Master Recherche IAC
Option 2
Robotique et agents autonomes

Michèle Sebag – Emanuel Aldea – Odalric-Ambrym Maillard
LRI

Jan. 23rd, 2015
I. Position of the problem

▶ Environment, Physical robot, Decision making
▶ AI: focus on decision making
▶ Brooks: Situated cognition, modular decision making
▶ Reactive behaviors

II. Vision and Localization

Emanuel Aldea

III. Action selection

Odalric Maillard
Decision making

Multi-armed bandits

- $K$ arms
- At $t$, select arm $a_t$ and get reward $r_t$
- Such that $\sum_t r_t$ maximum

Settings

- Standard: each arm with its distribution $\nu_i$
- Adversarial: opponent chooses sequence $r_{i,t}$ for arm $i$.  
- Contextual bandits
This course

- Sequential decision making
- Several states
- State, Action $\rightarrow$ Next state
- Reward is delayed...
Overview

Introduction

RL Algorithms

Values
Value functions
Optimal policy
Temporal differences and eligibility traces
Q-learning
Partial summary

Monte-Carlo Tree Search

Example: Computer-Go
Algorithm
Algorithm
Multi-Armed Bandits
Random phase
Evaluation and Propagation
Advanced MCTS
Rapid Action Value Estimate
Improving the rollout policy
Using prior knowledge

Open problems
Reinforcement Learning

Generalities

- An agent, spatially and temporally situated
- Stochastic and uncertain environment
- Goal: select an action in each time step,
- ... in order maximize expected cumulative reward over a time horizon

What is learned?

A policy = strategy = \{ state \mapsto action \}
Reinforcement Learning

Context
An unknown world.
Some actions, in some states, bear rewards with some delay [with some probability]

Goal: find policy (state $\rightarrow$ action)
maximizing the expected reward

Goal:

4 rooms
4 hallways
4 unreliable primitive actions

up
left  right Fail 33% of the time
down

8 multi-step options
(to each room’s 2 hallways)

Given goal location, quickly plan shortest route
Reinforcement Learning, example

World  You are in state 34.
       Your immediate reward is 3. You have 3 actions

Robot  I’ll take action 2

World  You are in state 77
       Your immediate reward is -7. You have 2 actions

Robot  I’ll take action 1

World  You are in state 34 (again)

   Markov Decision Property: actions/rewards only depend on the current state.
Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction to the animal will — others things being equal — be more firmly connected with the situation, so that when it recurs, they will more likely to recur; those which are accompanied or closely followed by discomfort to the animal will — others things being equal — have their connection with the situation weakened, so that when it recurs, they will less likely to recur; the greater the satisfaction or discomfort, the greater the strengthening or weakening of the link. Thorndike, 1911.
Formal background

Notations

- State space $S$
- Action space $A$
- Transition model $p(s, a, s') \mapsto [0, 1]$
- Reward $r(s)$

Goal

- Find policy $\pi : S \mapsto A$

$$\text{Maximize } E[\pi] = \text{Expected cumulative reward}$$

(detail later)
Applications

- Robotics
  Navigation, football, walk,
- Control
  Helicopter, elevators, telecom, smart grids, manufacturing, ...
- Operation research
  Transport, scheduling, ...
- Games
  Backgammon, Othello, Tetris, Go, ...
- Other
  Computer Human Interfaces, ML (Feature Selection, Active learning, Natural Language Processing,...)
Position of the problem

3 interleaved tasks

- Learn a world model \((p, r)\)
- Decide/select (the best) action
- Explore the world

Sources

- Sutton & Barto, Reinforcement Learning, MIT Press, 1998
- http://www.eecs.umich.edu/~baveja/NIPS05RLTutorial/
Particular case

If the transition model is known

Reinforcement learning → Optimal control
What’s hard

Curse of dimensionality

- State: features size, texture, color, ... ...
  \(|S|\) exponential wrt number of features
- Not all features are always relevant

Example:

<table>
<thead>
<tr>
<th>see</th>
<th>swann</th>
<th>white</th>
<th>—</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>swann</td>
<td>black</td>
<td>take a video</td>
</tr>
<tr>
<td></td>
<td>bear</td>
<td>—</td>
<td>flee</td>
</tr>
</tbody>
</table>
What’s hard

Curse of dimensionality

- State: features size, texture, color, ... ... 
  $|S|$ exponential wrt number of features
- Not all features are always relevant

Example:

<table>
<thead>
<tr>
<th>see</th>
<th>swann</th>
<th>white</th>
<th>—</th>
</tr>
</thead>
<tbody>
<tr>
<td>swann</td>
<td>black</td>
<td>take a video</td>
<td></td>
</tr>
<tr>
<td>bear</td>
<td>—</td>
<td>flee</td>
<td></td>
</tr>
</tbody>
</table>

Time horizon — Bounded rationality

- T.h. is infinite: eternity.
- Finite, unknown: reach the goal asap
- Finite: reach the goal in $T$ time steps
- Bounded rationality: find as fast as possible a decent policy (finding an approximation of the goal).
Overview
Introduction

RL Algorithms
Values
Value functions
Optimal policy
Temporal differences and eligibility traces
Q-learning
Partial summary

Monte-Carlo Tree Search
Example: Computer-Go
Algorithm
Algorithm
Multi-Armed Bandits
Random phase
Evaluation and Propagation
Advanced MCTS
Rapid Action Value Estimate
Improving the rollout policy
Using prior knowledge
Formalisation

Notations

- State space $S$
- Action space $A$
- Transition model
  - deterministic: $s' = t(s, a)$
  - probabilistic: $p(s, a, s') \in [0, 1]$.
- Reward $r(s)$ bounded
- Time horizon $H$ (finite or infinite)

Goal

- Find policy (strategy) $\pi : S \mapsto A$
- which maximizes (discounted) cumulative reward from now to timestep $H$
  \[ \sum_t r(s_t) \]
Formalisation

Notations

- State space $S$
- Action space $A$
- Transition model
  - deterministic: $s' = t(s, a)$
  - probabilistic: $p(s, a, s') \in [0, 1]$.
- Reward $r(s)$ bounded
- Time horizon $H$ (finite or infinite)

Goal

- Find policy (strategy) $\pi : S \mapsto A$
- which maximizes (discounted) cumulative reward from now to timestep $H$

$$\sum_{t=1}^{H} \gamma^t r(s_t) \quad \gamma < 1$$
Formalisation

Notations

- State space $S$
- Action space $A$
- Transition model
  - deterministic: $s' = t(s, a)$
  - probabilistic: $p(s, a, s') \in [0, 1]$.
- Reward $r(s)$ bounded
- Time horizon $H$ (finite or infinite)

Goal

- Find policy (strategy) $\pi : S \mapsto A$
- which maximizes (discounted) cumulative reward from now to timestep $H$

$$\mathbb{E}_{s_0, \pi} \left[ \sum_{t=1}^{\infty} \gamma^t r(s_t) \right]$$
Markov Decision Process

But can we define $P_{ss}', \text{ and } r(s)$?

- YES, if all necessary information is in $s$
- NO, otherwise
  - If state is partially observable
  - If environment (reward and transition distribution) is changing
    Reward for *first* photo of an object by the satellite

The Markov assumption

$$P(s_{h+1}|s_0 \ a_0 \ s_1 \ a_1 \ldots s_h \ a_h) = P(s_{h+1}|s_h \ a_h)$$

Everything you need to know is the current (state, action).
Find the treasure

Single reward: on the treasure.
Wandering robot

Nothing happens...
The robot finds it
Robot updates its value function

\[ V(s, a) = \text{“distance“ to the treasure on the trajectory.} \]
Reinforcement learning

* Robot most often selects $a = \arg \max V(s, a)$
* and sometimes explores (selects another action).
Reinforcement learning

* Robot most often selects \( a = \arg \max V(s, a) \)
* and sometimes explores (selects another action).
* Lucky exploration: finds the treasure again
Updates the value function

* Value function tells how far you are from the treasure given the known trajectories.
Finally

* Value function tells how far you are from the treasure
Finally

Let’s be greedy: selects the action maximizing the value function
Exercize

Uniform policy

- States: squares
- Actions: north, south, east, west.
- Rewards: -1 if you would get outside; 10 in A; 5 in B
- Transitions: as expected, except: A → A'; B → B'.

Compute the value function

A → A', reward 10
B → B', reward 5
Underlying: Dynamic programming

**Principle**
- Recursively decompose the problem into subproblems
- Solve and propagate

**An example**
\[ \ell(\text{shortest path } (A, B)) \leq \ell(\text{sp}(A, C)) + \ell(\text{sp}(C, B)) \]
Approaches

- Value function
  - Value iteration
  - Policy iteration

- Temporal differences

- Q-learning

- Direct policy search optimization in the $\pi$ space
  Stochastic optimization
Policy and value function 1/3

Finite horizon, deterministic transition

\[ V_\pi(s_0) = r(s_0) + \sum_{h=1}^{H} r(s_h) \]

where \( s_{h+1} = t(s_h, a_h = \pi(s_h)) \)
Policy and value function 1/3

Finite horizon, deterministic transition

\[ V_\pi(s_0) = r(s_0) + \sum_{h=1}^{H} r(s_h) \]

where \( s_{h+1} = t(s_h, a_h = \pi(s_h)) \)

Finite horizon, stochastic transition

\[ V_\pi(s_0) = r(s_0) + \sum_{h=1}^{H} p(s_{h-1}, a_{h-1} = \pi(s_{h-1}), s_h) r(s_h) \]

where \( s_{h+1} = s \) with proba \( p(s_h, a_h = \pi(s_h), s) \)
Policy and value function, 2/3

Finite horizon, stochastic transition

\[ V_\pi(s_0) = r(s_0) + \sum_{h=1}^{H} p(s_{h-1}, a_{h-1} = \pi(s_{h-1}), s_h) r(s_h) \]

where \( s_{h+1} = s \) with proba \( p(s_h, a_h = \pi(s_h), s) \)

Infinite horizon, stochastic transition

\[ V_\pi(s_0) = r(s_0) + \sum_{h=1}^{H} \gamma^h p(s_{h-1}, a_{h-1} = \pi(s_{h-1}), s_h) r(s_h) \]

with discount factor \( \gamma, 0 < \gamma < 1 \)

Remark

\( \gamma < 1 \rightarrow V < \infty \)

\( \gamma \) small \( \rightarrow \) myopic agent.
Value function and Q-value function

Value function

\[ V : S \mapsto \mathbb{R} \]

\( V_\pi(s) \): utility of state \( s \) when following policy \( \pi \)

Improving \( \pi \) by using \( V_\pi \) requires to know the transition model:

\[ \pi(s) \rightarrow \arg \max_a p(s, a, s') V_\pi(s') \]

Q function

\[ Q : (S \times A) \mapsto \mathbb{R} \]

\( Q_\pi(s, a) \): utility of selecting action \( a \) in state \( s \) when following policy \( \pi \)

Improving \( \pi \) by using \( Q_\pi \) is straightforward:

\[ \pi(s) \rightarrow \arg \max_a Q_\pi(s, a) \]
Optimal policies

From value function to a better policy

$$\pi(s) = \arg\max_a \{ p(s, a, s') V_\pi(s') \}$$

From policies to optimal value function

$$V^*(s) = \max_\pi V_\pi(s)$$

From value function to optimal policy

$$\pi^*(s) = \arg\max_a \{ p(s, a, s') V^*(s') \}$$
Linear and dynamic programming

If transition model and reward function are known

**Step 1**

\[ \pi(s) := \arg \max_a \left\{ \sum_{s'} p(s, a, s') \left( r(s') + \gamma V(s') \right) \right\} \]

**Step 2**

\[ V(s) := \sum_{s'} p(s, a = \pi(s), s') \left( r(s') + \gamma V(s') \right) \]

**Properties**

Converges eventually toward the optimum if all states, actions are considered.
Value iteration

Iterate

\[ V_{k+1}(s) := \max_a \left\{ \sum_{s'} p(s, a, s') \left( r(s') + \gamma V_k(s') \right) \right\} \]

Stop when

\[ \max_s |V_{k+1}(s) - V_k(s)| < \epsilon \]

Initialisation

- arbitrary
- educated is better

see Inverse Reinforcement Learning
Policy iteration

Principle

- Modify $\pi$  
- Update $V$ until convergence

Getting faster

- Don’t wait until $V$ has converged before modifying $\pi$. 
Discussion

Policy and value iteration

▶ Must wait until the end of the episode
▶ Episodes might be long

Can we update $V$ on the fly?

▶ I have estimates of how long it takes to go to RER, to catch the train, to arrive at Cité-U
▶ Something happens on the way (bump into a friend, chat, delay, miss the train,...)
▶ I can update my estimates of when I’ll be home...
TD(0)

1. Initialize $V$ and $\pi$
2. Loop on episode
   2.1 Initialize $s$
   2.2 Repeat
      Select action $a = \pi(s)$
      Observe $s'$ and reward $r$
      $V(s) \leftarrow V(s) + \alpha \left( r + \gamma V(s') - V(s) \right)$
      $s \leftarrow s'$
   2.3 Until $s'$ terminal state
Discussion

Update on the spot?

- Might be brittle
- Instead one can consider several steps

\[ R = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) \]

Find an intermediate between

- Policy iteration

\[ R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \]

- TD(0)

\[ R_t = r_{t+1} + \gamma V_t(s_{t+1}) \]
$TD(\lambda)$, intuition

$$R_t^\lambda = (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} R_t^{(n)} + \lambda^{T-t-1} R_t$$
TD(\(\lambda\)), intuition, followed by:

\[
\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)
\]
TD($\lambda$)

1. Initialize $V$ and $\pi$

2. Loop on episode
   2.1 Initialize $s$
   2.2 Repeat

   \[
   a = \pi(s) \\
   \text{Observe } s' \text{ and reward } r \\
   \delta \leftarrow r + V(s') - V(s) \\
   e(s) \leftarrow e(s) + 1
   \]

   For all $s''$

   \[
   V(s'') \leftarrow V(s'') + \alpha \delta e(s'') \\
   e(s'') \leftarrow \gamma \lambda e(s'')
   \]

   \[
   s \leftarrow s'
   \]

2.3 Until $s'$ terminal state
Q-learning

Principle: Iterate

▶ During an episode (from initial state until reaching a final state)
▶ At some point explore and choose another action;
▶ If it improves, update $Q(s, a)$:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[ r(s_{t+1}) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

Equivalent to

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t)(1 - \alpha) + \alpha [r(s_{t+1}) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})]$$
Partial summary

Notations

- State space $S$
- Action space $A$
- Transition model
  - deterministic: $s' = t(s, a)$
  - probabilistic: $p(s, a, s') \in [0, 1]$.
- Reward $r(s)$ bounded
- Time horizon $H$ (finite or infinite)

Policy $\pi \leftrightarrow$ Value function $V(s)$ (or $Q(s, a)$)

1. Update $V$ Iterate [until convergence]
2. Modify $\pi$
Reinforcement Learning, 2

Strengths
- Optimality guarantees (converge to global optimum)...

Weaknesses
- ...if each state is visited often, and each action is tried in each state
- Number of states: exponential wrt number of features
Behavioral cloning

*Input*
- Traces \((s_t, a_t)\) of expert

*Supervised learning*
- Learn \(\hat{h}(s_t) = a_t\)

*Limitations*
- Expert’s mistakes
- Mistakes of \(\hat{h}\): unbounded consequences
Inverse Reinforcement Learning

Abbeel, Ng, 2004

Input

▶ Traces \((s_t, a_t)\) of expert

Supervised learning

▶ Learn \(V\) t.q. \(V(s_t, a_t) > V(s_t, a')\)

Limitations

▶ Expert’s mistakes
▶ Requires appropriate representation

more ?

http://videolectures.net/ecmlpkdd2012_abbeel_learning_robotics/?q=Abbeel
Overview

Introduction

RL Algorithms

Values

Value functions

Optimal policy

Temporal differences and eligibility traces

Q-learning

Partial summary

Monte-Carlo Tree Search

Example: Computer-Go

Algorithm

Algorithm

Multi-Armed Bandits

Random phase

Evaluation and Propagation

Advanced MCTS

Rapid Action Value Estimate

Improving the rollout policy

Using prior knowledge

Open problems

Direct Value learning

Preference learning

Validation

Discussion
Any-time Reinforcement Learning

- **Standard 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:

- 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 $2 \times 10^{170}$
- 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 $S$
- Action space $A$
- 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

Introduction

RL Algorithms

Values

Value functions

Optimal policy

Temporal differences and eligibility traces

Q-learning

Partial summary

Monte-Carlo Tree Search

Example: Computer-Go

Algorithm

Algorithm

Multi-Armed Bandits

Random phase

Evaluation and Propagation

Advanced MCTS

Rapid Action Value Estimate

Improving the rollout policy

Using prior knowledge
Monte-Carlo Tree Search

Gradually grow the search tree:

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

Gradually grow the search tree:

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

Gradually grow the search tree:

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

Gradually grow the search tree:

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

Gradually grow the search tree:

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

Gradually grow the search tree:

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

Gradually grow the search tree:

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

- Returned solution:
  - Path visited most often
Monte-Carlo Tree Search

Gradually grow the search tree:

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

Gradually grow the search tree:

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

- Returned solution:
  - Path visited most often
Monte-Carlo Tree Search

Gradually grow the search tree:

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

Gradually grow the search tree:

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

Gradually grow the search tree:

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

- Returned solution:
  - Path visited most often
Monte-Carlo Tree Search

Gradually grow the search tree:

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

Gradually grow the search tree:

- Iterate Tree-Walk
  - Building Blocks
    - Select next action
  - 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 $a^* = \operatorname{argmax} \{ \hat{\mu}(s, a), tr(s, a) \in \mathcal{T} \}$
   \[ r \leftarrow \text{TreeWalk}(\mathcal{T}, tr(s, a^*)) \]
Else
   $\mathcal{A}_s = \{ \text{admissible actions not yet visited in } s \}$
   Select $a^*$ in $\mathcal{A}_s$
   Add $tr(s, a^*)$ as child node of $s$
   \[ r \leftarrow \text{RandomWalk}(tr(s, a^*)) \]
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 $r$

\[ A_{\text{rnd}} \leftarrow \{\} \quad // \text{store the set of actions visited in the random phase} \]

**While** $u$ is not final state

- Uniformly select an admissible action $a$ for $u$
  \[ A_{\text{rnd}} \leftarrow A_{\text{rnd}} \cup \{a\} \]
  \[ u \leftarrow \text{tr}(u, a) \]

**EndWhile**

\[ r = \text{Evaluate}(u) \quad //\text{reward vector of the tree-walk} \]

Return $r$
Monte-Carlo Tree Search

Properties of interest

- Consistency: \( \Pr(\text{finding optimal path}) \to 1 \) 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

2011  20 wins out of 20 games in 7x7 with minimal computer komi
2011  First win against a pro (6D), H2, 13×13
2011  First win against a pro (9P), H2.5, 13×13
2011  First win against a pro in Blind Go, 9×9
2010  Gold medal in TAAI, all categories
      19×19, 13×13, 9×9
2009  Win against a pro (5P), 9×9 (black)
2009  Win against a pro (5P), 9×9 (black)
2008  in against a pro (5P), 9×9 (white)
2007  Win against a pro (5P), 9×9 (blitz)
2009  Win against a pro (8P), 19×19 H9
2009  Win against a pro (1P), 19×19 H6
2008  Win against a pro (9P), 19×19 H7
Overview
Introduction
RL Algorithms
Values
Value functions
Optimal policy
Temporal differences and eligibility traces
Q-learning
Partial summary
Monte-Carlo Tree Search
Example: Computer-Go
Algorithm
Algorithm
Multi-Armed Bandits
Random phase
Evaluation and Propagation
Advanced MCTS
Rapid Action Value Estimate
Improving the rollout policy
Using prior knowledge
Direct Value learning
Preference learning
Validation
Discussion
Action selection as a Multi-Armed Bandit problem

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

**Lifelong learning**

**Exploration vs Exploitation** Dilemma

- Play the best arm so far?
- But there might exist better arms...
The multi-armed bandit (MAB) problem

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

$$n_{i,t} = \sum_{u=1}^{t} \mathbb{1}_{i^*_u = i} \quad \text{number of times i has been selected}$$

$$\hat{\mu}_{i,t} = \frac{1}{n_{i,t}} \sum_{i^*_u = i} r_u \quad \text{average reward of arm i}$$

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

$\Leftrightarrow$

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: $\epsilon$-greedy selection

At each time $t$,

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

$$= \arg\max\{\hat{\mu}_{1,t}, \ldots, \hat{\mu}_{K,t}\}$$

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

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

Optimal regret rate: $\log(t)$
Upper Confidence Bound

\[ \text{Select} = \arg\max \left\{ \hat{\mu}_{i,t} + \sqrt{C \frac{\log(\sum n_{j,t})}{n_{i,t}}} \right\} \]

Decision: Optimism in front of unknown!

Auer et al. 2002
Upper Confidence bound, followed

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

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

Extensions and variants

- Tune $c_e$ control the exploration/exploitation trade-off
- UCB-tuned: take into account the standard deviation of $\hat{\mu}_i$:

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

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

Introduction

RL Algorithms

Values
Value functions
Optimal policy
Temporal differences and eligibility traces
Q-learning

Partial summary

Monte-Carlo Tree Search

Example: Computer-Go
Algorithm
Algorithm
Multi-Armed Bandits
Random phase
Evaluation and Propagation
Advanced MCTS
Rapid Action Value Estimate
Improving the rollout policy
Using prior knowledge
Monte-Carlo Tree Search. Random phase

Gradually grow the search tree:

- Iterate Tree-Walk
  - Building Blocks
    - Select next action
  - Add a node
    - Grow a leaf of the search tree
  - Select next action bis
    - Random phase, roll-out
  - Compute instant reward
  - Update information in visited nodes

- Returned solution:
  - Path visited most often
Random phase – Roll-out policy

Monte-Carlo-based

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

Monte-Carlo-based

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

Brügman 93
Silver et al. 07
Evaluation and Propagation

The tree-walk returns an evaluation $r$ and $\text{win(\text{black})}$

**Propagate**

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

\[
\begin{align*}
n_{s,a} &\leftarrow n_{s,a} + 1 \\
\hat{\mu}_{s,a} &\leftarrow \hat{\mu}_{s,a} + \frac{1}{n_{s,a}} (r - \mu_{s,a})
\end{align*}
\]
Evaluation and Propagation

The tree-walk returns an evaluation $r$ \hspace{1cm} \text{win(black)}

Propagate

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

\[ n_{s,a} \leftarrow n_{s,a} + 1 \]
\[ \hat{\mu}_{s,a} \leftarrow \hat{\mu}_{s,a} + \frac{1}{n_{s,a}}(r - \mu_{s,a}) \]

Variants

\[ \hat{\mu}_{s,a} \leftarrow \begin{cases} 
\min\{\hat{\mu}_x, x \text{ child of } (s, a)\} & \text{if } (s, a) \text{ is a black node} \\
\max\{\hat{\mu}_x, x \text{ child of } (s, a)\} & \text{if } (s, a) \text{ is a white node}
\end{cases} \]

Kocsis & Szepesvári, 06
Dilemma

- smarter roll-out policy →
  more computationally expensive →
  less tree-walks on a budget

- frugal roll-out →
  more tree-walks →
  more confident evaluations
Overview

Introduction

RL Algorithms
- Values
- Value functions
- Optimal policy
- Temporal differences and eligibility traces
- Q-learning

Partial summary

Monte-Carlo Tree Search
- Example: Computer-Go
- Algorithm

Multi-Armed Bandits
- Random phase
- Evaluation and Propagation

Advanced MCTS
- Rapid Action Value Estimate
- Improving the rollout policy
- Using prior knowledge
Action selection revisited

Select \( a^* = \arg\max \left\{ \hat{\mu}_{s,a} + \sqrt{c_e \frac{\log(n_s)}{n_{s,a}}} \right\} \)

- Asymptotically optimal
- But visits the tree infinitely often!

Being greedy is excluded

Frugal and consistent

Select \( a^* = \arg\max \frac{\text{Nb win}(s,a) + 1}{\text{Nb loss}(s,a) + 2} \)

Further directions

- Optimizing the action selection rule

Berthier et al. 2010

Maes et al., 11
Controlling the branching factor

What if many arms?

- Continuous heuristics
  Use a small exploration constant $c_ε$

- Discrete heuristics
  Progressive Widening
  Coulom 06; Rolet et al. 09

Limit the number of considered actions to $\left\lfloor b\sqrt{n(s)} \right\rfloor$
(usually $b = 2$ or $4$)

Introduce a new action when
$\left\lfloor b\sqrt{n(s) + 1} \right\rfloor > \left\lfloor b\sqrt{n(s)} \right\rfloor$
(which one? See RAVE, below).
RAVE: Rapid Action Value Estimate

Motivation

- It needs some time to decrease the variance of $\hat{\mu}_{s,a}$
- Generalizing across the tree?

$$RAVE(s, a) = \text{average } \{\hat{\mu}(s', a), s \text{ parent of } s'\}$$
Rapid Action Value Estimate, 2

Using RAVE for action selection
In the action selection rule, replace $\hat{\mu}_{s,a}$ by

$$\alpha \hat{\mu}_{s,a} + (1 - \alpha) \left( \beta \text{RAVE}_\ell(s, a) + (1 - \beta) \text{RAVE}_g(s, a) \right)$$

$$\alpha = \frac{n_{s,a}}{n_{s,a} + c_1} \quad \beta = \frac{n_{\text{parent}(s)}}{n_{\text{parent}(s)} + c_2}$$

Using RAVE with Progressive Widening
- PW: introduce a new action if $\lceil b\sqrt{n(s)} + 1 \rceil > \lfloor b\sqrt{n(s)} \rfloor$
- Select promising actions: it takes time to recover from bad ones
- Select $\arg\max \text{RAVE}_\ell(\text{parent}(s))$. 
A limit of RAVE

- Brings information from bottom to top of tree
- Sometimes harmful:

B2 is the only good move for white
B2 only makes sense as first move (not in subtrees)
⇒ RAVE rejects B2.
Improving the roll-out policy $\pi$

$\pi_0$ Put stones uniformly in empty positions

$\pi_{\text{random}}$ Put stones uniformly in the neighborhood of a previous stone

$\pi_{\text{MoGo}}$ Put stones matching patterns prior knowledge

$\pi_{\text{RLGO}}$ Put stones optimizing a value function Silver et al. 07

Beware! Silver et al. 07

$\pi$ better $\pi'$ $\not\Rightarrow$ $\text{MCTS}(\pi)$ better $\text{MCTS}(\pi')$
Improving the roll-out policy $\pi$, followed

$\pi_{RLGO}$ against $\pi_{random}$

$\pi_{RLGO}$ against $\pi_{MoGo}$

Evaluation error on 200 test cases
Interpretation

What matters:

- Being **biased** is more harmful than being weak...
- Introducing a stronger but biased rollout policy \( \pi \) is detrimental.

if there exist situations where you (wrongly) think you are in good shape then you go there and you are in bad shape...
Using prior knowledge

Assume a value function $Q_{\text{prior}}(s, a)$

- Then when action $a$ is first considered in state $s$, initialize

$$n_{s,a} = n_{\text{prior}}(s, a) \quad \text{equivalent experience / confidence of priors}$$
$$\mu_{s,a} = Q_{\text{prior}}(s, a)$$

The best of both worlds

- Speed-up discovery of good moves
- Does not prevent from identifying their weaknesses
Overview

Introduction

RL Algorithms

Values
Value functions
Optimal policy
Temporal differences and eligibility traces
Q-learning
Partial summary

Monte-Carlo Tree Search

Example: Computer-Go
Algorithm
Algorithm
Multi-Armed Bandits
Random phase
Evaluation and Propagation
Advanced MCTS
Rapid Action Value Estimate
Improving the rollout policy
Using prior knowledge

Direct Value learning
Preference learning
Validation
Discussion
Parallelization. 1 Distributing the roll-outs

Distributing roll-outs on different computational nodes does not work.
Parallelization. 2 With shared memory

- Launch tree-walks in parallel on the same MCTS
- (micro) lock the indicators during each tree-walk update.

Use virtual updates to enforce the diversity of tree walks.
Parallelization. 3. Without shared memory

- Launch one MCTS per computational node
- $k$ times per second
  - Select nodes with sufficient number of simulations $> 0.05 \times \# \text{total simulations}$
  - Aggregate indicators

**Good news**
Parallelization with and without shared memory can be combined.
It works!

<table>
<thead>
<tr>
<th>32 cores against</th>
<th>Winning rate on $9 \times 9$</th>
<th>Winning rate on $19 \times 19$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$75.8 \pm 2.5$</td>
<td>$95.1 \pm 1.4$</td>
</tr>
<tr>
<td>2</td>
<td>$66.3 \pm 2.8$</td>
<td>$82.4 \pm 2.7$</td>
</tr>
<tr>
<td>4</td>
<td>$62.6 \pm 2.9$</td>
<td>$73.5 \pm 3.4$</td>
</tr>
<tr>
<td>8</td>
<td>$59.6 \pm 2.9$</td>
<td>$63.1 \pm 4.2$</td>
</tr>
<tr>
<td>16</td>
<td>$52 \pm 3.$</td>
<td>$63 \pm 5.6$</td>
</tr>
<tr>
<td>32</td>
<td>$48.9 \pm 3.$</td>
<td>$48 \pm 10$</td>
</tr>
</tbody>
</table>

Then:

▶ Try with a bigger machine! and win against top professional players!
▶ Not so simple... there are diminishing returns.
Increasing the number $N$ of tree-walks

<table>
<thead>
<tr>
<th>$N$</th>
<th>2$N$ against $N$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Winning rate on $9 \times 9$</td>
</tr>
<tr>
<td>1,000</td>
<td>71.1 ± 0.1</td>
</tr>
<tr>
<td>4,000</td>
<td>68.7 ± 0.2</td>
</tr>
<tr>
<td>16,000</td>
<td>66.5 ± 0.9</td>
</tr>
<tr>
<td>256,000</td>
<td>61 ± 0.2</td>
</tr>
</tbody>
</table>
The limits of parallelization

R. Coulom

Improvement in terms of performance against humans

≈

Improvement in terms of performance against computers

≈

Improvements in terms of self-play
Overview

Introduction

RL Algorithms

Values
Value functions
Optimal policy
Temporal differences and eligibility traces
Q-learning
Partial summary

Monte-Carlo Tree Search

Example: Computer-Go
Algorithm
Algorithm
Multi-Armed Bandits
Random phase
Evaluation and Propagation
Advanced MCTS
Rapid Action Value Estimate
Improving the rollout policy
Using prior knowledge
Failure: Semeai
Failure: Semeai
Failure: Semeai
Failure: Semeai
Failure: Semeai
Failure: Semeai
Failure: Semeai
Failure: Semeai
Failure: Semeai

Why does it fail

- First simulation gives 50%
- Following simulations give 100% or 0%
- But MCTS tries other moves: doesn’t see all moves on the black side are equivalent.
Implication 1

MCTS does not detect invariance → too short-sighted and parallelization does not help.
Implication 2

MCTS does not build abstractions $\rightarrow$ too short-sighted and parallelization does not help.