Reinforcement Learning

Michèle Sebag ; TP : Diviyan Kalainathan
TAO, CNRS − INRIA − Université Paris-Sud

Dec. 11th, 2017

Where we are

MDP Main Building block

General settings

	Model-based	Model-free
Finite	Dynamic Programming	**Discrete RL**
Infinite	(optimal control)	Continuous RL

More about the Exploration *vs* Exploitation Dilemma

This course: **Multi-Armed Bandits ; Monte-Carlo Tree Search**

Overview

Action selection as a Multi-Armed Bandit problem

Lai, Robbins 85

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

Lifelong learning

Exploration vs **Exploitation** Dilemma

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

Formalization

- K options a.k.a. arms
- Arms are independent
- The i-th arm yields a reward r drawn iid along distribution ν_i
  In the following, $\nu_i = \text{Bernoulli}(\mu_i)$
  (return 1 with proba μ_i, 0 otherwise).

Goals

- Find the best arm:
$$i^* = \arg\max_i \mathbb{E}[\nu_i]$$
- Find a policy $\pi : t \rightarrow i_t$, gets reward r_t s.t. the sum of rewards is maximal in expectation
$$\pi = \arg\max \mathbb{E}[r_0 + r_1 + \dots$$

Applications

- Find the best cure/drug for a disease.
  $r = 1$ if patient is cured, 0 otherwise

- Find the best ad for a Web site/user
  $r = 1$ if user clicks on the ad, 0 otherwise

- Find the best action for a robot
  $r = 1$ if the robot grasps the banana, 0 otherwise
  (What is different here ?)

The multi-armed bandit (MAB) problem

Algorithmic setting

Unknown parameters: K unknown probability distributions on $[0, 1]$

Known parameters: the set of arms $1 \dots K$, the number of rounds T

For each round $t = 1, 2, \dots, T$

(1) the learner chooses $i_t \in 1 \dots K$ according to its own strategy.

(2) the learner incurs and observes the reward $r_t \sim \nu_{i_t}$ independently from the past given rewards.

T: time horizon

When T unknown, algorithm is *anytime*

The multi-armed bandit (MAB) problem

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

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

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

$\Leftrightarrow$

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

Objective

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

$\Leftrightarrow$

$$\textbf{Minimize Regret } (t) = \sum_{u=1}^{t} (r \sim \nu^* - r_u)$$

Regret: extra-loss incurred w.r.t. the oracle (who knows i^*).

Why using the regret ?
"Kind of" normalization w.r.t. problem difficulty: the more difficult the problem, the lower the oracle's gain; what matters is how well one fares compared to the expert.
(Additive normalization).

Overview

Notations

- $n_{i,t}$: number of times i has been selected up to t
- $\hat{\mu}_{i,t}$ empirical reward of i-th arm as of t

$$\hat{\mu}_{i,t} = \frac{1}{n_{i,t}} \sum_{u=1}^{t} r_u . \mathbb{I}_{i_u = i}$$

  with $\mathbb{I}_e = 1$ iff e holds true
- $\mu_i = \mathbb{E}[\nu_i]$
- Δ_i: margin of i-th arm

$$\Delta_i = \mu^* - \mu_i$$

Scientific questions

- How does the regret increase with T (linear ? quadratic ? logarithmic ?)
- What are the factors of difficulty of the MAB problem ?

Greedy algorithm

- Draw once each arm

$$\hat{\mu}_i = r \sim \nu_i$$

- At time u, select arm i_t s.t.

$$i_t = argmax\{\hat{\mu}_{i,t-1}, i = 1 \ldots K\}$$

Example

- 2 arms:
  - arm 1, $\mu_1 = .8$;
  - arm 2, $\mu_2 = .2$.
- Assume the first two drawings yield:
  - arm 1, $r_1 = 0$;
  - arm 2, $r_2 = 1$.
- What happens ?

The ϵ-greedy algorithm

At each time t,

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

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

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

What is the regret ?

The ϵ-greedy algorithm

At each time t,

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

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

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

What is the regret ?

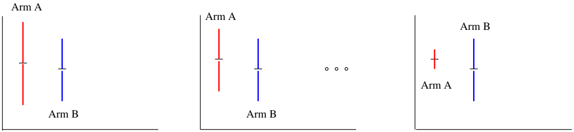
$$\textbf{Regret } (t) > \varepsilon t \frac{1}{K} \sum_i \Delta_i$$

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

Upper Confidence Bound

Auer et al. 2002

$$\text{Select } i_t = \text{argmax} \left\{ \hat{\mu}_{i,t} + \sqrt{2\frac{log(t)}{n_{i,t}}} \right\}$$



Decision: Optimism in front of unknown !

Upper Confidence bound, 2

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

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

Then

$$Regret(t) \leq 8 \sum_{i \neq i^*} \frac{1}{\Delta_i} log(t) + \left(1 + \frac{\pi^2}{3}\right) \sum_i \Delta_i$$

Proof

$$Regret(t) = \sum_{i \neq i^*} n_{i,t} \Delta_i$$

Upper Confidence bound, 3

The very useful Hoeffding inequality

Given $r_1, \dots r_n$ iid in $[0,1]$ drawn after p, with expectation μ,
Define empirical mean $\hat{\mu}_n = 1/n \sum_{u=1}^{n} r_u$, then

$$\begin{aligned}
\mathbb{P}\left(\hat{\mu}_n - \mu \geq \varepsilon\right) &\leq \exp\left(-2\,\varepsilon^2 n\right),\\
\mathbb{P}\left(\mu - \hat{\mu}_n \geq \varepsilon\right) &\leq \exp\left(-2\,\varepsilon^2 n\right),\\
\mathbb{P}\left(|\hat{\mu}_n - \mu| \geq \varepsilon\right) &\leq 2\exp\left(-2\,\varepsilon^2 n\right)
\end{aligned}$$

Sketch of the proof

Auer et al., 02

$$\text{Regret(t)} = \sum_{i \neq i^*} \Delta_i \times n_{i,t}$$

with $n_{i,t}$ = number of times i-th arm is played until step t.
Let $\ell_i = \frac{8 ln(t)}{\Delta_i^2}$. Then, for $n_{i,t} > \ell_i$,

$$\mu_i + 2\sqrt{\frac{2 ln(t)}{n_{i,t}}} < \mu^*$$

For $n_{i,t} > \ell_i$, wrong choice (one selects the i-th arm instead of the optimal i^* one)
$\Rightarrow$ $\widehat{\mu^*}$ is underestimated and $\widehat{\mu_i^*}$ is overestimated:

$$\begin{array}{ll} (A) & \widehat{\mu^*} < \mu^* - \sqrt{\frac{2 ln(t)}{n_{i^*,t}}} \\ (B) & \widehat{\mu_i^*} > \mu_i^* + \sqrt{\frac{2 ln(t)}{n_{i,t}}} \end{array}$$

Hoeffding $\Rightarrow$
Events (A) and (B) occur with probability less than $exp\{-4\, ln(t)\} = t^{-4}$

Sketch of the proof, 2

Hence:

$$\mathbb{E}[n_{i,t}] \leq \ell_i + \sum_{t=1}^{\infty} \sum_{n_{i,t}=\ell_i}^{t-1} (P(A) + P(B))$$

(first term: assume that it's always wrong in the first ℓ_i steps;
second term, $n_{i,t} \geq \ell_i$; if it goes wrong, the two estimates are far from their expectations.

$$\mathbb{E}[n_{i,t}] \leq \frac{8 ln(t)}{\Delta_i} + \sum_{t=\ell}^{\infty} 2t^{-4}$$

with

$$\sum_{t=1}^{\infty} t^{-4} = \frac{\pi^4}{90} \approx 1.09$$

Which concludes the proof (UCB regret is logarithmic):

$$Regret(t) \leq 8 \sum_{i \neq i*} \frac{1}{\Delta_i} log(t) + \frac{\pi^4}{90} \sum_i \Delta_i$$

Overview

Around MAB algorithms

- UCB is great, but not optimal. See KL-UCB Garivier et al. 2012
- In practice, play with C. control the exploration/exploitation trade-off
- Take into account the standard deviation of $\hat{\mu}_i$: Select $i_t = \text{argmax}$

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

- When there are **many** arms: tendency to over-explore...

Extensions

- When there is some side information: contextual bandits
- When arm distributions are not stationary: restless bandits

A particular algorithm: BESA

Best Empirical Sampled Average Baransi Maillard 2014

Intuition

- Case 1: you compare two arms with same number of reward samples.
  Easy: take the one with best average.

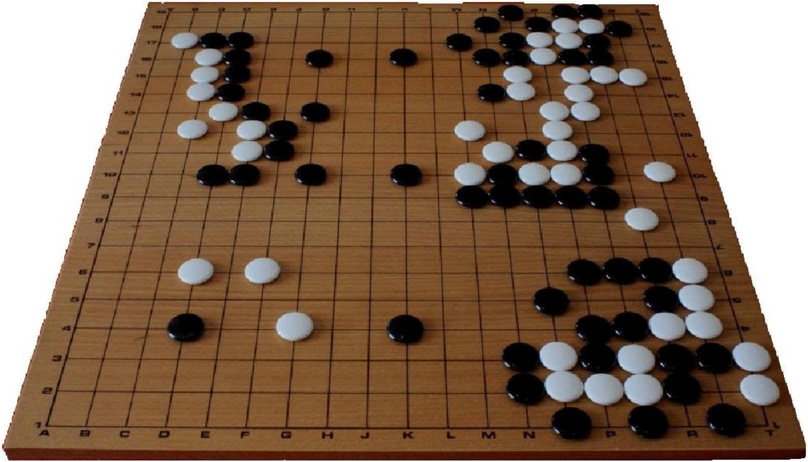
- Case 2: there is an arm A with many samples, and an arm B with few samples (say k).
  Easy: subsample k rewards for arm A and get back to Case 1.

Nota-bene

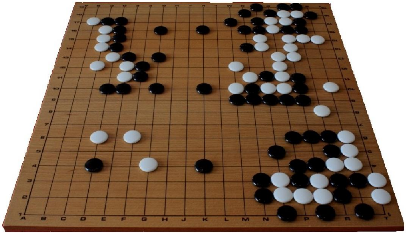
Same results with one hyper-parameter less $==$ much better.

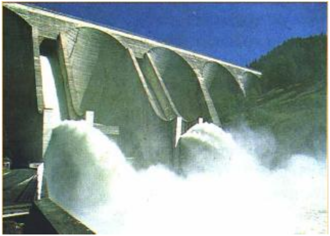
Overview

MCTS: computer-Go as explanatory example



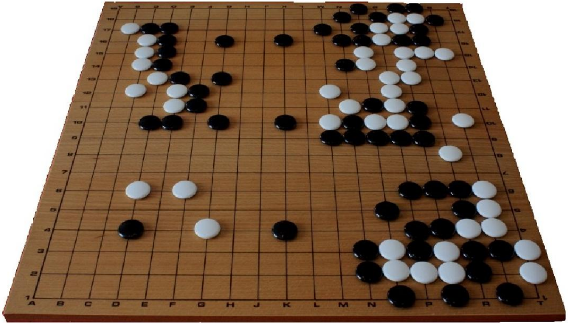
Not just a game: same approaches apply to optimal energy policy

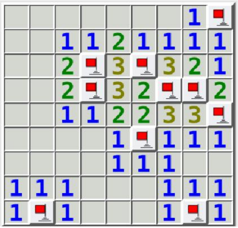




MCTS for computer-Go and MineSweeper

Go: deterministic transitions
MineSweeper: probabilistic transitions

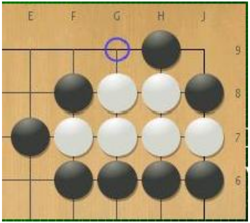




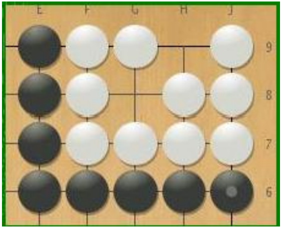
The game of Go in one slide

Rules

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



group w/o degree freedom is killed



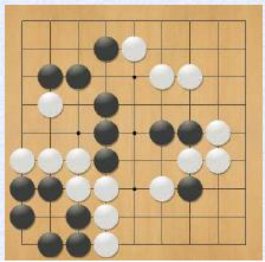
a group with two eyes can't be killed

- The goal is to control the max. territory

Go as a sequential decision problem

Features

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



Setting

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

Baseline strategies do not apply:

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

Monte-Carlo Tree Search

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

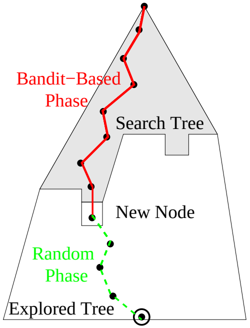
most promising subtrees are more explored and developed

Overview

Monte-Carlo Tree Search. Random phase

Gradually grow the search tree:

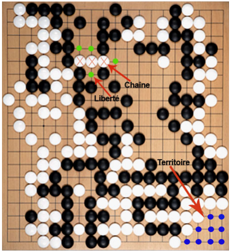
- Iterate Tree-Walk
  - Building Blocks
    - Select next action
      Bandit phase
    - Add a node
      Grow a leaf of the search tree
    - **Select next action bis**
      Random phase, roll-out
    - Compute instant reward
      Evaluate
    - Update information in visited nodes
      Propagate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



Random phase – Roll-out policy

Monte-Carlo-based Brügman 93

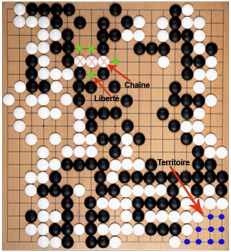
1. Until the goban is filled,
   add a stone (black or white in turn)
   at a uniformly selected empty position
2. Compute $r = \text{Win(black)}$
3. The outcome of the tree-walk is r



Random phase — Roll-out policy

Monte-Carlo-based Brügman 93

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



Improvements ?

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

Evaluation and Propagation

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

Propagate

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

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

Evaluation and Propagation

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

Propagate

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

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

Variants

Kocsis & Szepesvári, 06

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

Dilemma

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

Overview

Action selection revisited

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

- Asymptotically optimal
- But visits the tree infinitely often !

Being greedy is excluded not consistent

Frugal and consistent

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

Further directions

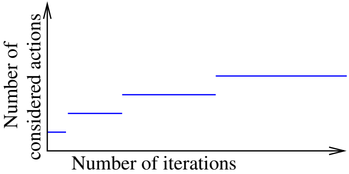
- Optimizing the action selection rule

Controlling the branching factor

What if many arms ? degenerates into exploration

- Continuous heuristics
  Use a small exploration constant c_e
- Discrete heuristics Progressive Widening
  Coulom 06; Rolet et al. 09

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



Introduce a new action when $\lfloor \sqrt[b]{n(s)+1} \rfloor > \lfloor \sqrt[b]{n(s)} \rfloor$
(which one ? See RAVE, below).

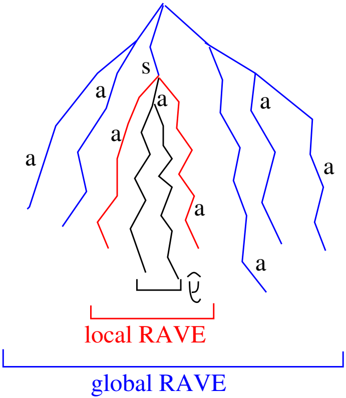
RAVE: Rapid Action Value Estimate

Gelly Silver 07

Motivation

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

$RAVE(s, a) =$
average $\{\hat{\mu}(s', a), s \text{ parent of } s'\}$



Rapid Action Value Estimate, 2

Using RAVE for action selection

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

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

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

Using RAVE with Progressive Widening

- PW: introduce a new action if $\lfloor \sqrt[b]{n(s)+1} \rfloor > \lfloor \sqrt[b]{n(s)} \rfloor$
- Select promising actions: it takes time to recover from bad ones
- Select argmax $RAVE_\ell(parent(s))$.

A limit of RAVE

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

B2 is the only good move for white
B2 only makes sense as first move (not in subtrees)
$\Rightarrow$ RAVE rejects B2.

Improving the roll-out policy π

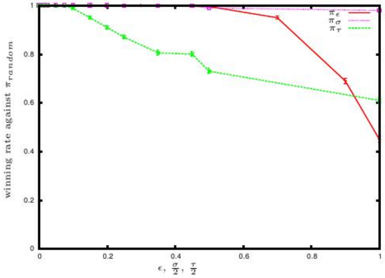
π_0	Put stones uniformly in empty positions	
π_{random}	Put stones uniformly in the neighborhood of a previous stone	
π_{MoGo}	Put stones matching patterns	prior knowledge
π_{RLGO}	Put stones optimizing a value function	Silver et al. 07

Beware! Gelly Silver 07

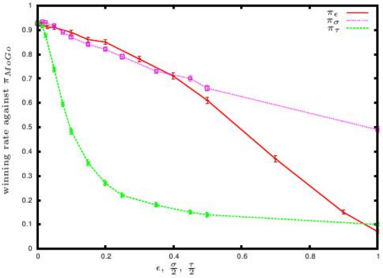
$$\pi \text{ better } \pi' \qquad \not\Rightarrow \qquad MCTS(\pi) \text{ better } MCTS(\pi')$$

Improving the roll-out policy π, followed

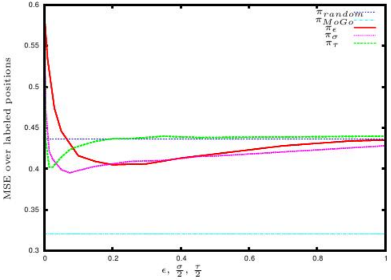
π_{RLGO} **against** π_{random}



π_{RLGO} **against** π_{MoGo}



Evaluation error on 200 test cases



Interpretation

What matters:

- Being **biased** is more harmful than being weak...
- Introducing a stronger but biased rollout policy π is detrimental.

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

Using prior knowledge

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

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

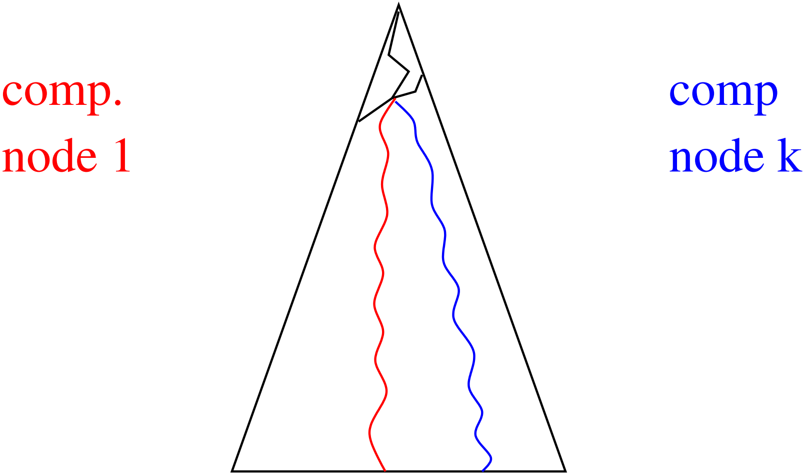
$$\begin{aligned} n_{s,a} &= n_{prior}(s,a) \quad \text{equivalent experience / confidence of priors} \\ \mu_{s,a} &= Q_{prior}(s,a) \end{aligned}$$

The best of both worlds

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

Overview

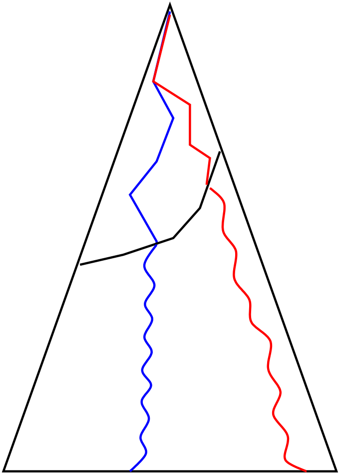
Parallelization. 1 Distributing the roll-outs



Distributing roll-outs on different computational nodes does not work.

Parallelization. 2 With shared memory

comp.
node 1

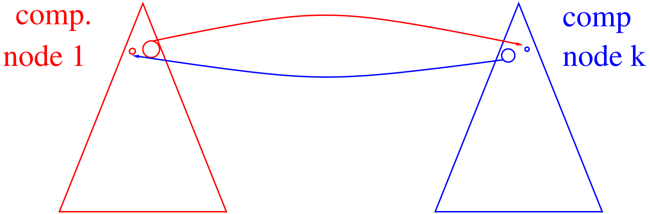


comp
node k

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

Use virtual updates to enforce the diversity of tree walks.

Parallelization. 3. Without shared memory



- Launch one MCTS per computational node
- k times per second $\qquad k = 3$
  - Select nodes with sufficient number of simulations $> .05 \times \#$ total simulations
  - Aggregate indicators

Good news
Parallelization with and without shared memory can be combined.

It works !

32 cores against	Winning rate on 9×9	Winning rate on 19×19
1	75.8 ± 2.5	95.1 ± 1.4
2	66.3 ± 2.8	82.4 ± 2.7
4	62.6 ± 2.9	73.5 ± 3.4
8	59.6 ± 2.9	63.1 ± 4.2
16	$52\pm 3.$	63 ± 5.6
32	$48.9\pm 3.$	48 ± 10

Then:

- Try with a bigger machine ! and win against top professional players !
- Not so simple... there are diminishing returns.

Increasing the number N of tree-walks

N	$2N$ against N	
	Winning rate on 9×9	Winning rate on 19×19
1,000	71.1 ± 0.1	90.5 ± 0.3
4,000	68.7 ± 0.2	$84.5 \pm 0,3$
16,000	66.5 ± 0.9	80.2 ± 0.4
256,000	$61 \pm 0,2$	58.5 ± 1.7

The limits of parallelization

R. Coulom

Improvement in terms of performance against humans

$\ll$

Improvement in terms of performance against computers

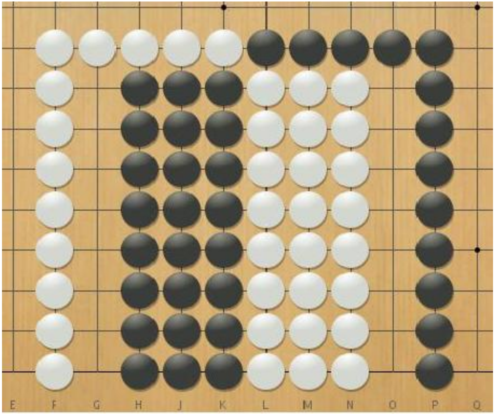
$\ll$

Improvements in terms of self-play

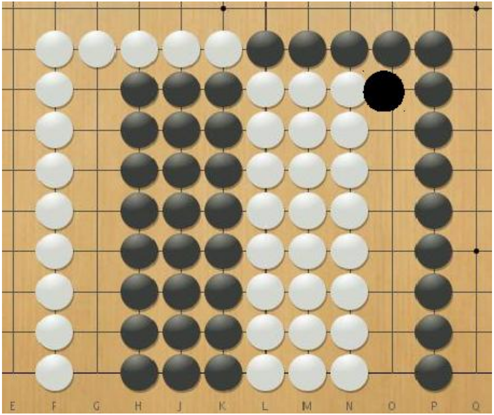
More: https://hal.inria.fr/inria-00512854/document

Overview

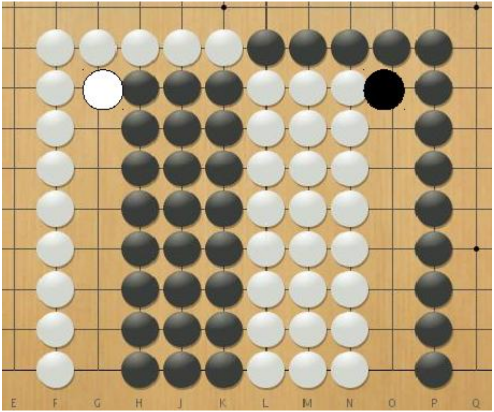
Failure: Semeai



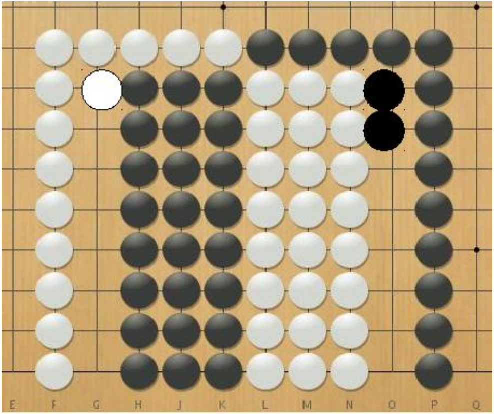
Failure: Semeai



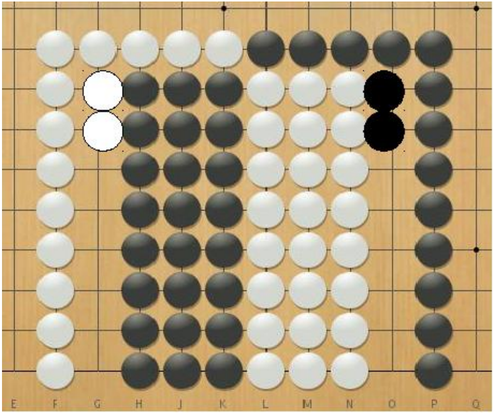
Failure: Semeai



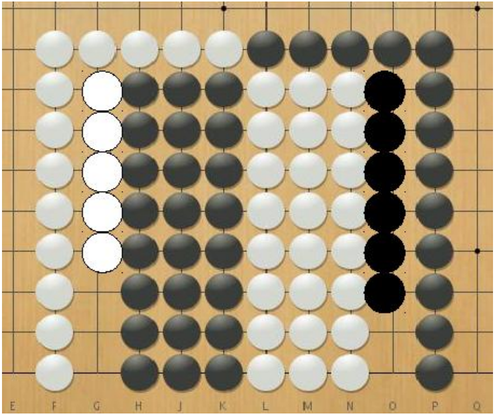
Failure: Semeai



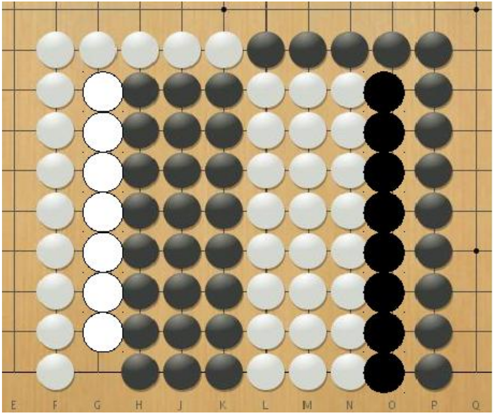
Failure: Semeai



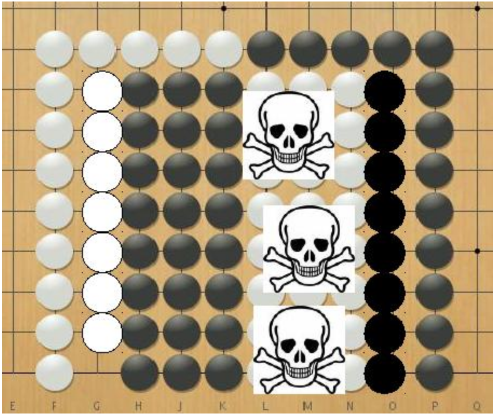
Failure: Semeai



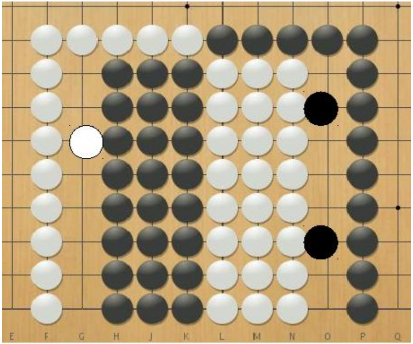
Failure: Semeai



Failure: Semeai



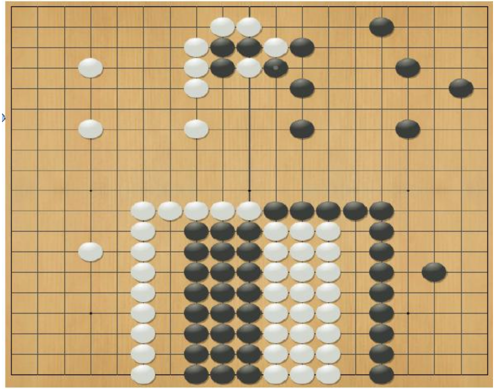
Failure: Semeai



Why does it fail

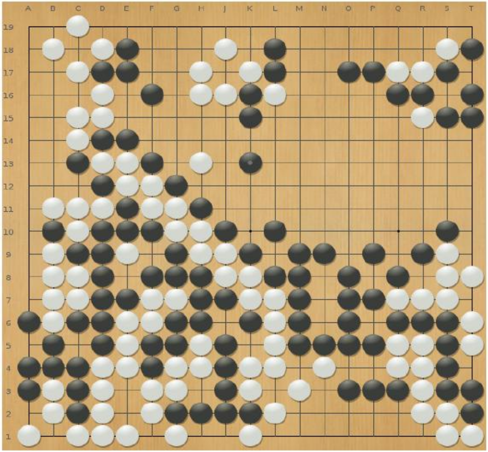
- First simulation gives 50%
- Following simulations give 100% or 0%
- But MCTS tries other moves: doesn't see all moves on the black side are equivalent.

Implication 1



MCTS does not detect invariance $\rightarrow$ too short-sighted and parallelization does not help.

Implication 2

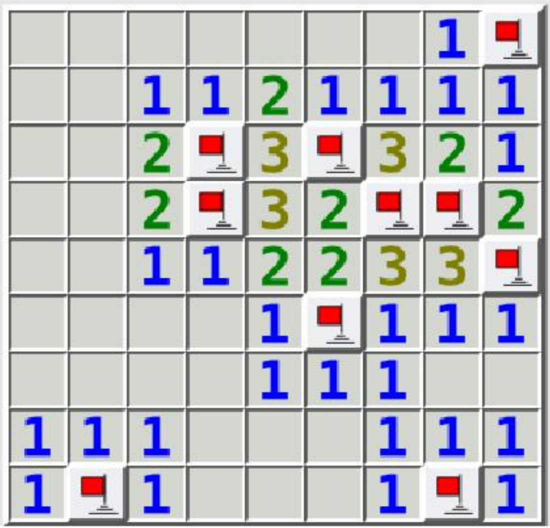


MCTS does not build abstractions $\rightarrow$ too short-sighted
and parallelization does not help.

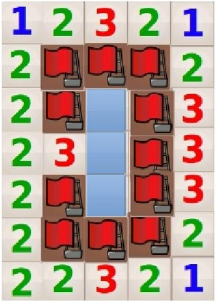
Overview

MCTS for one-player game

- The MineSweeper problem
- Combining CSP and MCTS

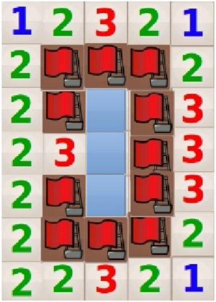


Motivation



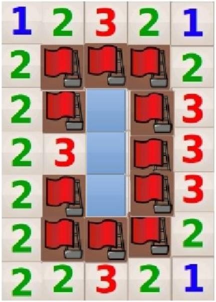
- All locations have same probability of death $\quad 1/3$
- Are then all moves equivalent ?

Motivation



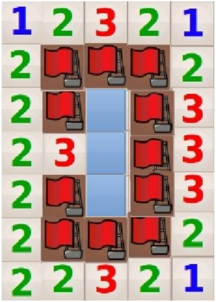
- All locations have same probability of death 1/3
- Are then all moves equivalent ? **NO !**

Motivation



- All locations have same probability of death 1/3
- Are then all moves equivalent ? **NO !**
- Top, Bottom: Win with probability 2/3

Motivation



- All locations have same probability of death   1/3
- Are then all moves equivalent ?   **NO !**
- Top, Bottom: Win with probability 2/3
- MYOPIC approaches LOSE.

MineSweeper, State of the art

Markov Decision Process Very expensive; 4×4 is solved

Single Point Strategy (SPS) local solver

CSP

- Each unknown location j, a variable $x[j]$
- Each visible location, a constraint, e.g. $loc(15) = 4 \rightarrow$

$$x[04] + x[05] + x[06] + x[14] + x[16] + x[24] + x[25] + x[26] = 4$$

- Find all N solutions
- $\mathrm{P}(\text{mine in } j) = \dfrac{\text{number of solutions with mine in } j}{N}$
- Play j with minimal $\mathrm{P}(\text{mine in } j)$

Constraint Satisfaction for MineSweeper

State of the art

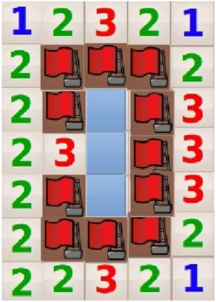
- 80% success *beginner* (9x9, 10 mines)
- 45% success *intermediate* (16x16, 40 mines)
- 34% success *expert* (30x40, 99 mines)

PROS

- Very fast

CONS

- Not optimal
- Beware of first move (opening book)



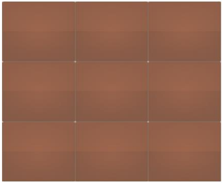
Upper Confidence Tree for MineSweeper

Couetoux Teytaud 11

- Cannot compete with CSP in terms of speed
- But consistent (find the optimal solution if given enough time)

Lesson learned

- Initial move matters
- UCT improves on CSP



- 3x3, 7 mines
- Optimal winning rate: 25%
- Optimal winning rate if uniform initial move: 17/72
- UCT improves on CSP by 1/72

UCT for MineSweeper

Another example

- 5x5, 15 mines
- GnoMine rule (first move gets 0)
- if 1st move is center, optimal winning rate is 100 %
- UCT finds it; CSP does not.

The best of both worlds

CSP

- Fast
- Suboptimal (myopic)

UCT

- Needs a generative model
- Asymptotic optimal

Hybrid

- UCT with generative model based on CSP

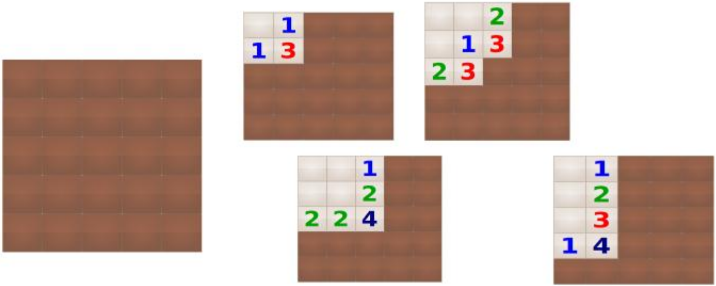
UCT needs a generative model

Given

- A state, an action
- **Simulate** possible transitions

Initial state, play top left

probabilistic transitions



Simulating transitions

- Using rejection (draw mines and check if consistent) SLOW
- Using CSP FAST

The algorithm: Belief State Sampler UCT

- One node created per simulation/tree-walk
- Progressive widening
- Evaluation by Monte-Carlo simulation
- Action selection: UCB tuned (with variance)
- Monte-Carlo moves
  - If possible, Single Point Strategy (can propose riskless moves if any)
  - Otherwise, move with null probability of mines (CSP-based)
  - Otherwise, with probability .7, move with minimal probability of mines (CSP-based)
  - Otherwise, draw a hidden state compatible with current observation (CSP-based) and play a safe move.

The results

- BSSUCT: Belief State Sampler UCT
- CSP-PGMS: CSP + initial moves in the corners

Format	CSP-PGMS	BSSUCT
4 mines on 4x4	64.7 %	**70.0% ± 0.6%**
1 mine on 1x3	100 %	100% (2000 games)
3 mines on 2x5	22.6%	**25.4 % ± 1.0%**
10 mines on 5x5	8.20%	9% (p-value: 0.14)
5 mines on 1x10	12.93%	**18.9% ± 0.2%**
10 mines on 3x7	4.50%	**5.96% ± 0.16%**
15 mines on 5x5	0.63%	**0.9% ± 0.1%**

Partial conclusion

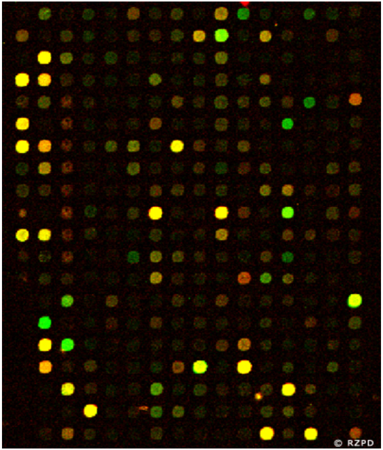
Given a myopic solver

- It can be combined with MCTS / UCT:
- Significant (costly) improvements

Overview

Feature Selection

BioInformatics



- 30 000 genes
- Find genes relevant to (cancer, obesity, you name it)

Position of the problem

Goals

- Selection
- Ranking

Formalization

Given feature set $\mathcal{F} = \{f_1, ..f_d\}$. Define

$$\begin{aligned} \mathcal{G} : \mathcal{P}(\mathcal{F}) &\mapsto \mathbb{R} \\ F \subset \mathcal{F} &\mapsto Err(F) = \text{ min error of models using } F \end{aligned}$$

Find $Argmin(\mathcal{G})$

Difficulties

- Combinatorial optimization problem (2^d)
- $\mathcal{F}$ unknown; noisy

Some approaches

- Filter approaches [1]
- Wrapper approaches
  - Tackling combinatorial optimization [2,3,4]
- 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

FS as A Markov Decision Process

Set of features $\mathcal{F}$

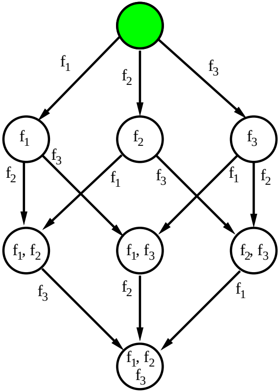
Set of states $\mathcal{S} = 2^{\mathcal{F}}$

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 $\underset{F \subseteq \mathcal{F}}{argmin}\ \mathbf{Err}\left(\mathcal{A}\left(F, D\right)\right)$

Optimal Policy

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

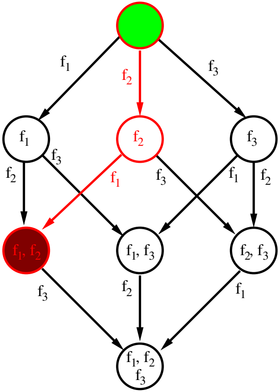
Final state following a policy F_π

Optimal policy $\pi^\star = \underset{\pi}{argmin}\ \mathbf{Err}\left(\mathcal{A}\left(F_\pi, \mathcal{E}\right)\right)$

Bellman's optimality principle

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

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



In practice

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

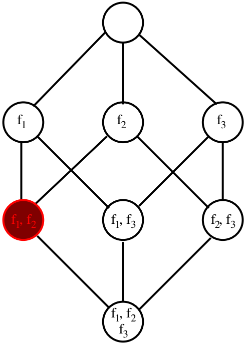
FS as a game

Exploration vs Exploitation tradeoff

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

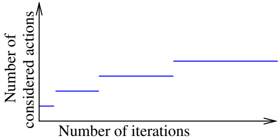
How ?

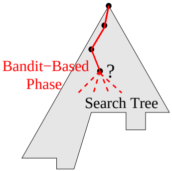
- Monte-Carlo Tree Search



FUSE: bandit-based phase
The many arms problem

- Bottleneck
  - A many-armed problem (hundreds of features)
    - $\Rightarrow$ 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 ($b < 1$)



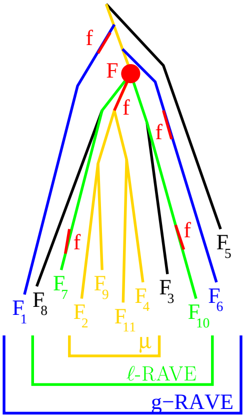


[1] S. Gelly, and D. Silver ICML'07
[2] R. Coulom Computer and Games 2006
[3] P. Rolet, M. Sebag, and O. Teytaud ECML'09

FUSE: bandit-based phase
Sharing information among nodes

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

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

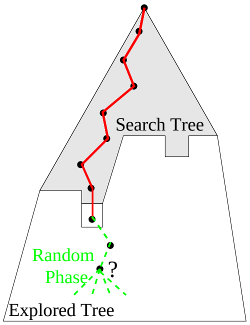


[1] S. Gelly, and D. Silver ICML'07

FUSE: random phase
Dealing with an unknown horizon

- Bottleneck
  - Finite unknown horizon
- Random phase policy

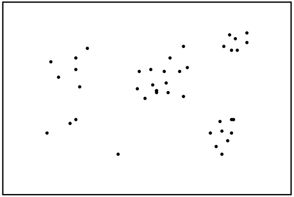
  With probability $1 - q^{|F|}$ stop
  Else • add a uniformly selected feature
  • $|F| = |F| + 1$
  Iterate



FUSE: reward(F)
Generalization error estimate

- Requisite
  - fast (to be computed 10^4 times)
  - unbiased
- Proposed reward
  - k-NN like
  - $+$ AUC criterion *
- Complexity: $\tilde{O}(mnd)$

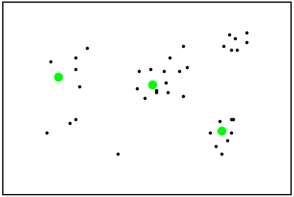
  d Number of selected features
  n Size of the training set
  m Size of sub-sample ($m \ll n$)



FUSE: reward(F)
Generalization error estimate

- Requisite
  - fast (to be computed 10^4 times)
  - unbiased
- Proposed reward
  - k-NN like
  - $+$ AUC criterion *
- Complexity: $\tilde{O}(mnd)$

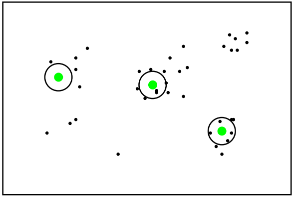
  d Number of selected features
  n Size of the training set
  m Size of sub-sample ($m \ll n$)



FUSE: reward(F)
Generalization error estimate

- Requisite
  - fast (to be computed 10^4 times)
  - unbiased
- Proposed reward
  - k-NN like
  - $+$ AUC criterion *
- Complexity: $\tilde{O}(mnd)$

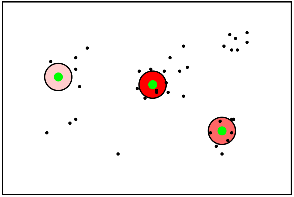
  d Number of selected features
  n Size of the training set
  m Size of sub-sample ($m \ll n$)



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 ($m \ll n$)



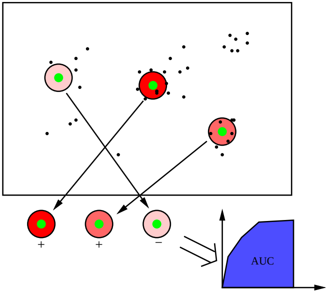
FUSE: reward(F)
Generalization error estimate

- 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

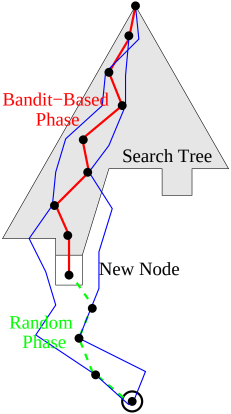
  n Size of the training set

  m Size of sub-sample ($m \ll n$)



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

$$\begin{array}{ccc} \text{Search tree (most visited path)} & \longleftrightarrow & \text{RAVE score} \\ \Downarrow & & \Downarrow \end{array}$$

- Result:

Wrapper approach	**Filter approach**
FUSE	**FUSE**^{R}

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

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"

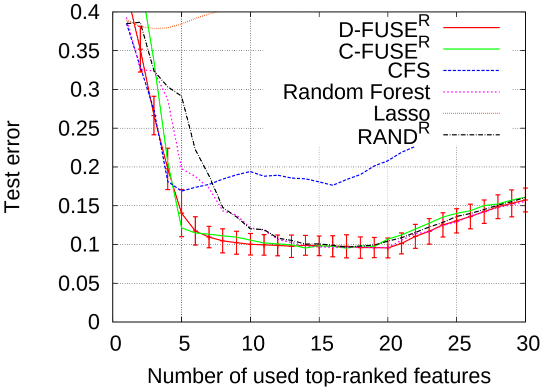
[1] NIPS'03

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 $\times$ 5 fold cross-validation)
- End learner
  - Hyper-parameters optimized by 5 fold cross-validation

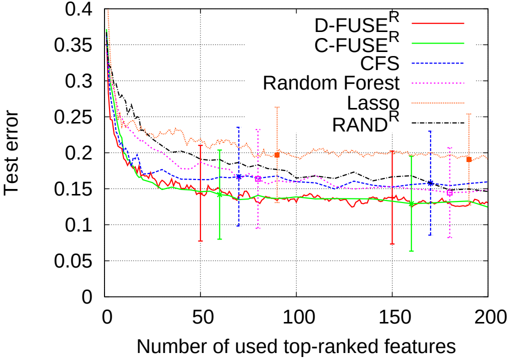
[1] M. A. Hall ICML'00
[2] J. Rogers, and S. R. Gunn SLSFS'05
[3] R. Tibshirani Journal of the Royal Statistical Society 94

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)

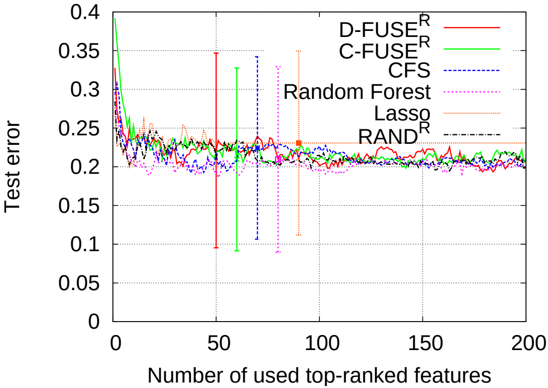
Results on Arcene after 200,000 iterations



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

[0] T-test "CFS vs. FUSE^R" with 100 features: p-value=0.036

Results on Colon after 200,000 iterations



- Remark
  - All equivalent

NIPS 2003 Feature Selection challenge

- Test error on the NIPS 2003 Feature Selection challenge
  - On an disjoint test set

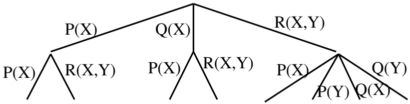
DATABASE	ALGORITHM	CHALLENGE ERROR	SUBMITTED FEATURES	IRRELEVANT FEATURES
MADELON	FSPP2 [1]	6.22% (1^{st})	12	0
	D-FUSE^R	6.50% (24^{th})	18	0
ARCENE	BAYES-NN-RED [2]	7.20% (1^{st})	100	0
	D-FUSE^R(ON ALL)	8.42% (3^{rd})	500	34
	D-FUSE^R	9.42% 500 (8^{th})	500	0

- Remarks
  - Selected features: accurate
  - Promising results

[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 and Perspectives

- 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
  - Extend to Feature construction
    - Inspired by [1]



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

Overview

Conclusion

Take-home message: MCTS/UCT

- enables any-time smart look-ahead for better sequential decisions in front of uncertainty.
- is an integrated system involving two main ingredients:
  - Exploration vs Exploitation rule UCB, UCBtuned, others
  - Roll-out policy
- can take advantage of prior knowledge

Caveat

- The UCB rule was not an essential ingredient of MoGo
- Refining the roll-out policy $\not\Rightarrow$ refining the system
  Many tree-walks might be better than smarter (biased) ones.

On-going

Extensions

- Continuous bandits: action ranges in a $\mathbb{R}$
- Contextual bandits: state ranges in $\mathbb{R}^d$
- Multi-objective sequential optimization
- Duelling bandits

Controlling the size of the search space

- Building abstractions
- Considering nested MCTS (partially observable settings, e.g. poker)