

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

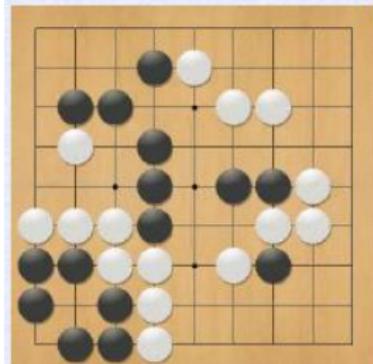
Validation expérimentale

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

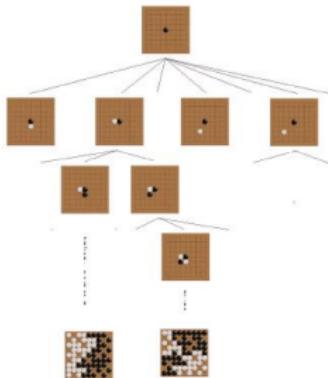
Brügman (1993)

1. While possible, add a stone (white, black)
2. Compute $\text{Win}(\text{black})$
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



Multi-Armed Bandits

Lai, Robbins 1985

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

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

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*

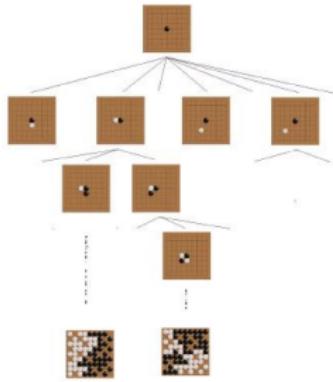
Decision: Optimism in front of unknown!

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

Variants

- ▶ Take into account standard deviation of $\hat{\mu}$
- ▶ 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

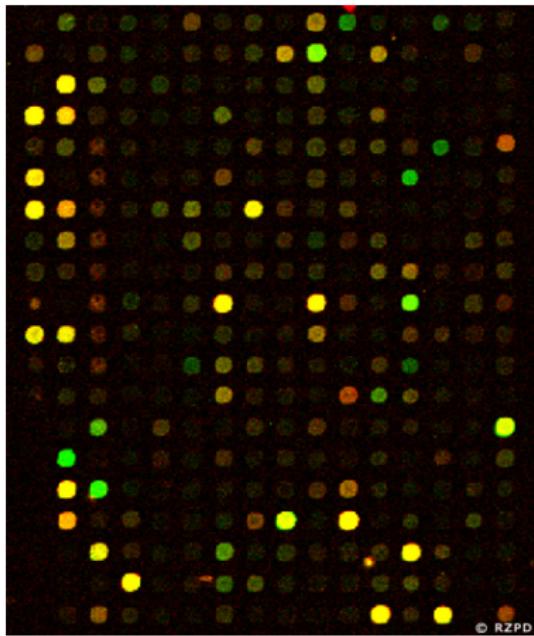
Algorithme BAAL

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 $\mathcal{F} = \{f_1, \dots, f_d\}$. Soit la fonction :

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

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

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 bactracker : 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
 - Mettre à jour \mathcal{L} .
- $$= \hat{\mathcal{G}}(F)$$

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

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}

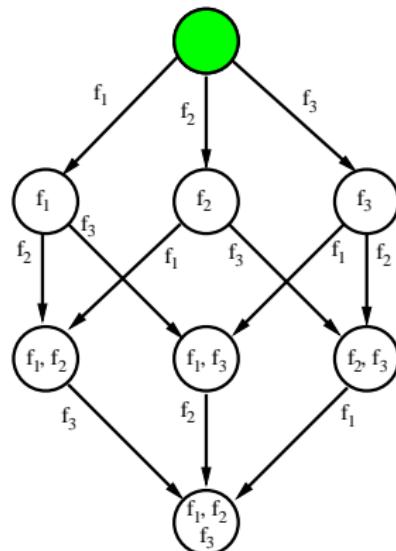
Set of states $\mathcal{S} = 2^{\mathcal{F}}$

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

Final state following a policy F_π

Optimal policy $\pi^* =$

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

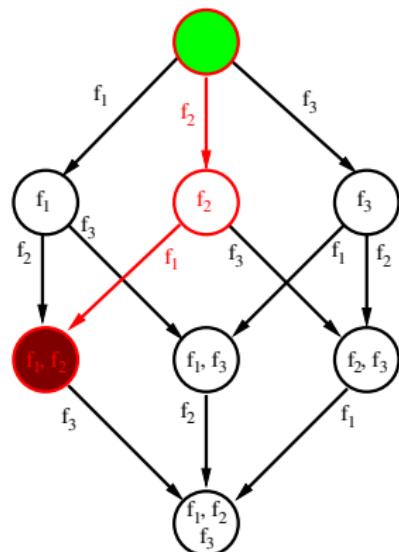
Bellman's optimality principle

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

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

In practice

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



FS as a game

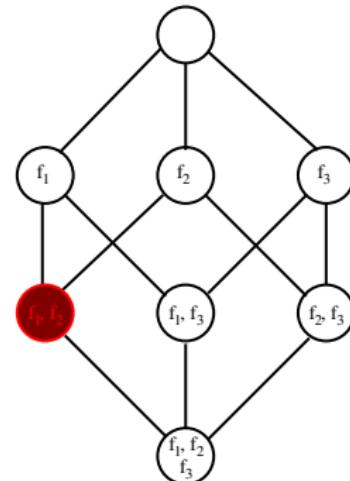
Exploration vs Exploitation tradeoff

- ▶ 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]
 - ▶ $\text{UCT} \subset \text{Monte-Carlo Tree Search}$
 - ▶ 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

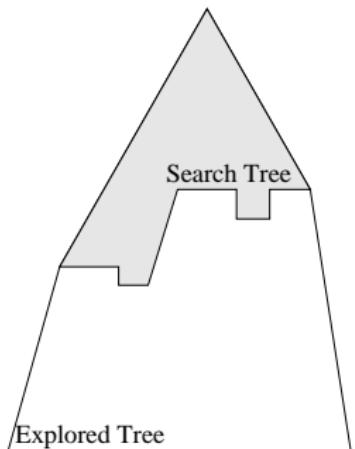
Algorithme BAAL

Validation expérimentale

Constructive Induction

The UCT scheme

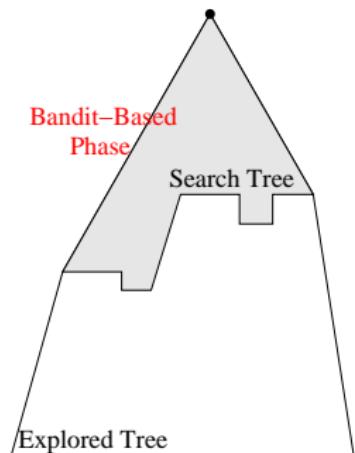
- ▶ Upper Confidence Tree (UCT) [1]
 - ▶ Gradually grow the search tree
 - ▶ Building Blocks
 - ▶ Select next action (bandit-based phase)
 - ▶ Add a node (leaf of the search tree)
 - ▶ Select next action bis (random phase)
 - ▶ Compute instant reward
 - ▶ Update information in visited nodes
 - ▶ Returned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



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

The UCT scheme

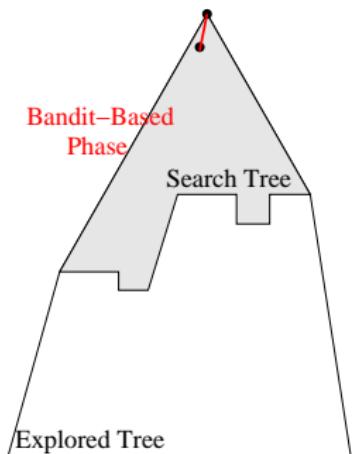
- ▶ Upper Confidence Tree (UCT) [1]
 - ▶ Gradually grow the search tree
 - ▶ Building Blocks
 - ▶ Select next action (bandit-based phase)
 - ▶ Add a node (leaf of the search tree)
 - ▶ Select next action bis (random phase)
 - ▶ Compute instant reward
 - ▶ Update information in visited nodes
 - ▶ Returned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



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

The UCT scheme

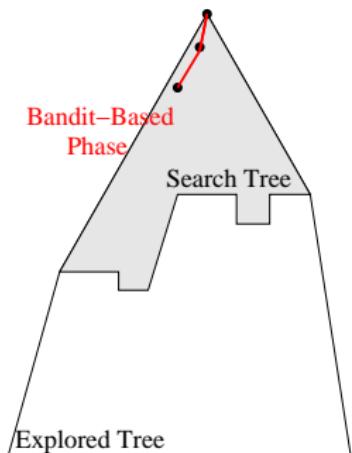
- ▶ Upper Confidence Tree (UCT) [1]
 - ▶ Gradually grow the search tree
 - ▶ Building Blocks
 - ▶ Select next action (bandit-based phase)
 - ▶ Add a node (leaf of the search tree)
 - ▶ Select next action bis (random phase)
 - ▶ Compute instant reward
 - ▶ Update information in visited nodes
 - ▶ Returned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



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

The UCT scheme

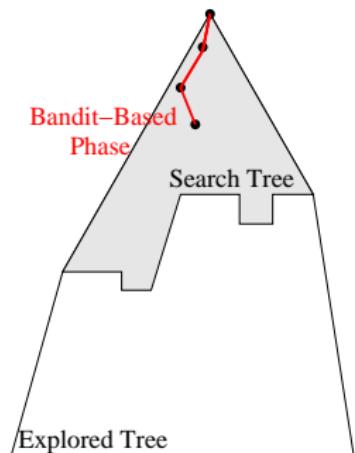
- ▶ Upper Confidence Tree (UCT) [1]
 - ▶ Gradually grow the search tree
 - ▶ Building Blocks
 - ▶ Select next action (bandit-based phase)
 - ▶ Add a node (leaf of the search tree)
 - ▶ Select next action bis (random phase)
 - ▶ Compute instant reward
 - ▶ Update information in visited nodes
 - ▶ Returned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



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

The UCT scheme

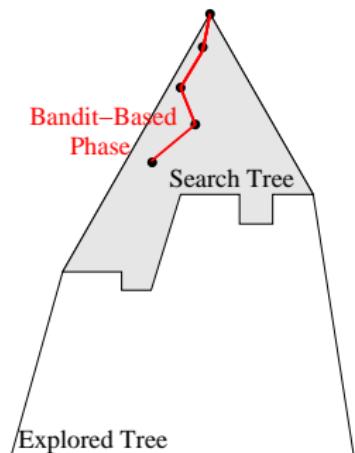
- ▶ Upper Confidence Tree (UCT) [1]
 - ▶ Gradually grow the search tree
 - ▶ Building Blocks
 - ▶ Select next action (bandit-based phase)
 - ▶ Add a node (leaf of the search tree)
 - ▶ Select next action bis (random phase)
 - ▶ Compute instant reward
 - ▶ Update information in visited nodes
 - ▶ Returned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



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

The UCT scheme

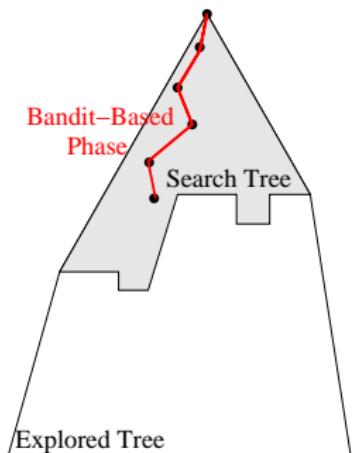
- ▶ Upper Confidence Tree (UCT) [1]
 - ▶ Gradually grow the search tree
 - ▶ Building Blocks
 - ▶ Select next action (bandit-based phase)
 - ▶ Add a node (leaf of the search tree)
 - ▶ Select next action bis (random phase)
 - ▶ Compute instant reward
 - ▶ Update information in visited nodes
 - ▶ Returned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



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

The UCT scheme

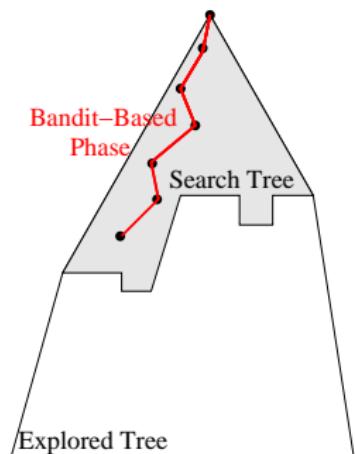
- ▶ Upper Confidence Tree (UCT) [1]
 - ▶ Gradually grow the search tree
 - ▶ Building Blocks
 - ▶ Select next action (bandit-based phase)
 - ▶ Add a node (leaf of the search tree)
 - ▶ Select next action bis (random phase)
 - ▶ Compute instant reward
 - ▶ Update information in visited nodes
 - ▶ Returned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



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

The UCT scheme

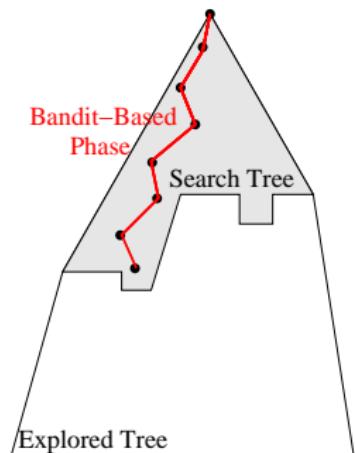
- ▶ Upper Confidence Tree (UCT) [1]
 - ▶ Gradually grow the search tree
 - ▶ Building Blocks
 - ▶ Select next action (bandit-based phase)
 - ▶ Add a node (leaf of the search tree)
 - ▶ Select next action bis (random phase)
 - ▶ Compute instant reward
 - ▶ Update information in visited nodes
 - ▶ Returned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



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

The UCT scheme

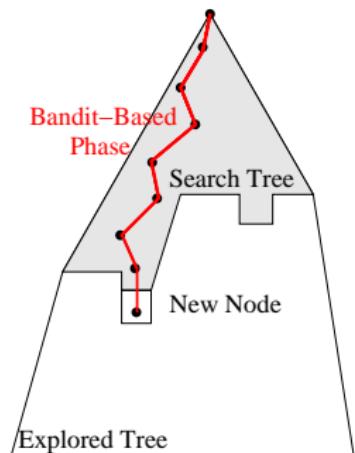
- ▶ Upper Confidence Tree (UCT) [1]
 - ▶ Gradually grow the search tree
 - ▶ Building Blocks
 - ▶ Select next action (bandit-based phase)
 - ▶ Add a node (leaf of the search tree)
 - ▶ Select next action bis (random phase)
 - ▶ Compute instant reward
 - ▶ Update information in visited nodes
 - ▶ Returned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



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

The UCT scheme

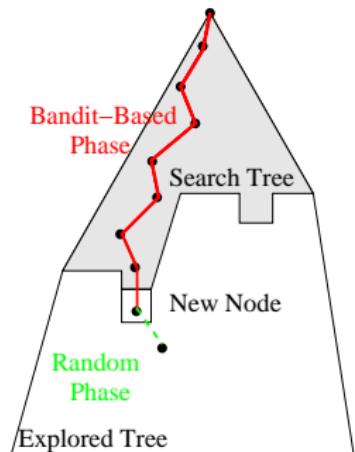
- ▶ Upper Confidence Tree (UCT) [1]
 - ▶ Gradually grow the search tree
 - ▶ Building Blocks
 - ▶ Select next action (bandit-based phase)
 - ▶ Add a node (leaf of the search tree)
 - ▶ Select next action bis (random phase)
 - ▶ Compute instant reward
 - ▶ Update information in visited nodes
 - ▶ Returned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



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

The UCT scheme

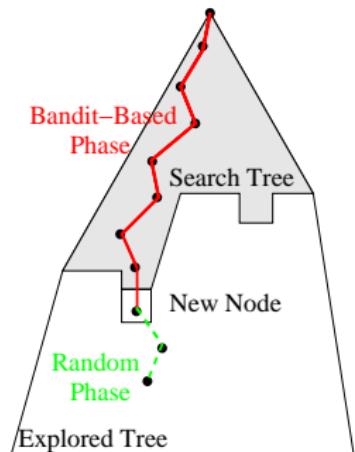
- ▶ Upper Confidence Tree (UCT) [1]
 - ▶ Gradually grow the search tree
 - ▶ Building Blocks
 - ▶ Select next action (bandit-based phase)
 - ▶ Add a node (leaf of the search tree)
 - ▶ Select next action bis (random phase)
 - ▶ Compute instant reward
 - ▶ Update information in visited nodes
 - ▶ Returned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



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

The UCT scheme

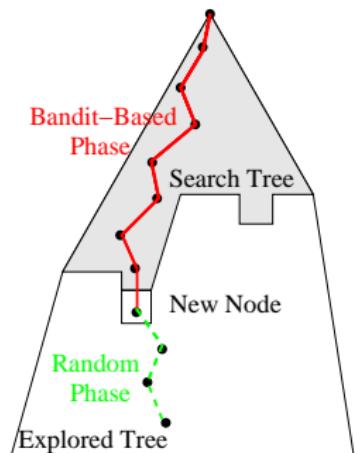
- ▶ Upper Confidence Tree (UCT) [1]
 - ▶ Gradually grow the search tree
 - ▶ Building Blocks
 - ▶ Select next action (bandit-based phase)
 - ▶ Add a node (leaf of the search tree)
 - ▶ Select next action bis (random phase)
 - ▶ Compute instant reward
 - ▶ Update information in visited nodes
 - ▶ Returned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



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

The UCT scheme

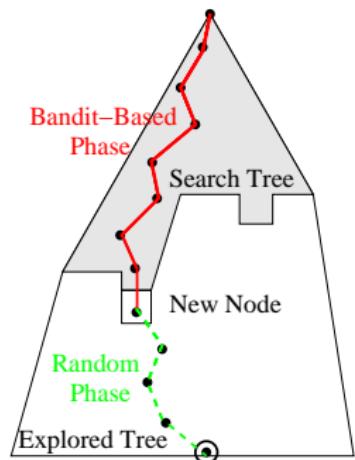
- ▶ Upper Confidence Tree (UCT) [1]
 - ▶ Gradually grow the search tree
 - ▶ Building Blocks
 - ▶ Select next action (bandit-based phase)
 - ▶ Add a node (leaf of the search tree)
 - ▶ Select next action bis (random phase)
 - ▶ Compute instant reward
 - ▶ Update information in visited nodes
 - ▶ Returned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



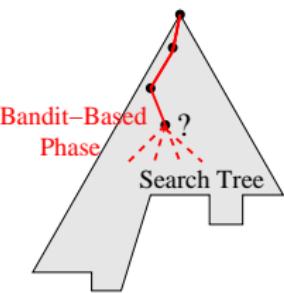
[1] L. Kocsis, and C. Szepesvári ECML'06

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

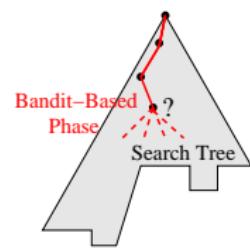
Validation expérimentale

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

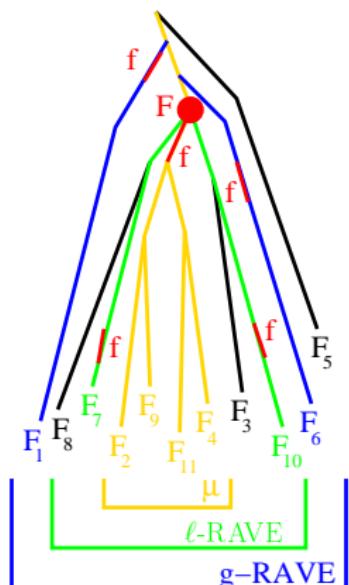
[3] P. Rolet, M. Sebag, and O. Teytaud ECML'09

FUSE: bandit-based phase

Sharing information among nodes

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

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

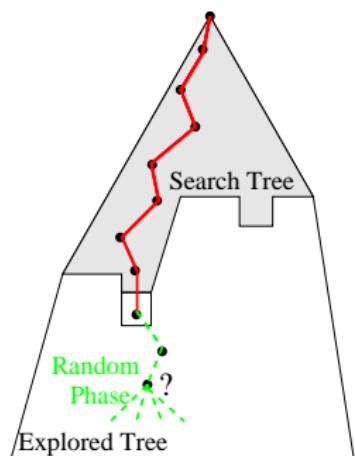


[1] S. Gelly, and D. Silver ICML'07

FUSE: random phase

Dealing with an unknown horizon

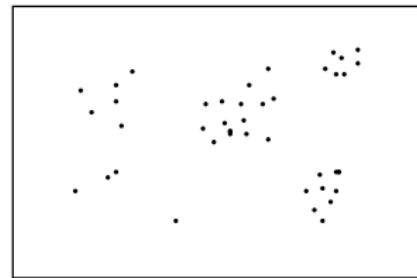
- ▶ Bottleneck
 - ▶ Finite unknown horizon
- ▶ Random phase policy
 - ↑ With probability $1 - q^{|F|}$ stop
 - | Else • add a uniformly selected feature
 - $|F| = |F| + 1$
 - | Iterate



FUSE: reward(F)

Generalization error estimate

- ▶ Requisite
 - ▶ fast (to be computed 10^4 times)
 - ▶ unbiased
- ▶ Proposed reward
 - ▶ k -NN like
 - ▶ + AUC criterion *
- ▶ Complexity: $\tilde{O}(md)$
 - d* Number of selected features
 - n* Size of the training set
 - m* Size of sub-sample ($m \ll n$)



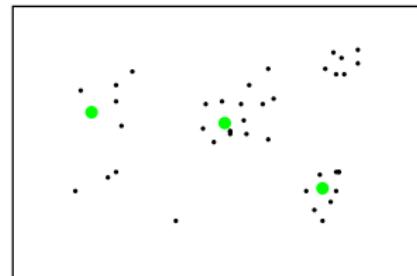
(*) Mann Whitney Wilcoxon test:

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

FUSE: reward(F)

Generalization error estimate

- ▶ Requisite
 - ▶ fast (to be computed 10^4 times)
 - ▶ unbiased
- ▶ Proposed reward
 - ▶ k -NN like
 - ▶ + AUC criterion *
- ▶ Complexity: $\tilde{O}(md)$
 - d* Number of selected features
 - n* Size of the training set
 - m* Size of sub-sample ($m \ll n$)



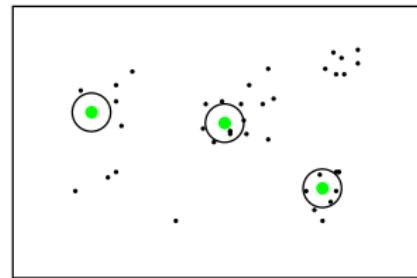
(*) Mann Whitney Wilcoxon test:

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

FUSE: reward(F)

Generalization error estimate

- ▶ Requisite
 - ▶ fast (to be computed 10^4 times)
 - ▶ unbiased
- ▶ Proposed reward
 - ▶ k -NN like
 - ▶ + AUC criterion *
- ▶ Complexity: $\tilde{O}(md)$
 - d* Number of selected features
 - n* Size of the training set
 - m* Size of sub-sample ($m \ll n$)



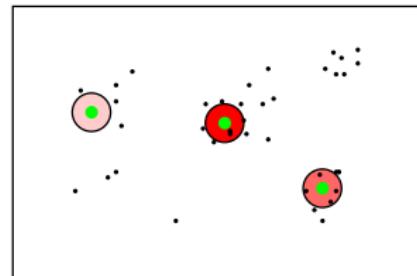
(*) Mann Whitney Wilcoxon test:

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

FUSE: reward(F)

Generalization error estimate

- ▶ Requisite
 - ▶ fast (to be computed 10^4 times)
 - ▶ unbiased
- ▶ Proposed reward
 - ▶ k -NN like
 - ▶ + AUC criterion *
- ▶ Complexity: $\tilde{O}(md)$
 - d* Number of selected features
 - n* Size of the training set
 - m* Size of sub-sample ($m \ll n$)



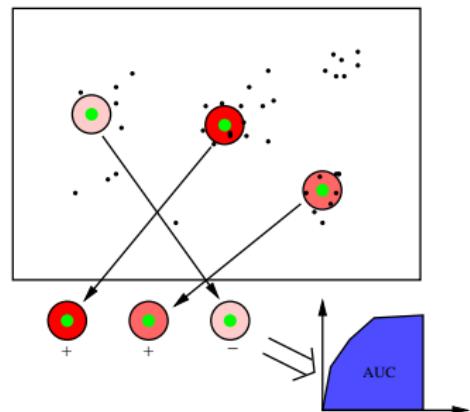
(*) Mann Whitney Wilcoxon test:

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

FUSE: reward(F)

Generalization error estimate

- ▶ Requisite
 - ▶ fast (to be computed 10^4 times)
 - ▶ unbiased
- ▶ Proposed reward
 - ▶ k -NN like
 - ▶ + AUC criterion *
- ▶ Complexity: $\tilde{O}(md)$
 - d* Number of selected features
 - n* Size of the training set
 - m* Size of sub-sample ($m \ll n$)

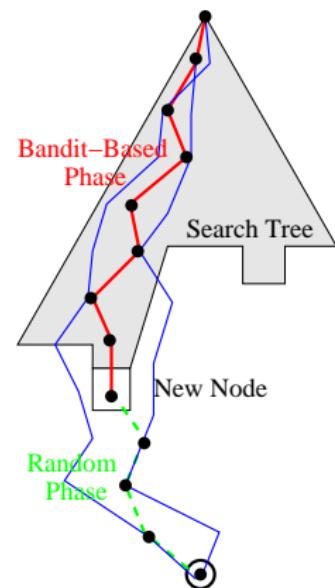


(*) Mann Whitney Wilcoxon test:

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

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



- ▶ 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
MADELON [1]	2,600	500	XOR-LIKE
ARCENE [1]	200	10,000	REDUNDANT FEATURES
COLON	62	2,000	“EASY”

[1] NIPS'03

Experimental setting

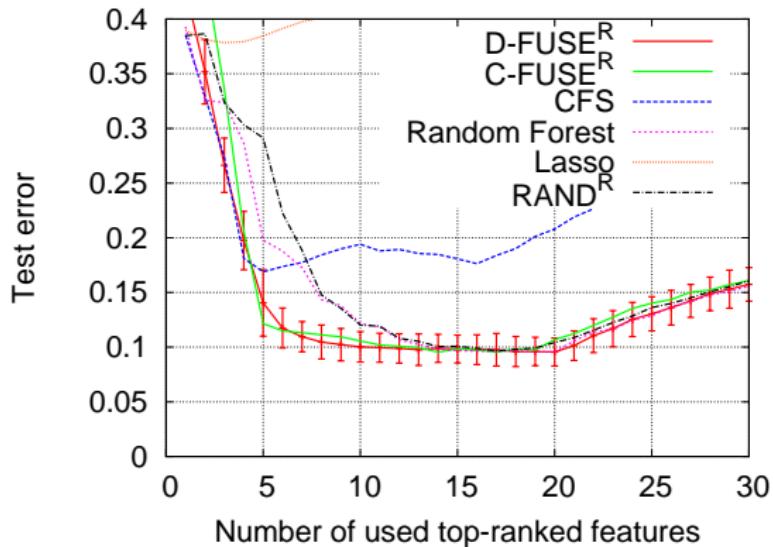
- ▶ Baselines
 - ▶ CFS (Constraint-based Feature Selection) [1]
 - ▶ Random Forest [2]
 - ▶ Lasso [3]
 - ▶ RAND^R: RAVE obtained by selecting 20 random features at each iteration
- ▶ Results averaged on 50 splits (10×5 fold cross-validation)
- ▶ End learner
 - ▶ Hyper-parameters optimized by 5 fold cross-validation

[1] M. A. Hall ICML'00

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

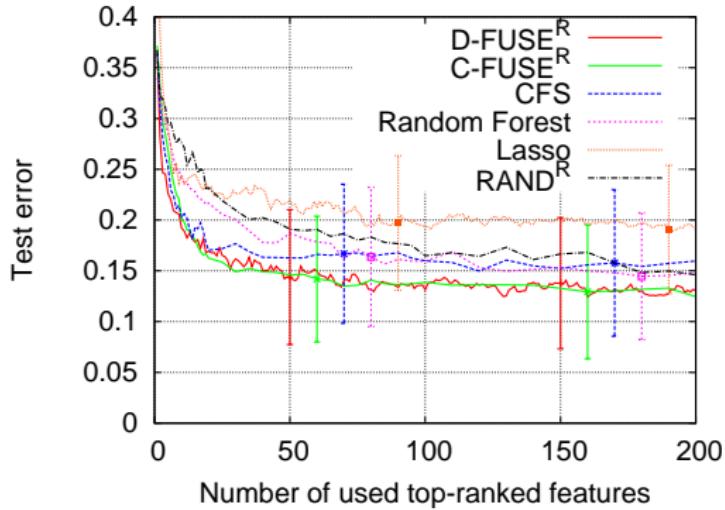
[3] R. Tibshirani Journal of the Royal Statistical Society 94

Results on Madelon after 200,000 iterations



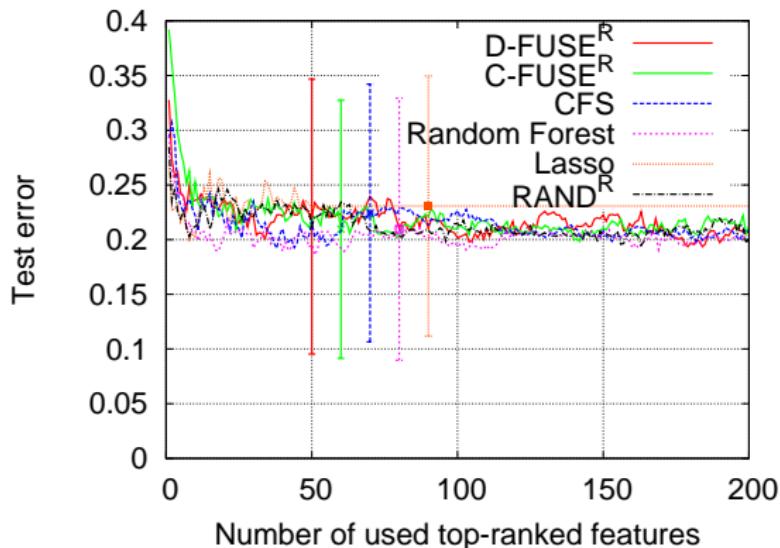
- ▶ **Remark:** FUSE^R = best of both worlds
 - ▶ Removes redundancy (like CFS)
 - ▶ Keeps conditionally relevant features (like Random Forest)

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

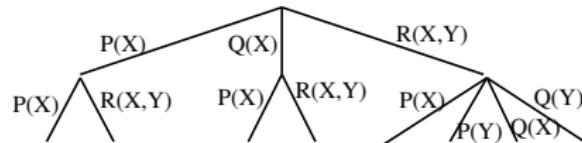
Conclusion

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