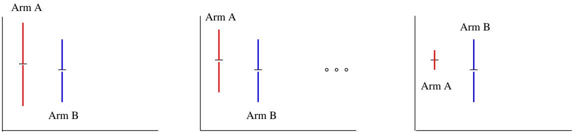
$$\text{Regret } (t) > \varepsilon t \frac{1}{K} \sum_i \Delta_i$$

Optimal regret rate: $log(t)$ Lai Robbins 85

Upper Confidence Bound

Auer et al. 2002

$$\text{Select } i_t^* = \text{argmax} \left\{ \hat{\mu}_{i,t} + \sqrt{C \frac{\log(\sum n_{j,t})}{n_{i,t}}} \right\}$$



Decision: Optimism in front of unknown !

Upper Confidence bound, followed

UCB achieves the optimal regret rate $log(t)$

$$\text{Select } i_t^* = \text{ argmax } \left\{ \hat{\mu}_{i,t} + \sqrt{c_e \frac{log(\sum n_{j,t})}{n_{i,t}}} \right\}$$

Extensions and variants

- Tune c_e control the exploration/exploitation trade-off
- UCB-tuned: take into account the standard deviation of $\hat{\mu}_i$:
  Select $i_t^* = \text{argmax}$

$$\left\{ \hat{\mu}_{i,t} + \sqrt{c_e \frac{log(\sum n_{j,t})}{n_{i,t}} + min\left(\frac{1}{4}, \hat{\sigma}^2_{i,t} + \sqrt{c_e \frac{log(\sum n_{j,t})}{n_{i,t}}}\right)} \right\}$$

- Many-armed bandit strategies
- Extension of UCB to trees: UCT Kocsis & Szepesvári, 06

Monte-Carlo Tree Search. Random phase

Gradually grow the search tree:

- Iterate Tree-Walk
  - Building Blocks
    - Select next action

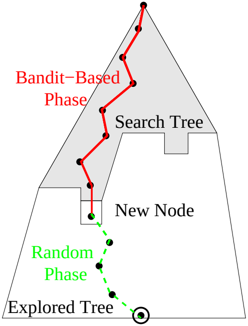
      Bandit phase
    - Add a node

      Grow a leaf of the search tree
    - **Select next action bis**

      Random phase, roll-out
    - Compute instant reward

      Evaluate
    - Update information in visited nodes

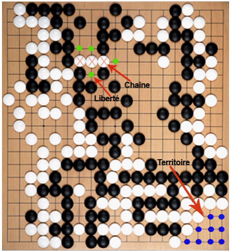
      Propagate
- Returned solution:
  - Path visited most often



Random phase — Roll-out policy

Monte-Carlo-based   Brügman 93

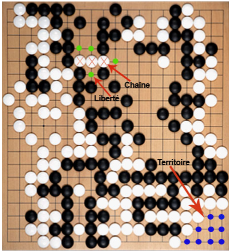
1. Until the goban is filled,
   add a stone (black or white in turn)
   at a uniformly selected empty position
2. Compute $r = \text{Win(black)}$
3. The outcome of the tree-walk is r



Random phase — Roll-out policy

Monte-Carlo-based Brügman 93

1. Until the goban is filled,
   add a stone (black or white in turn)
   at a uniformly selected empty position
2. Compute $r = \text{Win(black)}$
3. The outcome of the tree-walk is r



Improvements ?

- Put stones randomly in the neighborhood of a previous stone
- Put stones matching patterns prior knowledge
- Put stones optimizing a value function Silver et al. 07

Evaluation and Propagation

The tree-walk returns an evaluation r win(black)

Propagate

- For each node (s, a) in the tree-walk

$$
\begin{aligned}
n_{s,a} &\leftarrow n_{s,a} + 1 \\
\hat{\mu}_{s,a} &\leftarrow \hat{\mu}_{s,a} + \frac{1}{n_{s,a}}(r - \mu_{s,a})
\end{aligned}
$$

Evaluation and Propagation

The tree-walk returns an evaluation r win(black)

Propagate

- For each node (s,a) in the tree-walk

$$
\begin{aligned}
n_{s,a} &\leftarrow n_{s,a} + 1 \\
\hat{\mu}_{s,a} &\leftarrow \hat{\mu}_{s,a} + \frac{1}{n_{s,a}}(r - \mu_{s,a})
\end{aligned}
$$

Variants Kocsis & Szepesvári, 06

$$
\hat{\mu}_{s,a} \leftarrow \left\{ \begin{array}{ll} min\{\hat{\mu}_x, x \text{ child of } (s,a)\} & \text{if } (s,a) \text{ is a black node} \\ max\{\hat{\mu}_x, x \text{ child of } (s,a)\} & \text{if } (s,a) \text{ is a white node} \end{array} \right.
$$

Dilemma

- smarter roll-out policy $\rightarrow$
  more computationally expensive $\rightarrow$
  less tree-walks on a budget
- frugal roll-out $\rightarrow$
  more tree-walks $\rightarrow$
  more confident evaluations