Optimized Lookahead Trees: A Bridge Between Lookahead Tree Policies and Direct Policy Search

Tobias Jung, Louis Wehenkel, Damien Ernst, and Francis Maes

Montefiore Institute

University of Liège

{tjung}@ulg.ac.be

Motto: Lookahead trees as a fancy way of parameterizing policies in direct policy search

Motivation

Problem statement

We consider general discrete-time systems with

- X state space (e.g., $\subset \mathbb{R}^{n_x}$, n_x dimensionality of X)
- A action space (e.g., $\subset \mathbb{R}^{n_u}$, n_u dimensionality of A)
- $f : X \times A \to X$ deterministic transition function
- $\varrho : X \times A \to \mathbb{R}$ stepwise performance measure ("rewards")
- no regularity assumptions on (f, ϱ) (e.g., linearity) (this rules out standard MPC)

Let $\pi : X \to A$ denote a stationary deterministic policy. Consider

$$V^\pi(x_0) := \lim_{T\to\infty} \sum_{t=0}^{T} \gamma^t \varrho(x_t, \pi(x_t)) \quad \text{where } x_{t+1} = f(x_t, \pi(x_t))$$

(infinite horizon discounted sum of rewards)

Goal: model based deterministic optimal control

$$\boxed{\text{Given } (f, \varrho), \text{we want to find the best policy } \pi^* = \operatorname*{argmax}_\pi \ V^\pi(x_0)}$$

Challenge: as we all know, solving the HJB equation is difficult, in particular if the dimensionality of the state space is large. (And we do not even want to think about high dim action spaces.)

Motivation I

Two simple but reliable ways of finding good control policies without touching the HJB equation:

Motivation I

Two simple but reliable ways of finding good control policies without touching the HJB equation:

I. Direct policy search (DPS):

1. Parameterize the policy directly, e.g.,

$$\pi(x;\theta) = \tau\Big(\sum_i \theta_i g_i(x)\Big) \text{ or } \pi(x;\theta,\xi) = \tau\Big(\sum_i \theta_i g_i(x;\xi_i)\Big)$$

   where τ is some transformation, g_i **suitable** features, and θ, ξ parameters.

2. Determine θ via black-box global optimization (derivative-free):

$$\theta^* = \operatorname*{argmax}_\theta \sum_{x_0 \in X_0} \sum_{t=0}^{H} \gamma^t \varrho\big(x_t, \pi(x_t,\theta)\big) \text{ where } x_{t+1} = f(x_t, \pi(x_t;\theta)),$$

   X_0 is a set of initial states and H the number of steps used to evaluate the performance of a parameter setting.

Examples: polynomials, RBFs, fuzzy-somethings, NN, recurrent NN, and whatnot

Motivation II

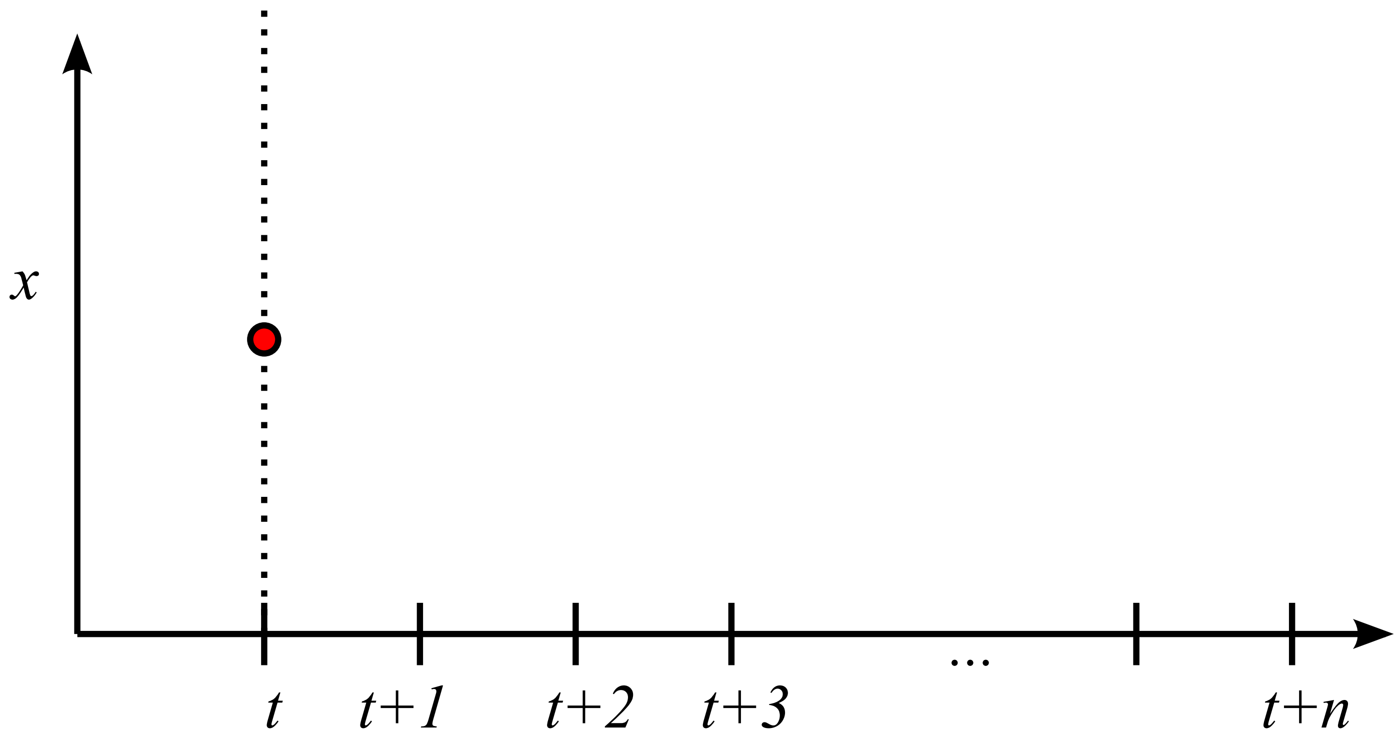
II. Lookahead tree policies (LT)

- Approaches that locally build a tree of limited depth every time a decision is required:

Motivation II

II. Lookahead tree policies (LT)

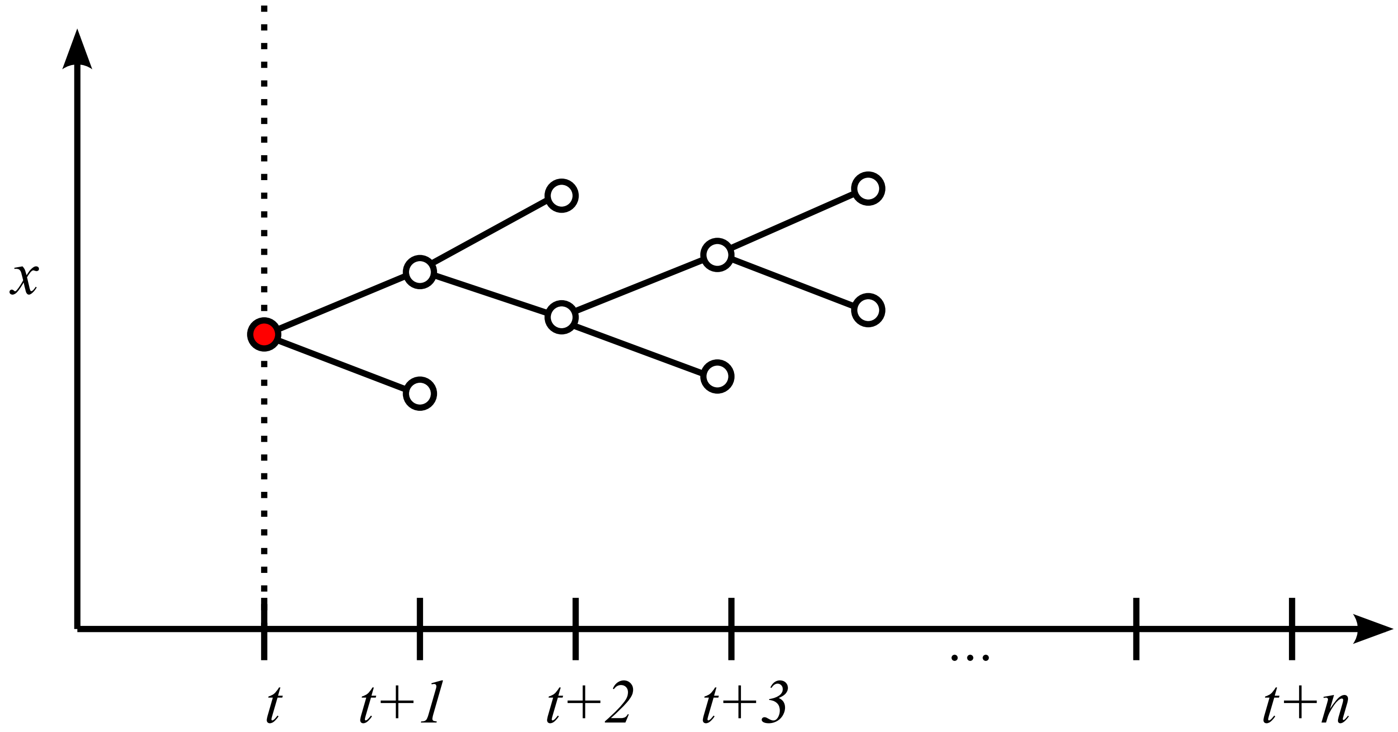
- Approaches that locally build a tree of limited depth every time a decision is required:
- Let x_t be the current state for which we want to compute an action:



Motivation II

II. Lookahead tree policies (LT)

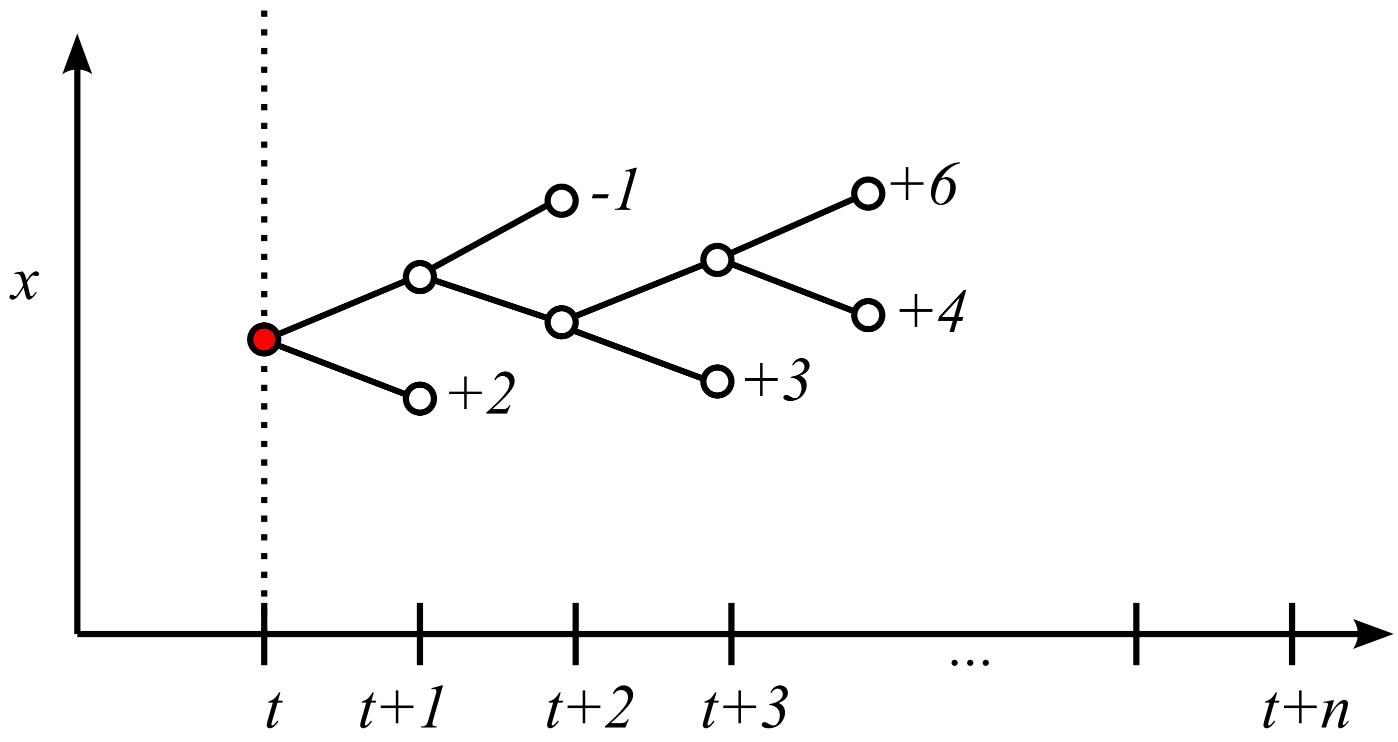
- Approaches that locally build a tree of limited depth every time a decision is required:
- Build a lookahead tree from x_t until budget of node expansions exhausted:



Motivation II

II. Lookahead tree policies (LT)

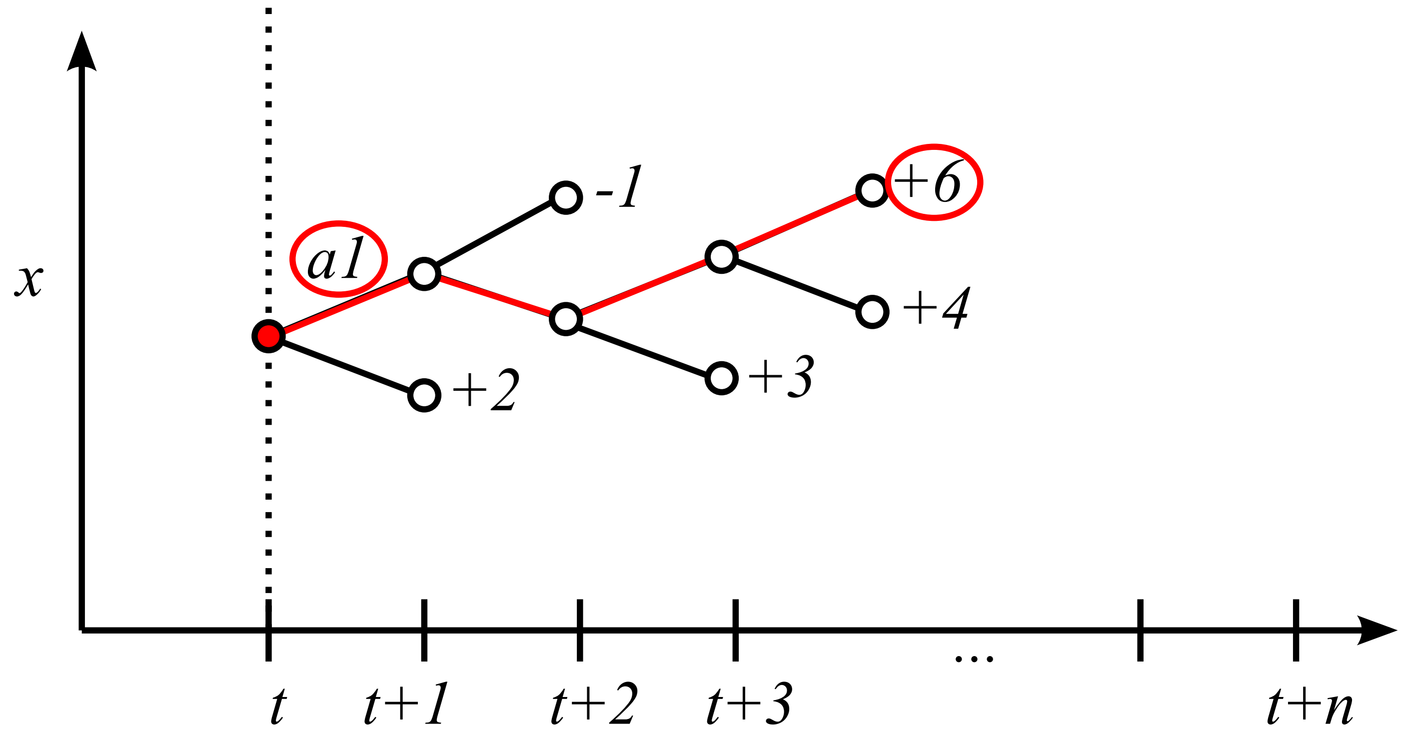
- Approaches that locally build a tree of limited depth every time a decision is required:
- Compute scores for the leaf nodes:



Motivation II

II. Lookahead tree policies (LT)

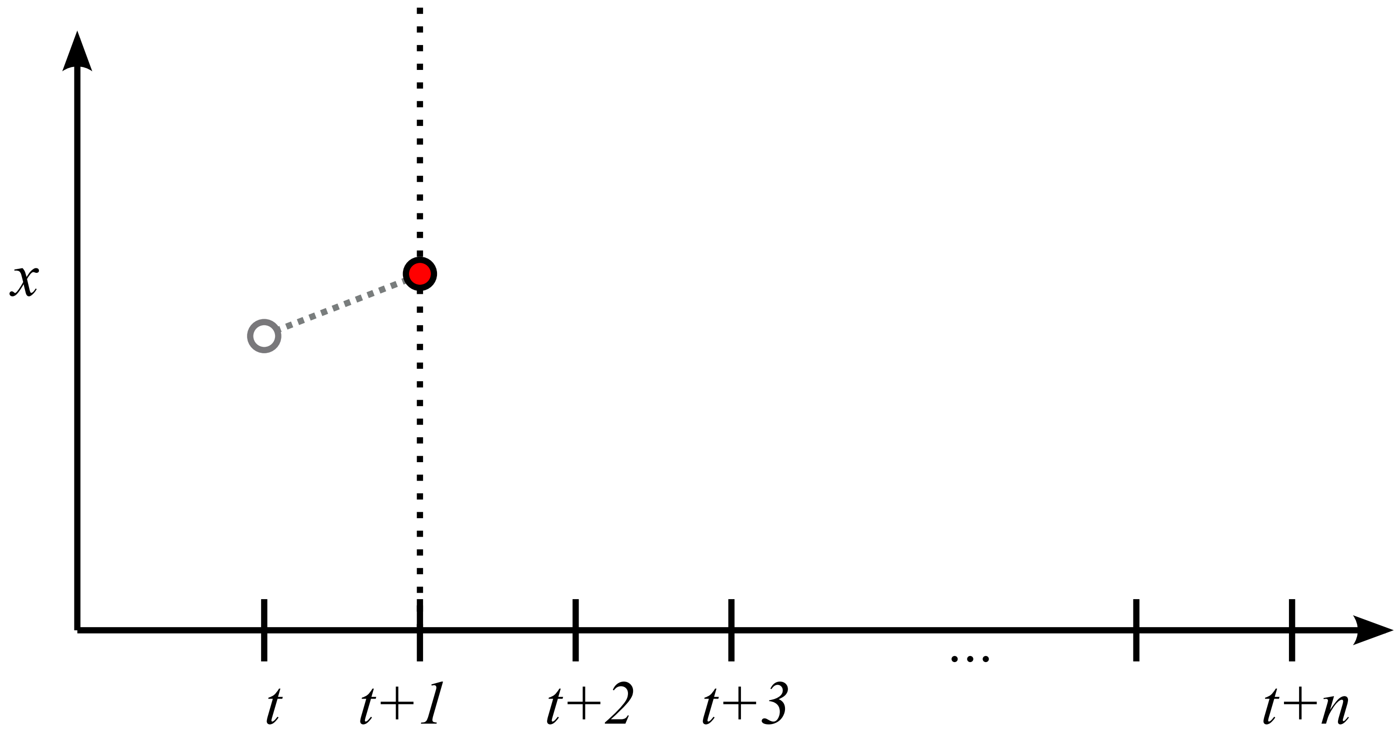
- Approaches that locally build a tree of limited depth every time a decision is required:
- Find best first action (leading to path with highest score):



Motivation II

II. Lookahead tree policies (LT)

- Approaches that locally build a tree of limited depth every time a decision is required:
- Make transition under best first action, discard tree, and start all over:



The good and the bad

Direct policy search

- $\oplus$ **Simple:** technically not very demanding (everybody can do it)
- $\oplus$ **Fairly powerful:** tends to produce good results, even for difficult nonlinear control problems
- $\oplus$ **Cheap:** low computational cost when deploying during runtime (online cost)
- $\ominus$ **Trial & error fiddling:** finding the right features g (offline cost)

The good and the bad

Direct policy search

- ⊕ **Simple:** technically not very demanding (everybody can do it)
- ⊕ **Fairly powerful:** tends to produce good results, even for difficult nonlinear control problems
- ⊕ **Cheap:** low computational cost when deploying during runtime (online cost)
- ⊖ **Trial & error fiddling:** finding the right features g (offline cost)

Lookahead tree policies

- ⊕ **Simple:** completely independent of the dimensionality of the state!
- ⊕ **Powerful:** excellent results, even for difficult nonlinear control problems
- ⊕ **Hassle-free:** can be deployed out-of-the-box (zero offline cost)
- ⊖ **Expensive:** large online cost to build trees of 'sufficient' depth

The good and the bad

Direct policy search

- ⊕ **Simple:** technically not very demanding (everybody can do it)
- ⊕ **Fairly powerful:** tends to produce good results, even for difficult nonlinear control problems
- ⊕ **Cheap:** low computational cost when deploying during runtime (online cost)
- ⊖ **Trial & error fiddling:** finding the right features g (offline cost)

Lookahead tree policies

- ⊕ **Simple:** completely independent of the dimensionality of the state!
- ⊕ **Powerful:** excellent results, even for difficult nonlinear control problems
- ⊕ **Hassle-free:** can be deployed out-of-the-box (zero offline cost)
- ⊖ **Expensive:** large online cost to build trees of 'sufficient' depth

Can we somehow combine the advantages of both?

LT+DPS=OLT

Point is: LT policies require *large* trees because they rely on generic heuristics.

LT+DPS=OLT

Point is: LT policies require *large* trees because they rely on generic heuristics.

Hence our idea: optimized lookahead tree policies (OLT)

- Parameterize the heuristics governing how the tree is build and scored, e.g.,

$$h(\underbrace{\mathtt{n}}_{\text{node}};\theta) = \sum_i \underbrace{\theta_i}_{\text{parameter}} \underbrace{g_i(\mathtt{n})}_{\text{feature}}$$

- **Offline:** optimize θ via global optimization for given domain *and* budget.
  (e.g., stochastic search, genetic algorithm, or my favorite: Gaussian process optimization)
- **Online:** use this θ and deploy the policy $\pi_{f,\varrho}(\cdot;\theta)$
- OLT can be seen as standard direct policy search with a nonstandard form of implicit policy representation.

LT+DPS=OLT

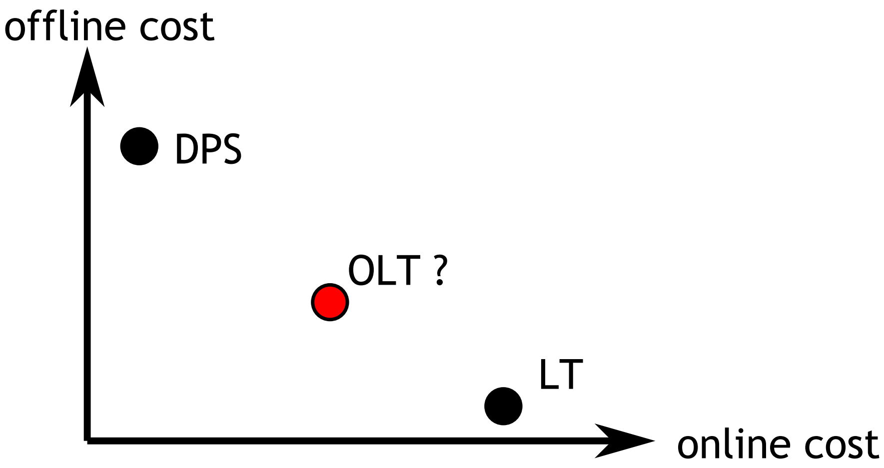
Point is: LT policies require *large* trees because they rely on generic heuristics.

Hence our idea: optimized lookahead tree policies (OLT)

- Parameterize the heuristics governing how the tree is build and scored, e.g.,

$$h(\underbrace{\mathtt{n}}_{\text{node}};\theta) = \sum_i \underbrace{\theta_i}_{\text{parameter}} \underbrace{g_i(\mathtt{n})}_{\text{feature}}$$

- **Offline:** optimize θ via global optimization for given domain *and* budget.
  (e.g., stochastic search, genetic algorithm, or my favorite: Gaussian process optimization)
- **Online:** use this θ and deploy the policy $\pi_{f,\varrho}(\cdot;\theta)$
- OLT can be seen as standard direct policy search with a nonstandard form of implicit policy representation.



OLT for finite and small action spaces

Basic algorithm I

How to parameterize the tree building process:

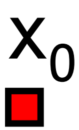
Notation:

- Let x_t be the current state for which we want to compute action $\pi_{f,\varrho}(x_t;\theta)$.
- Let $\mathcal{T}$ be the list of open/unexplored nodes.
- Every node corresponds to a sequence of actions applied from the root.
- Every node $\mathtt{n} \in \mathcal{T}$ is a `struct` object of type $\aleph$ with members:
  - $\mathtt{n}.x$ the underlying state
  - $\mathtt{n}.d$ the depth from the root
  - $\mathtt{n}.\varrho$ the incoming reward obtained from the parent
  - $\mathtt{n}.r$ cumulative reward on path root$\rightarrow$ $\mathtt{n}.x$.
  - $\mathtt{n}.\pi$ first action on path

Algorithm:

- While curr_expansions $<$ budget
  1. Find node with highest expansion score: $\mathtt{n}^* := \mathrm{argmax}_{\mathtt{n}\in\mathcal{T}}\,\mathrm{exp\_score}(\mathtt{n};\theta)$.
  2. For each action a
     - Simulate transition from $(\mathtt{n}^*.x, a)$: $x' = f(\mathtt{n}^*.x, a)$.
     - Add new node for x' and compute its expansion score.
  3. Remove $\mathtt{n}^*$.
- Return policy action: $\pi_{f,\varrho}(x_t;\theta)$:
  1. Find node with highest action selection score: $\mathtt{n}^* := \mathrm{argmax}_{\mathtt{n}\in\mathcal{T}}\,\mathrm{act\_score}(\mathtt{n};\theta)$.
  2. Return first action: $\pi_{f,\varrho}(x_t;\theta) = \mathtt{n}^*.\pi$

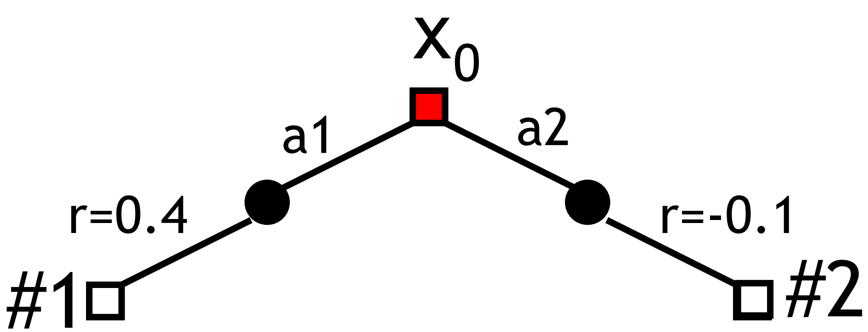
Illustration



List

Node exp_score

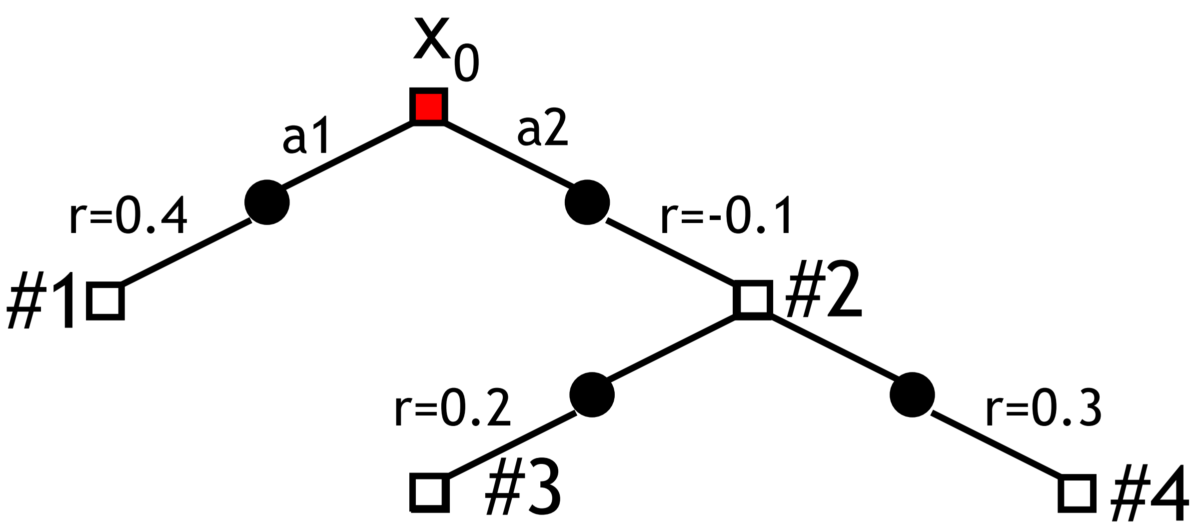
Illustration



List

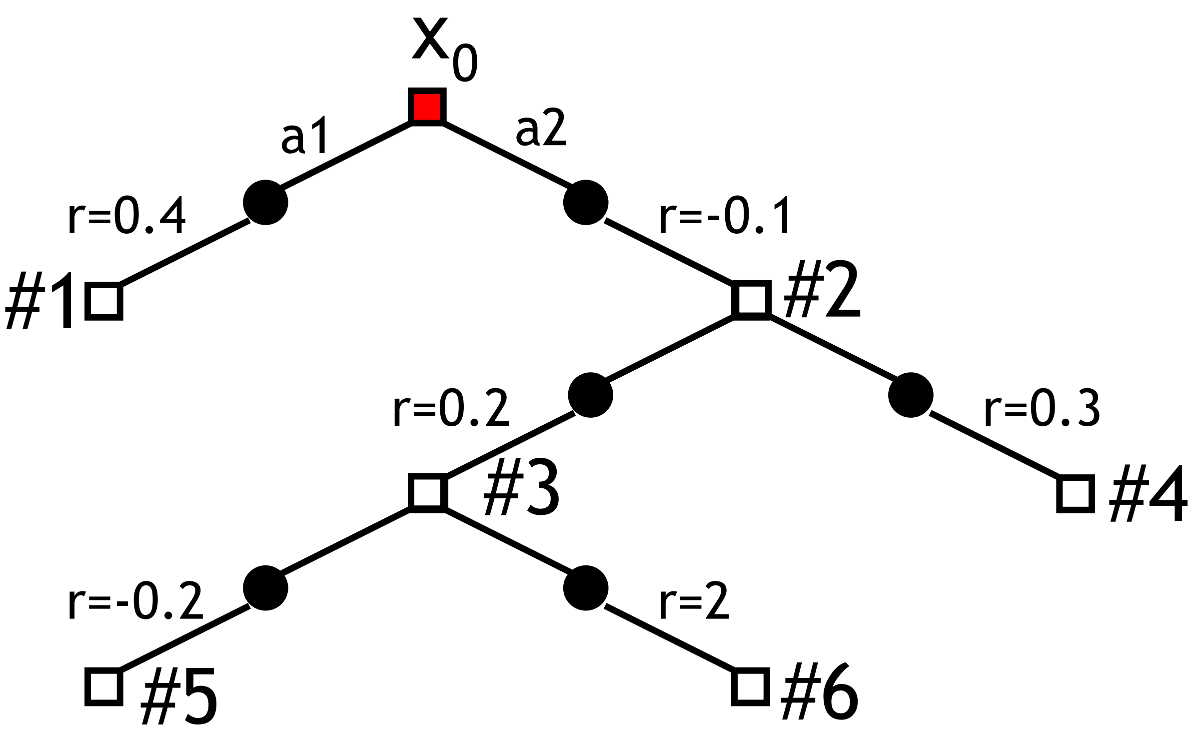
Node	exp_score
#1	0.45
#2	0.97

Illustration



List

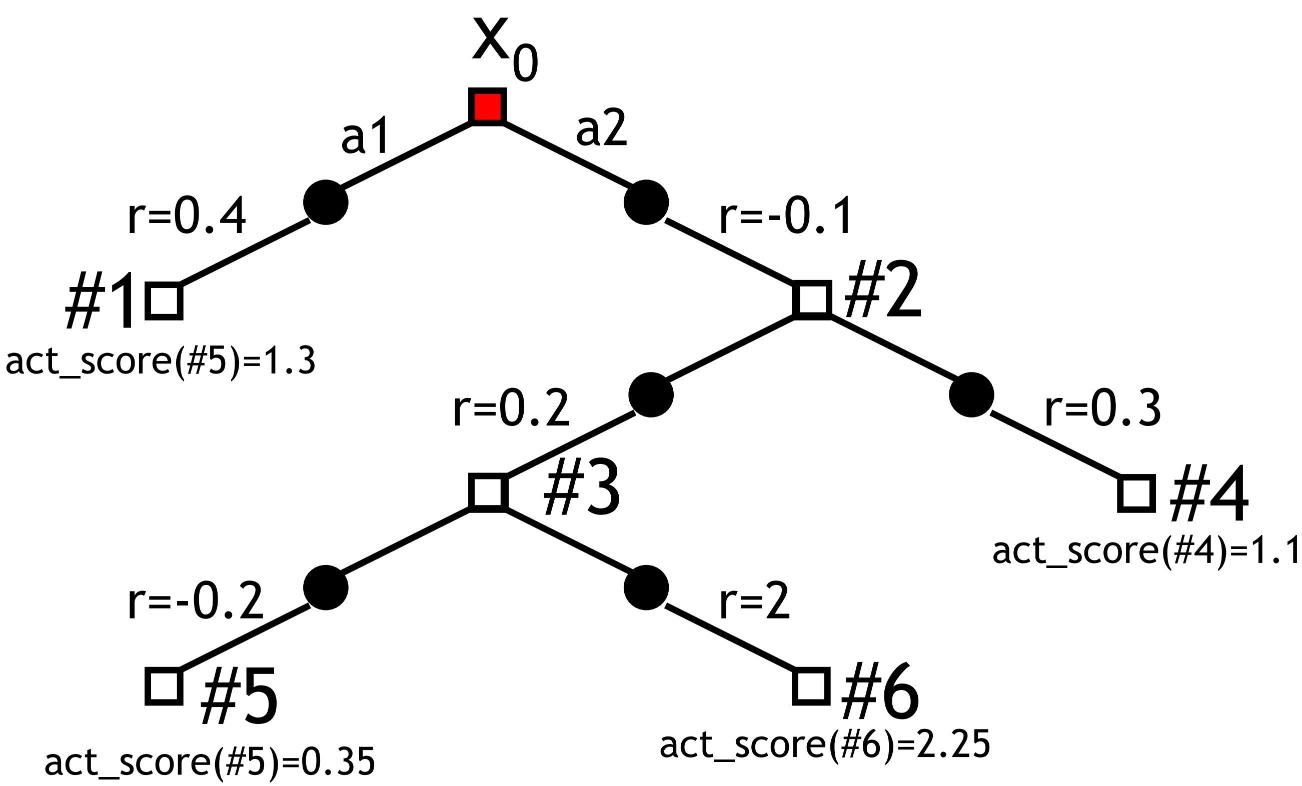
Node	exp_score
#1	0.45
~~#2~~	~~0.97~~
#3	2.01
#4	1.98



List

Node	exp_score
#1	0.45
~~#2~~	~~0.97~~
~~#3~~	~~2.01~~
#4	1.98
#5	1.97
#6	1.72

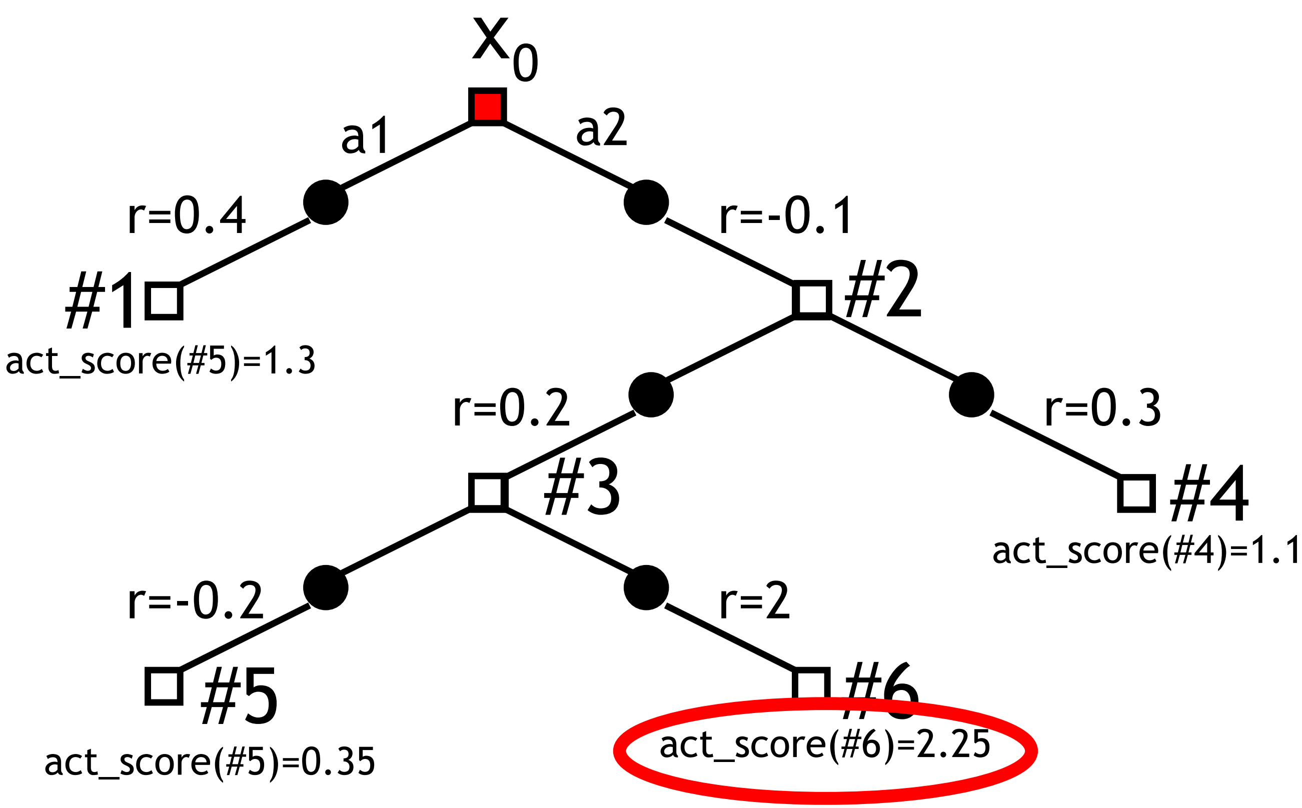
Illustration



List

Node	exp_score
#1	0.45
~~#2~~	~~0.97~~
~~#3~~	~~2.01~~
#4	1.98
#5	1.97
#6	1.72

Illustration



List

Node	exp_score
#1	0.45
~~#2~~	~~0.97~~
~~#3~~	~~2.01~~
#4	1.98
#5	1.97
#6	1.72

Tree building heuristics

Note: conceptually, all LT policies can be specified by implementing the two abstract functions

- Node expansion heuristic: determines how the tree is build.
  (Assigns scores to intermediate nodes; the node with the highest score gets expanded next.)
- Action selection heuristic: once the tree is built, determines what the best first action is.
  (Assigns scores to leaf nodes; best first action is the one that lies on the path to the highest scored leaf node.)

Examples: here are some generic (domain-independent) choices

- **Uniform:** (Maes et al., EWRL'11)

$$\begin{aligned} \text{exp\_score}(\mathtt{n}) &= -\mathtt{n}.d \quad \text{(breadth first)} \\ \text{act\_score}(\mathtt{n}) &= \mathtt{n}.r + R_{min}\gamma^{\mathtt{n}.d}/(1-\gamma) \end{aligned}$$

- **Greedy-1:** (Maes et al., EWRL'11)

$$\begin{aligned} \text{exp\_score}(\mathtt{n}) &= \mathtt{n}.\varrho \\ \text{act\_score}(\mathtt{n}) &= \mathtt{n}.r + R_{min}\gamma^{\mathtt{n}.d}/(1-\gamma) \end{aligned}$$

- **U-score:** (upper and lower bounds for $V^*(x_t)$) (Hren & Munos., EWRL'08)

$$\begin{aligned} \text{exp\_score}(\mathtt{n}) &= \mathtt{n}.r + R_{max}\gamma^{\mathtt{n}.d}/(1-\gamma) \\ \text{act\_score}(\mathtt{n}) &= \mathtt{n}.r + R_{min}\gamma^{\mathtt{n}.d}/(1-\gamma) \end{aligned}$$

Parameterized tree building heuristics

How do we parameterize these heuristics?

- **Node expansion heuristic:** is a simple linearly parameterized function

$$\text{exp\_score}(\mathbf{n};\theta) = \sum_{i=1}^{3}\sum_{j=1}^{n_x} \theta_{ij} g_{ij}(\mathbf{n})$$

with

$$\begin{aligned} g_{1j}(\mathbf{n}) &= \mathbf{n}.x_j \\ g_{2j}(\mathbf{n}) &= \mathbf{n}.x_j \cdot \mathbf{n}.r \\ g_{3j}(\mathbf{n}) &= \mathbf{n}.x_j \cdot \mathbf{n}.d \end{aligned}$$

Note that we use in all our experiments the same parameterization.

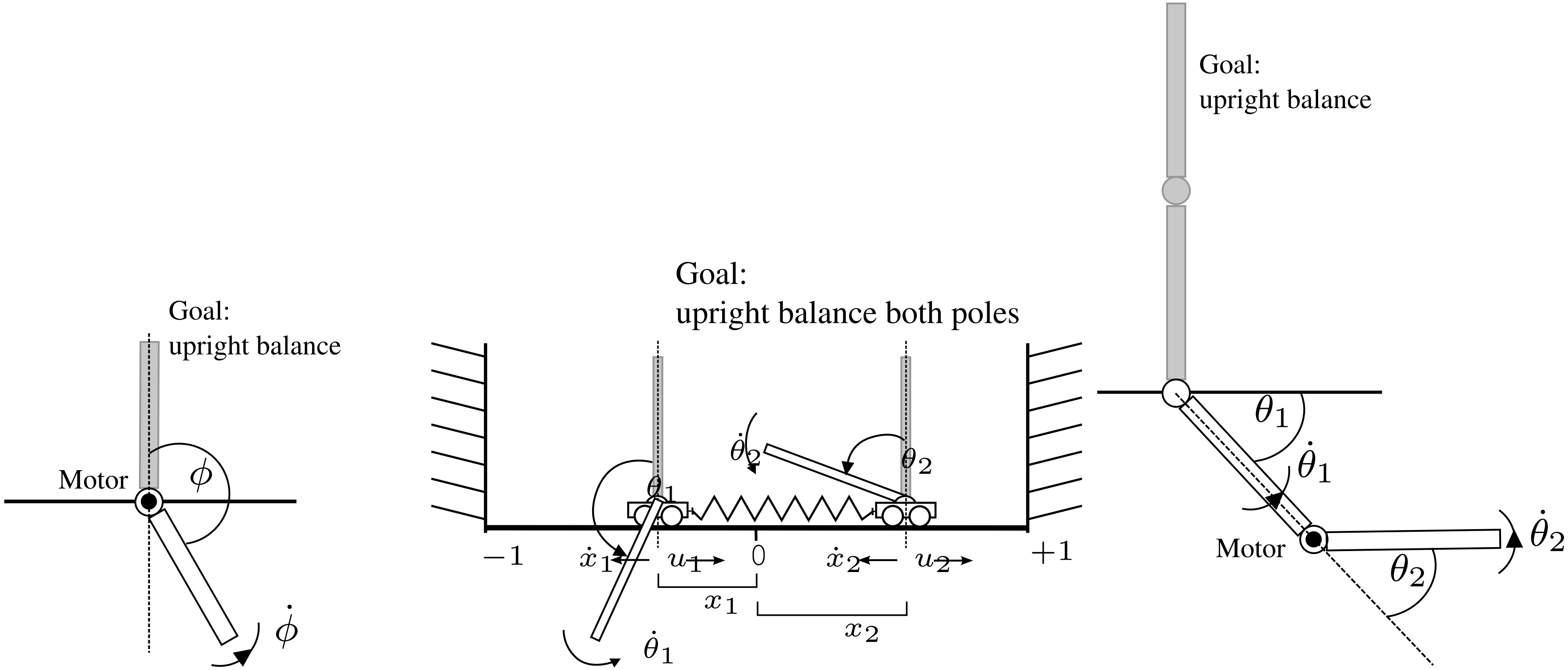
- **Action selection heuristic:** same as before in (Maes et al., EWRL'11),(Hren & Munos, EWRL'08)

$$\text{act\_score}(\mathbf{n};\theta) \equiv \text{act\_score}(\mathbf{n}) = \mathbf{n}.r + \gamma^{\mathbf{n}.d} R_{min}/(1-\gamma)$$

(Thus does not depend on any parameters.)

Experiments

The benchmark domains



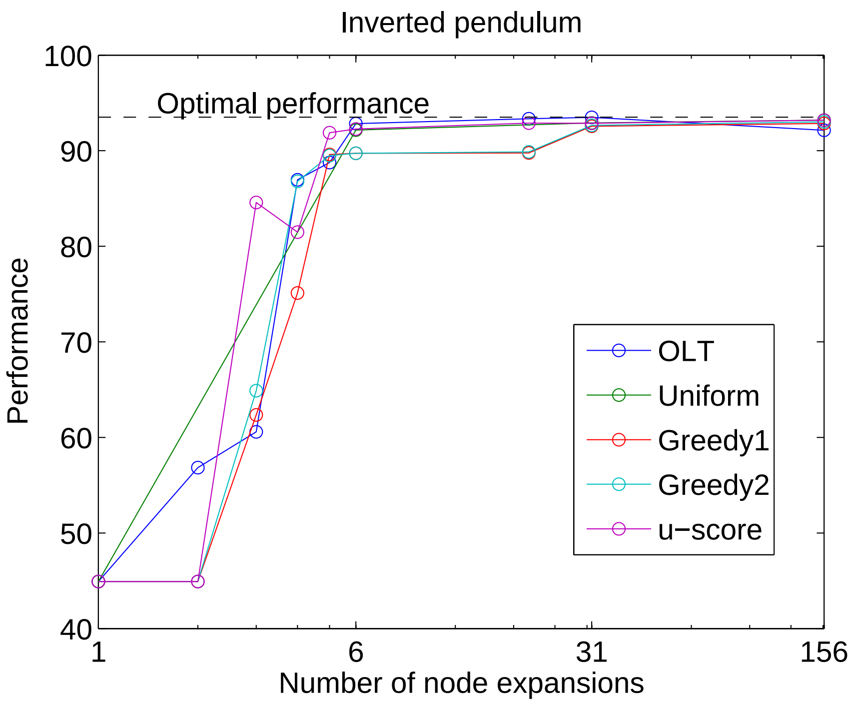
The domains:

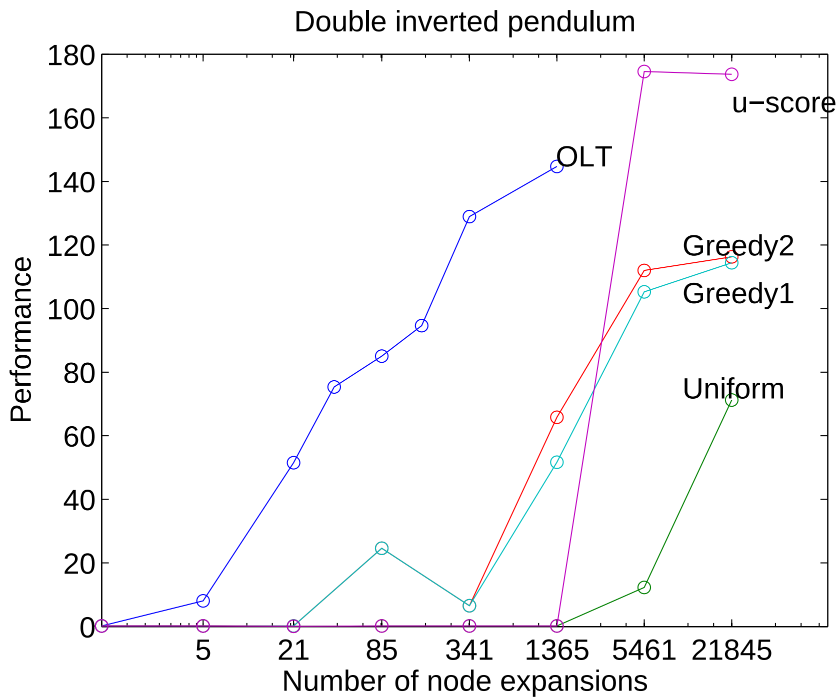
- **Inverted pendulum:** 2-dimensional state space, 5 actions (discretized)
- **Double pendulum:** 8-dimensional state space, 4 actions (discretized)
- **Acrobot handstand:** 4-dimensional state space, 3 actions (discretized + LQR-balance)
- **HIV STI drug treatment:** 6-dimensional state space, 4 actions (discretized)

  (Adams et al., Math. Biosciences and Engineering, Vol.1, 2004)

Results 1: OLT vs LT

Experimental protocol: OLT is optimized using Gaussian process optimization (500 fn evals)

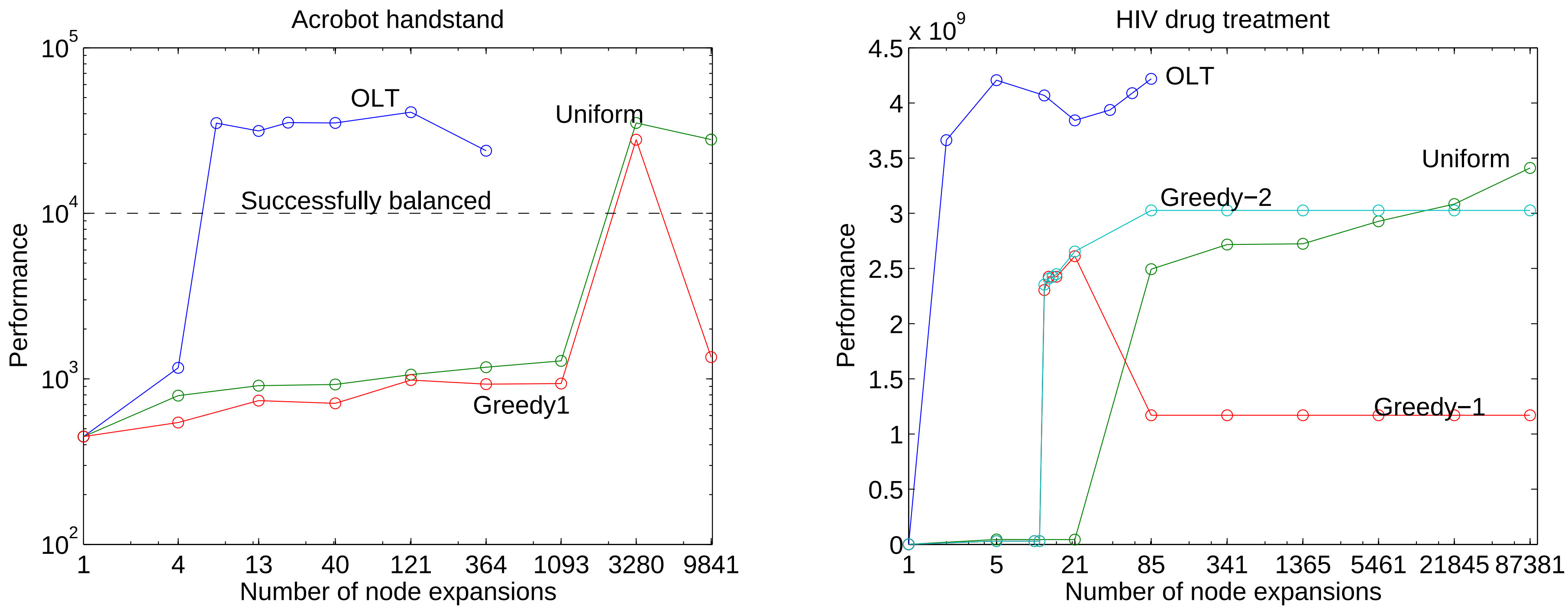




Comparison in terms of online complexity: how much does optimizing help as opposed to using the generic heuristics?

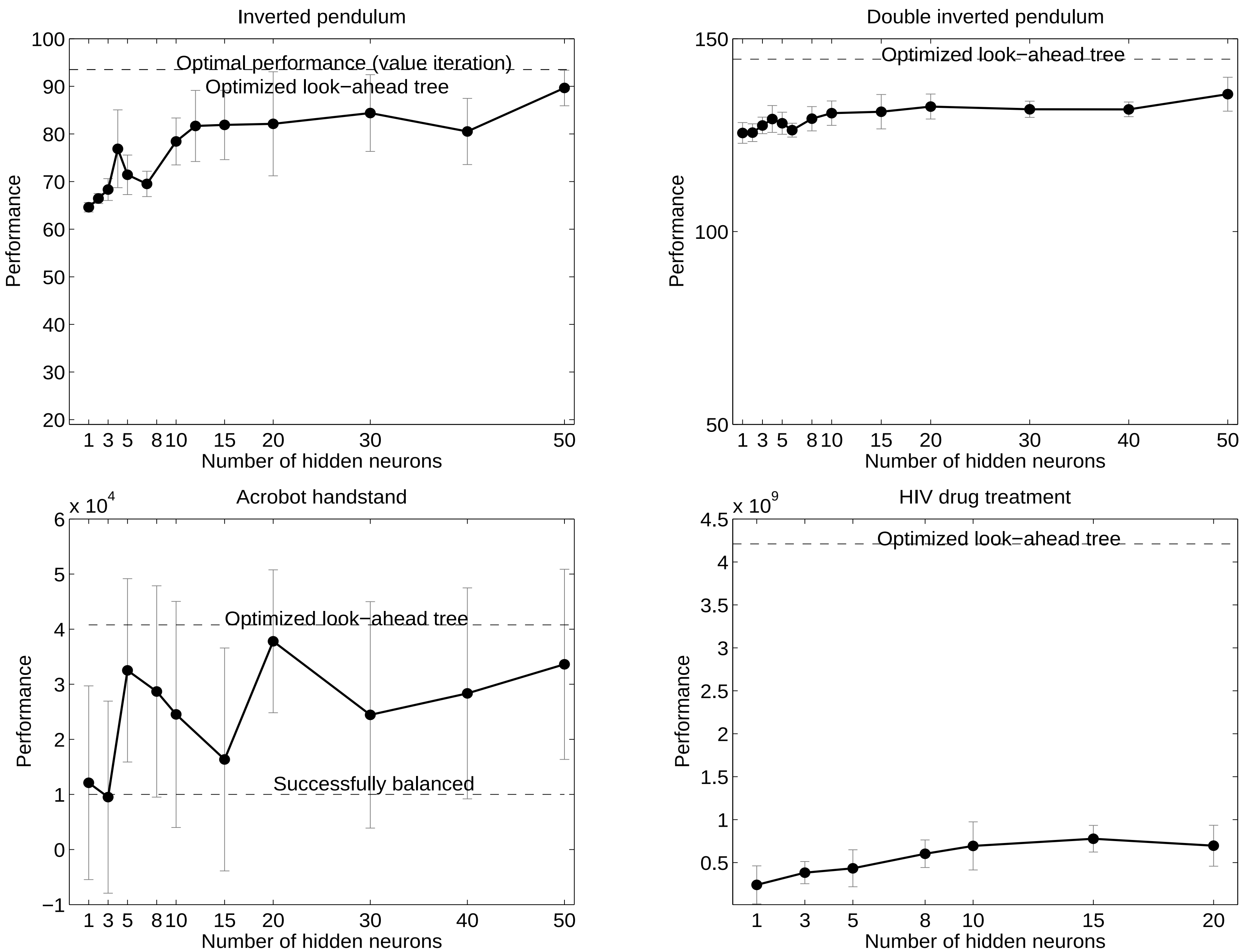
Results 2: OLT vs LT

Experimental protocol: OLT is optimized using Gaussian process optimization (500 fn evals)



Comparison in terms of online complexity: how much does optimizing help as opposed to using the generic heuristics?

Results 3: OLT vs DPS



OLT for continuous action spaces

Recursive action splitting

Idea: turn the continuous-action problem into a discrete-action problem. Cf. (Pazis & Lagoudakis, ICML'09)

Transformed problem: $(f, \varrho) \longrightarrow (f', \varrho')$

- New state space = old state space + a partition of the action space
- New action space = $\{split\_left, split\_right, go\}$
  (which refine the partition of the action space of the current state)
- The center of a partition (in 1D an interval) encodes the action under 'go'
- New transition model:

$$\begin{aligned} f'((x,\alpha), split\_left) &= (x, left(\alpha)) \\ f'((x,\alpha), split\_right) &= (x, right(\alpha)) \\ f'((x,\alpha), go) &= (f(x, center(\alpha)), A) \end{aligned}$$

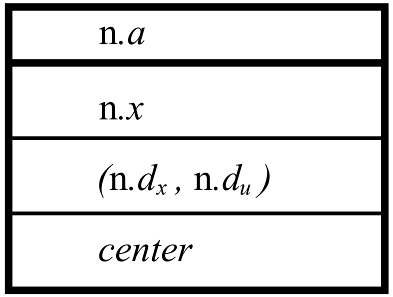
- New reward model:

$$\varrho'((x,\alpha), a) = \begin{cases} \varrho(x, center(\alpha)) & \text{if } a = go \\ 0 & \text{otherwise.} \end{cases}$$

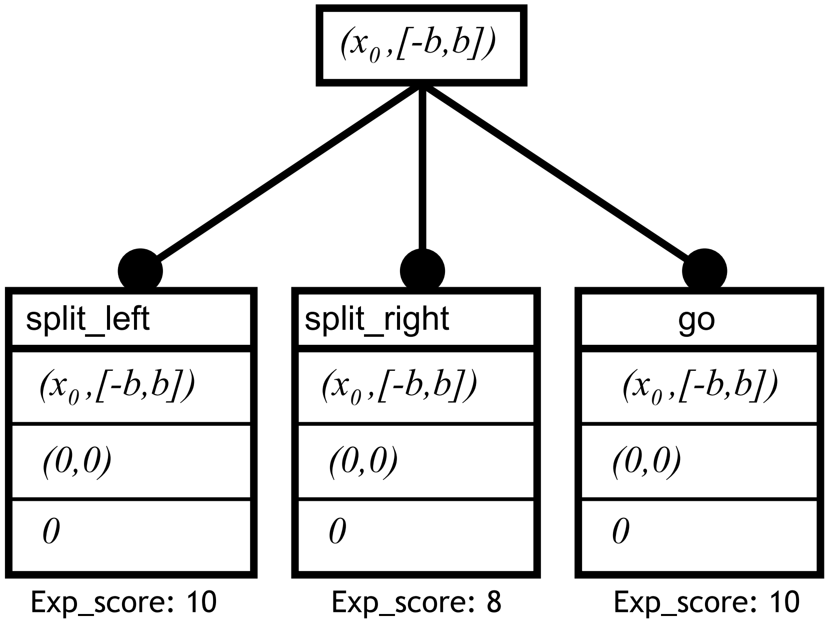
Illustration

Illustration

Nodes:

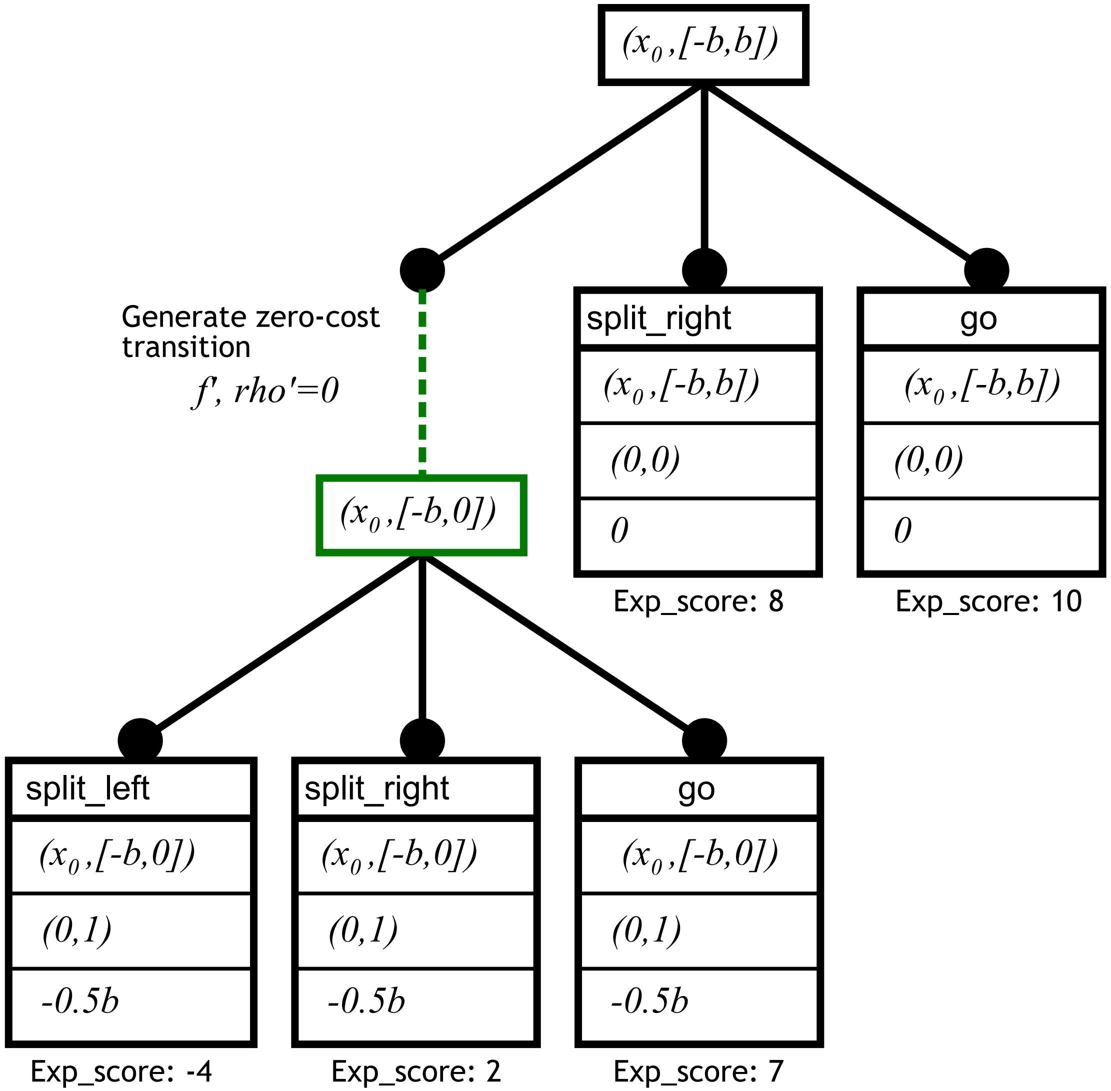


Initial look-ahead tree:



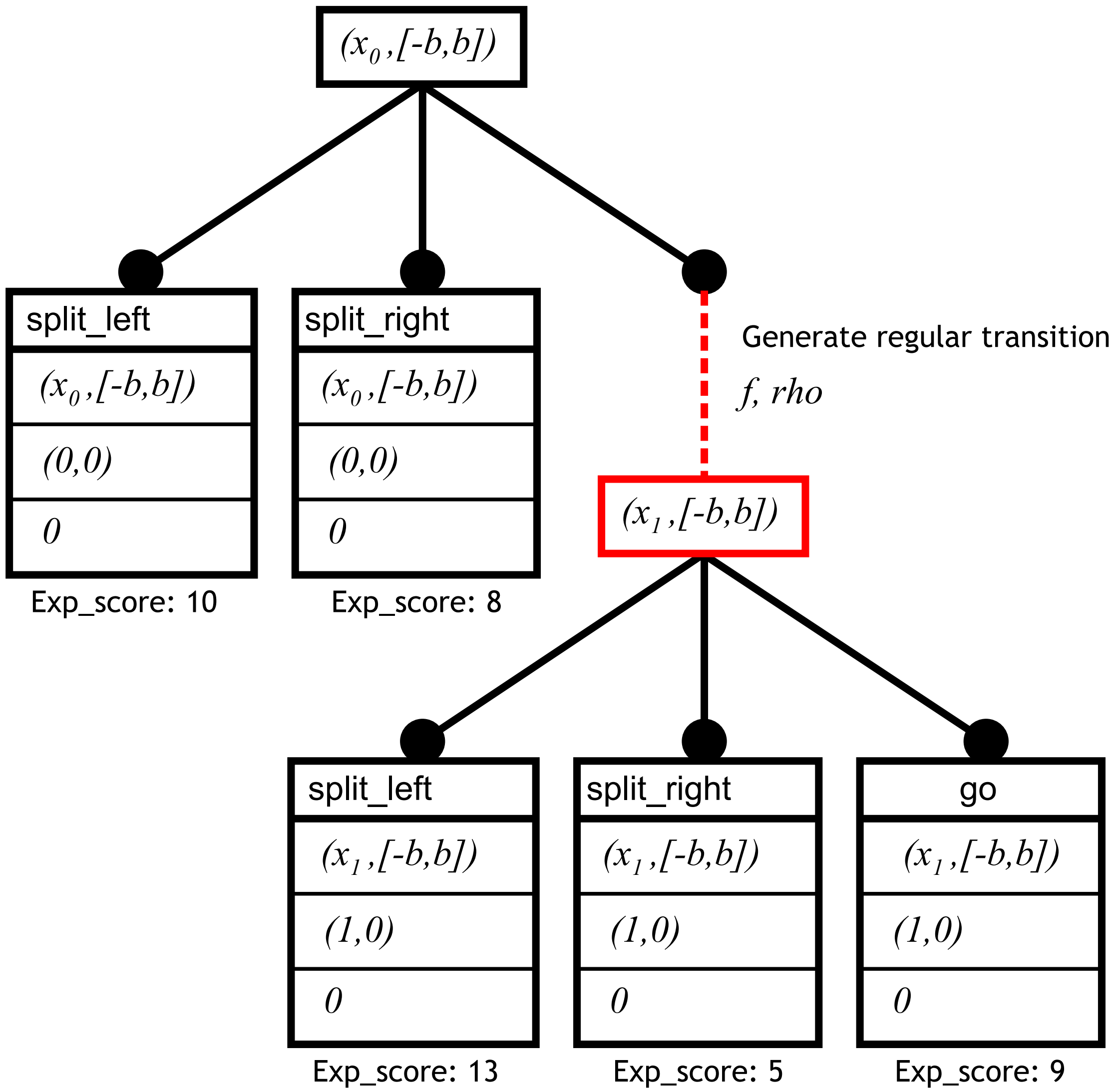
Illustration

Expanding the "split_left" node:

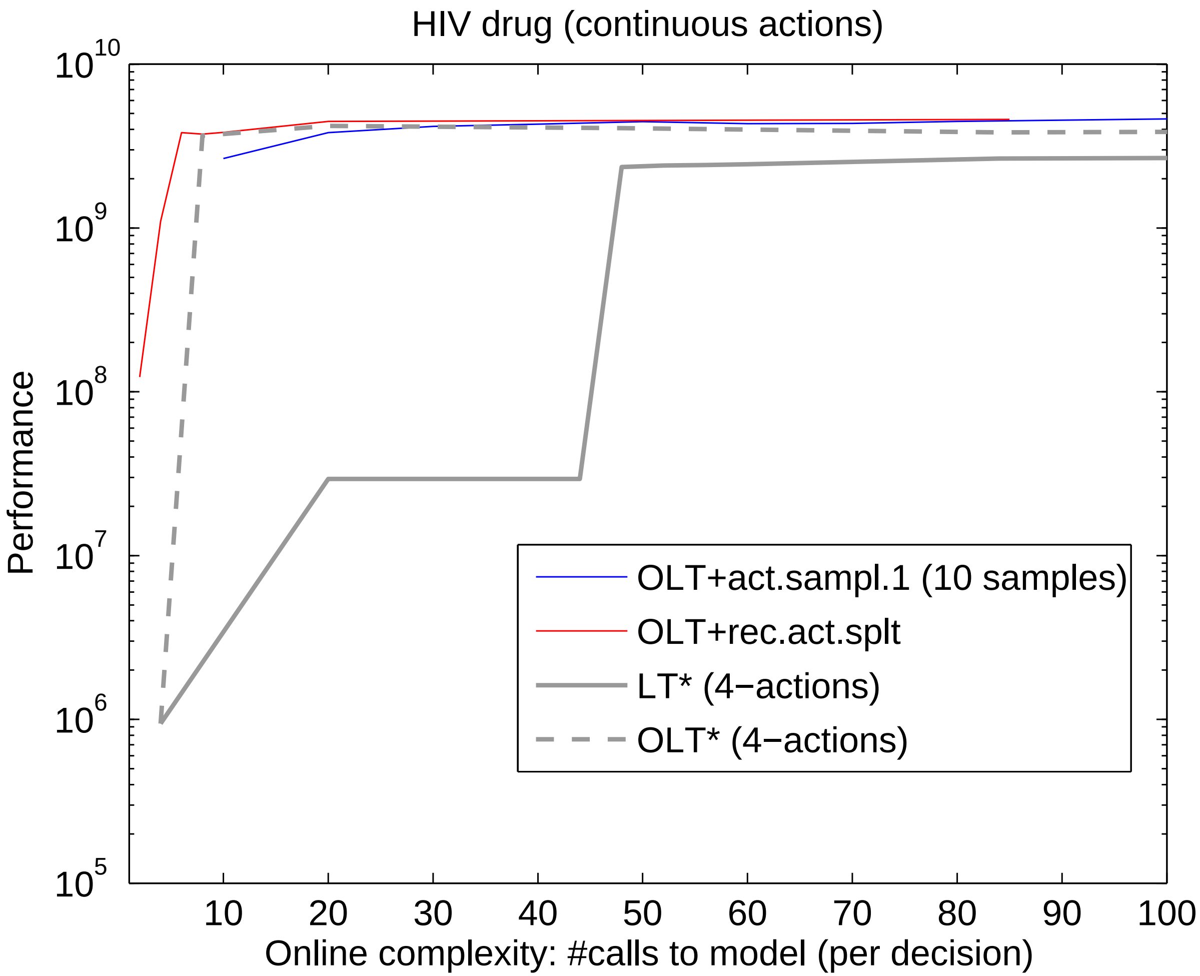


Illustration

Expanding the "go" node:



Results for HIV drug treatment



Some numbers:

Method	Perf.
4 actions	
FQI	4.22e9
LT	4.21e9
OLT	4.22e9
continuous actions	
OLT+rec.act.splt	4.78e9

Take-home message

3 simple ways of solving optimal control without worrying about value functions

Take-home message

3 simple ways of solving optimal control without worrying about value functions

	Performance	Online cost	Offline cost
Direct policy search	good – very good	very low	very high

Take-home message

3 simple ways of solving optimal control without worrying about value functions

	Performance	**Online cost**	**Offline cost**
Direct policy search	good – very good	very low	very high
Lookahead tree policies	very good	very high	zero

Take-home message

3 simple ways of solving optimal control without worrying about value functions

	Performance	Online cost	Offline cost
Direct policy search	good – very good	very low	very high
Lookahead tree policies	very good	very high	zero
Optimized lookahead tree policies	very good	low	medium – high