Meta-learning of exploration-exploitation strategies in reinforcement learning.

Damien Ernst

University of Liège, Belgium

8th of April 2013
University College London

Exploration-exploitation in RL

Reinforcement Learning (RL): an agent interacts with an environment in which it perceives information about its current state and takes actions. The environment, in return, provides a reward signal. The agent's objective is to maximize the (expected) cumulative reward signal. He may have a priori knowledge on its environment before taking an action and has limited computational resources.

Exploration-exploitation: The agent needs to take actions so as to gather information that may help him to discover how to obtain high rewards in the long-term (exploration) and, at the same time, he has to exploit at best its current information to obtain high short-term rewards.

Pure exploration: The agent first interacts with its environment to collect information without paying attention to the rewards. The information is used afterwards to take actions leading to high rewards.

Pure exploration with model of the environment: Typical problem met when an agent with limited computational resources computes actions based on (pieces of) simulated trajectories.

Example 1: Multi-armed bandit problems

Definition: A gambler (agent) has T coins, and at each step he may choose among one of K slots (or arms) to allocate one of these coins, and then earns some money (his reward) depending on the response of the machine he selected. Rewards are sampled from an unknown probability distribution dependent on the selected arm. Goal of the agent: to collect the largest cumulated reward once he has exhausted his coins.

Standard solutions: Index based policies; use an index for ranking the arms and pick at each play the arm with the highest index; the index for each arm k is a **small formula** that takes for example as input the average rewards collected ($\overline{r}_k$), the standard deviations of these rewards ($\overline{\sigma}_k$), the total number of plays t, the number of times arm k has been played (T_k), etc.

Upper Confidence Bound index as example: $\overline{r}_k + \sqrt{\frac{2 \ln t}{T_k}}$

Example 2: Tree search

Tree search: Pure exploration problem with model of the environment. The agent represents possible strategies by a tree. He cannot explore the tree exhaustively because of limited computational resources. When computational resources are exhausted, he needs to take an action. Often used with time receding horizon strategies.

Example of tree exploration algorithms: A^*, Monte Carlo Tree Search, optimistic planning for deterministic systems, etc. In all these algorithms, terminal nodes to be developed are chosen based on **small formulas**.

Example 3: Exploration-exploitation in a MDP

Markov Decision Process (MDP): At time t, the agent is in state $x_t \in X$ and may choose any action $u_t \in U$. The environment responds by randomly moving into a new state $x_{t+1} \sim P(\cdot|x_t, u_t)$, and giving the agent the reward $r_t \sim \rho(x_t, u_t)$.

An approach for interacting with MDPs when P and r unknown: a layer that learns (directly or indirectly) an approximate state-action value function $Q : X \times U \to \mathbb{R}$ atop of which comes the exploration/exploitation policy. Typical exploration-exploration policies are made of **small formulas**.

ϵ-Greedy: $u_t = \arg\max_{u \in U} Q(x_t, u)$ with probability $1 - \epsilon$ and u_t at random wiht probability ϵ.

Softmax policy: $u_t \sim P_{sm}(\cdot)$ where $P_{sm}(u) = \frac{exp^{Q(x_t,u)/\tau}}{\sum_{u \in U} exp^{Q(x_t,u)/\tau}}$.

Pros and cons for small formulas

Pros: (i) Lend themselves to a theoretical analysis (ii) No computing time required for designing a solution (iii) No a priori knowledge on the problem required.

Cons: (i) Formulas published not used as such in practice and solutions often engineered to the problem at hand (typically the case for Monte-Carlo Tree Search (MCTS) techniques) (ii) Computing time often available before starting the exploration-exploitation task (iii) A priori knowledge on the problem often available and not exploited by the formulas.

Learning for exploration-(exploitation) in RL

1. Define a set of training problems (*TP*) for exploration-(exploitation). A priori knowledge can (should) be used for defining the set *TP*.

2. Define a rich set of candidate exploration-(exploitation) strategies (*CS*) with numerical parametrizations/with formulas.

3. Define a performance criterion $PC : CS \times TP \rightarrow \mathbb{R}$ such that $PC(strategy, training\_problem)$ gives the performance of *strategy* on *training_problem*.

4. Solve $\arg\max_{strategy \in CS} \sum_{training\_prob. \in TP} PC(strategy, training\_prob.)$ using an optimisation tool.

NB: *approach inspired by current methods for tuning the parameters of existing formulas; differs by the fact that (i) it searches in much richer spaces of strategies (ii) the search procedure is much more systematic.*

Example 1: Learning for multi-armed bandit problems

Multi-armed bandit problem: Let $i \in \{1, 2, \ldots, K\}$ be the K arms of the bandit problem. ν_i is the reward distribution of arm i and μ_i its expected value. T is the total number of plays. b_t is the arm played by the agent at round t and $r_t \sim \nu_{b_t}$ is the reward it obtains.

Information: $H_t = [b_1, r_1, b_2, r_2, \ldots, b_t, r_t]$ is a vector that gathers all the information the agent has collected during the first t plays. $\mathcal{H}$ the set of all possible histories of any length t.

Policy: The agent's policy $\pi : \mathcal{H} \to \{1, 2, \ldots, K\}$ processes at play t the information H_{t-1} to select the arm $b_t \in \{1, 2, \ldots, K\}$: $b_t = \pi(H_{t-1})$.

Notion of regret: Let $\mu^* = \max_k \mu_k$ be the expected reward of the optimal arm. The *regret* of π is : $R_T^\pi = T\mu^* - \sum_{t=1}^T r_t$. The *expected regret* is $E[R_T^\pi] = \sum_{k=1}^K E[T_k(T)](\mu^* - \mu_k)$ where $T_k(T)$ is the number of times the policy has drawn arm k on the first T rounds.

Objective: Find a policy π^* that for a given K minimizes the expected regret, ideally for any T and any $\{\nu_i\}_{i=1}^K$ (equivalent to maximizing the expected sum of rewards obtained).
Index-based policy: (i) During the first K plays, play sequentially each arm (ii) For each $t > K$, compute for every machine k the score $index(H_{t-1}^k, t)$ where H_{t-1}^k is the history of rewards for machine k (iii) Play the arm with the largest score.

1. Define a set of training problems (*TP*) for exploration-exploitation. A priori knowledge can (should) be used for defining the set *TP*.

In our simulations a training set is made of 100 bandit problems with Bernouilli distributions, two arms and the same horizon T. Every Bernouilli distribution is generated by selecting at random in $[0, 1]$ its expectation.
Three training sets generated, one for each value of the *training horizon* $T \in \{10, 100, 1000\}$.

NB: *In the spirit of supervised learning, the learned strategies are evaluated on test problems different from the training problems. The first three test sets are generated using the same procedure. The second three test sets are also generated in a similar way but by considering truncated Gaussian distributions in the interval* $[0, 1]$ *for the rewards. The mean and the standard deviation of the Gaussian distributions are selected at random in* $[0, 1]$*. The test sets count 10,000 elements.*

2. Define a rich set of candidate exploration-exploitation strategies (*CS*).

Candidate exploration-exploitation strategies are index-based policies.

Set of index-based policies contains all the policies that can be represented by: $index(H_t^k, t) = \theta \cdot \phi(H_t^k, t)$ where θ is the vector of parameters to be optimized and $\phi(\cdot, \cdot)$ a hand-designed feature extraction function.

We suggest to use as feature extraction function the one that leads the indexes:

$$\sum_{i=0}^{p} \sum_{j=0}^{p} \sum_{k=0}^{p} \sum_{l=0}^{p} \theta_{i,j,k,l} (\sqrt{\ln t})^i \cdot (\frac{1}{\sqrt{T_k}})^j \cdot (\overline{r}_k)^k \cdot (\overline{\sigma}_k)^l$$

where $\theta_{i,j,k,l}$ are parameters.

$p = 2$ in our simulations $\Rightarrow$ 81 parameters to be learned.

3. Define a performance criterion $PC : CS \times TP \rightarrow \mathbb{R}$ such that $PC(strategy, training\_problem)$ gives the performance of *strategy* on *training_problem*.

Performance criterion of an index-based policy on a multi-armed bandit problem is chosen as the expected regret obtained by this policy.

NB: *Again in the spirit of supervised learning, a regularization term could be added to the above mentioned performance criterion to avoid overfitting.*

4. Solve $\arg\max_{strategy \in CS} \sum_{training\_prob. \in TP} PC(strategy, training\_prob.)$ using an optimisation tool.

Objective function has a complex relation with the parameter and may contain many local minima. Global optimisation approaches are advocated.

Estimation of Distribution Algorithms (EDA) are used here as global optimization tool. EDAs rely on a probabilistic model to describe promising regions of the search space and to sample good candidate solutions. This is performed by repeating iterations that first *sample* a population of n_p candidates using the *current* probabilistic model and then *fit* a *new* probabilistic model given the $b < n_p$ best candidates.

Simulation results

Policy quality on a test set measured by its average expected regret on this test set.

Learned index based-policies compared with 5 other index-based policies: UCB1, UCB1-BERNOULLI, UCB2, UCB1-NORMAL, UCB-V.

These 5 policies are made of small formulas which may have parameters. Two cases studied: (i) default values for the parameters or (ii) parameters optimized on a training set.

Policy	Training Horizon	Parameters	Bernoulli test problems T=10	T=100	T=1000
Policies based on small formulas. Default parameters					
UCB1	-	$C = 2$	1.07	5.57	20.1
UCB1-BERNOULLI	-		**0.75**	**2.28**	**5.43**
UCB1-NORMAL	-		1.71	13.1	31.7
UCB2	-	$\alpha = 10^{-3}$	0.97	3.13	7.26
UCB-V	-	$c = 1, \zeta = 1$	1.45	8.59	25.5
ϵ_n-GREEDY	-	$c = 1, d = 1$	1.07	3.21	11.5
Learned policies					
	T=10	...	**0.72**	2.37	15.7
	T=100	*(81 parameters)*	0.76	**1.82**	5.81
	T=1000	...	0.83	2.07	**3.95**

Observations: **(i)** UCB1-BERNOULLI is the best policy based on small formulas. **(ii)** The learned policy for a training horizon is always better than any policy based on a small formula if the test horizon is the same. **(iii)** Performances of the learned policies with respect to the formulas based policies degrade when the training horizon is not equal to the test horizon.

Policy	Training Horizon	Parameters	Bernoulli test problems T=10	T=100	T=1000
Policies based on small formulas. Optimized parameters					
UCB1	T=10	$C = 0.170$	**0.74**	**2.05**	4.85
	T=100	$C = 0.173$	0.74	**2.05**	**4.84**
	T=1000	$C = 0.187$	0.74	2.08	4.91
UCB2	T=10	$\alpha = 0.0316$	*0.97*	3.15	7.39
	T=100	$\alpha = 0.000749$	0.97	*3.12*	7.26
	T=1000	$\alpha = 0.00398$	0.97	3.13	*7.25*
UCB-V	T=10	$c = 1.542, \zeta = 0.0631$	*0.75*	2.36	5.15
	T=100	$c = 1.681, \zeta = 0.0347$	0.75	*2.28*	7.07
	T=1000	$c = 1.304, \zeta = 0.0852$	0.77	2.43	*5.14*
ϵ_n-GREEDY	T=10	$c = 0.0499, d = 1.505$	*0.79*	3.86	32.5
	T=100	$c = 1.096, d = 1.349$	0.95	*3.19*	14.8
	T=1000	$c = 0.845, d = 0.738$	1.23	3.48	*9.93*
Learned policies					
	T=10	...	**0.72**	2.37	15.7
	T=100	*(81 parameters)*	0.76	**1.82**	5.81
	T=1000	...	0.83	2.07	**3.95**

Main observation: The learned policy for a training horizon is always better than any policy based on a small formula if the test horizon is the same.

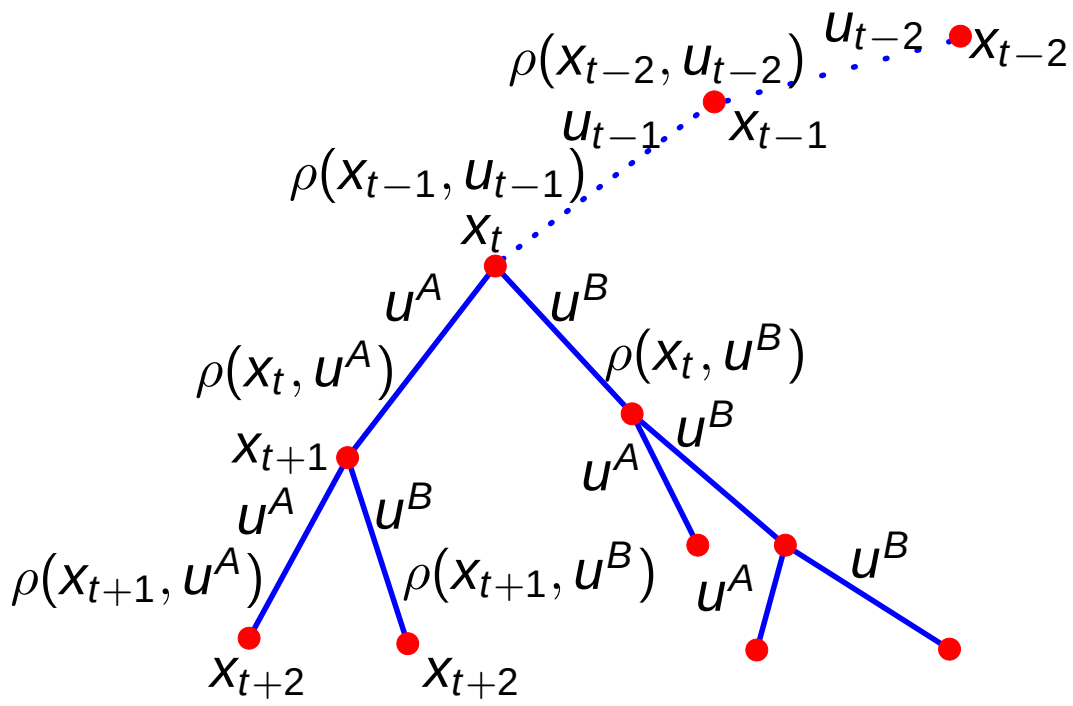
Policy	Training Horizon	Parameters	Gaussian test problems T=10	T=100	T=1000
Policies based on small formulas. Optimized parameters					
UCB1	T=10	$C = 0.170$	1.05	6.05	32.1
	T=100	$C = 0.173$	1.05	6.06	32.3
	T=1000	$C = 0.187$	1.05	6.17	33.0
UCB2	T=10	$\alpha = 0.0316$	1.28	7.91	40.5
	T=100	$\alpha = 0.000749$	1.33	8.14	40.4
	T=1000	$\alpha = 0.00398$	1.28	7.89	40.0
UCB-V	T=10	$c = 1.542, \zeta = 0.0631$	**1.01**	5.75	26.8
	T=100	$c = 1.681, \zeta = 0.0347$	**1.01**	**5.30**	**27.4**
	T=1000	$c = 1.304, \zeta = 0.0852$	1.13	5.99	27.5
ϵ_n-GREEDY	T=10	$c = 0.0499, d = 1.505$	**1.01**	7.31	67.57
	T=100	$c = 1.096, d = 1.349$	1.12	6.38	46.6
	T=1000	$c = 0.845, d = 0.738$	1.32	6.28	37.7
Learned policies					
	T=10	...	**0.97**	6.16	55.5
	T=100	*(81 parameters)*	1.05	**5.03**	29.6
	T=1000	...	1.12	5.61	**27.3**

Main observation: A learned policy is always better than any policy based on a small formula if the test horizon is equal to the training horizon.

Example 2: Learning for tree search. The case of look-ahead trees for deterministic planning

Deterministic planning: At time t, the agent is in state $x_t \in X$ and may choose any action $u_t \in U$. The environment responds by moving into a new state $x_{t+1} = f(x_t, u_t)$, and giving the agent the reward $r_t = \rho(x_t, u_t)$. The agent wants to maximize its *return* defined as: $\sum_{t=0}^{\infty} \gamma^t r_t$ where $\gamma \in [0, 1[$. Functions f and r are known.

Look-ahead tree: Particular type of policy that represents at every instant t the set of all possible trajectories starting from x_t by a tree. A look-ahead tree policy explores this tree until the computational resources (measured here in terms of number of tree nodes developed) are exhausted. When exhausted, it outputs the first action of the branch along which the highest discounted rewards are collected.



Scoring the terminal nodes for exploration: The tree is developed incrementally by always expending first the terminal node which has the highest score. Scoring functions are usually small formulas. (“best-first search” tree-exploration approach).

Objective: Find an exploration strategy for the look-ahead tree policy that maximizes the return of the agent, whatever the initial state of the system.

1. Define a set of training problems (*TP*) for exploration-(exploitation). A priori knowledge can (should) be used for defining the set *TP*.

The training problems are made of the elements of the deterministic planning problem. They only differ by the initial state.

If information is available on the true initial state of the problem, it should be used for defining the set of initial states X_0 used for defining the training set.

If the initial state is known perfectly well, one may consider only one training problem.

2. Define a rich set of candidate exploration strategies (*CS*).

The candidate exploration strategies all grow the tree incrementally, expending always the terminal node which has the highest score according to a scoring function.

The set of scoring functions contain all the functions that take as input a *tree* and a *terminal_node* and that can be represented by: $\theta \cdot \phi(tree, terminal\_node)$ where θ is the vector of parameters to be optimized and $\phi(\cdot, \cdot)$ a hand-designed feature extraction function.

For problem where $X \subset \mathbb{R}^m$, we use as ϕ: $(x[1], \ldots, x[m], dx[1], \ldots, dx[m], rx[1], \ldots, rx[m])$ where x is the state associated with the *terminal_node*, d is its depth and r is the reward collected before reaching this node.

3. Define a performance criterion $PC : CS \times TP \rightarrow \mathbb{R}$ such that $PC(strategy, training\_problem)$ gives the performance of *strategy* on *training_problem*.

The performance criterion of a scoring function is the return on the test problem of the look-ahead policy it leads to.

The computational budget for the tree development when evaluating the performance criterion should be chosen equal to the computational budget available when controlling the 'real system'.

4. Solve $\arg\max_{strategy \in CS} \sum_{training\_prob. \in TP} PC(strategy, training\_prob.)$ using an optimisation tool.

Objective function has a complex relation with the parameter and may contain many local minima. Global optimisation approaches are advocated.

Estimation of Distribution Algorithms (EDA) are used here as global optimization tool.

NB: *Gaussian processes for global optimisation have also been tested. They are more complex optimisation tools and they have been found to be able to require significantly less evaluation of objective function to identify a near-optimal solution.*

Simulation results

Small formulas based scoring functions used for comparison:

$$score^{mindepth} = -(\text{depth\_terminal\_node})$$

$$score^{greedy1} = \text{reward\_obtained\_before\_reaching\_terminal\_node}$$

$$score^{greedy2} = \text{sum\_of\_disc.\_rew.\_from\_top\_to\_terminal\_node}$$

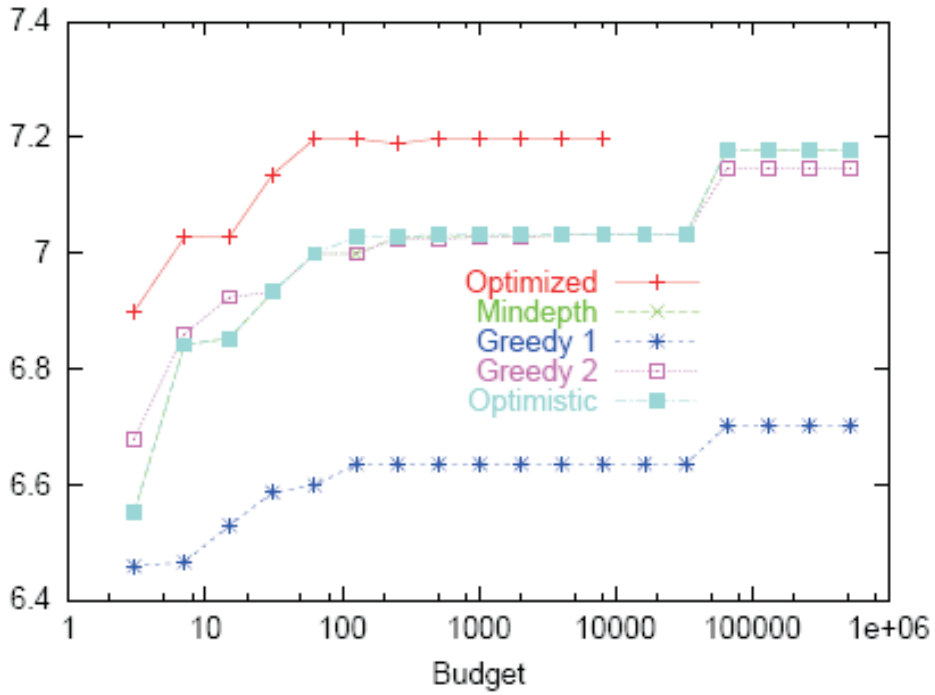
$$score^{optimistic} = score^{greedy2} + \frac{\gamma^{depth\_terminal\_node}}{1-\gamma} B_r$$

where B_r is an upperbound on the reward function.

In the results hereafter, performance of a scoring function evaluated by averaging the return of the agent over a set of initial states. Results generated for different computational budgets. The score function has always been optimized for the budget used for performance evaluation.

2-dimensional test problem

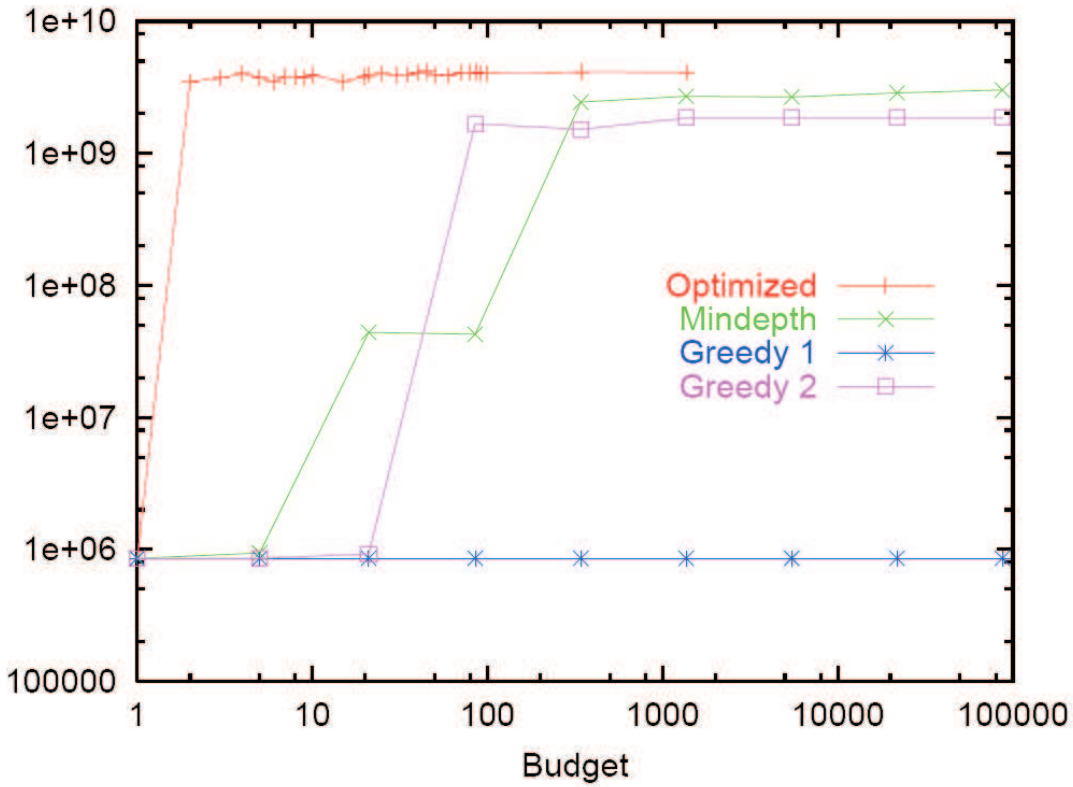
Dynamics: $(y_{t+1}, v_{t+1}) = (y_t, v_t) + (v_t, u_t)0.1$; $X = \mathbb{R}^2$; $U = \{-1, +1\}$; $\rho((y_t, v_t), u_t) = \max(1 - y_{t+1}^2, 0)$; $\gamma = 0.9$. The initial states chosen at random in $[-1, 1] \times [-2, 2]$ for building the training problems. Same states used for evaluation.

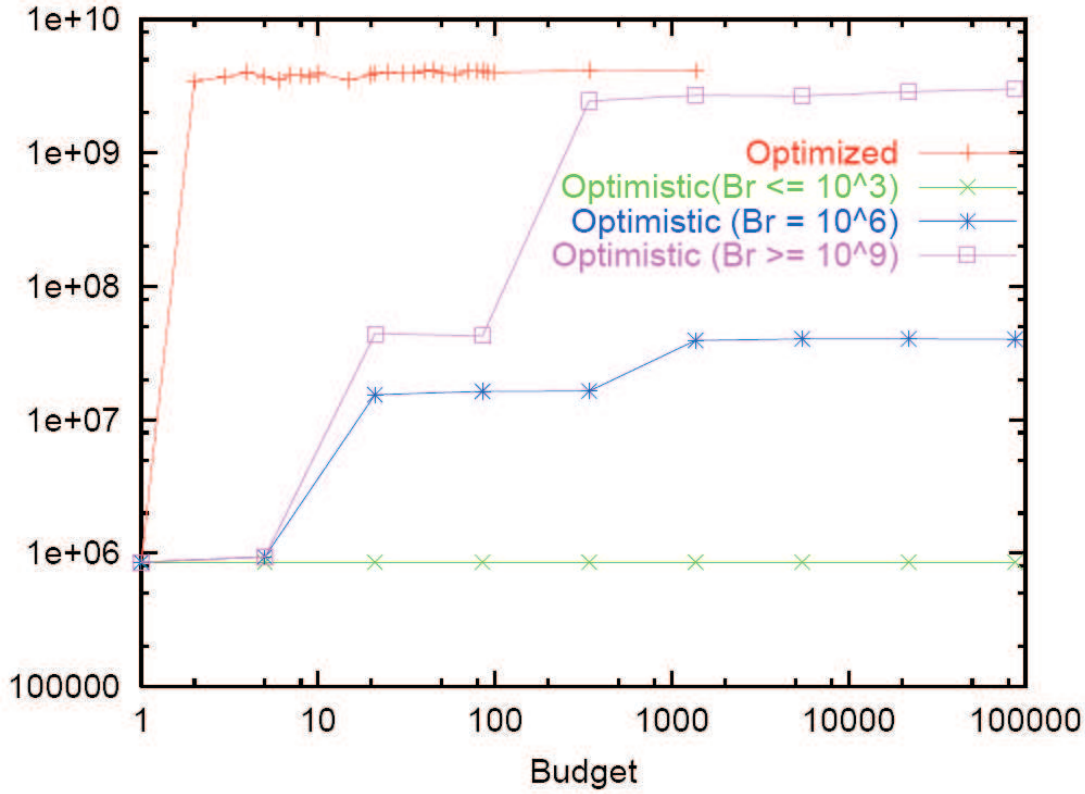


Main observation: (i) Optimized trees outperform other methods (ii) Small computational budget needed for reaching near-optimal performances.

HIV infection control problem

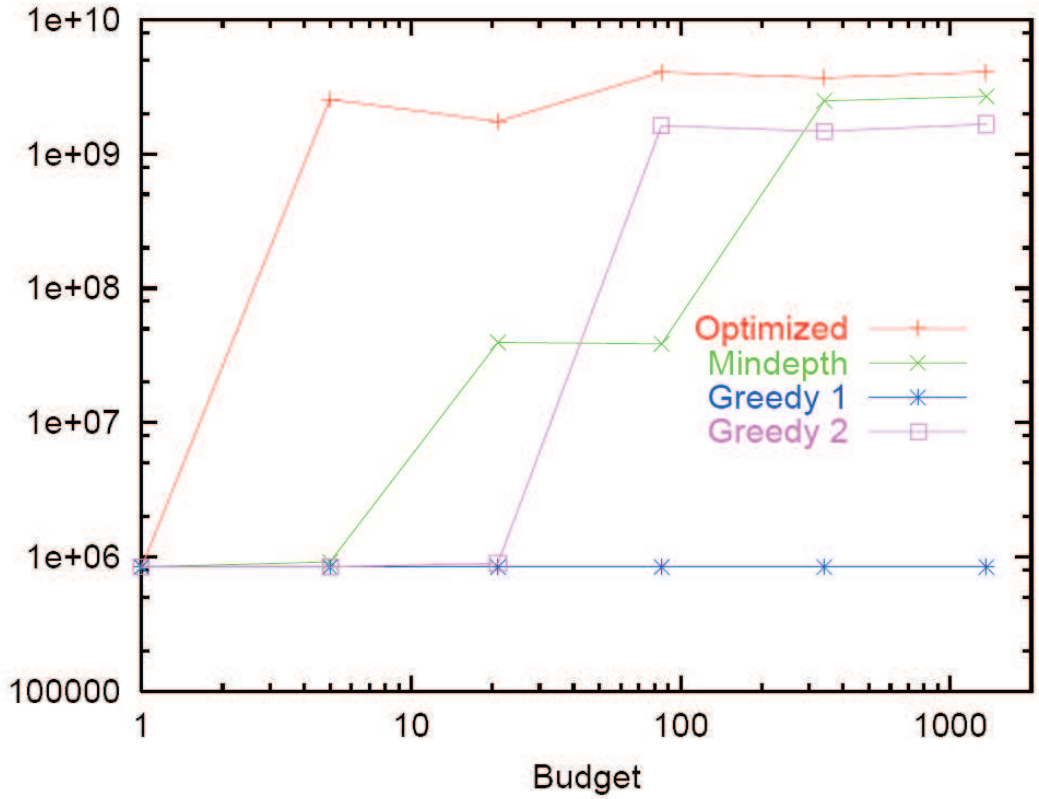
Problem with six state variables, two control actions and a highly non linear dynamics. A single training 'initial state' is used. Policies first evaluated when the system starts from this state.





Main observations: (i) Optimized look ahead trees clearly outperforms other methods. (ii) Very small computational budget required for reaching near optimal performances.

Policies evaluated when the system starts from a state rather far from the initial state used for training:



Main observation: Optimized look ahead trees perform very well even when the 'training state' is not the test state.

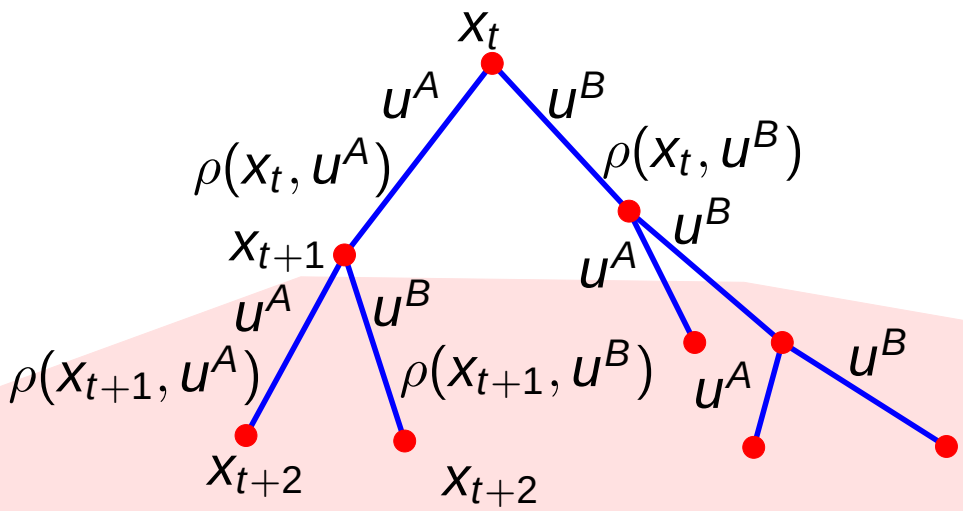
A side note on these optimized look-ahead trees

Optimizing look-ahead trees is a direct policy search method for which parameters to be optimized are those of the tree exploration procedure rather than those of standard function approximators (e.g, neural networks, RBFs) as it is usually the case.

Require very little memory (compared to dynamic programming methods and even other policy search techniques); not that many parameters need to be optimized; robust with respect to the initial states chosen for training, etc.

Right problem statement for comparing different techniques: *"How to compute the best policy given an amount of off-line and on-line computational resources knowing fully the environment and not the initial state?"*

Complement to other techniques rather than competitor:
look-ahead tree policies could be used as an “on-line correction mechanism” for policies computed off-line.



Other policy used for scoring the nodes
(for tree exploration and selection of action u_t)

Example 3: Learning for exploration-exploitation in finite MDPs

Instantiation of the general learning paradigm for exploration-exploitation for RL to this case is very much similar to what has been described for multi-bandit problems.

Good results already obtained by considering large set of canditate exploration-exploitation policies made of small formulas.

Next step would be to experiment the approach on MDPs with continuous spaces.

Automatic learning of (small) formulas for exploration-exploitation?

Remind: There are pros for using small formulas !

The learning approach for exploration-exploitation in RL could help to find new (small) formulas people have not thought of before by considering as candidate strategies a large set of (small) formulas.

An example of formula for index-based policies found to perform very well on bandit problems by such an approach: $\overline{r}_k + \frac{c}{T_k}$.

Conclusion and future works

Learning for exploration/exploitation: **excellent performances** that suggest a bright future for this approach.

Several directions for future research:

Optimisation criterion: Approach targets an exploration-exploitation policy that gives the best average performances on a set of problems. Risk adverse criteria could also be thought of. Regularization terms could also be added to avoid overfitting.

Design of customized optimisation tools: Particularly relevant when the set of candidate policies is made of formulas.

Theoretical analysis: E.g., what are the properties of the strategies learned in generalization?

Connection with Bayesian RL

"*Automatic discovery of ranking formulas for playing with multi-armed bandits*". F. Maes, L. Wehenkel and D. Ernst. In Proceedings of the 9th European Workshop on Reinforcement Learning (EWRL 2011), Athens, Greece, September 9-11, 2011. (12 pages).
"*Learning to play K-armed bandit problems*". F. Maes, L. Wehenkel and D. Ernst. In Proceedings of the 4th International Conference on Agents and Artificial Intelligence (ICAART 2012), Vilamoura, Algarve, Portugal, 6-8 February 2012. (8 pages).
"*0ptimized look-ahead tree search policies*". F. Maes, L. Wehenkel and D. Ernst. In Proceedings of the 9th European Workshop on Reinforcement Learning (EWRL 2011), Athens, Greece, September 9-11, 2011. (12 pages).
"*Learning exploration/exploitation strategies for single trajectory reinforcement learning*". M. Castronovo, F. Maes, R. Fonteneau and D. Ernst. In Proceedings of the 10th European Workshop on Reinforcement Learning (EWRL 2012), Edinburgh, Scotland, June 30-July 1 2012. (8 pages).
"*Meta learning of exploration/exploitation strategies: the multi-armed bandit case*". F. Maes, L. Wehenkel and D. Ernst. In Springer Selection of papers of ICAART 2012.
"*Optimized look-ahead tree policies: a bridge between look-ahead tree policies and direct policy search*". T. Jung, D. Ernst, L.Wehenkel and F. Maes. To appear in International Journal of Adaptive Control and Signal Processing.
"*Monte-Carlo search algorithm discovery for one player games*". F. Maes, D. Lupien St-Pierre and D. Ernst. To appear in IEEE Transactions on Computational Intelligence and AI in Games.