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 $(f, \varrho)$ (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

**Given** $(f, \varrho)$, we want to find the best policy $\pi^* = \arg\max_\pi V^\pi(x_0)$

**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.)
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:

1. **Direct policy search (DPS):**
   1. Parameterize the policy directly, e.g.,
      \[ \pi(x; \theta) = \tau\left(\sum \theta_i g_i(x)\right) \text{ or } \pi(x; \theta, \xi) = \tau\left(\sum \theta_i g_i(x; \xi)\right) \]
      where \( \tau \) is some transformation, \( g_i \) suitable features, and \( \theta, \xi \) parameters.
   2. Determine \( \theta \) via black-box global optimization (derivative-free):
      \[ \theta^* = \argmax_{\theta} \sum_{x_0 \in X_0} \sum_{t=0}^{H} \gamma^t q(x_t, \pi(x_t, \theta)) \text{ where } x_{t+1} = f(x_t, \pi(x_t; \theta)), \]
      \( X_0 \) 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.
II. Lookahead tree policies (LT)

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

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

\[
h(n; \theta) = \sum_{i} \theta_i g_i(n)\]

- Offline: optimize \( \theta \) via global optimization for given domain and budget.
  (e.g., stochastic search, genetic algorithm, or my favorite: Gaussian process optimization)

- Online: use this \( \theta \) and deploy the policy \( \pi_{f,g}(\cdot; \theta) \)

- OLT can be seen as standard direct policy search with a nonstandard form of implicit policy representation.
**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 built and scored, e.g.,

\[ h(n; \theta) = \sum_{i} \theta_i g_i(n) \]

- **Offline:** optimize \( \theta \) via global optimization for given domain and budget. (e.g., stochastic search, genetic algorithm, or my favorite: Gaussian process optimization)
- **Online:** use this \( \theta \) and deploy the policy \( \pi_{f,g}(\cdot; \theta) \)
- 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 $\pi_{f,\theta}(x_t; \theta)$.
- Let $\mathcal{T}$ be the list of open/unexplored nodes.
- Every node corresponds to a sequence of actions applied from the root.
- Every node $n \in \mathcal{T}$ is a struct object of type $\mathbb{N}$ with members:
  - $n.x$ the underlying state
  - $n.d$ the depth from the root
  - $n.\varrho$ the incoming reward obtained from the parent
  - $n.r$ cumulative reward on path root $\rightarrow n.x$
  - $n.\pi$ first action on path

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

<table>
<thead>
<tr>
<th>Node</th>
<th>exp_score</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>0.45</td>
</tr>
<tr>
<td>#2</td>
<td>0.97</td>
</tr>
</tbody>
</table>
Illustration

List

<table>
<thead>
<tr>
<th>Node</th>
<th>exp_score</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>0.45</td>
</tr>
<tr>
<td>#2</td>
<td>0.97</td>
</tr>
<tr>
<td>#3</td>
<td>2.01</td>
</tr>
<tr>
<td>#4</td>
<td>1.98</td>
</tr>
<tr>
<td>#5</td>
<td>1.97</td>
</tr>
<tr>
<td>#6</td>
<td>1.72</td>
</tr>
</tbody>
</table>
Illustration

List

<table>
<thead>
<tr>
<th>Node</th>
<th>exp_score</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>0.45</td>
</tr>
<tr>
<td>#2</td>
<td>0.97</td>
</tr>
<tr>
<td>#3</td>
<td>2.01</td>
</tr>
<tr>
<td>#4</td>
<td>1.98</td>
</tr>
<tr>
<td>#5</td>
<td>1.97</td>
</tr>
<tr>
<td>#6</td>
<td>1.72</td>
</tr>
</tbody>
</table>
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 built.
  (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)*
  \[\text{exp\_score}(n) = -n.d \quad \text{(breadth first)}\]
  \[\text{act\_score}(n) = n.r + R_{min} \gamma^{n.d} / (1 - \gamma)\]

- **Greedy-1:** *(Maes et al., EWRL’11)*
  \[\text{exp\_score}(n) = n.q\]
  \[\text{act\_score}(n) = n.r + R_{min} \gamma^{n.d} / (1 - \gamma)\]

- **U-score:** *(upper and lower bounds for \(V^*(x_t)\)) *(Hren & Munos., EWRL’08)*
  \[\text{exp\_score}(n) = n.r + R_{max} \gamma^{n.d} / (1 - \gamma)\]
  \[\text{act\_score}(n) = n.r + R_{min} \gamma^{n.d} / (1 - \gamma)\]
Parameterized tree building heuristics

How do we parameterize these heuristics?

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

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

with

\[ g_{1j}(n) = n.x_j \]
\[ g_{2j}(n) = n.x_j \cdot n.r \]
\[ g_{3j}(n) = n.x_j \cdot n.d \]

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)

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

(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)

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?
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

Inverted pendulum

Optimal performance (value iteration)
Optimized look–ahead tree

Performance vs Number of hidden neurons

Double inverted pendulum

Performance vs Number of hidden neurons

Acrobot handstand

Successfully balanced

Performance vs Number of hidden neurons

HIV drug treatment

Performance vs Number of hidden neurons
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:** \((f, \varrho) \rightarrow (f', \varrho')\)

- New state space = old state space + a partition of the action space
- New action space = \(\{\text{split\_left}, \text{split\_right}, \text{go}\}\)
  (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:

\[
\begin{align*}
    f'(x, \alpha, \text{split\_left}) &= (x, \text{left}(\alpha)) \\
    f'(x, \alpha, \text{split\_right}) &= (x, \text{right}(\alpha)) \\
    f'(x, \alpha, \text{go}) &= (f(x, \text{center}(\alpha)), A)
\end{align*}
\]

- New reward model:

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

**Nodes:**

- n.a
- n.x
- (n.d_x, n.d_y)
- center

**Initial look-ahead tree:**

- Root: (x_0, [-b, b])
  - Split left: (x_0, [-b, b])
    - (0, 0)
    - 0
    - Exp_score: 10
  - Split right: (x_0, [-b, b])
    - (0, 0)
    - 0
    - Exp_score: 8
  - Go: (x_0, [-b, b])
    - (0, 0)
    - 0
    - Exp_score: 10
Expanding the "split_left" node:

Generate zero-cost transition
\( f', \rho' = 0 \)

\( (x_0, [-b, 0]) \)

Exp_score: 8

<table>
<thead>
<tr>
<th>split_left</th>
<th>split_right</th>
<th>go</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_0, [-b, 0])</td>
<td>(x_0, [-b, b])</td>
<td>(x_0, [-b, b])</td>
</tr>
<tr>
<td>(0, 1)</td>
<td>(0, 0)</td>
<td>(0, 0)</td>
</tr>
<tr>
<td>-0.5b</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Exp_score: -4
Exp_score: 2
Exp_score: 7
Expanding the "go" node:

\[(x_0, [-b, b])\]

\[\text{split_left}\]
\[(x_0, [-b, b])\]
\[(0,0)\]
\[0\]
Exp_score: 10

\[\text{split_right}\]
\[(x_0, [-b, b])\]
\[(0,0)\]
\[0\]
Exp_score: 8

\[\text{Generate regular transition}\]
\[f, \rho\]

\[(x_1, [-b, b])\]

\[\text{split_left}\]
\[(x_1, [-b, b])\]
\[(1,0)\]
\[0\]
Exp_score: 13

\[\text{split_right}\]
\[(x_1, [-b, b])\]
\[(1,0)\]
\[0\]
Exp_score: 5

\[\text{go}\]
\[(x_1, [-b, b])\]
\[(1,0)\]
\[0\]
Exp_score: 9
### Results for HIV drug treatment

#### Performance

<table>
<thead>
<tr>
<th>Method</th>
<th>Perf.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>4 actions</strong></td>
<td></td>
</tr>
<tr>
<td>FQI</td>
<td>4.22e9</td>
</tr>
<tr>
<td>LT</td>
<td>4.21e9</td>
</tr>
<tr>
<td>OLT</td>
<td>4.22e9</td>
</tr>
<tr>
<td><strong>Continuous actions</strong></td>
<td></td>
</tr>
<tr>
<td>OLT+rec.act.splt</td>
<td>4.78e9</td>
</tr>
</tbody>
</table>

**Online complexity:** #calls to model (per decision)

---

Some numbers:

<table>
<thead>
<tr>
<th>Method</th>
<th>Perf.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>4 actions</strong></td>
<td></td>
</tr>
<tr>
<td>FQI</td>
<td>4.22e9</td>
</tr>
<tr>
<td>LT</td>
<td>4.21e9</td>
</tr>
<tr>
<td>OLT</td>
<td>4.22e9</td>
</tr>
<tr>
<td><strong>Continuous actions</strong></td>
<td></td>
</tr>
<tr>
<td>OLT+rec.act.splt</td>
<td>4.78e9</td>
</tr>
</tbody>
</table>
3 simple ways of solving optimal control without worrying about value functions
3 simple ways of solving optimal control without worrying about value functions

<table>
<thead>
<tr>
<th>Direct policy search</th>
<th>Performance</th>
<th>Online cost</th>
<th>Offline cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>good – very good</td>
<td>very low</td>
<td>very high</td>
</tr>
</tbody>
</table>
### Take-home message

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

<table>
<thead>
<tr>
<th></th>
<th>Performance</th>
<th>Online cost</th>
<th>Offline cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct policy search</td>
<td>good – very good</td>
<td>very low</td>
<td>very high</td>
</tr>
<tr>
<td>Lookahead tree policies</td>
<td>very good</td>
<td>very high</td>
<td>zero</td>
</tr>
</tbody>
</table>
### Take-home message

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

<table>
<thead>
<tr>
<th></th>
<th>Performance</th>
<th>Online cost</th>
<th>Offline cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct policy search</td>
<td>good – very good</td>
<td>very low</td>
<td>very high</td>
</tr>
<tr>
<td>Lookahead tree policies</td>
<td>very good</td>
<td>very high</td>
<td>zero</td>
</tr>
<tr>
<td>Optimized lookahead tree policies</td>
<td>very good</td>
<td>low</td>
<td>medium – high</td>
</tr>
</tbody>
</table>