M1 — Apprentissage

Michèle Sebag – Benoit Barbot

LRI – LSV

24 février 2014
Overview

Introduction

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

Game of Go

Monte-Carlo Tree Search

Next

Experimental validation
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
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 $\pi : S \mapsto A$
  
  Maximize $E[\pi] = \text{Expected cumulative reward}$

(details 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 $\rightarrow$ 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>swann</td>
<td>black</td>
<td>—</td>
<td>take a video</td>
</tr>
<tr>
<td>bear</td>
<td>—</td>
<td>flee</td>
<td></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

<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.
- NEVER
- 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

Game of Go

Monte-Carlo Tree Search

Next

Experimental validation
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 $P_{ss'}^a$, 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
- Transition model: as expected (South, North etc, except: in A, any action sends you in A'; in B any action sends you in B'.

Compute the value function
**Underlying: Dynamic programming**

**Principle**
- Recursively decompose the problem in 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

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

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

Infinite horizon, stochastic transition

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

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**

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

\[ V_\pi(s) : \text{utility of state } s \text{ when following policy } \pi \]

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**

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

\[ Q_\pi(s, a) : \text{utility of selecting action } a \text{ in state } s \text{ when following policy } \pi \]

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

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

1. Initialize $V$ and $\pi$
2. Loop on episode
   2.1 Initialize $s$
   2.2 Repeat
      
      $a = \pi(s)$
      Observe $s'$ 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/
Overview

Introduction

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

Game of Go

Monte-Carlo Tree Search

Next

Experimental validation
Go as AI Challenge
Go as AI Challenge, foll’d

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 AI Challenge, foll’d

Features

- Number of games $2 \cdot 10^{170} \sim$ number of atoms in universe.
- Branching factor: 200 ($\sim 30$ for chess)
- No good heuristic function to assess a position
- Local and global features (symmetries, freedom, ...)
- A move might make a difference some dozen plies later

Where is the difficulty?

- You can’t grow the full tree
- You can’t safely cut branches
- You can’t be greedy
Principles of MoGo

Gelly Wang 07, Gelly Silver 07

- A weak but unbiased assessment function: Monte Carlo-based
- Allowing the machine to play against itself and build its own strategy

Exploration vs Exploitation dilemma
Assessing a position Fast and frugal...

Monte-Carlo-based Brügman (1993)

1. While possible, add a stone (white, black)
2. Compute Win(black)
3. Repeat and average

Remark: The point is to be unbiased
if there exist situations where you (wrongly) think you are in good shape
then you go there
and you are in bad shape...
Build a strategy: Monte-Carlo Tree Search

In a given situation:
   Select a move Multi-Armed Bandit

In the end:
1. Assess the final move Monte-Carlo
2. Update reward for all moves
Select a move

**Exploration vs Exploitation**

Dilemma

**Multi-Armed Bandits**

- In a casino, one wants to maximize one’s gains *while playing*
- Play the best arms so far?
- But there might exist better arms...

Lai, Robbins 85

Exploitation

Exploration
Multi-Armed Bandits, foll’d

Auer et al. 01, 02; Kocsis Szepesvári 06

For each arm (move)

- Reward: Bernoulli variable $\sim \mu_i, 0 \leq \mu_i \leq 1$
- Empirical estimate: $\hat{\mu}_i \pm \text{Confidence} \left( n_i \right)$

Decision: Optimism in front of unknown!

Select $i^* = \arg\max \hat{\mu}_i + C \sqrt{\frac{\log(\sum n_j)}{n_i}}$
Multi-Armed Bandits, foll’d

Auer et al. 01, 02; Kocsis Szepesvári 06  

For each arm (move)

- Reward: Bernoulli variable $\sim \mu_i, 0 \leq \mu_i \leq 1$
- Empirical estimate: $\hat{\mu}_i \pm \text{Confidence (} n_i \text{)}$

Decision: Optimism in front of unknown!

Select $i^* = \arg\max \hat{\mu}_i + C \sqrt{\frac{\log(\sum n_j)}{n_i}}$

Arm A  Arm B

Arm A  Arm B

Arm A  Arm B

Arm B

Arm A
Multi-Armed Bandits, foll’d

Auer et al. 01, 02; Kocsis Szepesvári 06  **Criterion: the regret of your strategy** \( \pi \)

- Not what you gain,
- But what you loose compared to the oracle

\[
Regret(\pi) = \sum_{t=1}^{T} (\mu^* - \mu(\pi(t))
\]

**Note**

- Optimal regret in \( \log(T) \)
- UCB achieves the optimal regret
- Compare to \( \epsilon \)-greedy...

Lai Robbins 85
Auer et al. 02
The UCT scheme

- Upper Confidence Tree (UCT) [1]
  - Gradually grow the search tree
  - Building Blocks
    - Select next action (bandit-based phase)
    - Add a node (leaf of the search tree)
    - Select next action bis (random phase)
    - Compute instant reward
    - Update information in visited nodes
  - Returned solution:
    - Path visited most often

L. Kocsis, and C. Szepesvári, 06
The UCT scheme

- Upper Confidence Tree (UCT) [1]
  - Gradually grow the search tree
  - Building Blocks
    - Select next action (bandit-based phase)
    - Add a node (leaf of the search tree)
    - Select next action bis (random phase)
    - Compute instant reward
    - Update information in visited nodes
  - Returned solution:
    - Path visited most often

L. Kocsis, and C. Szepesvári, 06
The UCT scheme

- Upper Confidence Tree (UCT) [1]
  - Gradually grow the search tree
  - Building Blocks
    - Select next action (bandit-based phase)
    - Add a node (leaf of the search tree)
    - Select next action bis (random phase)
    - Compute instant reward
    - Update information in visited nodes
  - Returned solution:
    - Path visited most often

L. Kocsis, and C. Szepesvári, 06
The UCT scheme

- Upper Confidence Tree (UCT) [1]
  - Gradually grow the search tree
  - Building Blocks
    - Select next action (bandit-based phase)
    - Add a node (leaf of the search tree)
    - Select next action bis (random phase)
    - Compute instant reward
    - Update information in visited nodes
  - Returned solution:
    - Path visited most often

L. Kocsis, and C. Szepesvári, 06
The UCT scheme

- Upper Confidence Tree (UCT) [1]
  - Gradually grow the search tree
  - Building Blocks
    - Select next action (bandit-based phase)
    - Add a node (leaf of the search tree)
    - Select next action bis (random phase)
    - Compute instant reward
    - Update information in visited nodes
  - Returned solution:
    - Path visited most often

L. Kocsis, and C. Szepesvári, 06
The UCT scheme

- Upper Confidence Tree (UCT) [1]
  - Gradually grow the search tree
  - Building Blocks
    - Select next action (bandit-based phase)
    - Add a node (leaf of the search tree)
    - Select next action bis (random phase)
    - Compute instant reward
    - Update information in visited nodes
  - Returned solution:
    - Path visited most often

L. Kocsis, and C. Szepesvári, 06
The UCT scheme

- Upper Confidence Tree (UCT) [1]
  - Gradually grow the search tree
  - Building Blocks
    - Select next action (bandit-based phase)
    - Add a node (leaf of the search tree)
    - Select next action bis (random phase)
    - Compute instant reward
    - Update information in visited nodes
- Returned solution:
  - Path visited most often

L. Kocsis, and C. Szepesvári, 06
The UCT scheme

- Upper Confidence Tree (UCT) [1]
  - Gradually grow the search tree
  - Building Blocks
    - Select next action (bandit-based phase)
    - Add a node (leaf of the search tree)
    - Select next action bis (random phase)
    - Compute instant reward
    - Update information in visited nodes
  - Returned solution:
    - Path visited most often

L. Kocsis, and C. Szepesvári, 06
The UCT scheme

- Upper Confidence Tree (UCT) [1]
  - Gradually grow the search tree
  - Building Blocks
    - Select next action (bandit-based phase)
    - Add a node (leaf of the search tree)
    - Select next action bis (random phase)
    - Compute instant reward
    - Update information in visited nodes
  - Returned solution:
    - Path visited most often

L. Kocsis, and C. Szepesvári, 06
The UCT scheme

- Upper Confidence Tree (UCT) [1]
  - Gradually grow the search tree
  - Building Blocks
    - Select next action (bandit-based phase)
    - Add a node (leaf of the search tree)
    - Select next action bis (random phase)
    - Compute instant reward
    - Update information in visited nodes
  - Returned solution:
    - Path visited most often

L. Kocsis, and C. Szepesvári, 06
The UCT scheme

- Upper Confidence Tree (UCT) [1]
  - Gradually grow the search tree
  - Building Blocks
    - Select next action (bandit-based phase)
    - Add a node (leaf of the search tree)
    - Select next action bis (random phase)
    - Compute instant reward
    - Update information in visited nodes
  - Returned solution:
    - Path visited most often

L. Kocsis, and C. Szepesvári, 06
The UCT scheme

- Upper Confidence Tree (UCT) [1]
  - Gradually grow the search tree
  - Building Blocks
    - Select next action (bandit-based phase)
    - Add a node (leaf of the search tree)
    - Select next action bis (random phase)
    - Compute instant reward
    - Update information in visited nodes
  - Returned solution:
    - Path visited most often

L. Kocsis, and C. Szepesvári, 06
The UCT scheme

- Upper Confidence Tree (UCT) [1]
  - Gradually grow the search tree
  - Building Blocks
    - Select next action (bandit-based phase)
    - Add a node (leaf of the search tree)
    - Select next action bis (random phase)
    - Compute instant reward
    - Update information in visited nodes
  - Returned solution:
    - Path visited most often

L. Kocsis, and C. Szepesvári, 06
The UCT scheme

- Upper Confidence Tree (UCT) [1]
  - Gradually grow the search tree
  - Building Blocks
    - Select next action (bandit-based phase)
    - Add a node (leaf of the search tree)
    - Select next action bis (random phase)
    - Compute instant reward
    - Update information in visited nodes
  - Returned solution:
    - Path visited most often

L. Kocsis, and C. Szepesvári, 06
Monte-Carlo Tree Search

Comments: MCTS grows an asymmetrical tree
- Most promising branches are more explored
- thus their assessment becomes more precise
- Guarantees of optimality

Going Beyond
- Take into account standard deviation of $\mu$
- Adjust constant $C$
- Needs heuristics to deal with many arms
- Share information among branches
BB1: Select Next Action

**Rules for the Multi-Armed Bandit phase**

- **Upper Confidence Bound**
  
  Select \( \text{argmax}_{a \in A} \hat{\mu}_a + \sqrt{\frac{c_e \log(T)}{n_a}} \)

- **Upper Confidence Bound (UCB1-tuned)**
  
  Select \( \text{argmax}_{a \in A} \hat{\mu}_a + \sqrt{\frac{c_e \log(T)}{n_a}} \min \left( \frac{1}{4}, \hat{\sigma}_a^2 + \sqrt{\frac{c_e \log(T)}{t_a}} \right) \)

\( T \) Total number of trials in current node  
\( n_a \) Number of trials for action \( a \)  
\( \hat{\mu}_a \) Empirical average reward for action \( a \)  
\( \hat{\sigma}_a^2 \) Empirical variance of reward for action \( a \)
BB1: Select Next Action, foll’d

Select \( \arg \max_{a \in A} \hat{\mu}_a + \sqrt{\frac{c_e \log(T)}{n_a}} \)

- Asymptotically optimal
- But visits the tree infinitely often!

Why not selecting always the best?

- Not consistent:
  if unlucky, good arm gets 0 first time it’s played, bad arm gets 1
  first time it’s played, always play bad arm...

Frugal and consistent

Select \( \arg \max_{a \in A} \frac{\text{Nb win} + 1}{\text{Nb loss} + 2} \)

Berthier et al. 2010
BB2: Control the branching factor

What if many arms? degenerate into exploration

- **Continuous heuristics**
  Use a small exploration constant $c_e$

- **Discrete heuristics** Progressive Widening
  Coulom 06; Rolet et al. 09
  Consider only $\lfloor T^b \rfloor$ actions ($b < 1$, usually $b = 1/2$)

When $n_a = 4, 9, 16, 25, \ldots$, you are allowed to consider a new arm.
BB3: Consider a new arm, which one?

Prefer good initialization: it takes time to recover from a bad one

Which good moves? Sharing information among nodes

Rapid Action Value Estimation (RAVE)

$$\text{RAVE}(n,m) = \text{average } \mu (n)_{m \text{ ancestor of } n}$$

$\hat{\mu}_a$ has a high variance $\rightarrow$ rather use

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

$$\alpha = \frac{n_a}{n_a + c_1} \quad \beta = \frac{n_{a,\ell}}{n_{a,\ell} + c_2}$$
BB4: The random phase

Note

- MoGo worked because of the smart random phase,
- not because of UCT.

Random phase, 3 variants

1. Put stones randomly
2. Put stones randomly in the neighborhood of a previous stone
3. Put stones matching patterns expertise
Many trials!

Failures

- Grafting expert knowledge
- Replacing the random phase by a smarter one
- Using value of the game instead of 0/1
- Compiling RAVE with a neural net

Successes

- Parallelizing
- Learning an opening book
Comparative results

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  MoGoTW
   19×19, 13×13, 9×9
2007 Win against a pro (5P), 9×9 (blitz)  MoGo
2008 in against a pro (5P), 9×9 (white)  MoGo
2009 Win against a pro (5P), 9×9 (black)  MoGo
2009 Win against a pro (5P), 9×9 (black)  MoGoTW
   200 Win against a pro (8P), 19×19 H9  MoGo
2008 Win against a pro (9P), 19×19 H7  MoGo
2009 Win against a pro (1P), 19×19 H6  MoGo
Comparative results
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.
Implications
Energy policy

Claim
Many problems can be phrased as optimization in front of the uncertainty.

- Adversarial setting: 2 two-player game
- Uniform setting: a single player game

Management of energy stocks under uncertainty
States and Decisions

States

- Amount of stock (60 nuclear, 20 hydro.)
- Varying: price, weather alea or archive
- Decision: release water from one reservoir to another
- Assessment: meet the demand, otherwise buy energy
When Machine Learning is Feature Selection

Bio-informatics

- 30 000 genes
- few examples (expensive)
- Goal: find the genes related to (disease, exposure to radiations, etc).
Feature Selection

Combinatorial optimization:
Set of features $\mathcal{F}$
Set of states $S = 2^\mathcal{F}$
Initial state $\emptyset$
Set of actions $A = \{\text{add } f, \ f \in \mathcal{F}\}$
Final state any state
Reward function $r : S \mapsto [0, 1]$

Goal: minimize generalization error
Find $\arg\min_{F \subseteq \mathcal{F}} \text{Err}(A(F, D))$
Feature Selection with MCTS: Fuse

Select next move: UCB + Progressive widening

Select new arm/feature: use RAVE

Random phase

- No fixed time horizon
- Introduce specific stopping actions
- In random phase, stop with probability $1 - q^d$ \quad (q < 1, user specified)
FUSE: reward($F$)

Generalization error estimate

- Requisite
  - fast (to be computed $10^4$ times)
  - unbiased

- Proposed reward
  - $k$-NN like
  - + AUC criterion *

- Complexity: $\tilde{O}(mnd)$
  - $d$ Number of selected features
  - $n$ Size of the training set
  - $m$ Size of sub-sample ($m \ll n$)

(*) Mann Whitney Wilcoxon test:

$$V(F) = \frac{|\{(x,y),(x',y')\} \in \mathcal{V}^2, N_{F,k}(x) < N_{F,k}(x'), y < y'\}|}{|\{(x,y),(x',y')\} \in \mathcal{V}^2, y < y'\}|$$
FUSE: reward($F$)

Generalization error estimate

- Requisite
  - fast (to be computed $10^4$ times)
  - unbiased

- Proposed reward
  - $k$-NN like
  - + AUC criterion *

- Complexity: $\tilde{O}(mnd)$
  - $d$ Number of selected features
  - $n$ Size of the training set
  - $m$ Size of sub-sample ($m \ll n$)

(*) Mann Whitney Wilcoxon test:

$$V(F) = \frac{|\{(x,y),(x',y')\in\mathcal{V}^2, \mathcal{N}_{F,k}(x) < \mathcal{N}_{F,k}(x'), y < y'\}|}{|\{(x,y),(x',y')\in\mathcal{V}^2, y < y'\}|}$$
FUSE: reward($F$)

Generalization error estimate

- **Requisite**
  - fast (to be computed $10^4$ times)
  - unbiased

- **Proposed reward**
  - $k$-NN like
  - + AUC criterion *

- **Complexity:** $\tilde{O}(mnd)$
  
  $d$: Number of selected features
  $n$: Size of the training set
  $m$: Size of sub-sample ($m \ll n$)

(*): Mann Whitney Wilcoxon test:

$$V(F) = \frac{|\{(x,y),(x',y')\} \in \mathcal{V}^2, \mathcal{N}_{F,k}(x) < \mathcal{N}_{F,k}(x'), y < y'\}|}{|\{(x,y),(x',y')\} \in \mathcal{V}^2, y < y'\}|$$
**FUSE: reward(\(F\))**

Generalization error estimate

- **Requisite**
  - fast (to be computed \(10^4\) times)
  - unbiased

- **Proposed reward**
  - \(k\)-NN like
  - + AUC criterion *

- **Complexity: \(\tilde{O}(mnd)\)**
  - \(d\) Number of selected features
  - \(n\) Size of the training set
  - \(m\) Size of sub-sample \((m \ll n)\)

\((*)\) Mann Whitney Wilcoxon test:

\[
V(F) = \frac{|\{((x,y),(x',y')) \in \mathcal{V}^2, \mathcal{N}_{F,k}(x) < \mathcal{N}_{F,k}(x'), y < y'\}|}{|\{((x,y),(x',y')) \in \mathcal{V}^2, y < y'\}|}
\]
FUSE: reward($F$)

Generalization error estimate

- Requisite
  - fast (to be computed $10^4$ times)
  - unbiased

- Proposed reward
  - $k$-NN like
  - + AUC criterion *

- Complexity: $\tilde{O}(mnd)$
  - $d$ Number of selected features
  - $n$ Size of the training set
  - $m$ Size of sub-sample ($m \ll n$)

(*) Mann Whitney Wilcoxon test:

$$V(F) = \frac{|\{((x,y),(x',y'))\in \mathcal{V}^2, \mathcal{N}_{F,k}(x)<\mathcal{N}_{F,k}(x'), y<y'\}|}{|\{((x,y),(x',y'))\in \mathcal{V}^2, y<y'\}|}$$
Experimental setting

Questions
- FUSE vs FUSE$^R$
- Continuous vs discrete exploration heuristics
- FS performance w.r.t. complexity of the target concept
- Convergence speed

Experiments on

<table>
<thead>
<tr>
<th>DATA SET</th>
<th>SAMPLES</th>
<th>FEATURES</th>
<th>PROPERTIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Madelon [1]</td>
<td>2,600</td>
<td>500</td>
<td>XOR-like</td>
</tr>
<tr>
<td>Arcene [1]</td>
<td>200</td>
<td>10,000</td>
<td>Redundant features</td>
</tr>
<tr>
<td>Colon</td>
<td>62</td>
<td>2,000</td>
<td>“Easy”</td>
</tr>
</tbody>
</table>

[1] NIPS'03
Experimental setting

- Baselines
  - CFS (Constraint-based Feature Selection) [1]
  - Random Forest [2]
  - Lasso [3]
  - RAND$^R$: RAVE obtained by selecting 20 random features at each iteration

- Results averaged on 50 splits (10 × 5 fold cross-validation)

- End learner
  - Hyper-parameters optimized by 5 fold cross-validation

[1] M. A. Hall ICML'00
Results on Madelon after 200,000 iterations

- **Remark:** $\text{FUSE}^R = \text{best of both worlds}$
  - Removes redundancy (like CFS)
  - Keeps conditionally relevant features (like Random Forest)
Results on Arcene after 200,000 iterations

- **Remark**: $\text{FUSE}^R = \text{best of both worlds}$
  - Removes redundancy (like CFS)
  - Keeps conditionally relevant features (like Random Forest)

T-test “CFS vs. $\text{FUSE}^R$” with 100 features: \( p\text{-value}=0.036 \)
Results on Colon after 200,000 iterations

Remark

All equivalent
# NIPS 2003 Feature Selection challenge

- Test error on a disjoint test set

<table>
<thead>
<tr>
<th>DATABASE</th>
<th>ALGORITHM</th>
<th>CHALLENGE ERROR</th>
<th>SUBMITTED FEATURES</th>
<th>IRRELEVANT FEATURES</th>
</tr>
</thead>
<tbody>
<tr>
<td>MadeLON</td>
<td>FSPP2 [1]</td>
<td>6.22% (1\textsuperscript{st})</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>D-FUSE\textsuperscript{R}</td>
<td>6.50% (24\textsuperscript{th})</td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>Arcene</td>
<td>Bayes-nn-red [2]</td>
<td>7.20% (1\textsuperscript{st})</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>D-FUSE\textsuperscript{R} (on all)</td>
<td>8.42% (3\textsuperscript{rd})</td>
<td>500</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>D-FUSE\textsuperscript{R}</td>
<td>9.42% 500 (8\textsuperscript{th})</td>
<td>500</td>
<td>0</td>
</tr>
</tbody>
</table>

Conclusion

Contributions

▶ Formalization of Feature Selection as a Markov Decision Process
▶ Efficient approximation of the optimal policy (based on UCT)
  ⇒ Any-time algorithm
▶ Experimental results
  ▶ State of the art
  ▶ High computational cost (45 minutes on Madelon)
Perspectives

- Use other end learners
- Revisit the reward see (Hand 2010) about AUC
- Extend to Feature construction
Extension to Feature Construction

**Context:** Data Mining from Relational Databases

- A star schema (customer, invoices, services...)
- Usually: propositionalize the database
- Alternative: Feature Construction
  Explore the language of queries
Lesson 1: When is MCTS relevant

Applicable when
- High dimension
- Complex model
- Delayed reward

Challenges
- When the game is highly non-observable
- When no baseline for the random phase
Lesson 2: Using by-products of MCTS

- FUSE with $N$ iterations:
  - each iteration i) follows a path; ii) evaluates a final node

- Result:
  - Search tree (most visited path) $\leftrightarrow$ RAVE score
  - $\Downarrow$
  - Wrapper approach $\leftrightarrow$ Filter approach

- On the feature subset, use end learner $\mathcal{A}$
  - Any Machine Learning algorithm
  - Support Vector Machine with Gaussian kernel in experiments
Perspectives

Extensions
- Continuous RAVE (continuous action / state spaces)
- Partially observable settings (poker)

Applications
- Feature construction
- Robotics (action selection)