16-782
Planning & Decision-making in Robotics

Learning in (Search-based) Planning

Maxim Likhachev
Robotics Institute
Carnegie Mellon University
Going into the Real-world

- Robot models and simple world interactions can be pre-encoded
- Planning on those models enables the robots to operate under benign/narrow conditions right away

- Real-world: real-time + going beyond what’s given
Learning in Search-based Planning

Speeding up planning

Learning cost function

Going beyond the prior model

Re-use of previous results within search (Phillips et al., ’12; Islam et al., ’18)
Learning heuristic functions (Bhardwaj et al., ’17; Paden & Frazzoli, ’17; Thayer et al., ’11)
Learning order of expansions (Choudhary et al., ’17)
Learning in Search-based Planning

Speeding up planning

Learning cost function

Going beyond the prior model

Crusher (from Ratliff et al., ‘09 paper)

Learning a cost function from demonstrations (Ratliff et al., ’09; Wulfmeier et al., ’17)
Learning in Search-based Planning

**Speeding up planning**

**Learning cost function**

**Going beyond the prior model**

*Online adaptation/learning of a prior model (e.g., Ordonez et al., ‘17)*

*Learning additional dimensions to reason over (Phillips et al., ’13)*

*Planning over learned skills (G. Konidaris et al., ‘18)*
Learning in Search-based Planning

Speeding up planning

Learning cost function

Going beyond the prior model

Re-use of previous results within search (Phillips et al., ’12; Islam et al., ’18)
Learning heuristic functions (Bhardwaj et al., ’17; Paden & Frazzoli, ’17; Thayer et al., ’11)
Learning order of expansions (Choudhary et al., ’17)
Experience Graphs [Phillips et al., RSS’12]

• Many planning tasks are repetitive
  - loading a dishwasher
  - opening doors
  - moving objects around a warehouse
  - …

• Can we re-use prior experience to accelerate planning, in the context of search-based planning?

• Especially useful for high-dimensional problems such as mobile manipulation!
Experience Graphs [Phillips et al., RSS’12]

Given a set of previous paths (experiences)…
Experience Graphs [Phillips et al., RSS’12]

Put them together into an $E$-graph (Experience graph)
Experience Graphs [Phillips et al., RSS’12]

Given a new planning query…
Experience Graphs [Phillips et al., RSS’12]

…would like to re-use E-graph to speed up planning in similar situations
Experience Graphs [Phillips et al., RSS’12]

…would like to re-use E-graph to speed up planning in similar situations

Re-use is via focusing search with a recomputed $h^\varepsilon()$ heuristic function:

$$h^\varepsilon(s_0) = \min_{\pi} \sum_{i=0}^{N-1} \min\{\varepsilon^\varepsilon h^G_i(s_i, s_{i+1}), c^\varepsilon(s_i, s_{i+1})\}$$
Experience Graphs [Phillips et al., RSS’12]

Re-use is via focusing search with a recomputed heuristic function:

$$h^\varepsilon(s_0) = \min_\pi \sum_{i=0}^{N-1} \min \{ \varepsilon h^G(s_i, s_{i+1}), c^\varepsilon(s_i, s_{i+1}) \}$$

**General idea:**

*Instead of biasing the search towards the goal, heuristics $h^\varepsilon(s)$ biases it towards a set of paths in Experience Graph*
Experience Graphs [Phillips et al., RSS’12]

Can be computed via a single Dijkstra’s search on the Experience Graph

\[ h^\varepsilon(s_0) = \min_\pi \sum_{i=0}^{N-1} \min \{ \varepsilon h^G(s_i, s_{i+1}), c^\varepsilon(s_i, s_{i+1}) \} \]
Experience Graphs [Phillips et al., RSS’12]

…would like to re-use E-graph to speed up planning in similar situations

Re-use is via focusing search with a recomputed $h^\varepsilon()$ heuristic function:

$$h^\varepsilon(s_0) = \min_\pi \sum_{i=0}^{N-1} \min\{\varepsilon^\varepsilon h^G(s_i, s_{i+1}), c^\varepsilon(s_i, s_{i+1})\}$$

heuristics $h^\varepsilon(s)$ is guaranteed to be $\varepsilon$-consistent
Experience Graphs [Phillips et al., RSS’12]

…would like to re-use E-graph to speed up planning in similar situations

Re-use is via focusing search with a recomputed $h^\varepsilon()$ heuristic function:

$$h^\varepsilon(s_0) = \min_{\pi} \sum_{i=0}^{N-1} \min\{\varepsilon h^G(s_i, s_{i+1}), c^\varepsilon(s_i, s_{i+1})\}$$

**Theorem 1:** Algorithm is complete with respect to the original graph

**Theorem 2:** The cost of the solution is within a given bound on sub-optimality
Learning in Search-based Planning

Speeding up planning

Learning cost function

Going beyond the prior model

Crusher (from Ratliff et al., ’09 paper)

*Learning a cost function from demonstrations (Ratliff et al., ’09; Wulfmeier et al., ’17)*
A bit of terminology

- Imitation Learning/Apprenticeship Learning/Learning from Demonstrations/Robot Programming by Demonstrations
  - Methods for programming robot behavior via demonstrations [Schaal & Atkeson, ‘94], [Abbeel & Ng, ’04], [Pomerleau et al., ‘89], [Ratliff & Bagnell, ‘06], [Billard, Calinon & Dillmann, ’13], [Sammut et al., ‘92], …

- Major classes of Imitation Learning:
  - Learning policies directly from demonstrated trajectories or supervised learning [Schaal & Atkeson, ‘94], [Pomerleau et al., ‘89], …
  - Learning a cost function (or reward function) from demonstrations and then using it to generate plans (policies) [Abbeel & Ng, ’04], [Ratliff & Bagnell, ‘06], …
A bit of terminology

- Imitation Learning/Apprenticeship Learning/Learning from Demonstrations/Robot Programming by Demonstrations
  - Methods for programming robot behavior via demonstrations [Schaal & Atkeson, ‘94], [Abbeel & Ng, ’04], [Pomerleau et al., ‘89], [Ratliff & Bagnell, ‘06], [Billard, Calinon & Dillmann, ’13], [Sammut et al., ‘92],…

- Major classes of Imitation Learning:
  - Learning policies directly from demonstrated trajectories or supervised learning [Schaal & Atkeson, ‘94], [Pomerleau et al., ‘89],…
  - Learning a cost function (or reward function) from demonstrations and then using it to generate plans (policies) [Abbeel & Ng, ’04], [Ratliff & Bagnell, ‘06], …
• Recover a cost function that makes given demonstrations optimal plans [Ratliff, Silver & Bagnell, ’09]
Example

- Consider a (simple) outdoor navigation example

Modeled as graph search
Example

- Consider a (simple) outdoor navigation example

Can we teach the planner to avoid slippery areas and driving close to the cliff (without manually tweaking a cost function)?

Modeled as graph search
Example

- Consider a (simple) outdoor navigation example

Can we teach the planner to avoid slippery areas and driving close to the cliff (without manually tweaking a cost function)?

= learning the “right” cost function
Consider a (simple) outdoor navigation example

Can we teach the planner to avoid slippery areas and driving close to the cliff (without manually tweaking a cost function)?

A user gives $N$ demonstrations of what paths are good. We want a cost function for which these demonstrated trajectories are least-cost plans.
Example

- Consider a (simple) outdoor navigation example

Demonstration $d_1$ on graph $G_1$

Demonstration $d_2$ on graph $G_2$
Example

- Consider a (simple) outdoor navigation example

Demonstration $d_1$ on graph $G_1$

Cost function – a function of features $\Phi$: $c(s,s') = f(\phi(s,s'))$

Compute cost function that makes these demonstrations optimal paths

Demonstration $d_2$ on graph $G_2$

Why not learn edge costs directly?
Consider a (simple) outdoor navigation example

Demonstration $d_1$ on graph $G_1$

Cost function – a function of features $\Phi$: $c(s,s') = f(\Phi(s,s'))$

Demonstration $d_2$ on graph $G_2$

What $\Phi$ would make sense in this example?
Consider a (simple) outdoor navigation example.

Demonstration $d_1$ on graph $G_1$:
- Start $S$ to Goal $G$.

Cost function – a function of features $\Phi$: $c(s,s') = f(\phi(s,s'))$.

Compute cost function that makes these demonstrations optimal paths.

Demonstration $d_2$ on graph $G_2$:
- Start $S$ to Goal $G$.

Example of $f()$?

slippery area

cliff
• Consider a (simple) outdoor navigation example

**Demonstration d₁ on graph G₁**

Compute cost function that makes these demonstrations optimal paths

Cost function – a function of features $\Phi$: $c(s,s') = f(\phi(s,s'))$

**Example of $f()$?**

Most common example: $f(\phi(s,s')) = \Sigma w_i \phi_i(s,s')$

**Demonstration d₂ on graph G₂**
• Consider a (simple) outdoor navigation example

\[ f(\phi(s,s')) = \sum w_i \phi_i(s,s') \]

For example:
\[ \phi_0: 1/(distance \ to \ slippery \ area) \]
\[ \phi_1: 1/(distance \ to \ cliff) \]
\[ \phi_2: length \ of \ the \ transition \]

Need to compute (learn) \( w_0, w_1, w_2 \) based on demonstrations

\begin{itemize}
  \item \( \phi_0 \): 1/(distance to slippery area)
  \item \( \phi_1 \): 1/(distance to cliff)
  \item \( \phi_2 \): Length of the transition
\end{itemize}
LEARCH (LEArning to searCH)

[Ratliff, Silver, Bagnell, 09]

Given demonstrations \{d_1,...,d_N\} on graphs \{G_1,...,G_N\} and features function \Phi

Need to compute \(c(s,s') = f(\phi(s,s'))\) s.t. \(d_i = \arg\min_{\pi_i} \sum_{i=1}^{N} c(\pi_i)\)

While (Not Converged)

for \(i=1...N\)

update edge costs in graph \(G_i\) using the current function \(f(\phi(,))\)

plan an optimal path \(\pi_i^* = \arg\min_{\pi_i} \sum_{k=0}^{\text{length}(\pi_i)-1} c(s_k, s_{k+1})\)

increase \(f(\phi(,))\) for edges \((u,v)\) s.t. \{(u,v) in \pi_i^* AND (u,v) not in d_i\}

decrease \(f(\phi(,))\) for edges \((u,v)\) s.t. \{(u,v) not in \pi_i^* AND (u,v) in d_i\}
Given demonstrations \{d_1, ..., d_N\} on graphs \{G_1, ..., G_N\} and features function \Phi
Need to compute \( c(s,s') = f(\phi(s,s')) \) s.t. \( d_i = \arg\min_{\pi_i} \sum_{i=1}^{N} c(\pi_i) \)

While (Not Converged)
for \( i=1 \ldots N \)
    update edge costs in graph \( G_i \) using the current function \( f(\phi(,)) \)
    plan an optimal path \( \pi_i^* = \arg\min_{\pi_i} \sum_{k=0}^{\text{length}(\pi_i)-1} c(s_k, s_{k+1}) \)
    increase \( f(\phi(,)) \) for edges \((u,v)\) s.t. \{\((u,v)\) in \( \pi_i^* \) AND \((u,v)\) not in \( d_i \)\}
    decrease \( f(\phi(,)) \) for edges \((u,v)\) s.t. \{\((u,v)\) not in \( \pi_i^* \) AND \((u,v)\) in \( d_i \)\}

Is \( \pi_i^* \) always guaranteed to converge to \( d_i \)?
Given demonstrations \{d_1,...,d_N\} on graphs \{G_1,...,G_N\} and features function \Phi
Need to compute \(c(s,s') = f(\phi(s,s'))\) s.t. \(d_i = \arg\min_{\pi_i} \sum_{i=1}^{N} c(\pi_i)\)

While (Not Converged)
for i=1...N
update edge costs in graph \(G_i\) using the current function \(f(\phi(),)\)
plan an optimal path \(\pi_i^* = \arg\min_{\pi_i} \sum_{k=0}^{\text{length}(\pi_i)-1} c(s_k, s_{k+1})\)
increase \(f(\phi(),)\) for edges \((u,v)\) s.t. \{\((u,v)\) in \(\pi_i^*\) AND \((u,v)\) not in \(d_i\)\}
decrease \(f(\phi(),)\) for edges \((u,v)\) s.t. \{\((u,v)\) not in \(\pi_i^*\) AND \((u,v)\) in \(d_i\)\}

Any problem with arbitrary decrease of \(f(\phi(),)?\)

Any solutions?
LEARCH (LEArning to searCH)

[Ratliff, Silver, Bagnell, 09]

Given demonstrations \( \{d_1, ..., d_N\} \) on graphs \( \{G_1, ..., G_N\} \) and features function \( \Phi \n\)

Need to compute \( c(s,s') = f(\phi(s,s')) \) s.t. \( d_i = \arg \min_{\pi_i} \sum_{i=1}^{N} c(\pi_i) \n\)

While (Not Converged)

for \( i=1 ... N \)

update edge costs in graph \( G_i \) using the current function \( f(\phi(,)) \n\)

plan an optimal path \( \pi_i^* = \arg \min_{\pi_i} \sum_{k=0}^{\text{length}(\pi_i)-1} c(s_k, s_{k+1}) \n\)

increase \( \log f(\phi(,)) \) for edges \( (u,v) \) s.t. \( \{ (u,v) \text{ in } \pi_i^* \text{ AND } (u,v) \text{ not in } d_i \} \n\)

decrease \( \log f(\phi(,)) \) for edges \( (u,v) \) s.t. \( \{ (u,v) \text{ not in } \pi_i^* \text{ AND } (u,v) \text{ in } d_i \} \n\)
Example

- Consider a (simple) outdoor navigation example

Suppose initial $w_0 = 0$. Any problem learning $W$?

Need a loss function that makes the algorithm learn harder to stay on the demonstrated paths (related to maximizing the margin in a classifier).

Demonstration $d_1$ on graph $G_1$
**LEARCH (LEArning to searCH)**

[Ratliff, Silver, Bagnell, 09]

Given demonstrations \( \{d_1, ..., d_N\} \) on graphs \( \{G_1, ..., G_N\} \) and features function \( \Phi \)

Need to compute \( c(s, s') = f(\phi(s, s')) \) s.t. \( d_i = \arg \min_{\pi_i} \sum_{i=1}^{N} c(\pi_i) \)

**While (Not Converged)**

for \( i = 1 \ldots N \)

update edge costs in graph \( G_i \) using the current function \( f(\phi(,)) \)

plan an optimal path \( \pi_i^* = \arg \min_{\pi_i} \sum_{k=0}^{\text{length}(\pi_i)-1} \{c(s_k, s_{k+1}) - l(s_k, s_{k+1})\} \)

increase \( \log f(\phi(,)) \) for edges \( (u,v) \) s.t. \( \{ (u,v) \in \pi_i^* \text{ AND } (u,v) \text{ not in } d_i \} \)

decrease \( \log f(\phi(,)) \) for edges \( (u,v) \) s.t. \( \{ (u,v) \text{ not in } \pi_i^* \text{ AND } (u,v) \text{ in } d_i \} \)

**Loss function penalizes being NOT on a demonstration path.**

For example, \( l(s, s') = 0 \) if \( (s, s') \) on \( d_i \) and \( l(s, s') > 1 \) otherwise
Given demonstrations \(\{d_1, \ldots, d_N\}\) on graphs \(\{G_1, \ldots, G_N\}\) and features function \(\Phi\) 

Need to compute \(c(s,s') = f(\phi(s,s'))\) s.t. \(d_i = \arg\min_{\pi_i} \sum_{i=1}^{N} c(\pi_i)\)

While (Not Converged)

for \(i=1\ldots N\) 

update edge costs in graph \(G_i\) using current function \(f(\phi, )\) 

plan an optimal path \(\pi_i^* = \arg\min_{\pi_i} \sum_{k=0}^{\text{length}(\pi_i)-1} \{c(s_k, s_{k+1}) - l(s_k, s_{k+1})\}\)

increase log \(f(\phi, )\) for edges \((u,v)\) s.t. \{\((u,v)\) in \(\pi_i^*\) AND \((u,v)\) not in \(d_i\)\}

decrease log \(f(\phi, )\) for edges \((u,v)\) s.t. \{\((u,v)\) not in \(\pi_i^*\) AND \((u,v)\) in \(d_i\)\}

How do we decide how to increase/decrease \(f(\phi, )\)?
LEARCH (LEArning to searCH)
[Ratliff, Silver, Bagnell, 09]

Given demonstrations \( \{d_1, ..., d_N\} \) on graphs \( \{G_1, ..., G_N\} \) and features function \( \Phi \)
Need to compute \( c(s, s') = f(\phi(s, s')) \) s.t. \( d_i = \arg\min_{\pi_i} \sum_{i=1}^{N} c(\pi_i) \)

While (Not Converged)

for \( i = 1 \ldots N \)

update edge costs in graph \( G_i \) using the current function \( f(\phi(,)) \)
plan an optimal path \( \pi_i^* = \arg\min_{\pi} \text{length } \pi_i \cdot \sum_{k=1}^{l-1} c(\pi_k) \cdot \pi_{k+1} \)

increase log \( f(\phi(,)) \) for edges \( (u, v) \) s.t. \{\( u, v \) in \( \pi_i^* \) AND \( u, v \) not in \( d_i \)\}

decrease log \( f(\phi(,)) \) for edges \( (u, v) \) s.t. \{\( u, v \) not in \( \pi_i^* \) AND \( u, v \) in \( d_i \)\}

How do we decide how to increase/decrease \( f(\phi(,)) \)?

Set \( dC \) vector as: +1 for all edges that need to be increased, and -1 for all edges that need to be decreased.

Recompute \( f(\phi(,)) \) to make a step in the direction of \( dC \)

For example, if \( f(\phi(s, s')) = \sum w_i \phi_i(s, s') = \Phi W \), then:

1. Solve for vector \( dW \) from \( \Phi dW = dC \) (e.g., \( dW = (\Phi^T \Phi)^{-1} \Phi^T dC \))
2. Update \( W \): \( W = W + \eta dW \)
What You Should Know…

- Types of learning in planning
- Why and when learning in planning is useful
- General idea for methods to learn plan faster
- General idea for learning cost function from demonstrations