

Master 2 Recherche

Apprentissage Statistique, Optimisation et Applications

Michèle Sebag – Balazs Kégl – Anne Auger

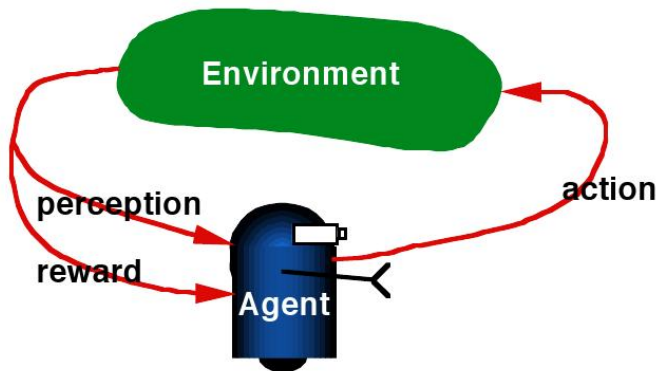
TAO: Theme Apprentissage & Optimisation

<http://tao.lri.fr/tiki-index.php>

18 janvier 2012



Apprentissage par Renforcement



Cas général

- ▶ Un agent est dans le temps et dans l'espace
- ▶ L'environnement est stochastique et incertain
- ▶ Le but est d'agir sur l'environnement
- ▶ de façon à maximiser une fonction de satisfaction (reward)

Qu'est-ce qu'on apprend ?

Une politique = une stratégie = (état \rightarrow action)

Apprentissage par Renforcement

Plan du cours

1. Contexte
2. Algorithmes
3. Exemple : jouer au Go
4. de MoGo à la sélection de variables.

MoGo

Apprentissage par Renforcement: Plan du cours

Contexte

Algorithms

Value functions

Optimal policy

Temporal differences and eligibility traces

Q-learning

Playing Go: MoGo

Feature Selection as a Game

Position du problème

Monte-Carlo Tree Search

Feature Selection: the FUSE algorithm

Experimental Validation

Active Learning as a Game

Position du problème

Algorithme BAAL

Validation expérimentale

Constructive Induction

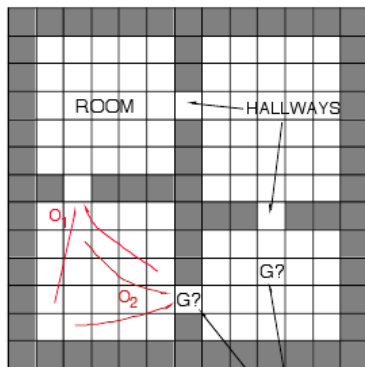
Apprentissage par Renforcement

Contexte

Le monde est inconnu.

Certaines actions, dans certains états, portent des fruits (*rewards*) avec un certain retard [avec une certaine probabilité].

Le but : trouver la politique (état \rightarrow action)
maximisant l'espérance de reward



4 rooms

4 hallways

4 unreliable
primitive actions



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

Given goal location,
quickly plan shortest route

Apprentissage par Renforcement, exemple

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.

Apprentissage par renforcement

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.

Formalisation

Formalisation

- ▶ Espace d'états \mathcal{S}
- ▶ Espace d'actions \mathcal{A}
- ▶ Fonction de transition $p(s, a, s') \mapsto [0, 1]$
- ▶ Reward $r(s)$

But

- ▶ Trouver politique $\pi : \mathcal{S} \mapsto \mathcal{A}$

Maximiser $E[\pi] =$ Espérance du reward cumulé

(détails après)

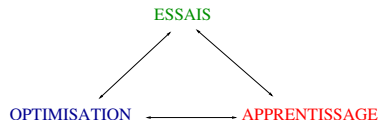
Quelques applications

- ▶ Robotique
Navigation, football, marche, jonglage
- ▶ Jeux
Backgammon, Othello, Tetris, Go, ...
- ▶ Contrôle
Hélicoptère, ascenseurs, telecom, grilles de calcul, gestion de processus industriels, ...
- ▶ Recherche opérationnelle
Transport, scheduling, ...
- ▶ Autres
Computer Human Interfaces, ...

Position du problème

Trois problèmes

- ▶ Apprendre le monde (p, r)
- ▶ Décider
- ▶ Faire des essais



Sources

- ▶ Sutton & Barto, Reinforcement Learning, MIT Press, 1998



<http://www.eecs.umich.edu/~baveja/NIPS05RLTutorial/>

Cas particulier

Quand on connaît la fonction de transition

Reinforcement learning \rightarrow Optimal control

Défis

Malédiction de la dimensionalité

- ▶ état : décrit par *taille, apparence, couleur, ...*
| \mathcal{S} | exponentiel en fonction du nombre d'attributs
- ▶ Mais tous les attributs ne sont pas toujours pertinents

Exemple:

voir	cygne	blanc	—
	cygne	noir	prendre une photo
	ours	—	fuir

Défis

Malédiction de la dimensionalité

- ▶ état : décrit par *taille, apparence, couleur, ...*
| \mathcal{S} | exponentiel en fonction du nombre d'attributs
- ▶ Mais tous les attributs ne sont pas toujours pertinents

Exemple:

voir	cygne	blanc	—
	cygne	noir	prendre une photo
	ours	—	fuir

Horizon – Rationalité limitée

- ▶ Horizon infini : on a l'éternité devant soi. JAMAIS
- ▶ Horizon fini inconnu : on veut une politique qui trouve le but aussi vite que possible
- ▶ Horizon fini : on veut une politique qui trouve le but après T pas de temps
- ▶ Rationalité limitée : on veut trouver **rapidement** une politique **raisonnable** (qui trouve une approximation du but)

Reinforcement learning

Contexte

Algorithms

Value functions

Optimal policy

Temporal differences and eligibility traces

Q-learning

Playing Go: MoGo

Feature Selection as a Game

Position du problème

Monte-Carlo Tree Search

Feature Selection: the FUSE algorithm

Experimental Validation

Active Learning as a Game

Position du problème

Algorithme BAAL

Validation expérimentale

Constructive Induction

Formalisation

Notations

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

bounded

Goal

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

Approaches

- ▶ Value function
 - ▶ Value iteration
 - ▶ Policy iteration
- ▶ Temporal differences
- ▶ Q-learning
- ▶ Direct policy search
optimization in the π 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 γ , $0 < \gamma < 1$

Remark

$\gamma < 1 \rightarrow V < \infty$

γ 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 π

Improving π by using V_π requires to know the transition model:

$$\pi(s) \rightarrow \arg \max P_{ss'}^a V_\pi(s')$$

Q function

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

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

Improving π by using Q_π is straightforward:

$$\pi(s) \rightarrow \arg \max Q_\pi(s, a)$$

Optimal policies

From value function to a better policy

$$\pi(s) = \operatorname{argmax}_a \{P_{ss'}^a V_\pi(s')\}$$

From policies to optimal value function

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

From value function to optimal policy

$$\pi^*(s) = \operatorname{argmax}_a \{P_{ss'}^a V^*(s')\}$$

Linear and dynamic programming

If transition model and reward function are known

Step 1

$$\pi(s) := \arg \max_a \left\{ \sum_{s'} P_{s,s'}^a (r(s') + \gamma V(s')) \right\}$$

Step 2

$$V(s) := \sum_{s'} P_{s,s'}^{a=\pi(s)} (r(s') + \gamma V(s'))$$

Properties

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

Value iteration

Bellman equation

Iterate

$$V_{k+1}(s) := \max_a \left\{ \sum_{s'} P_{s,s'}^a (r(s') + \gamma V_k(s')) \right\}$$

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 π step 1
- ▶ Update V until convergence step 2

Getting faster

- ▶ Don't wait until V has converged before modifying π .

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 π
2. Loop on episode
 - 2.1 Initialize s
 - 2.2 Repeat

Select action $a = \pi(s)$

Observe s' and reward r

$$V(s) \leftarrow V(s) + \alpha \underbrace{(r + \gamma V(s') - V(s))}_R$$

$$s \leftarrow s'$$

- 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

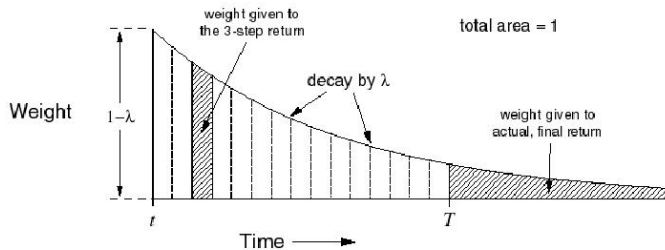
- ▶ Policy iteration

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots$$

- ▶ TD(0)

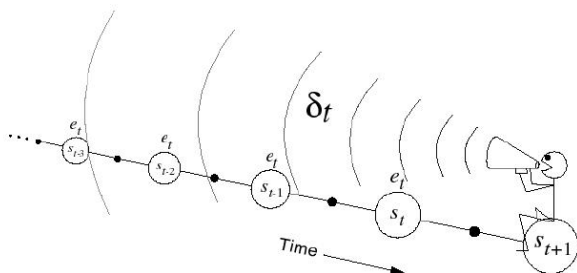
$$R_t = r_{t+1} + \gamma V_t(s_{t+1})$$

TD(λ), intuition



$$R_t^\lambda = (1-\lambda) \underbrace{\sum_{n=1}^{T-t-1} \lambda^{n-1} R_t^{(n)}}_{\text{}} + \underbrace{\lambda^{T-t-1} R_T}_{\text{}}$$

TD(λ), intuition, followed



$$\delta_t = r_{t+1} + \mathcal{W}_t(s_{t+1}) - V_t(s_t)$$

TD(λ)

1. Initialize V and π
2. Loop on episode
 - 2.1 Initialize s
 - 2.2 Repeat

$$a = \pi(s)$$

Observe s' and reward r

$$\delta \leftarrow r + V(s') - V(s)$$

$$e(s) \leftarrow e(s) + \delta$$

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 $Q(s, a)$:

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \times \left[\underbrace{r(s_{t+1})}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})}_{\text{max future value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right]$$

The equation shows the update rule for the Q-value of state s_t and action a_t . The current value is updated by adding a fraction α (learning rate) of the difference between the current value and a new value. The new value is the sum of the immediate reward $r(s_{t+1})$ and the discounted maximum future value $\gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$.

Equivalent to

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

Apprentissage par Renforcement: Plan du cours

Contexte

Algorithms

Value functions

Optimal policy

Temporal differences and eligibility traces

Q-learning

Playing Go: MoGo

Feature Selection as a Game

Position du problème

Monte-Carlo Tree Search

Feature Selection: the FUSE algorithm

Experimental Validation

Active Learning as a Game

Position du problème

Algorithme BAAL

Validation expérimentale

Constructive Induction

Go as AI Challenge

Features

- ▶ Number of games $2 \cdot 10^{170} \sim$ number of atoms in universe.
- ▶ Branching factor: 200 (~ 30 for chess)
- ▶ Assessing a game ?
- ▶ Local and global features (symmetries, freedom, ...)



Gelly Silver 2007

Principles of MoGo

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

Weak unbiased assessment

Monte-Carlo-based

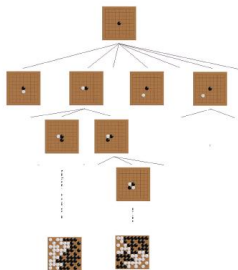
Brügman (1993)

1. While possible, add a stone (white, black)
2. Compute Win(black)
3. Average on 1-2



Remark: The point is to be unbiased if there exists situations where you (wrongly) think you're in good shape then you go there and you're 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
2. Update reward for all moves

Monte-Carlo

Select a move

Exploration vs Exploitation Dilemma



Lai, Robbins 1985

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...

Exploitation

Exploration

Multi-Armed Bandits, foll'd

Auer et al. 2001, 2002; Kocsis Szepesvari 2006

For each arm (move)

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

Decision: Optimism in front of unknown!

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

Multi-Armed Bandits, foll'd

Auer et al. 2001, 2002; Kocsis Szepesvari 2006

For each arm (move)

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

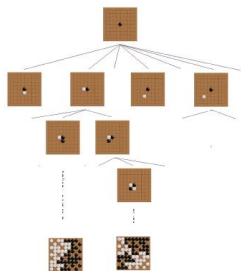
Decision: Optimism in front of unknown!

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

Variants

- ▶ Take into account standard deviation of $\hat{\mu}_i$
- ▶ Trade-off controlled by C
- ▶ Progressive widening

Monte-Carlo Tree Search



Comments: MCTS grows an asymmetrical tree

- ▶ Most promising branches are more explored
- ▶ thus their assessment becomes more precise
- ▶ Needs heuristics to deal with many arms...
- ▶ Share information among branches

MoGo: World champion in 2006, 2007, 2009
First to win over a 7th Dan player in 19×19

Apprentissage par Renforcement: Plan du cours

Contexte

Algorithms

Value functions

Optimal policy

Temporal differences and eligibility traces

Q-learning

Playing Go: MoGo

Feature Selection as a Game

Position du problème

Monte-Carlo Tree Search

Feature Selection: the FUSE algorithm

Experimental Validation

Active Learning as a Game

Position du problème

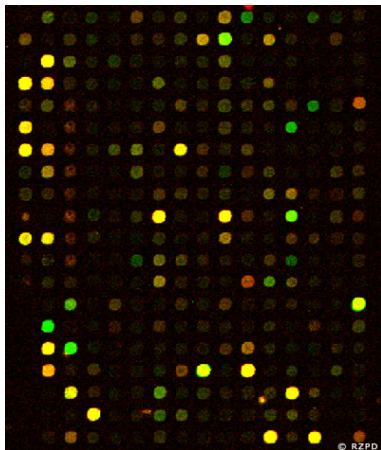
Algorithme BAAL

Validation expérimentale

Constructive Induction

Quand l'apprentissage c'est la sélection d'attributs

Bio-informatique



- ▶ 30 000 gènes
- ▶ peu d'exemples (chers)
- ▶ but : trouver les gènes pertinents

Position du problème

Buts

- Sélection : trouver un sous-ensemble d'attributs
- Ordre/Ranking : ordonner les attributs

Formulation

Soient les attributs $\mathcal{F} = \{f_1, ..f_d\}$. Soit la fonction :

$$\mathcal{G} : \mathcal{P}(\mathcal{F}) \mapsto \mathbb{R}$$

$$F \subset \mathcal{F} \mapsto Err(F) = \text{erreur min. des hypothèses fondées sur } F$$

Trouver $Argmin(\mathcal{G})$

Difficultés

- Un problème d'optimisation combinatoire (2^d)
- D'une fonction \mathcal{F} inconnue...

Approches

Filter

méthode univariée

Définir $score(f_i)$; ajouter itérativement les attributs maximisant $score$

ou retirer itérativement les attributs minimisant $score$

+ simple - pas cher

- optima très locaux

Rq : on peut backtrack : meilleurs optima, mais plus cher

Wrapping

méthode multivariée

Mesurer la qualité d'attributs en rapport avec d'autres attributs :

estimer $\mathcal{G}(f_{i1}, \dots, f_{ik})$

- cher : une estimation = un pb d'apprentissage.

+ optima meilleurs

Méthodes hybrides.

Approches filtre

Notations

Base d'apprentissage : $\mathcal{E} = \{(x_i, y_i), i = 1..n, y_i \in \{-1, 1\}\}$
 $f(x_i)$ = valeur attribut f pour exemple (x_i)

Gain d'information

arbres de décision

$$p([f = v]) = Pr(y = 1 | f(x_i) = v)$$
$$QI([f = v]) = -p \log p - (1 - p) \log (1 - p)$$
$$QI = \sum_v p(v) QI([f = v])$$

Corrélation

$$\text{corr}(f) = \frac{\sum_i f(x_i) \cdot y_i}{\sqrt{\sum_i (f(x_i))^2 \times \sum_i y_i^2}} \propto \sum_i f(x_i) \cdot y_i$$

Approches wrapper

Principe générer/tester

Etant donné une liste de candidats $\mathcal{L} = \{f_1, \dots, f_p\}$

- Générer un candidat F
- Calculer $\mathcal{G}(F)$
 - apprendre h_F à partir de $\mathcal{E}|_F$
 - tester h_F sur un ensemble de test $= \hat{\mathcal{G}}(F)$
- Mettre à jour \mathcal{L} .

Algorithmes

- hill-climbing / multiple restart
- algorithmes génétiques Vafaie-DeJong, IJCAI 95
- (*) programmation génétique & feature construction. Krawiec, GPEH 01

Approches a posteriori

Principe

- Construire des hypothèses
- En déduire les attributs importants
- Eliminer les autres
- Recommencer

Algorithme : SVM Recursive Feature Elimination Guyon et al. 03

- SVM linéaire $\rightarrow h(x) = \text{sign}(\sum w_i \cdot f_i(x) + b)$
- Si $|w_i|$ est petit, f_i n'est pas important
- Eliminer les k attributs ayant un poids min.
- Recommencer.

Limites

Hypothèses linéaires

- Un poids par attribut.

Quantité des exemples

- Les poids des attributs sont liés.
- La dimension du système est liée au nombre d'exemples.

Or le pb de FS se pose souvent quand il n'y a pas assez d'exemples

Some references

- ▶ Filter approaches [1]
- ▶ Wrapper approaches
 - ▶ Tackling combinatorial optimization [2,3,4]
 - ▶ Exploration vs Exploitation dilemma
- ▶ Embedded approaches
 - ▶ Using the learned hypothesis [5,6]
 - ▶ Using a regularization term [7,8]
 - ▶ Restricted to linear models [7] or linear combinations of kernels [8]

- [1] K. Kira, and L. A. Rendell ML'92
- [2] D. Margaritis NIPS'09
- [3] T. Zhang NIPS'08
- [4] M. Boullé J. Mach. Learn. Res. 07
- [5] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik Mach. Learn. 2002
- [6] J. Rogers, and S. R. Gunn SLSFS'05
- [7] R. Tibshirani Journal of the Royal Statistical Society 94
- [8] F. Bach NIPS'08

Feature Selection

Optimization problem

Find $F^* = \operatorname{argmin} \mathbf{Err}(\mathcal{A}, F, \mathcal{E})$

\mathcal{F} : Set of features

F : Feature subset

\mathcal{E} : Training data set

\mathcal{A} : Machine Learning algorithm

\mathbf{Err} : Generalization error

Feature Selection Goals

- ▶ Reduced Generalization Error
- ▶ More cost-effective models
- ▶ More understandable models

Bottlenecks

- ▶ Combinatorial optimization problem: find $F \subseteq \mathcal{F}$
- ▶ Generalization error unknown

FS as A Markov Decision Process

Set of features \mathcal{F}

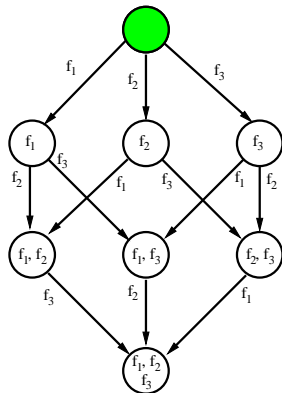
Set of states $\mathcal{S} = 2^{\mathcal{F}}$

Initial state \emptyset

Set of actions $A = \{\text{add } f, f \in \mathcal{F}\}$

Final state any state

Reward function $V : \mathcal{S} \mapsto [0, 1]$



Goal: Find $\underset{F \subseteq \mathcal{F}}{\operatorname{argmin}} \operatorname{Err}(\mathcal{A}(F, D))$

Optimal Policy

Policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$

Final state following a policy F_π

Optimal policy $\pi^* =$

$$\underset{\pi}{\operatorname{argmin}} \mathbf{Err}(\mathcal{A}(F_\pi, \mathcal{E}))$$

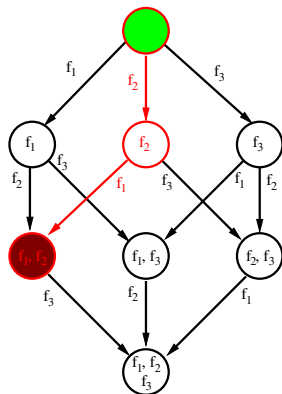
Bellman's optimality principle

$$\pi^*(F) = \underset{f \in \mathcal{F}}{\operatorname{argmin}} V^*(F \cup \{f\})$$

$$V^*(F) = \begin{cases} \mathbf{Err}(\mathcal{A}(F)) & \text{if } \text{final}(F) \\ \min_{f \in \mathcal{F}} V^*(F \cup \{f\}) & \text{otherwise} \end{cases}$$

In practice

- ▶ π^* intractable \Rightarrow approximation using UCT
- ▶ Computing $\mathbf{Err}(F)$ using a fast estimate



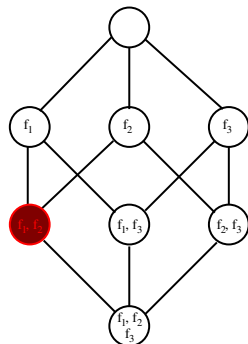
FS as a game

Exploration vs Exploitation tradeoff

- ▶ Virtually explore the whole lattice
- ▶ Gradually focus the search on most promising F s
- ▶ Use a frugal, unbiased assessment of F

How ?

- ▶ Upper Confidence Tree (UCT) [1]
 - ▶ $UCT \subset Monte\text{-}Carlo\ Tree\ Search$
 - ▶ UCT tackles tree-structured optimization problems



[1] L. Kocsis, and C. Szepesvári ECML'06

Apprentissage par Renforcement: Plan du cours

Contexte

Algorithms

Value functions

Optimal policy

Temporal differences and eligibility traces

Q-learning

Playing Go: MoGo

Feature Selection as a Game

Position du problème

Monte-Carlo Tree Search

Feature Selection: the FUSE algorithm

Experimental Validation

Active Learning as a Game

Position du problème

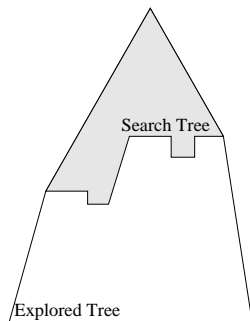
Algorithme BAAL

Validation expérimentale

Constructive Induction

The UCT scheme

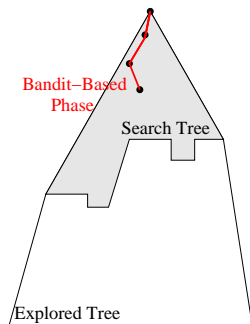
- ▶ Upper Confidence Tree (UCT) [1]
 - ▶ Gradually grow the search tree
 - ▶ Building Blocks
 - ▶ Select next action (bandit-based phase)
 - ▶ Add a node (leaf of the search tree)
 - ▶ Select next action bis (random phase)
 - ▶ Compute instant reward
 - ▶ Update information in visited nodes
 - ▶ Returned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



[1] L. Kocsis, and C. Szepesvári ECML'06

The UCT scheme

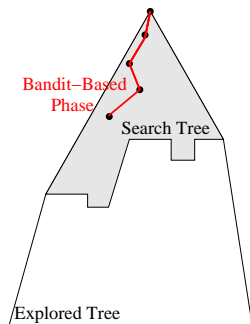
- ▶ Upper Confidence Tree (UCT) [1]
 - ▶ Gradually grow the search tree
 - ▶ Building Blocks
 - ▶ Select next action (bandit-based phase)
 - ▶ Add a node (leaf of the search tree)
 - ▶ Select next action bis (random phase)
 - ▶ Compute instant reward
 - ▶ Update information in visited nodes
 - ▶ Returned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



[1] L. Kocsis, and C. Szepesvári ECML'06

The UCT scheme

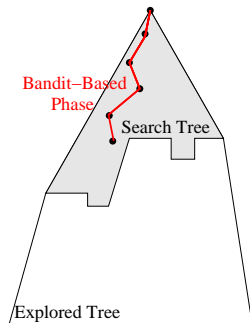
- ▶ Upper Confidence Tree (UCT) [1]
 - ▶ Gradually grow the search tree
 - ▶ Building Blocks
 - ▶ Select next action (bandit-based phase)
 - ▶ Add a node (leaf of the search tree)
 - ▶ Select next action bis (random phase)
 - ▶ Compute instant reward
 - ▶ Update information in visited nodes
 - ▶ Returned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



[1] L. Kocsis, and C. Szepesvári ECML'06

The UCT scheme

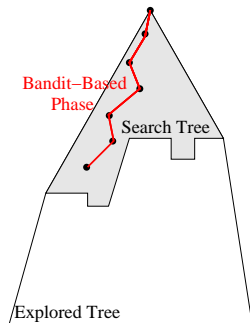
- ▶ Upper Confidence Tree (UCT) [1]
 - ▶ Gradually grow the search tree
 - ▶ Building Blocks
 - ▶ Select next action (bandit-based phase)
 - ▶ Add a node (leaf of the search tree)
 - ▶ Select next action bis (random phase)
 - ▶ Compute instant reward
 - ▶ Update information in visited nodes
 - ▶ Returned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



[1] L. Kocsis, and C. Szepesvári ECML'06

The UCT scheme

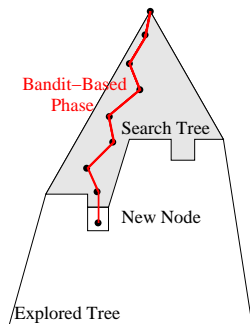
- ▶ Upper Confidence Tree (UCT) [1]
 - ▶ Gradually grow the search tree
 - ▶ Building Blocks
 - ▶ Select next action (bandit-based phase)
 - ▶ Add a node (leaf of the search tree)
 - ▶ Select next action bis (random phase)
 - ▶ Compute instant reward
 - ▶ Update information in visited nodes
 - ▶ Returned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



[1] L. Kocsis, and C. Szepesvári ECML'06

The UCT scheme

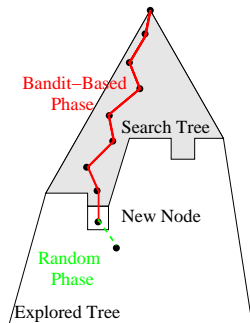
- ▶ Upper Confidence Tree (UCT) [1]
 - ▶ Gradually grow the search tree
 - ▶ Building Blocks
 - ▶ Select next action (bandit-based phase)
 - ▶ Add a node (leaf of the search tree)
 - ▶ Select next action bis (random phase)
 - ▶ Compute instant reward
 - ▶ Update information in visited nodes
 - ▶ Returned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



[1] L. Kocsis, and C. Szepesvári ECML'06

The UCT scheme

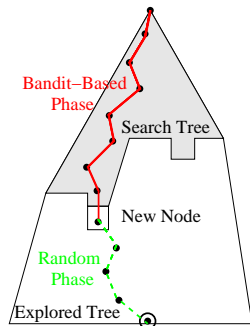
- ▶ Upper Confidence Tree (UCT) [1]
 - ▶ Gradually grow the search tree
 - ▶ Building Blocks
 - ▶ Select next action (bandit-based phase)
 - ▶ Add a node (leaf of the search tree)
 - ▶ Select next action bis (random phase)
 - ▶ Compute instant reward
 - ▶ Update information in visited nodes
 - ▶ Returned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



[1] L. Kocsis, and C. Szepesvári ECML'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



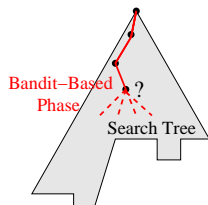
[1] L. Kocsis, and C. Szepesvári ECML'06

Multi-Arm Bandit-based phase

- ▶ Upper Confidence Bound (UCB1-tuned) [1]

- ▶ Select $\underset{a \in A}{\operatorname{argmax}} \hat{\mu}_a + \sqrt{\frac{c_e \log(T)}{n_a} \min\left(\frac{1}{4}, \hat{\sigma}_a^2 + \sqrt{\frac{c_e \log(T)}{t_a}}\right)}$

- ▶ 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



[1] P. Auer, N. Cesa-Bianchi, and P. Fischer ML'02

Apprentissage par Renforcement: Plan du cours

Contexte

Algorithms

Value functions

Optimal policy

Temporal differences and eligibility traces

Q-learning

Playing Go: MoGo

Feature Selection as a Game

Position du problème

Monte-Carlo Tree Search

Feature Selection: the FUSE algorithm

Experimental Validation

Active Learning as a Game

Position du problème

Algorithme BAAL

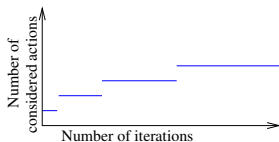
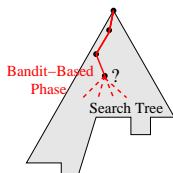
Validation expérimentale

Constructive Induction

FUSE: bandit-based phase

The many arms problem

- ▶ Bottleneck
 - ▶ A many-armed problem (hundreds of features)
 - ⇒ need to guide UCT
- ▶ How to control the number of arms?
 - ▶ **Continuous heuristics [1]**
 - ▶ Use a small exploration constant c_e
 - ▶ **Discrete heuristics [2,3]: Progressive Widening**
 - ▶ Consider only $\lfloor T^b \rfloor$ actions ($b < 1$)



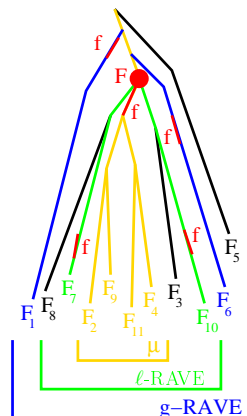
- [1] S. Gelly, and D. Silver ICML'07
- [2] R. Coulom Computer and Games 2006
- [3] P. Rolet, M. Sebag, and O. Teytaud ECML'09

FUSE: bandit-based phase

Sharing information among nodes

- ▶ How to share information among nodes?
 - ▶ Rapid Action Value Estimation (RAVE) [1]

$\text{RAVE}(f) = \text{average reward when } f \in F$

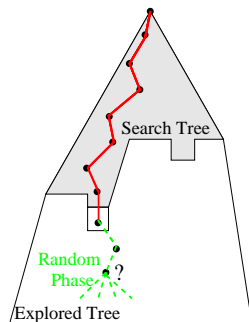


[1] S. Gelly, and D. Silver ICML'07

FUSE: random phase

Dealing with an unknown horizon

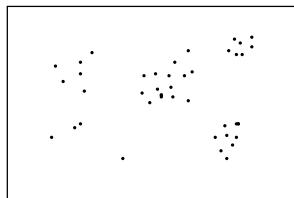
- ▶ Bottleneck
 - ▶ Finite unknown horizon
- ▶ Random phase policy
 - ↪ With probability $1 - q^{|F|}$ stop
 - | Else • add a uniformly selected feature
 - | • $|F| = |F| + 1$
 - | Iterate



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 ($m \ll n$)



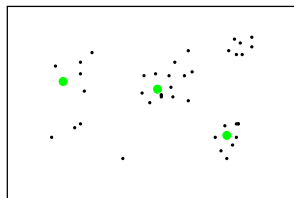
(*) Mann Whitney Wilcoxon test:

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

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 ($m \ll n$)



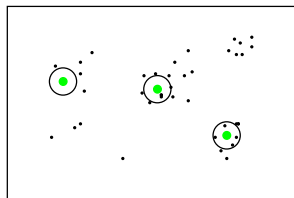
(*) Mann Whitney Wilcoxon test:

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

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 ($m \ll n$)



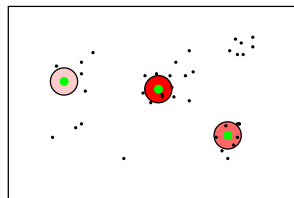
(*) Mann Whitney Wilcoxon test:

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

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 ($m \ll n$)



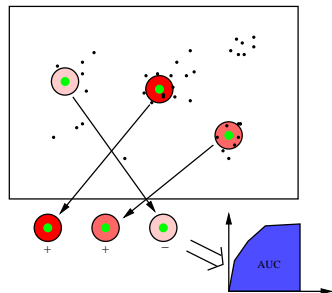
(*) Mann Whitney Wilcoxon test:

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

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 ($m \ll n$)

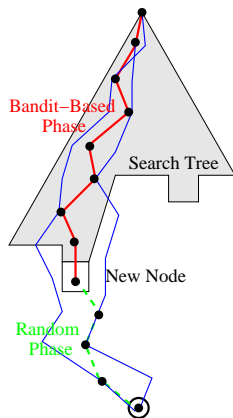


(*) Mann Whitney Wilcoxon test:

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

FUSE: update

- ▶ Explore a graph
 - ⇒ Several paths to the same node
- ▶ Update only current path



The FUSE algorithm

- ▶ N iterations:
each iteration i) follows a path; ii) evaluates a final node
- ▶ Result:

Search tree (most visited path) \longleftrightarrow RAVE score



Wrapper approach

FUSE



Filter approach

FUSE^R

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

Apprentissage par Renforcement: Plan du cours

Contexte

Algorithms

Value functions

Optimal policy

Temporal differences and eligibility traces

Q-learning

Playing Go: MoGo

Feature Selection as a Game

Position du problème

Monte-Carlo Tree Search

Feature Selection: the FUSE algorithm

Experimental Validation

Active Learning as a Game

Position du problème

Algorithme BAAL

Validation expérimentale

Constructive Induction

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

DATA SET	SAMPLES	FEATURES	PROPERTIES
MADOLON [1]	2,600	500	XOR-LIKE
ARCENE [1]	200	10,000	REDUNDANT FEATURES
COLON	62	2,000	“EASY”

[1] NIPS'03

Experimental setting

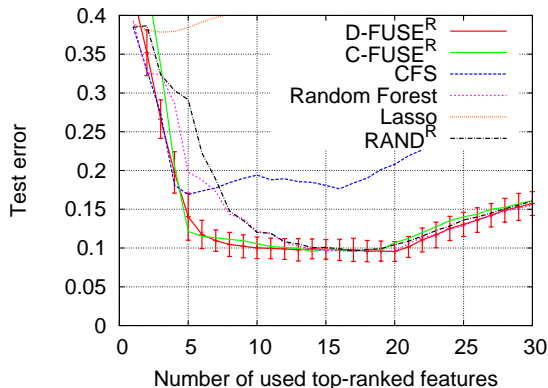
- ▶ Baselines
 - ▶ CFS (Constraint-based Feature Selection) [1]
 - ▶ Random Forest [2]
 - ▶ Lasso [3]
 - ▶ RAND^R : RAVE obtained by selecting 20 random features at each iteration
- ▶ Results averaged on 50 splits (10×5 fold cross-validation)
- ▶ End learner
 - ▶ Hyper-parameters optimized by 5 fold cross-validation

[1] M. A. Hall ICML'00

[2] J. Rogers, and S. R. Gunn SLSFS'05

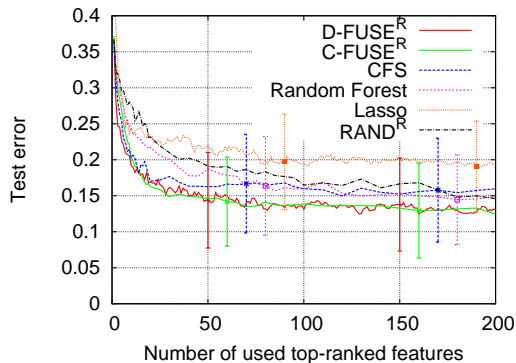
[3] R. Tibshirani Journal of the Royal Statistical Society 94

Results on Madelon after 200,000 iterations



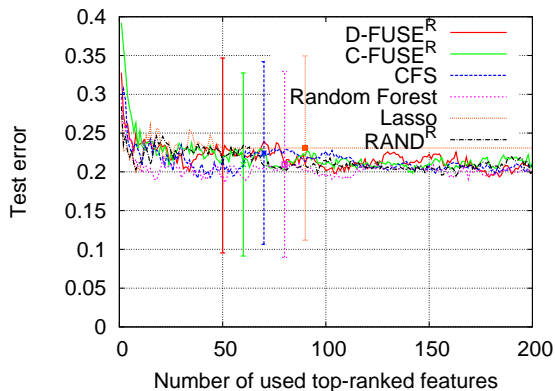
- ▶ **Remark:** FUSE^R = best of both worlds
 - ▶ Removes redundancy (like CFS)
 - ▶ Keeps conditionally relevant features (like Random Forest)

Results on Arcene after 200,000 iterations



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

Results on Colon after 200,000 iterations



- ▶ Remark
 - ▶ All equivalent

NIPS 2003 Feature Selection challenge

- ▶ Test error on a disjoint test set

DATABASE	ALGORITHM	CHALLENGE ERROR	SUBMITTED FEATURES	IRRELEVANT FEATURES
MADELON	FSPP2 [1]	6.22% (1 st)	12	0
	D-FUSE ^R	6.50% (24 th)	18	0
	BAYES-NN-RED [2]	7.20% (1 st)	100	0
ARCENE	D-FUSE ^R (ON ALL)	8.42% (3 rd)	500	34
	D-FUSE ^R	9.42% 500 (8 th)	500	0

[1] K. Q. Shen, C. J. Ong, X. P. Li, E. P. V. Wilder-Smith Mach. Learn. 2008

[2] R. M. Neal, and J. Zhang Feature extraction, foundations and applications, Springer 2006

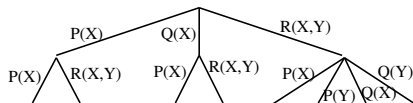
Conclusion

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

- ▶ Other end learners
- ▶ Revisit the reward see (Hand 2010) about AUC
- ▶ Extend to Feature construction along [1]



[1] F. de Mesmay, A. Rimmel, Y. Voronenko, and M. Püschel ICML'09

Apprentissage par Renforcement: Plan du cours

Contexte

Algorithms

Value functions

Optimal policy

Temporal differences and eligibility traces

Q-learning

Playing Go: MoGo

Feature Selection as a Game

Position du problème

Monte-Carlo Tree Search

Feature Selection: the FUSE algorithm

Experimental Validation

Active Learning as a Game

Position du problème

Algorithme BAAL

Validation expérimentale

Constructive Induction

Active Learning, position of the problem

Supervised learning, the setting

- ▶ Target hypothesis h^*
- ▶ Training set $\mathcal{E} = \{(x_i, y_i), i = 1 \dots n\}$
- ▶ Learn h_n from \mathcal{E}

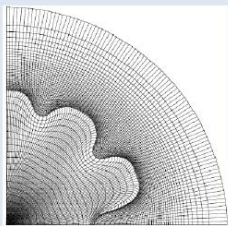
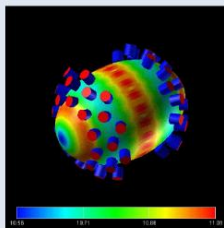
Criteria

- ▶ Consistency: $h_n \rightarrow h^*$ when $n \rightarrow \infty$.
- ▶ Sample complexity: number of examples needed to reach the target with precision ϵ

$$\epsilon \rightarrow n_\epsilon \text{ s.t. } \|h_n - h^*\| < \epsilon$$

Motivations

- Given x , obtaining $h^*(x)$ is costly
- Goal: reduce sample complexity while keeping generalization error low
- Motivating application: numerical engineering



=> Learn simplified models with only ~ 100 examples

Active Learning, definition

Passive learning

iid examples

$$\mathcal{E} = \{(x_i, y_i), i = 1 \dots n\}$$

Active learning

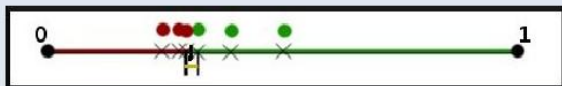
x_{n+1} selected depending on $\{(x_i, y_i), i = 1 \dots n\}$

In the best case, exponential improvement:

PASSIVE:



ACTIVE:



State of the art

Let H be the hypothesis space.

Realizable assumption: $h^* \in H$

Then, exponential improvements. Freund et al. 1997; Dasgupta 2005; Balcan et al. 2010.

Noisy case: improvement depends on noise model

Balcan et al. 2006; Hanneke 2007; Dasgupta et al. 2008.

Realizable batch case

PhD Philippe Rolet, 23 dec. 2010.

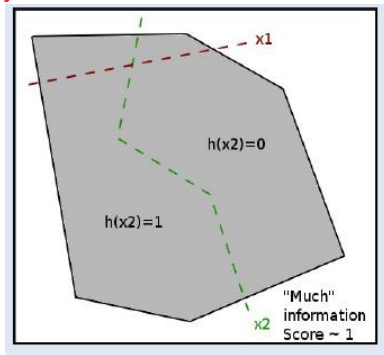
How it works

Principle

- ▶ Design a measure of the information brought by an instance
- ▶ Iteratively select the best instance

Example: query by committee

Seung et al. 92



Active Learning

Optimization problem

- ▶ T : time horizon (number of instances to select)
- ▶ States $s_t = \{(x_i, h^*(x_i)), i = 1 \dots t\}$
- ▶ Action: select x_{t+1}
- ▶ \mathcal{A} : Machine Learning algorithm
- ▶ **Err**: Generalization error

Find Sampling strategy S minimizing $\mathbb{E}\mathbf{Err}(\mathcal{A}(S_T(h^*), h^*))$

Bottlenecks

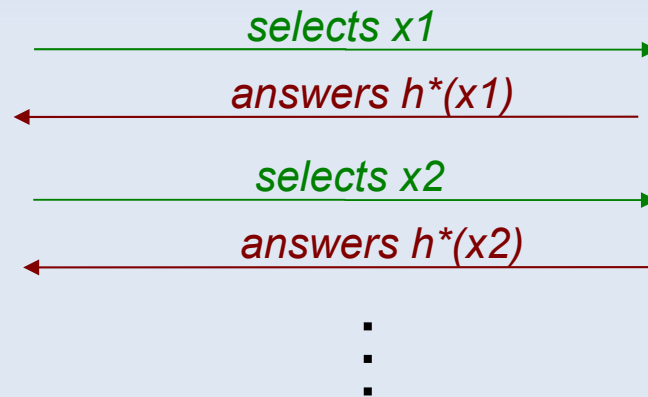
- ▶ Combinatorial optimization problem - in a continuous space
- ▶ Generalization error unknown

Optimal Strategy for AL

- Learning algorithm \mathcal{A}
- Finite Horizon T
- Sampling strategy S_T
- Target concept h^*



Learner \mathcal{A}



T-size training set $S_T(h^*)$
 $\{(x_1, h^*(x_1)), \dots, (x_T, h^*(x_T))\}$



Target Concept h^*
(a.k.a. Oracle)

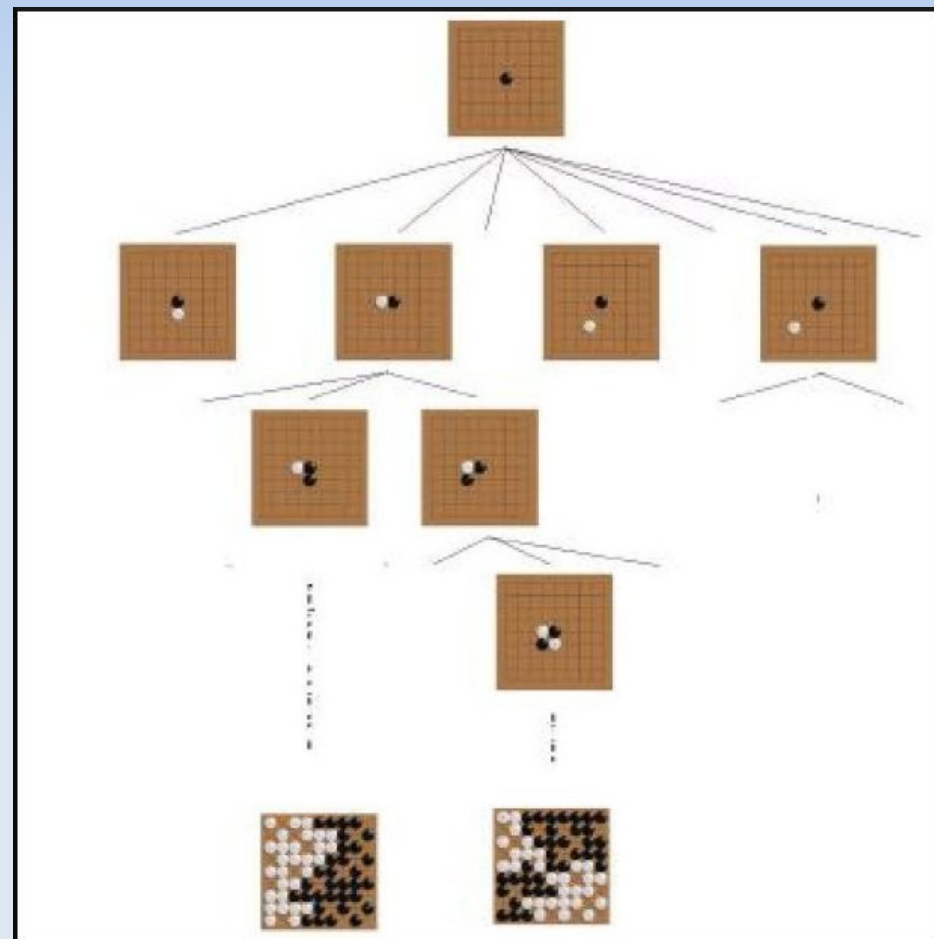
- **Goal:** $\operatorname{argmin} E[\operatorname{Err}(\mathcal{A}(S_T(h^*)), h^*)]$

Optimal Strategy for AL

- AL modeled as a Markov decision process:
 - **State space:** all possible training sets of size $\leq T$
 - **Action space:** instances x available for query
 - **Transition function:** $P(s_{t+1} | s_t, x)$
 - **Reward function:** gen. err. $Err(\mathcal{A}(S_T(h)), h)$
- Optimal policy $\pi^* \rightarrow$ Optimal AL strategy

Active Learning: a 1-Player Game

- Bottlenecks:
 - Large state space
 - Large action space
 - Cannot use h^* directly
- Approx. sol. inspired from Go: AL as a game
 - Coulom 06, Chaslot et al. 06,
Gelly&Sliver 07
- Browse game tree
- Estimate move values with *Monte-Carlo* simulations



Apprentissage par Renforcement: Plan du cours

Contexte

Algorithms

Value functions

Optimal policy

Temporal differences and eligibility traces

Q-learning

Playing Go: MoGo

Feature Selection as a Game

Position du problème

Monte-Carlo Tree Search

Feature Selection: the FUSE algorithm

Experimental Validation

Active Learning as a Game

Position du problème

Algorithme BAAL

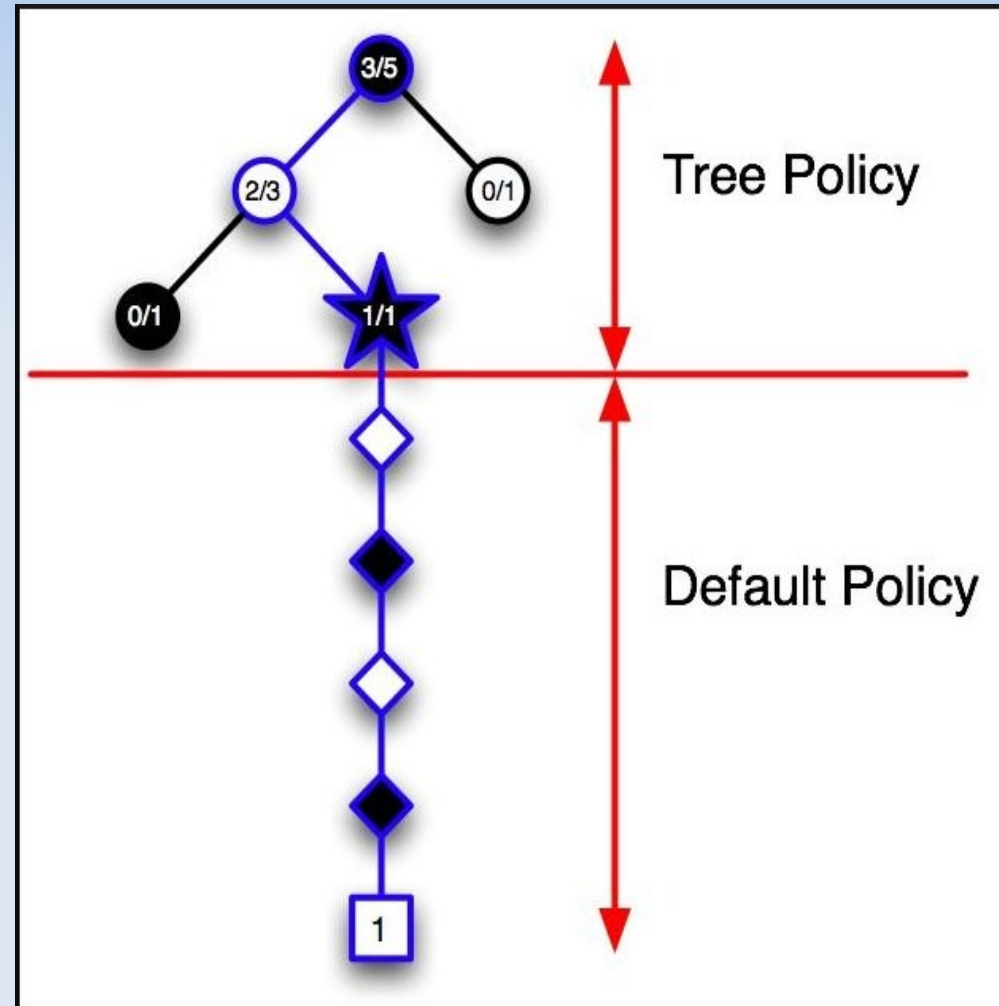
Validation expérimentale

Constructive Induction

The BAAL Algorithm

=> Bandit-based
Active Learner

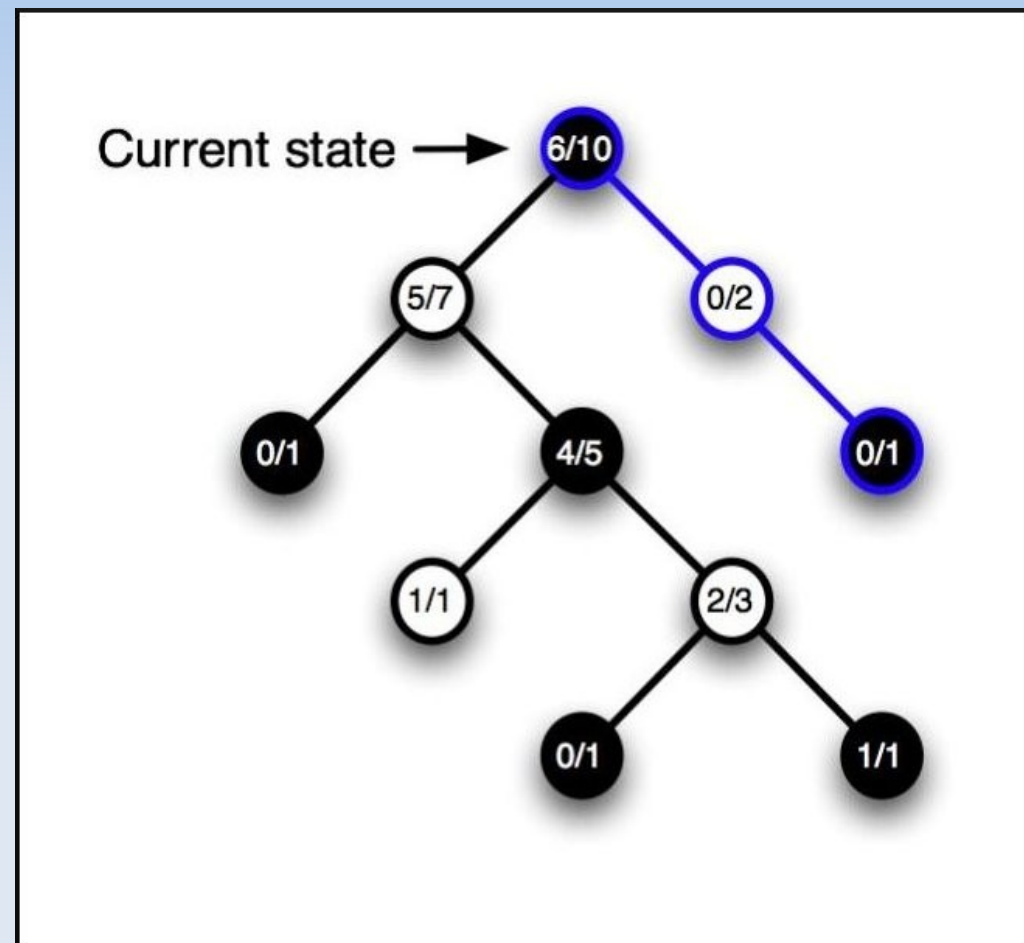
- Simulation planning with Multi-armed bandits
- Asymmetric tree growth
More exploration for promising moves



BAAL: Exploration v. Exploitation

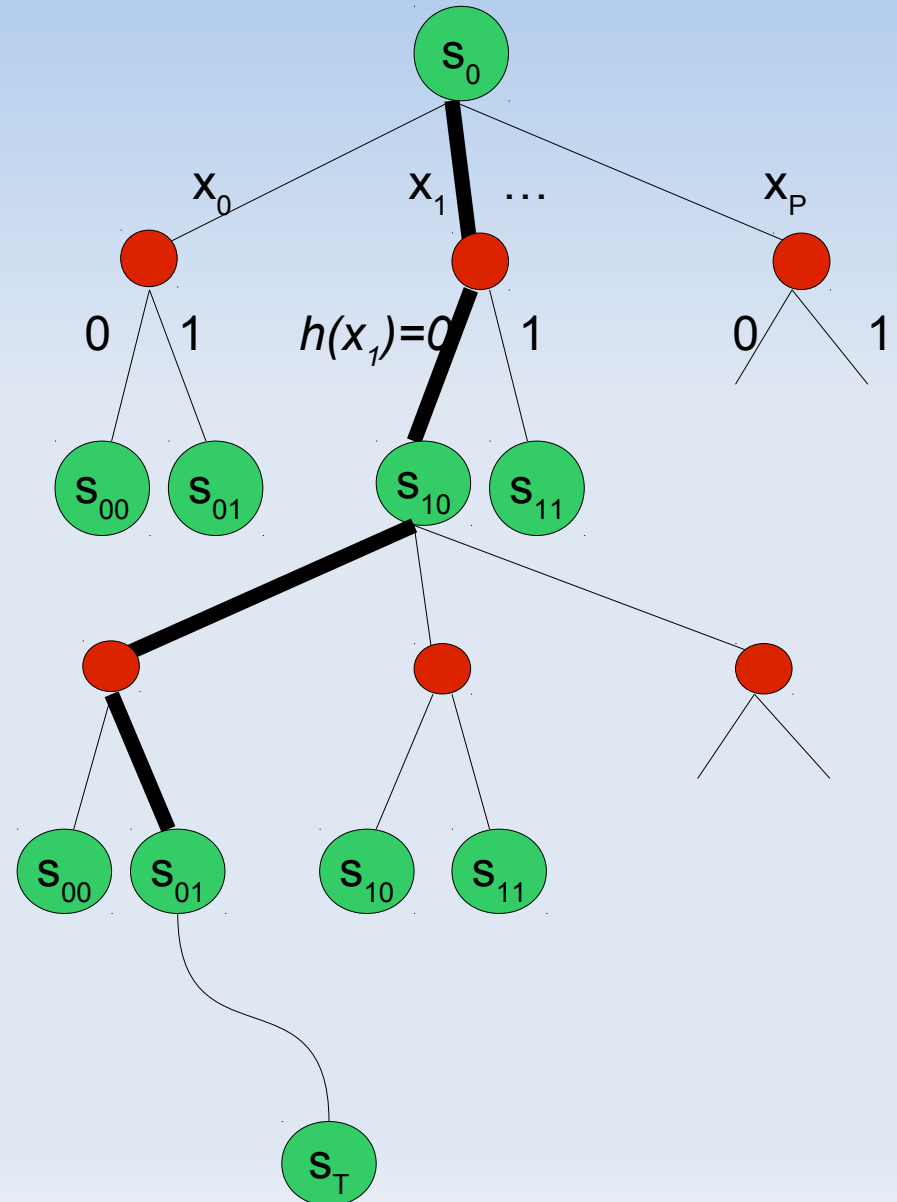
- UCB: balance exploration and exploitation
Auer, 2002
- UCT = UCB for trees
Kocsis&Szepesvari, 2006

$$\hat{\mu}_i + C \sqrt{\frac{\log(\sum_j n_j)}{n_i}}$$



BAAL: Outline

```
BAAL( $P_H, s_0, T, N$ )  
for  $i=1$  to  $N$  do  
   $h = \text{DrawSurrogateHypothesis}(s_0)$   
  Tree-Walk( $s_0, T, h$ )  
end for  
Return  $x = \arg \max_{x' \in \mathcal{X}} \{n(s \cup \{x'\})\}$   
  
Tree-Walk( $s, t, h$ )  
Increment  $n(s)$   
if  $t > 0$  then  
   $\mathcal{X}(s) = \text{ArmSet}(s, n(s))$   
  Select  $x^* = \text{UCB}(s, \mathcal{X}(s))$   
  Get label  $h(x^*)$  from surrogate  
   $r = \text{Tree-Walk}(s \cup \{(x^*, h(x^*))\}, t - 1, h)$   
else  
  Compute  $r = \text{Err}(\mathcal{A}(s), h)$   
end if  
 $r(s) \leftarrow (1 - \frac{1}{n(s)})r(s) + \frac{1}{n(s)} r$   
Return  $r$ 
```



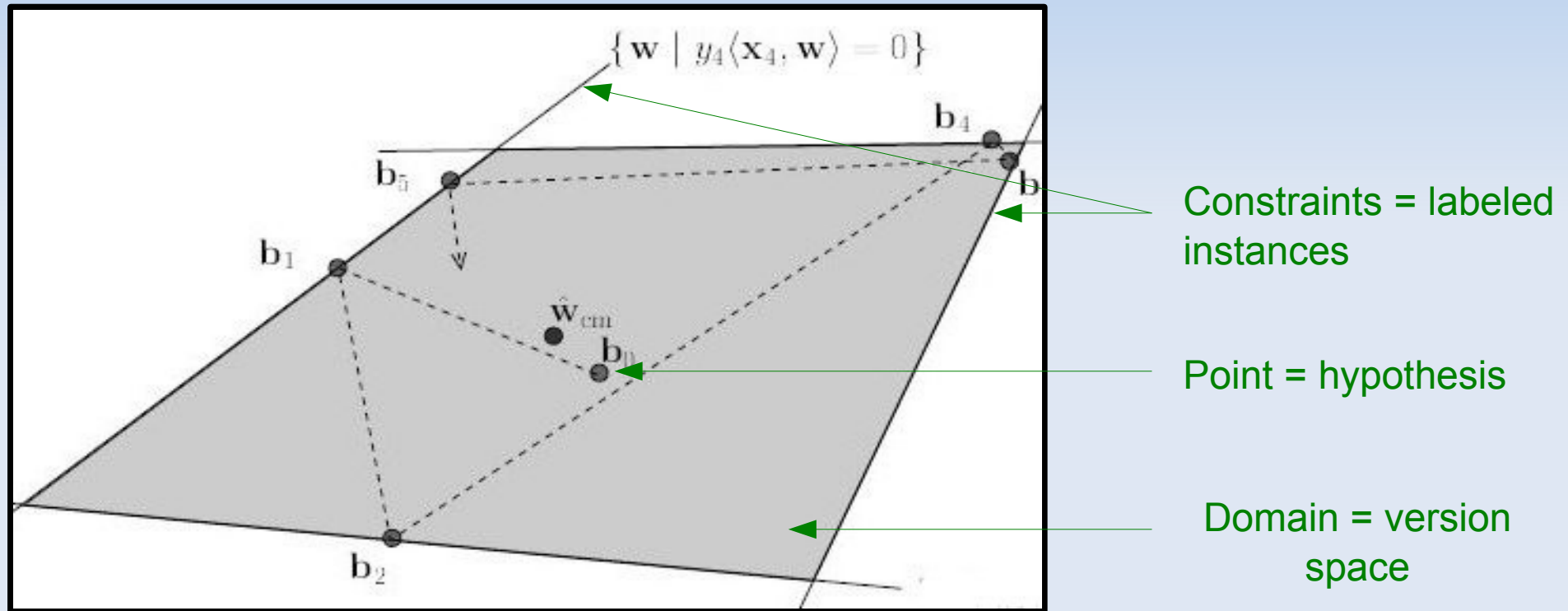
Baal: Continuous action space

- UCB is designed for finite action spaces
- AL: action space = \mathbb{R}^D
- Control the number of arms: Coulom, 2007
progressive widening Wang, Audibert, Munos, 2008
instances \sim ($\#$ visits) $^{1/4}$
- Select new instances
 - In a random order
 - Following a given heuristic (e.g. QbC heuristic)

Baal: draw surrogate hypotheses

- *Billiard* algorithms

Rujan 97,
Comets et. al. 09, ...



- **Sound:** provably converges to uniform draw
- **Scalable** w.r.t. dimension, # constraints

Apprentissage par Renforcement: Plan du cours

Contexte

Algorithms

Value functions

Optimal policy

Temporal differences and eligibility traces

Q-learning

Playing Go: MoGo

Feature Selection as a Game

Position du problème

Monte-Carlo Tree Search

Feature Selection: the FUSE algorithm

Experimental Validation

Active Learning as a Game

Position du problème

Algorithme BAAL

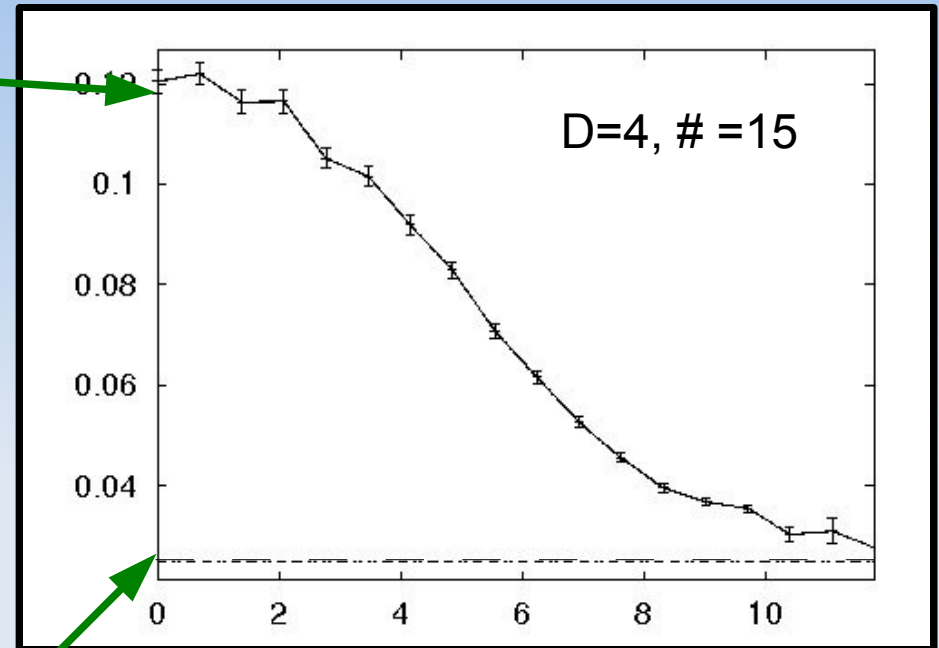
Validation expérimentale

Constructive Induction

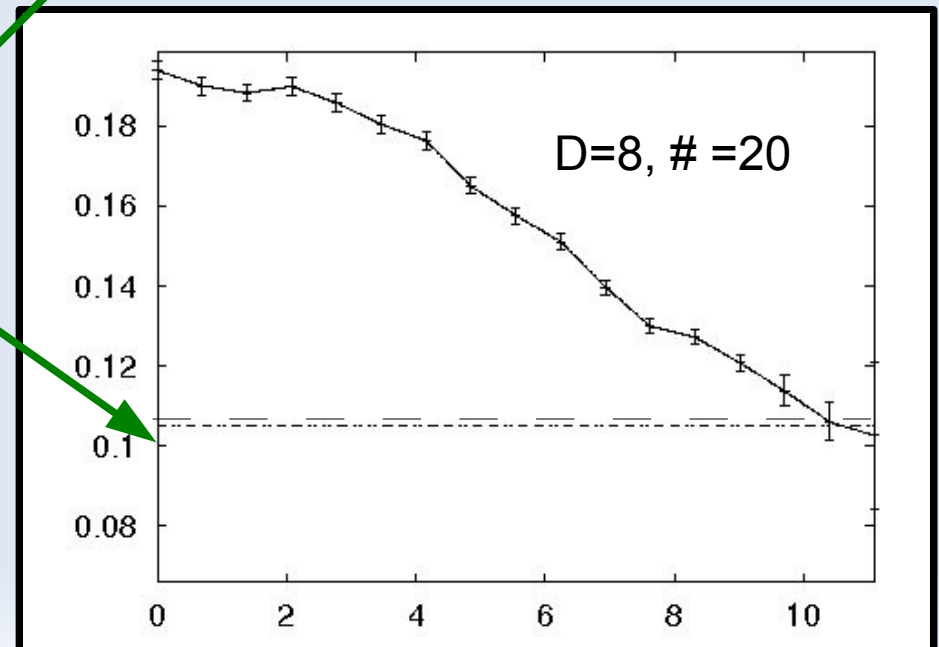
Some results

- Setting:
 - Linear sep. of \mathbb{R}^D
 - Dimension : 4, 8
 - # queries: 15, 20
- X-axis: $\log(\# \text{ sims})$
- Y-axis: Gen. Error

Passive learning



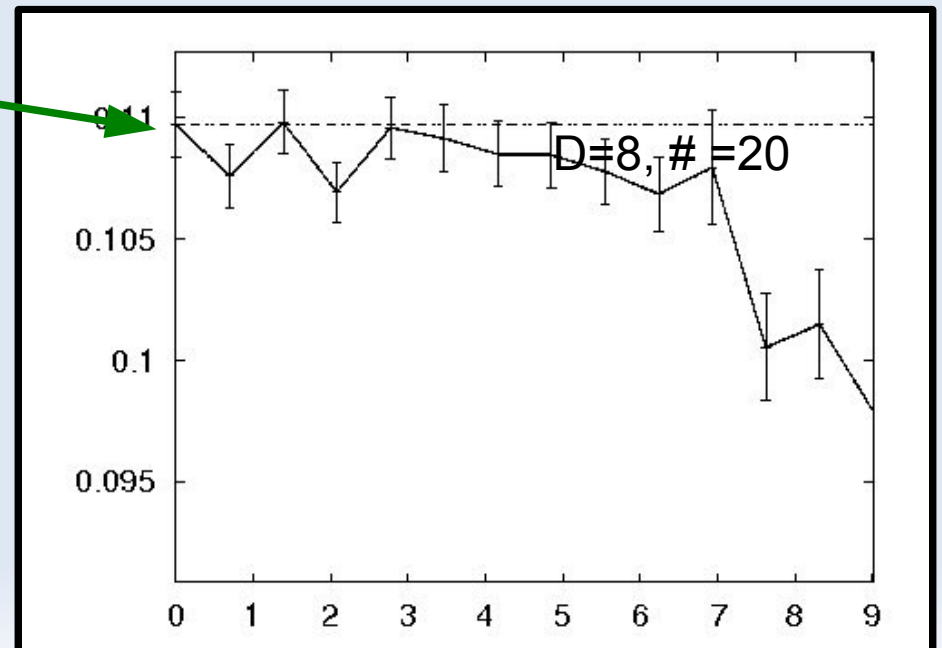
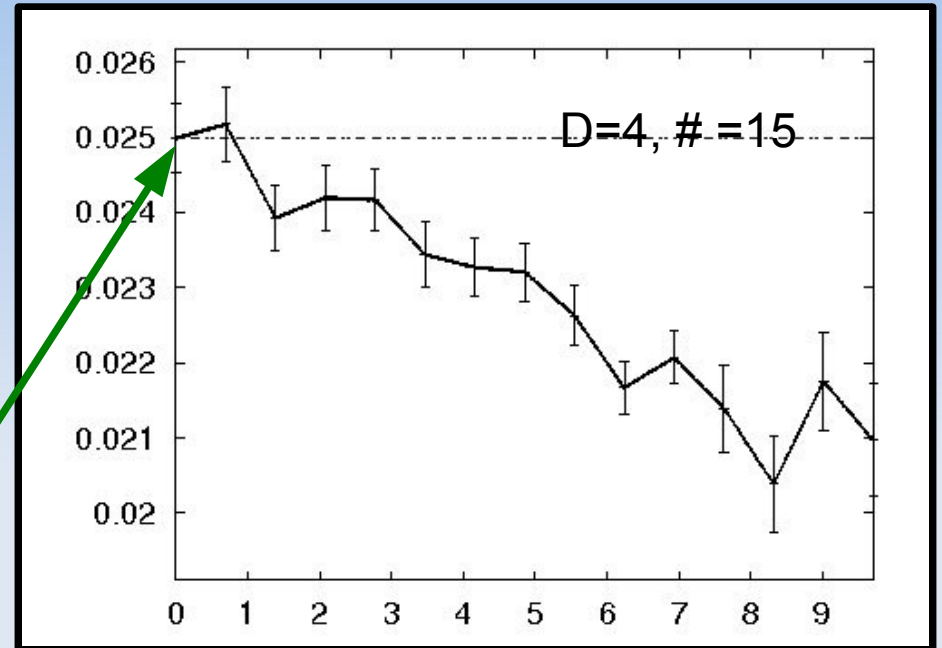
Almost optimal AL
(QbC-based)



Some results

- Combining with AL criteria (inspired from QbC)
- Best of both worlds!

Almost optimal AL
(QbC-based)

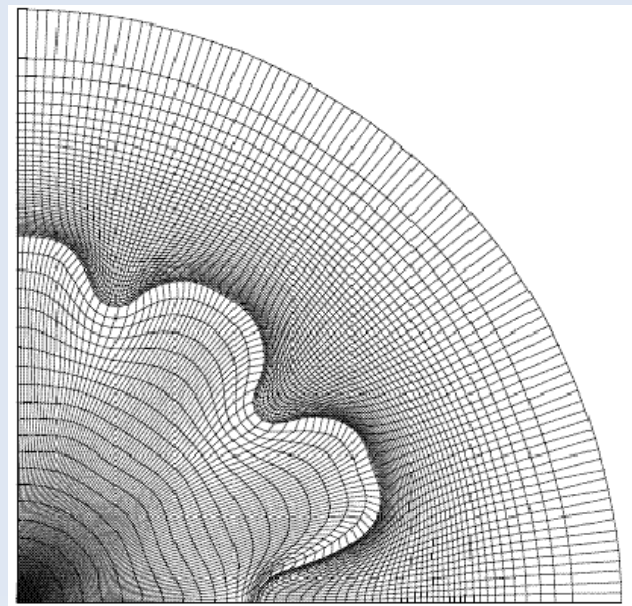


Partial Conclusion on BAAL

- A new approach to AL: *AL as a Game*
- Boosts heuristic to optimal strategy (*provably*)
- *Anytime* algorithm
- Straightforward extension to Optimization

Rolet, Sebag, Teytaud, 2009b

- **Perspectives:**
 - Kernelized Baal
 - Numerical engineering application



Apprentissage par Renforcement: Plan du cours

Contexte

Algorithms

Value functions

Optimal policy

Temporal differences and eligibility traces

Q-learning

Playing Go: MoGo

Feature Selection as a Game

Position du problème

Monte-Carlo Tree Search

Feature Selection: the FUSE algorithm

Experimental Validation

Active Learning as a Game

Position du problème

Algorithme BAAL

Validation expérimentale

Constructive Induction

KDD 2009 – Orange

Targets

1. Churn
2. Appetency
3. Up-selling

Core Techniques

1. Feature Selection
2. Bounded Resources
3. Parameterless methods

