

16-832 Spring'26
Integrated Planning and Learning

*Integrating learning into planning:
learning cost function*
Part 1

Maxim Likhachev
Robotics Institute
Carnegie Mellon University

Papers to cover

- *"Learning to search: Functional gradient techniques for imitation learning"* by Ratliff et al., 2009
- *"Learning Navigation Costs from Demonstration via Differentiable Planning"* by Wang et al., 2020 – *in the next round*

What?

- An approach to learn a cost function that makes given demonstrations optimal plans [Ratliff, Silver & Bagnell, '09]

"Learning to search: Functional gradient techniques for imitation learning" by Ratliff et al., Autonomous Robots, 2009

Why?

- For many planning domains, cost function is hard-to-design and sometimes is entirely unknown (e.g., “natural” behavior of a vehicle).
- Is often important for robot-human scenarios: “Natural” behavior solicits “natural” response from human actors

"Learning to search: Functional gradient techniques for imitation learning" by Ratliff et al., Autonomous Robots, 2009

Related Work

- Feature expectation matching [Abbeel & Ng, '04]
 - Assumes demonstrations are optimal
 - Requires the planner to generate stochastic policies (i.e., randomize over actions)

"Learning to search: Functional gradient techniques for imitation learning" by Ratliff et al., Autonomous Robots, 2009

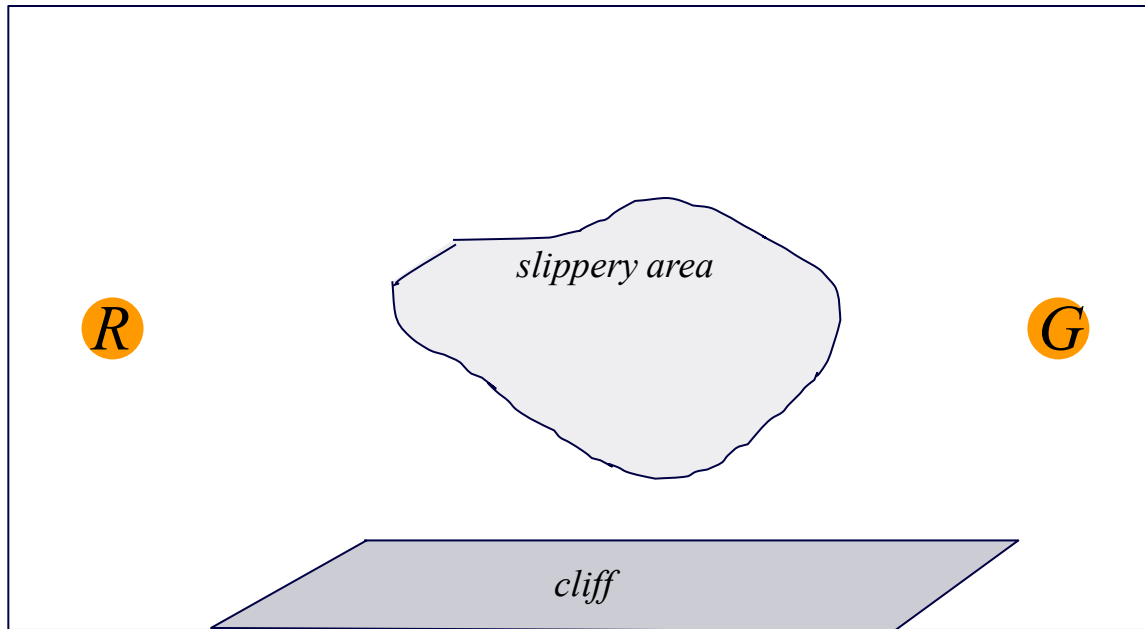
Assumptions

- Assumes we have enough demonstrations to recover the cost function
- Assumes full observability
- During demonstrations, the robot is “on” the graph/MDP that represents the planning problem

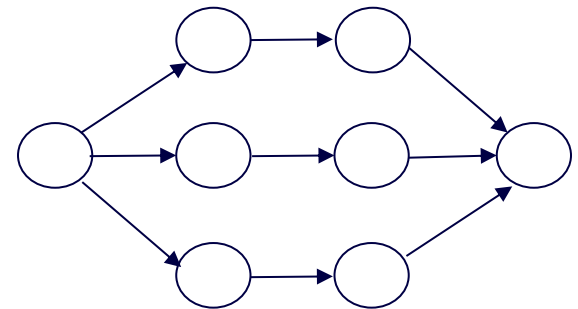
"Learning to search: Functional gradient techniques for imitation learning" by Ratliff et al., Autonomous Robots, 2009

Example

- Consider a (simple) outdoor navigation example



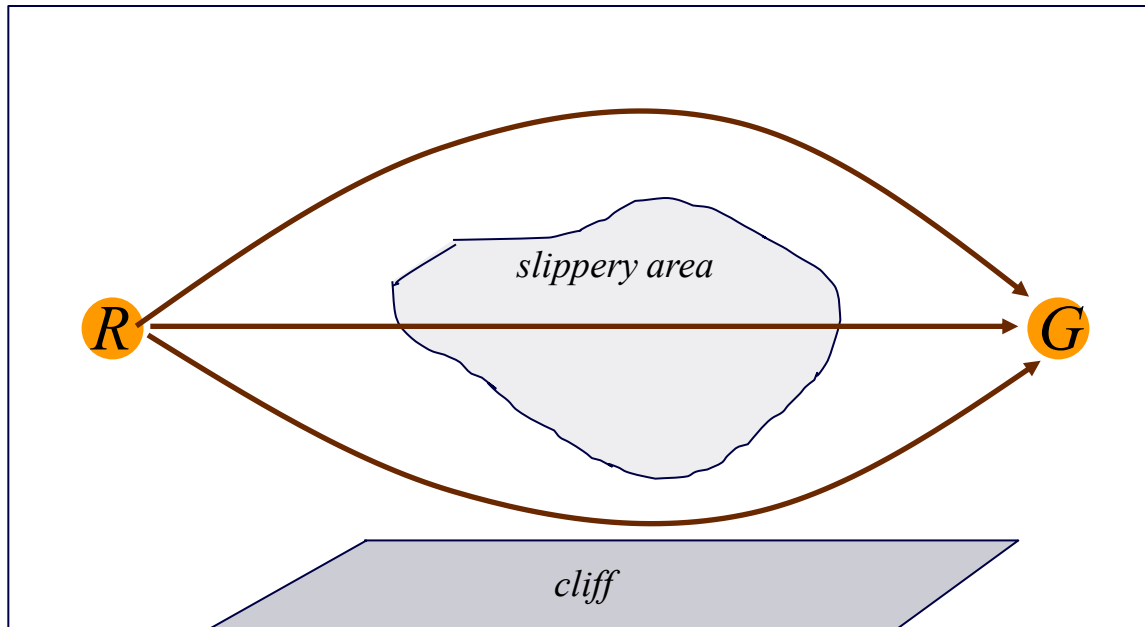
Modeled as graph search



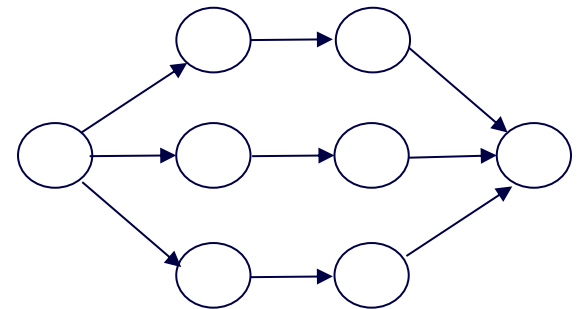
Example

- 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)?



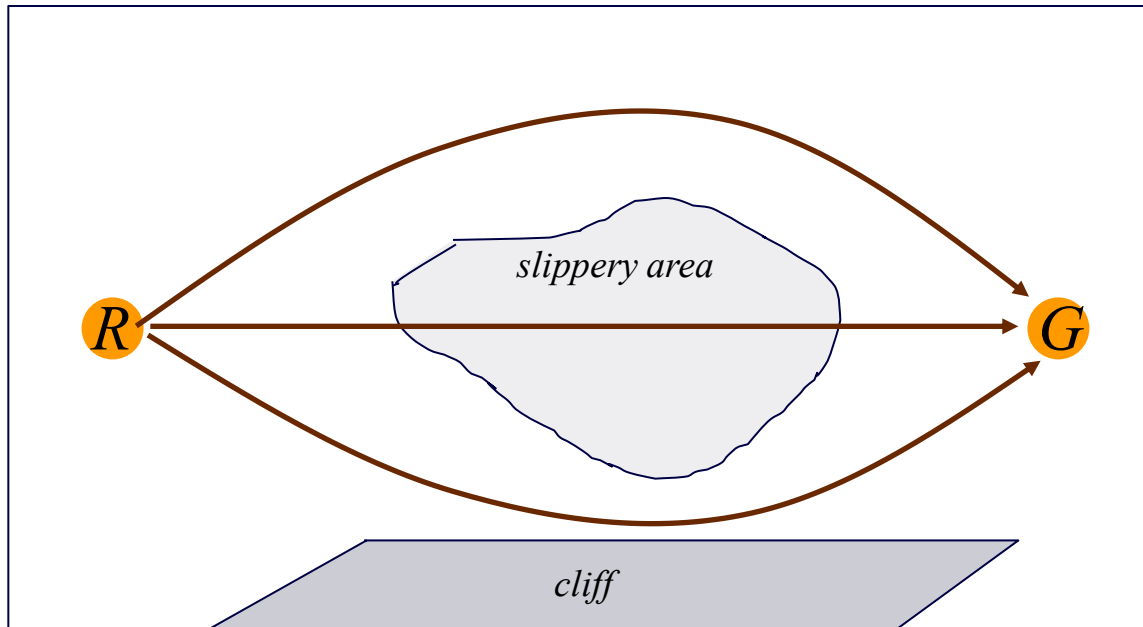
Modeled as graph search



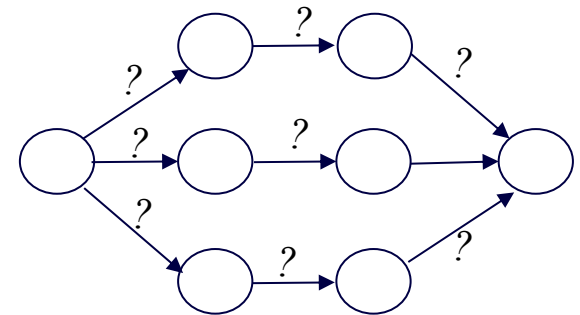
Example

- 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



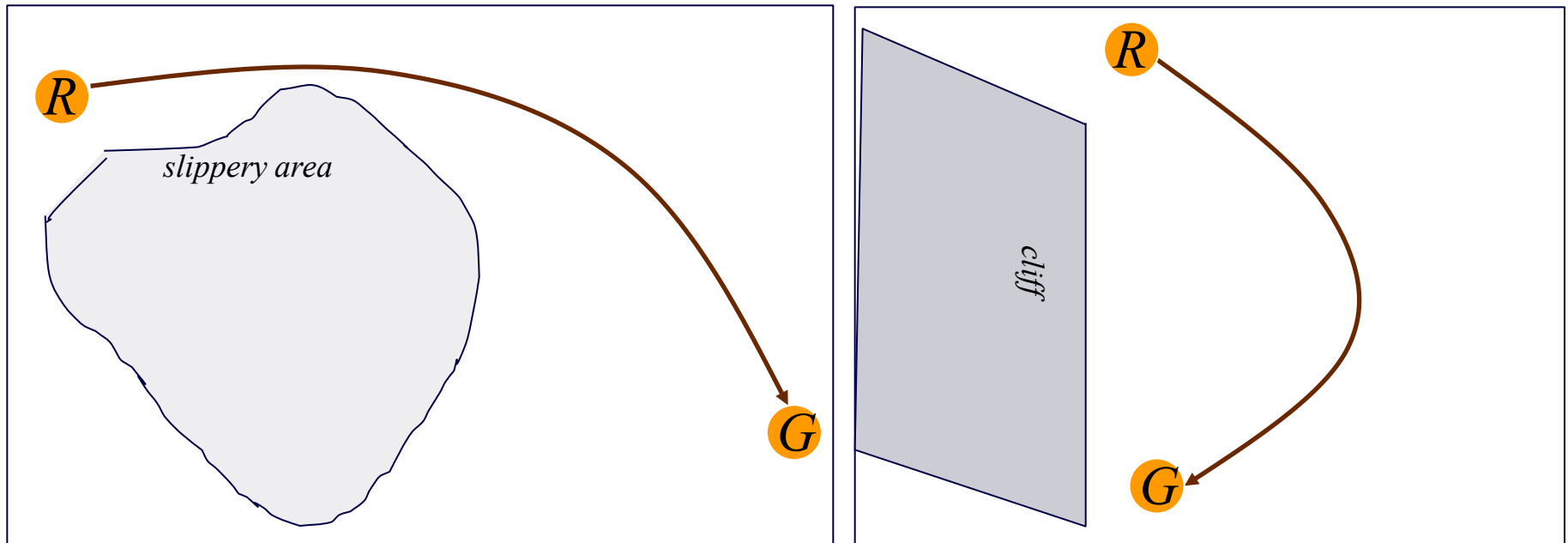
Example

- 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)?

A user gives N demonstrations of what paths are good.

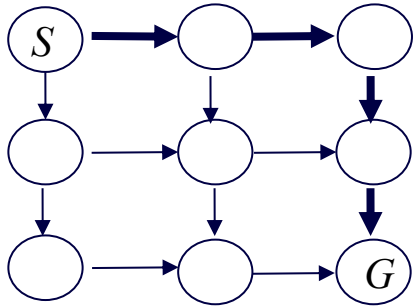
We want a cost function for which these demonstrated trajectories are least-cost plans



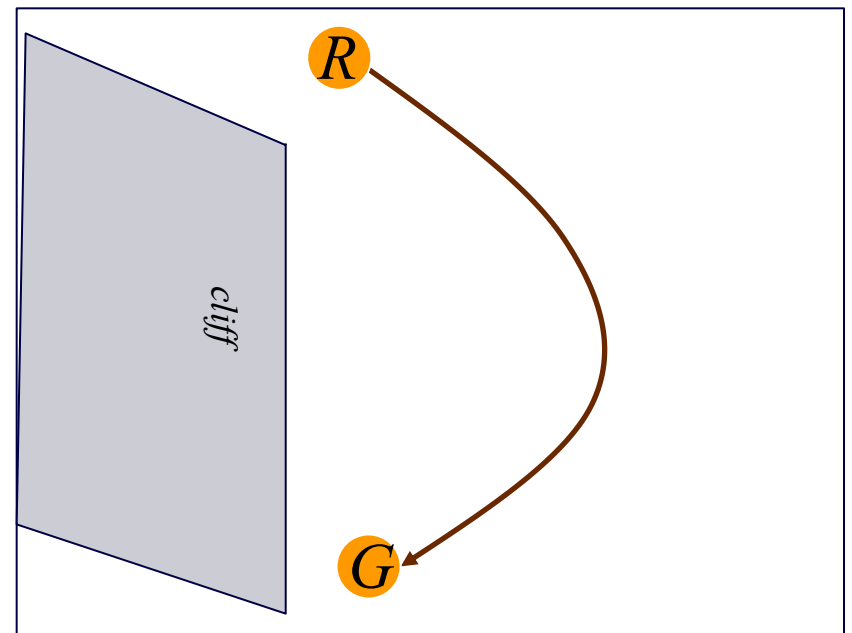
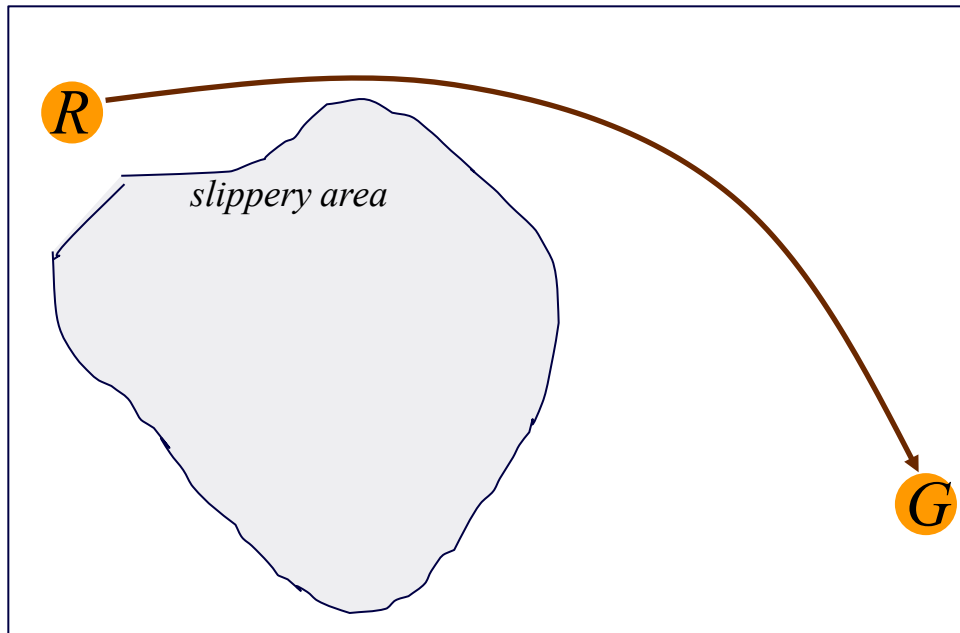
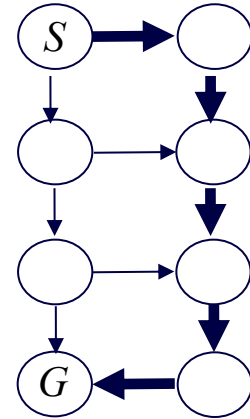
Example

- Consider a (simple) outdoor navigation example

Demonstration d_1 on graph G_1



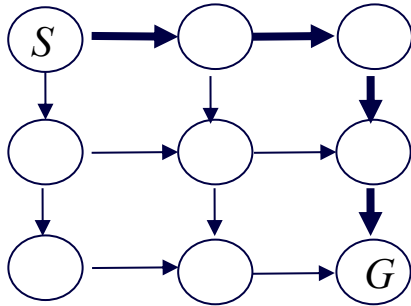
Demonstration d_2 on graph G_2



Example

- Consider a (simple) outdoor navigation example

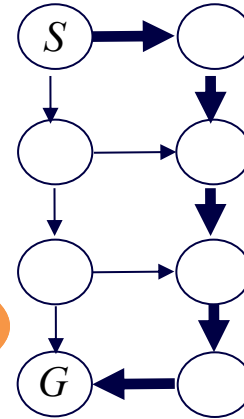
Demonstration d_1 on graph G_1



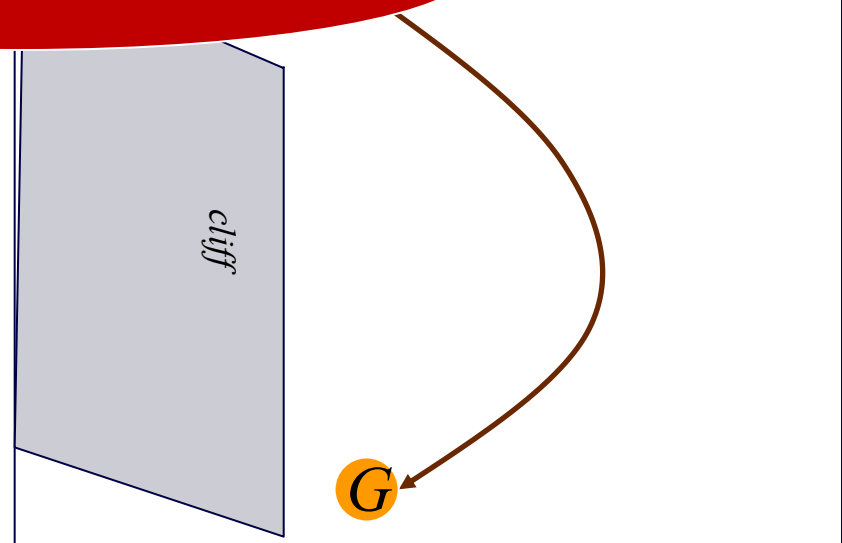
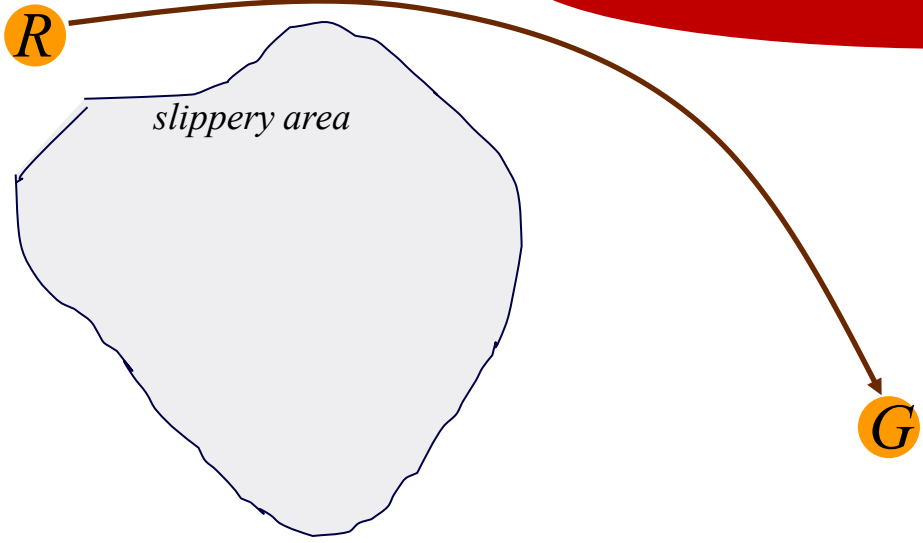
Compute cost function that makes these demonstrations optimal paths

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

Demonstration d_2 on graph G_2



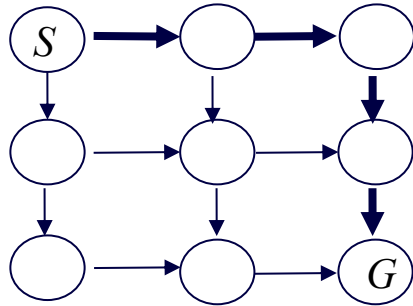
Why not learn edge costs directly?



Example

- Consider a (simple) outdoor navigation example

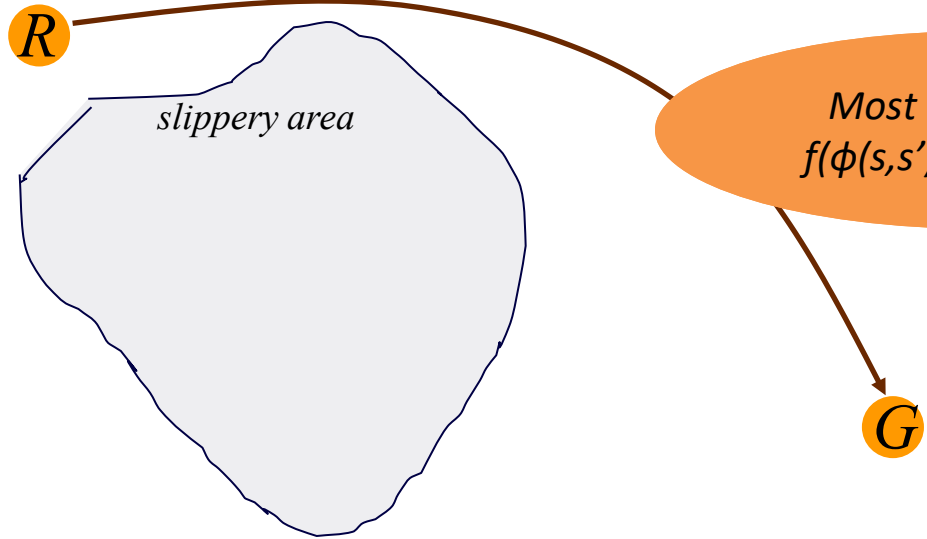
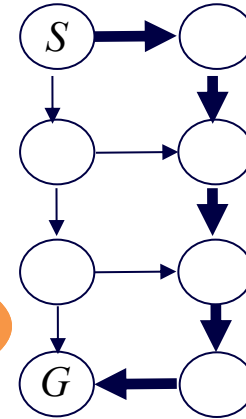
Demonstration d_1 on graph G_1



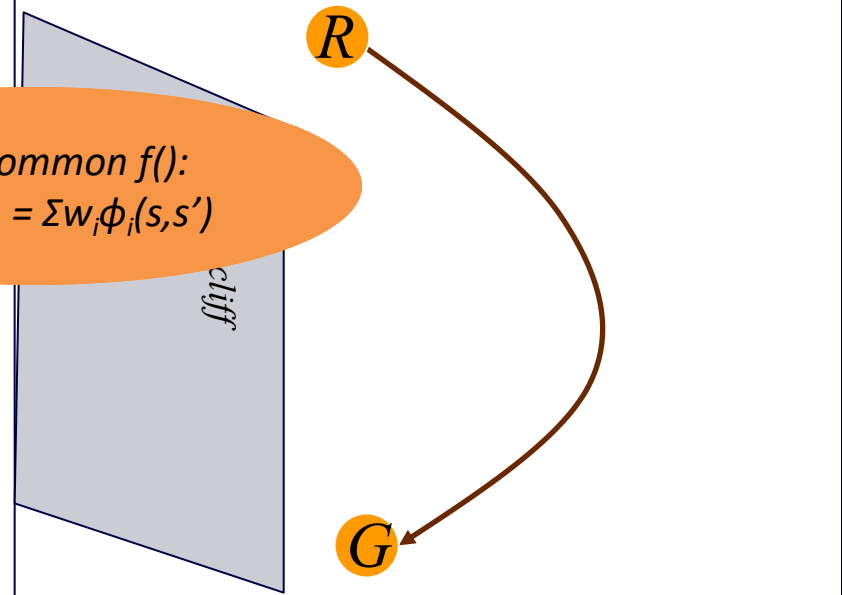
Compute cost function that makes these demonstrations optimal paths

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

Demonstration d_2 on graph G_2



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



Example

- Consider a (simple) outdoor navigation example

For example:

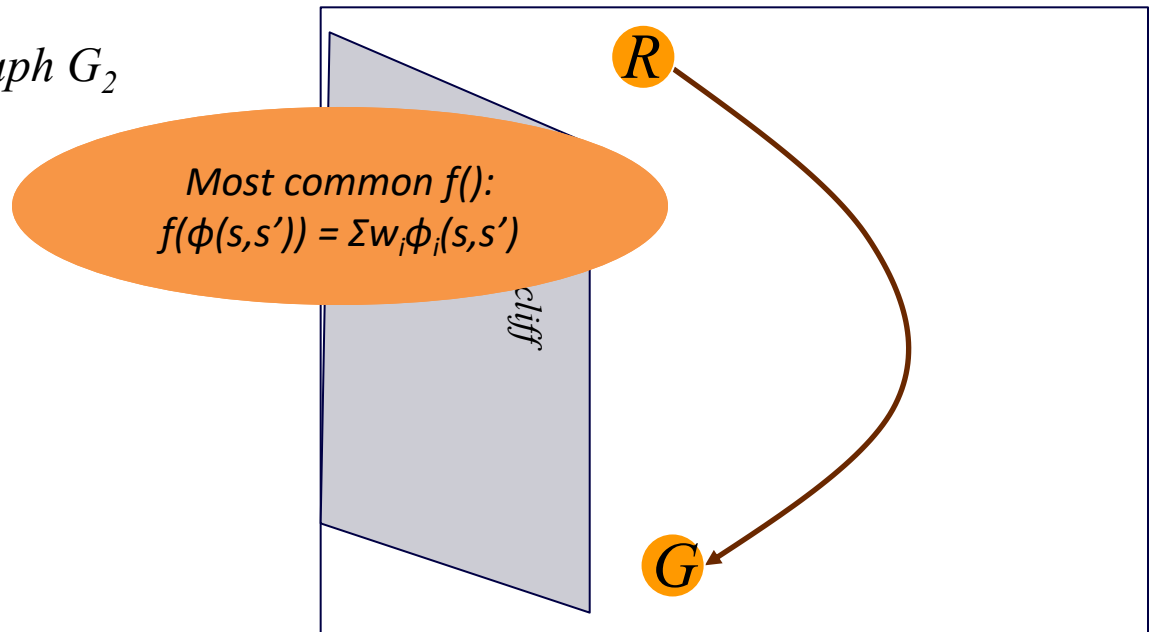
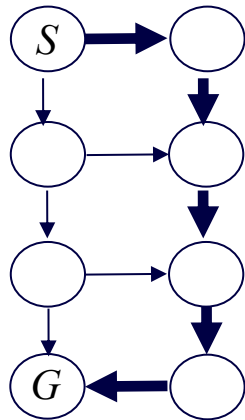
ϕ_0 : $1/(\text{distance to slippery area})$

ϕ_1 : $1/(\text{distance to cliff})$

ϕ_2 : length of the transition

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

Demonstration d_2 on graph G_2



LEARCH (LEArning to searCH)

[Ratliff, Silver, Bagnell, 09]

Given demonstrations $\{d_1, \dots, d_N\}$ on graphs $\{G_1, \dots, G_N\}$ and features function Φ
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 $i=1 \dots N$

update edge costs in graph G_i using the current function $f(\Phi(\cdot))$

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 $f(\Phi(\cdot))$ for edges (u, v) s.t. $\{(u, v) \text{ in } \pi_i^* \text{ AND } (u, v) \text{ not in } d_i\}$

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

LEARCH (LEArning to searCH)

[Ratliff, Silver, Bagnell, 09]

Given demonstrations $\{d_1, \dots, d_N\}$ on graphs $\{G_1, \dots, G_N\}$ and features function Φ
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 $i=1 \dots N$

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. $\{(u, v) \text{ in } \pi_i^* \text{ AND } (u, v) \text{ not in } d_i\}$

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

Is π_i^ always guaranteed to converge to d_i ?*

LEARCH (LEArning to searCH)

[Ratliff, Silver, Bagnell, 09]

Given demonstrations $\{d_1, \dots, d_N\}$ on graphs $\{G_1, \dots, G_N\}$ and features function Φ
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 $i=1 \dots N$

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. $\{(u, v) \text{ in } \pi_i^* \text{ AND } (u, v) \text{ not in } d_i\}$

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

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

Any solutions?

LEARCH (LEArning to searCH)

[Ratliff, Silver, Bagnell, 09]

Given demonstrations $\{d_1, \dots, d_N\}$ on graphs $\{G_1, \dots, G_N\}$ and features function Φ
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 $i=1 \dots N$

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

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

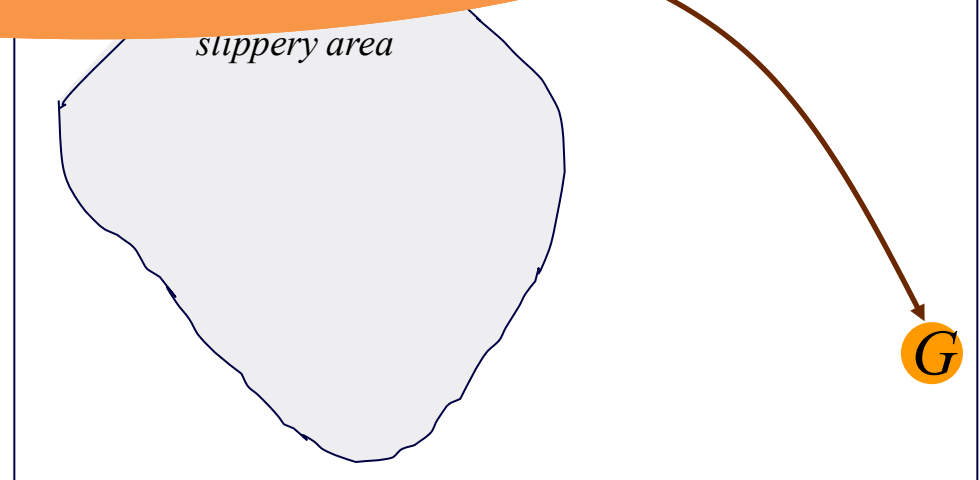
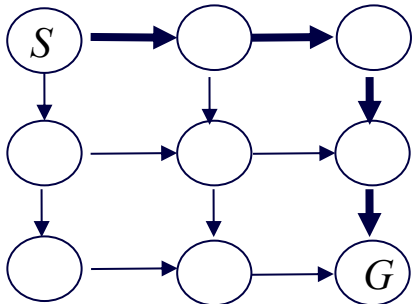
Example

- 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 $\{d_1, \dots, d_N\}$ on graphs $\{G_1, \dots, G_N\}$ and features function Φ
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 $i=1 \dots N$

update edge costs in graph G_i using the current function $f(\Phi(\cdot))$

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

decrease $\log f(\Phi(\cdot))$ 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

LEARCH (LEArning to searCH)

[Ratliff, Silver, Bagnell, 09]

Given demonstrations $\{d_1, \dots, d_N\}$ on graphs $\{G_1, \dots, G_N\}$ and features function Φ
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 $i=1 \dots N$ *How do we decide how to increase/decrease $f(\Phi(\cdot))$?*

update edge costs in graph G_i using the cost function $J(\Phi(\cdot))$

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

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

LEARCH (LEArning to searCH)

[Ratliff, Silver, Bagnell, 09]

Given demonstrations $\{d_1, \dots, d_N\}$ on graphs $\{G_1, \dots, G_N\}$ and features function Φ
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 $i=1 \dots N$ How do we decide how to increase/decrease $f(\phi(\cdot))$?

update edge costs in Φ

plan

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

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

increase $\log J(\Phi(s, s'))$

decrease $\log J(\Phi(s, s'))$

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 : $W = W + \eta dW$

Limitations

- Relies on repeated calls to optimal planner
- Assumes full observability of the state

"Learning to search: Functional gradient techniques for imitation learning" by Ratliff et al., Autonomous Robots, 2009