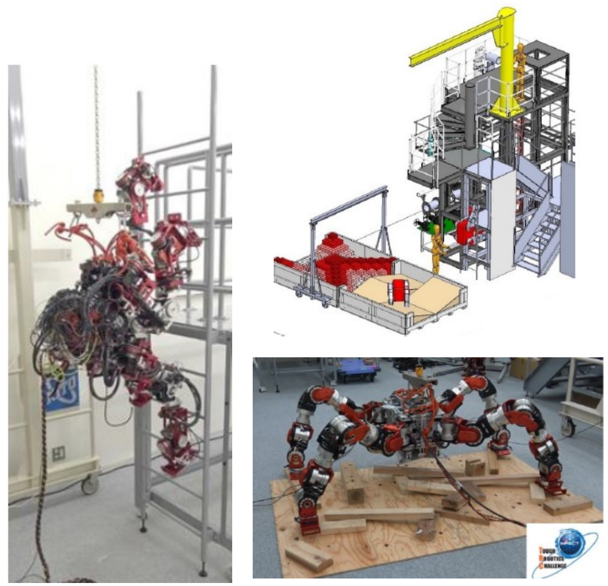
16-350
Planning & Decision-making in Robotics

Learning in Planning

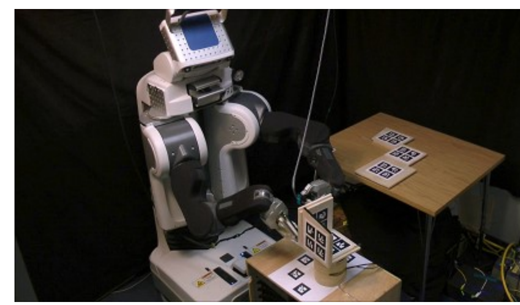
Maxim Likhachev
Robotics Institute
Carnegie Mellon University

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



Waseda/Mitsubishi robot



- **Real-world: real-time + going beyond what's given**

Learning in Search-based Planning

Speeding up planning

Learning cost function

Going beyond the given model

Waseda/
Mitsubishi

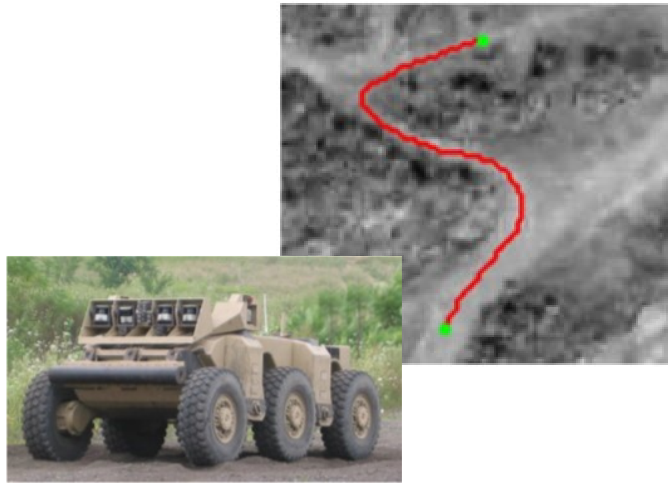
Re-use of previous results within search (Phillips et al.,'12; Islam et al.,'18)
Learning heuristic functions (Bhardwaj et al.,'17; Paden & Frazzoli,'17; Thayer et al.,'11)
Learning order of expansions (Choudhary et al.,'17)

Learning in Search-based Planning

Speeding up planning

Learning cost function

Going beyond the given model



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

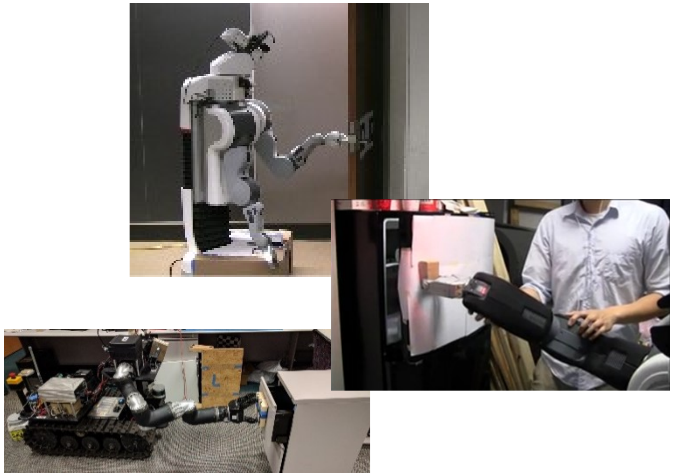
Learning a cost function from demonstrations (Ratliff et al.,’09; Wulfmeier et al.,’17)

Learning in Search-based Planning

Speeding up planning

Learning cost function

Going beyond the given 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 given model

Waseda/ Mitsubishi

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]

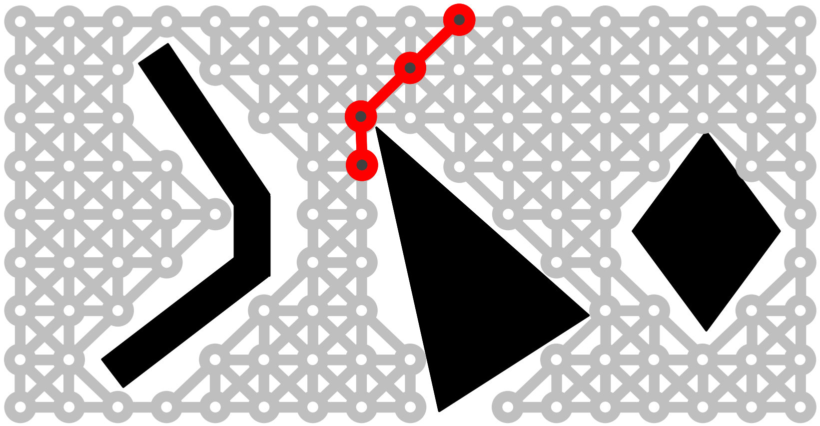
- Many planning tasks are repetitive
  - loading a dishwasher
  - opening doors
  - moving objects around a warehouse
  - …

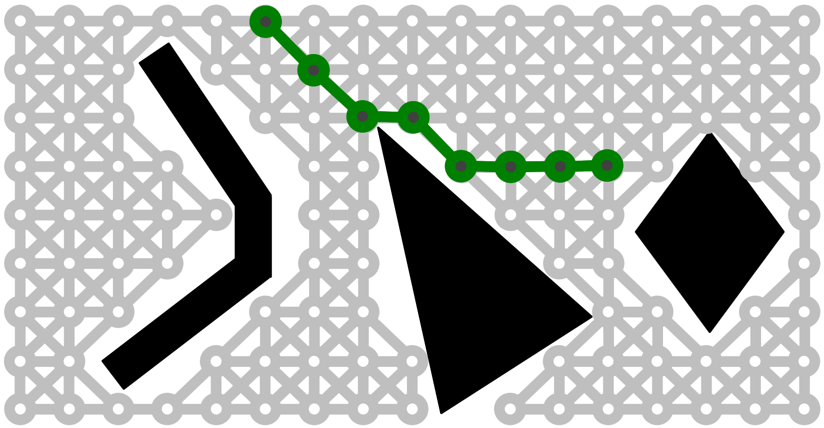
- Can we re-use prior experience to accelerate planning, in the context of search-based planning?

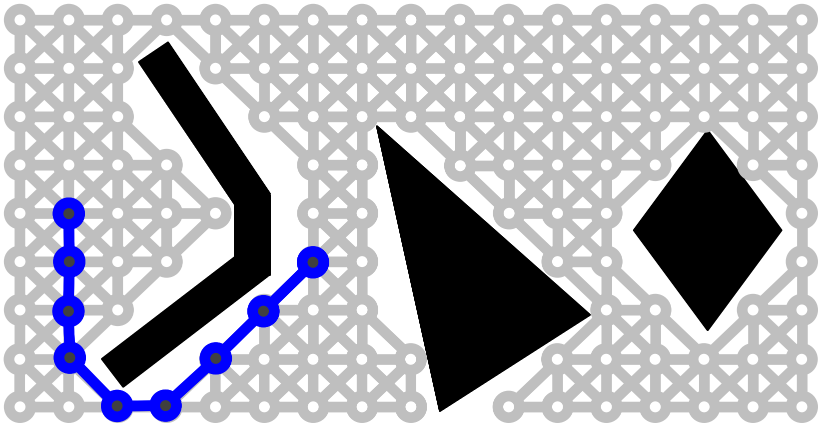
- Especially useful for high-dimensional problems such as mobile manipulation!

Experience Graphs [Phillips et al., RSS'12]

Given a set of previous paths (experiences)…

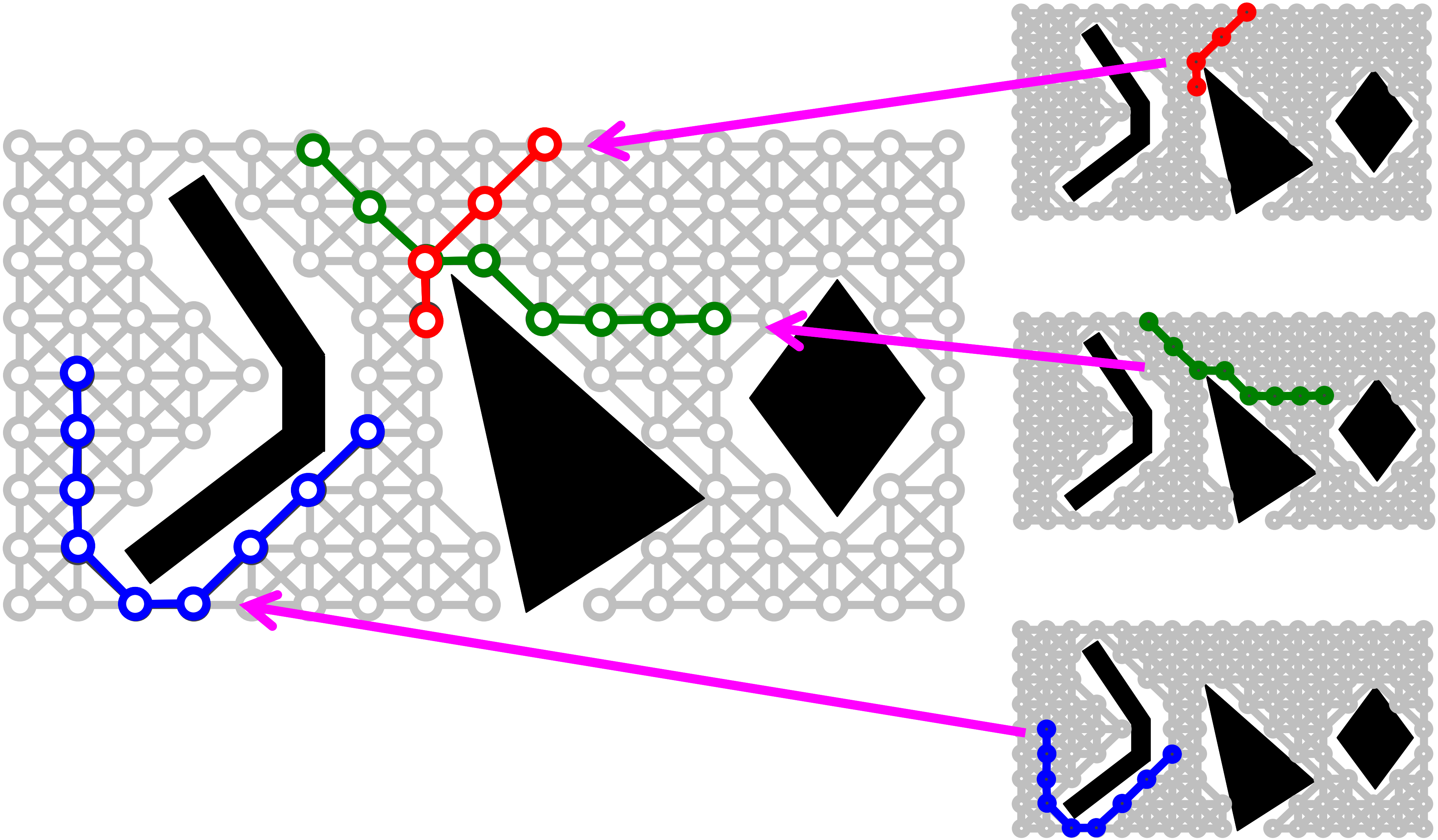






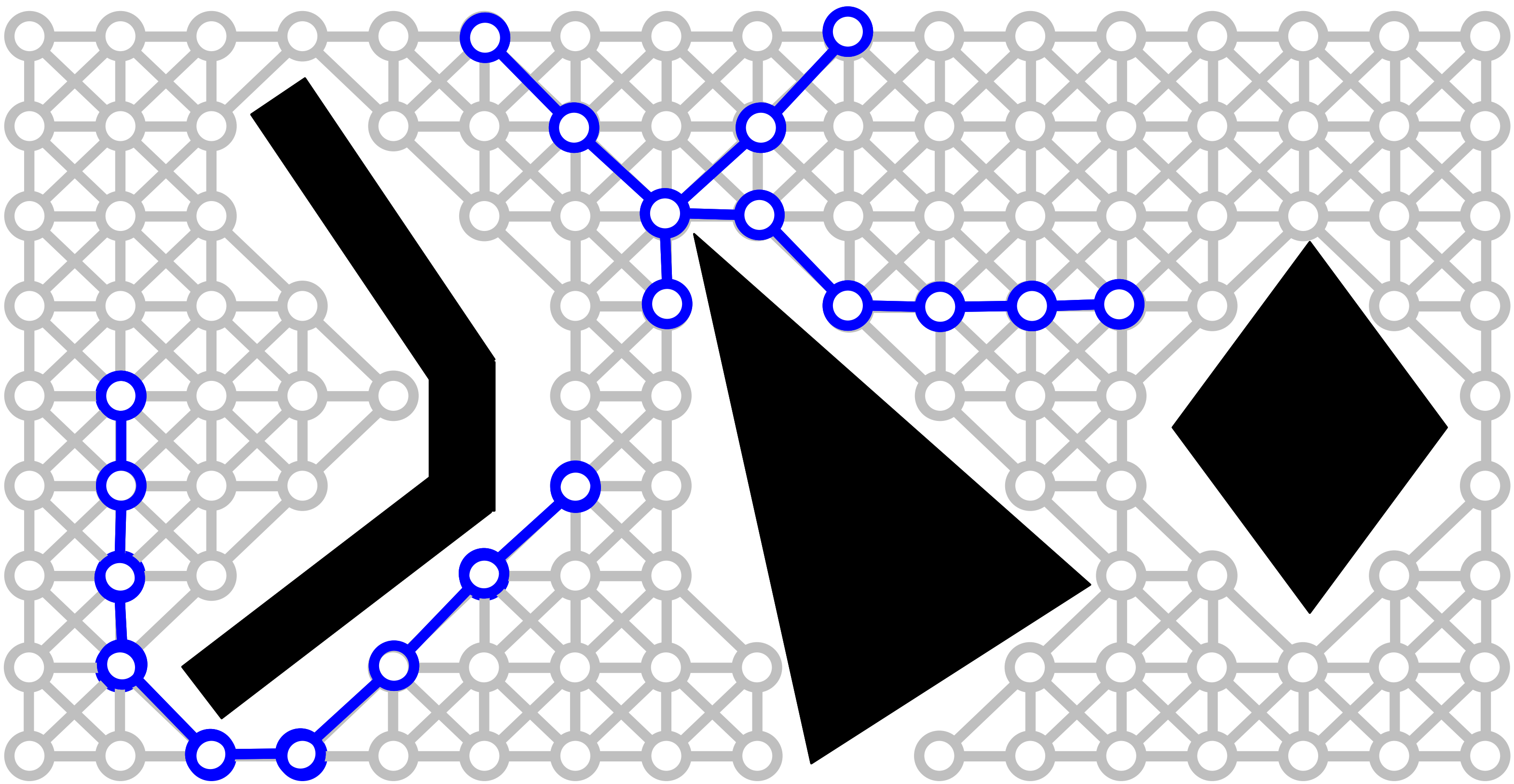
Experience Graphs [Phillips et al., RSS'12]

Put them together into an *E*-graph (Experience graph)



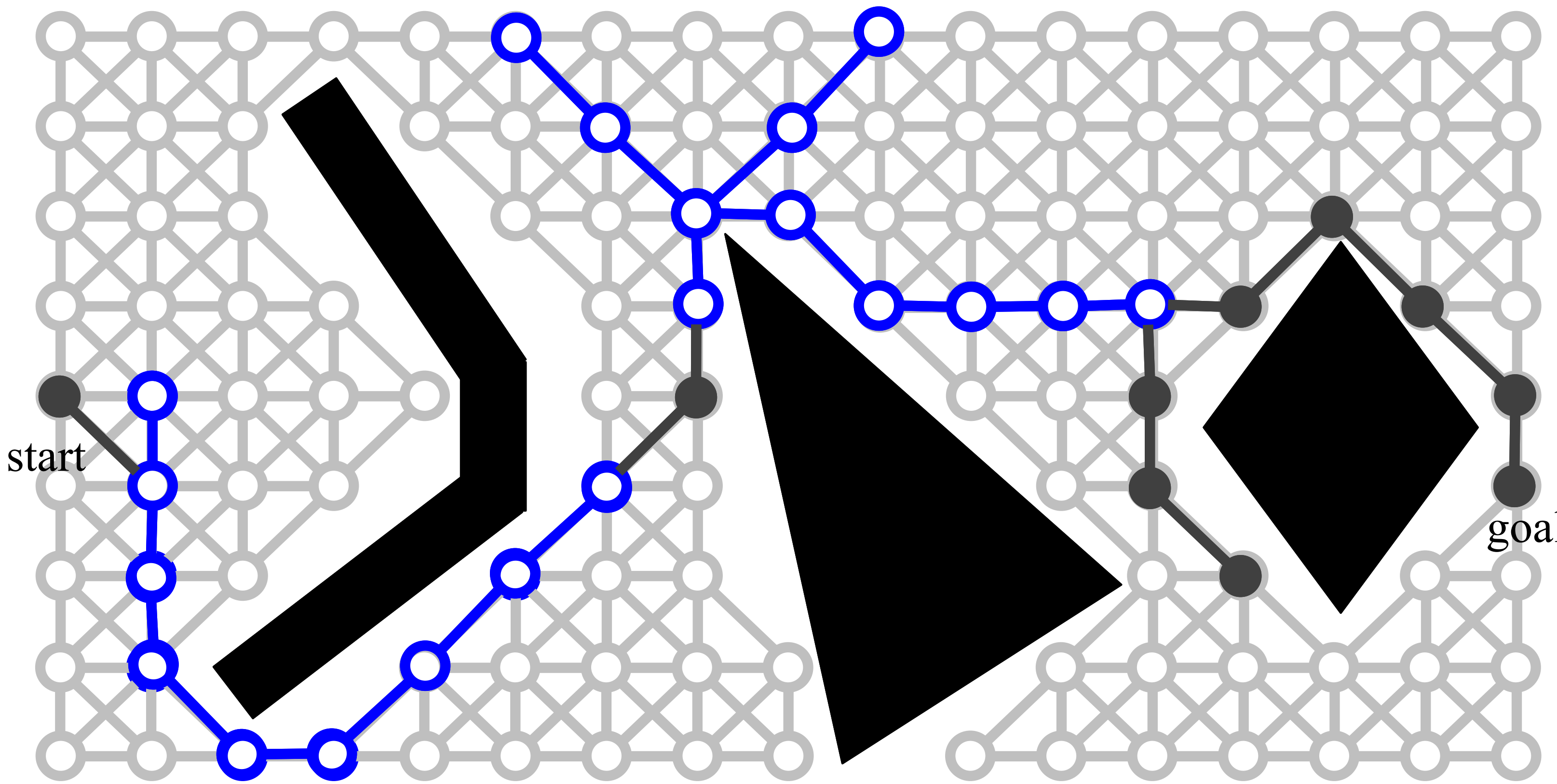
Experience Graphs [Phillips et al., RSS'12]

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

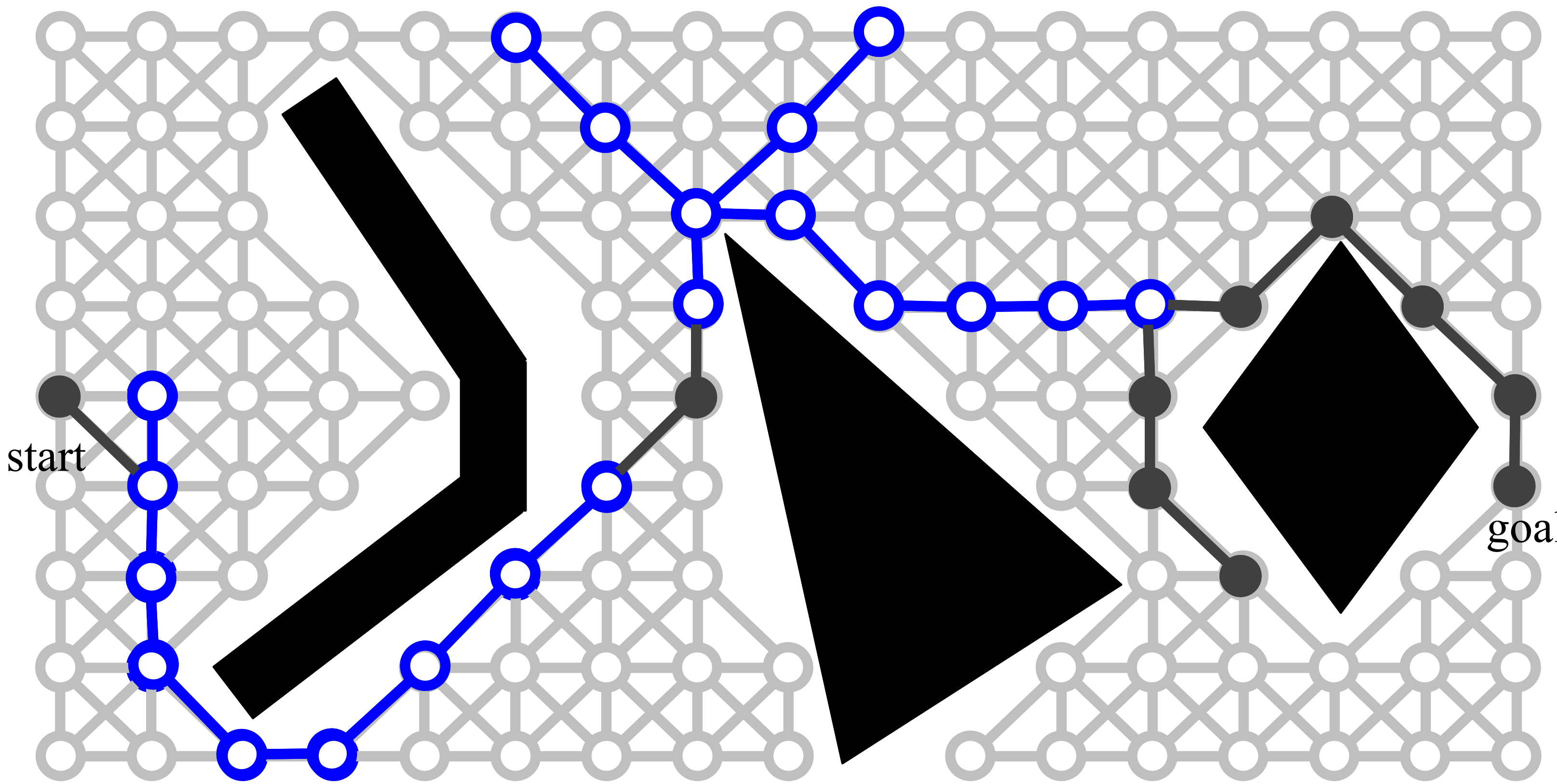


Experience Graphs [Phillips et al., RSS'12]

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

Re-use is via focusing search with a recomputed *h^ε()* heuristic function:

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



Experience Graphs [Phillips et al., RSS'12]

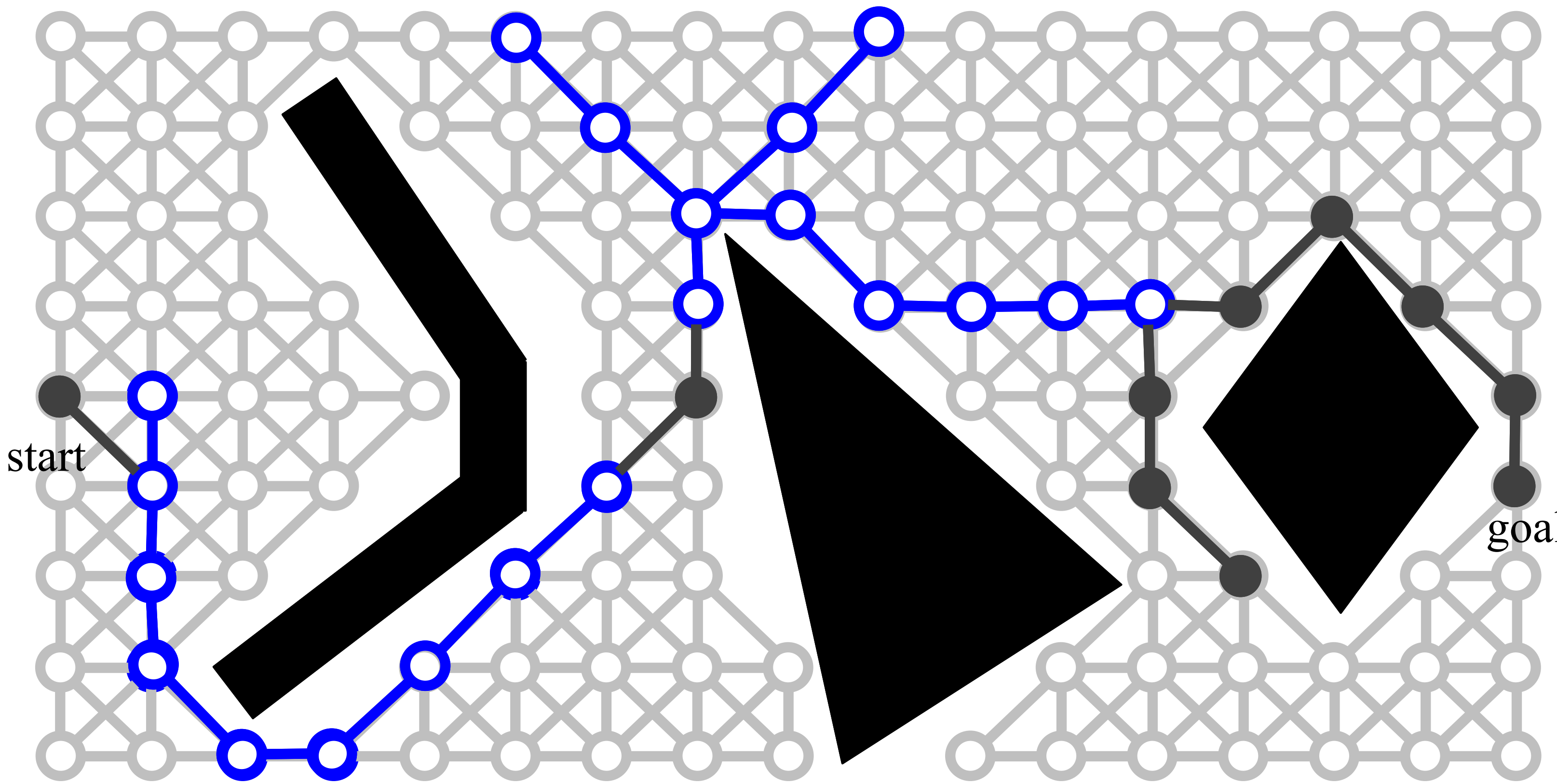
…w… …ions

General idea:

Instead of biasing the search towards the goal, heuristics $h^{\varepsilon}(s)$ biases it towards a set of paths in Experience Graph

Re-use is via … …tic function:

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



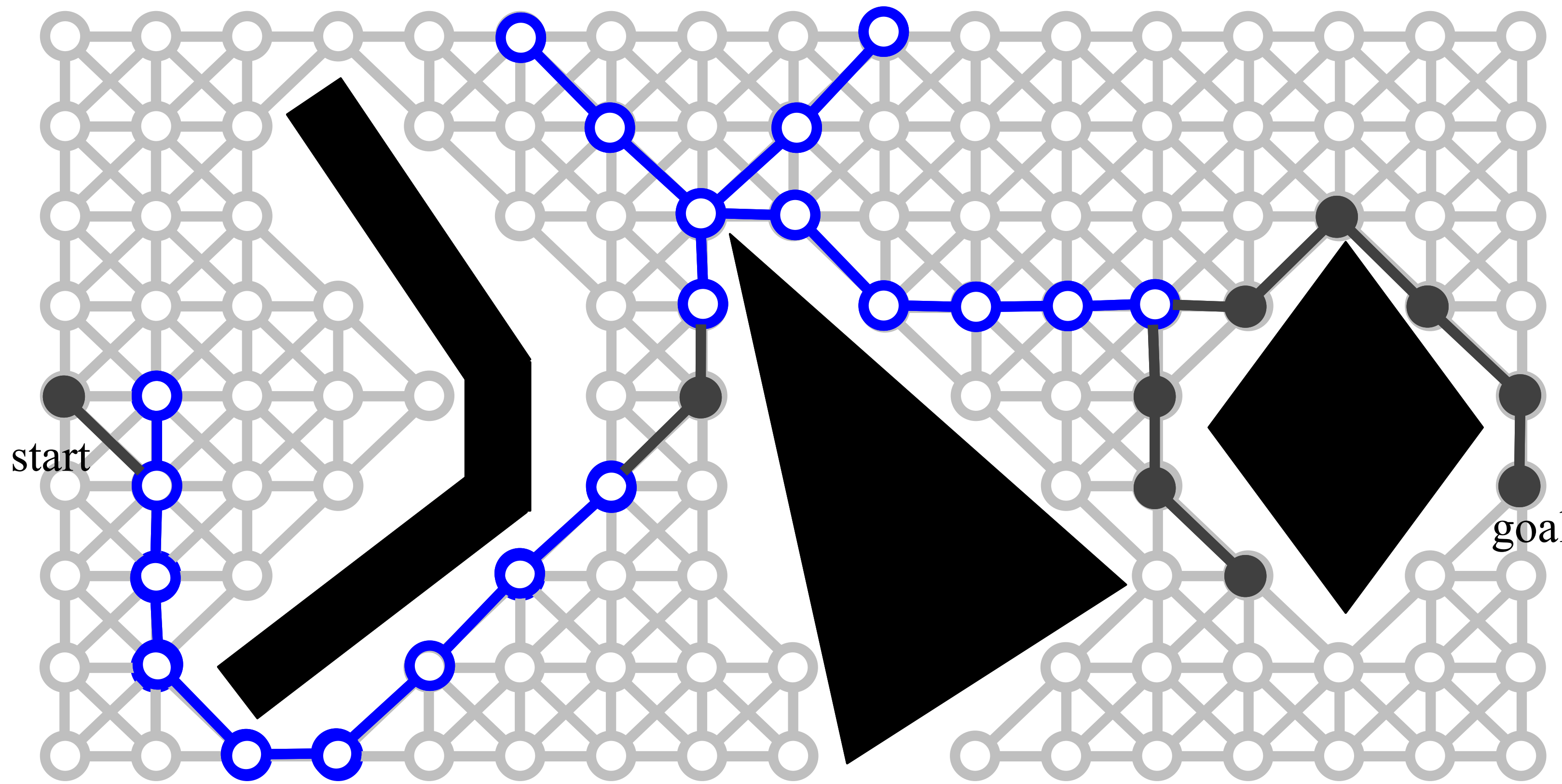
Experience Graphs [Phillips et al., RSS'12]

…would li… …ituations

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

Re-use … …unction:

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

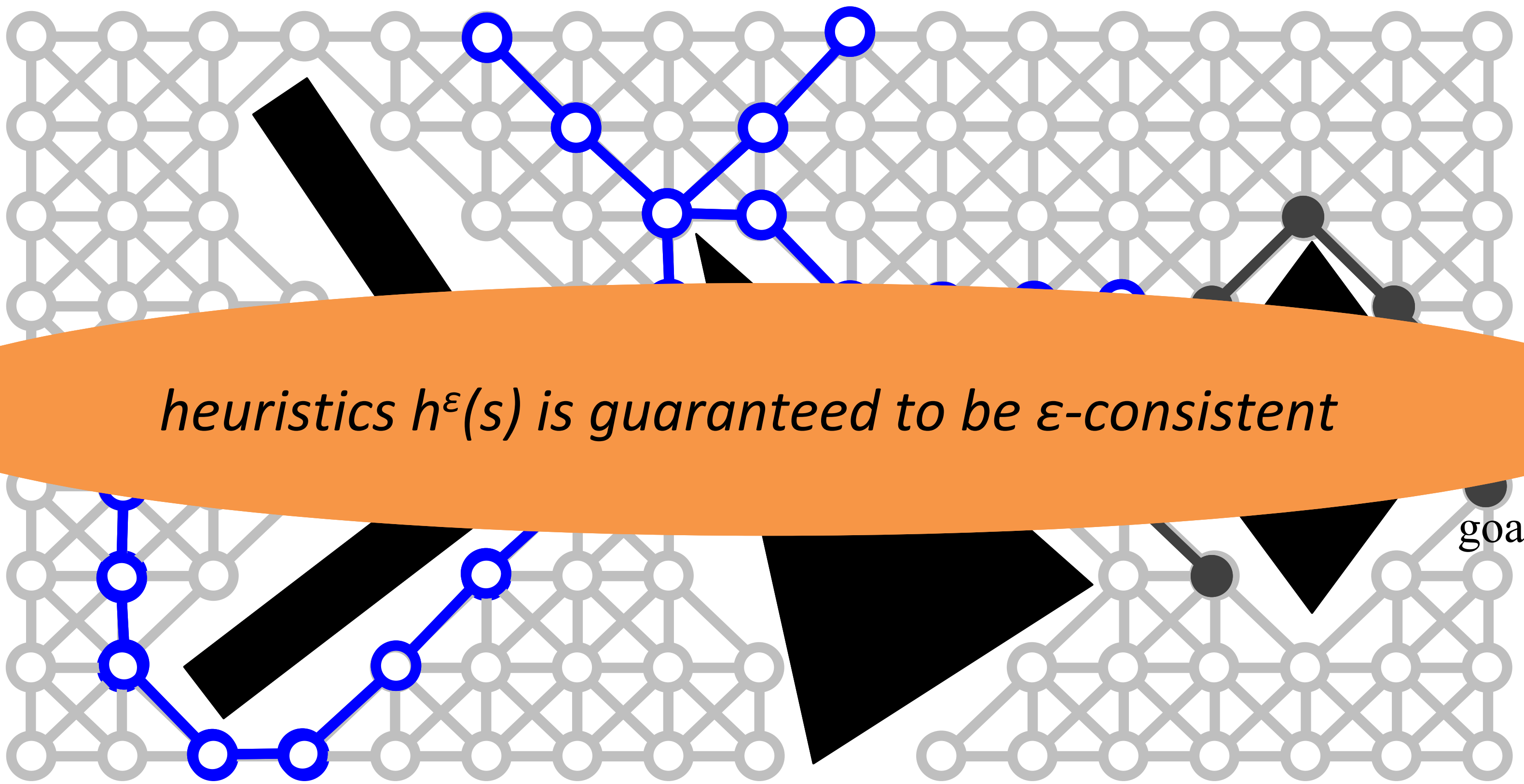


Experience Graphs [Phillips et al., RSS'12]

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

Re-use is via focusing search with a recomputed *h^ε()* heuristic function:

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



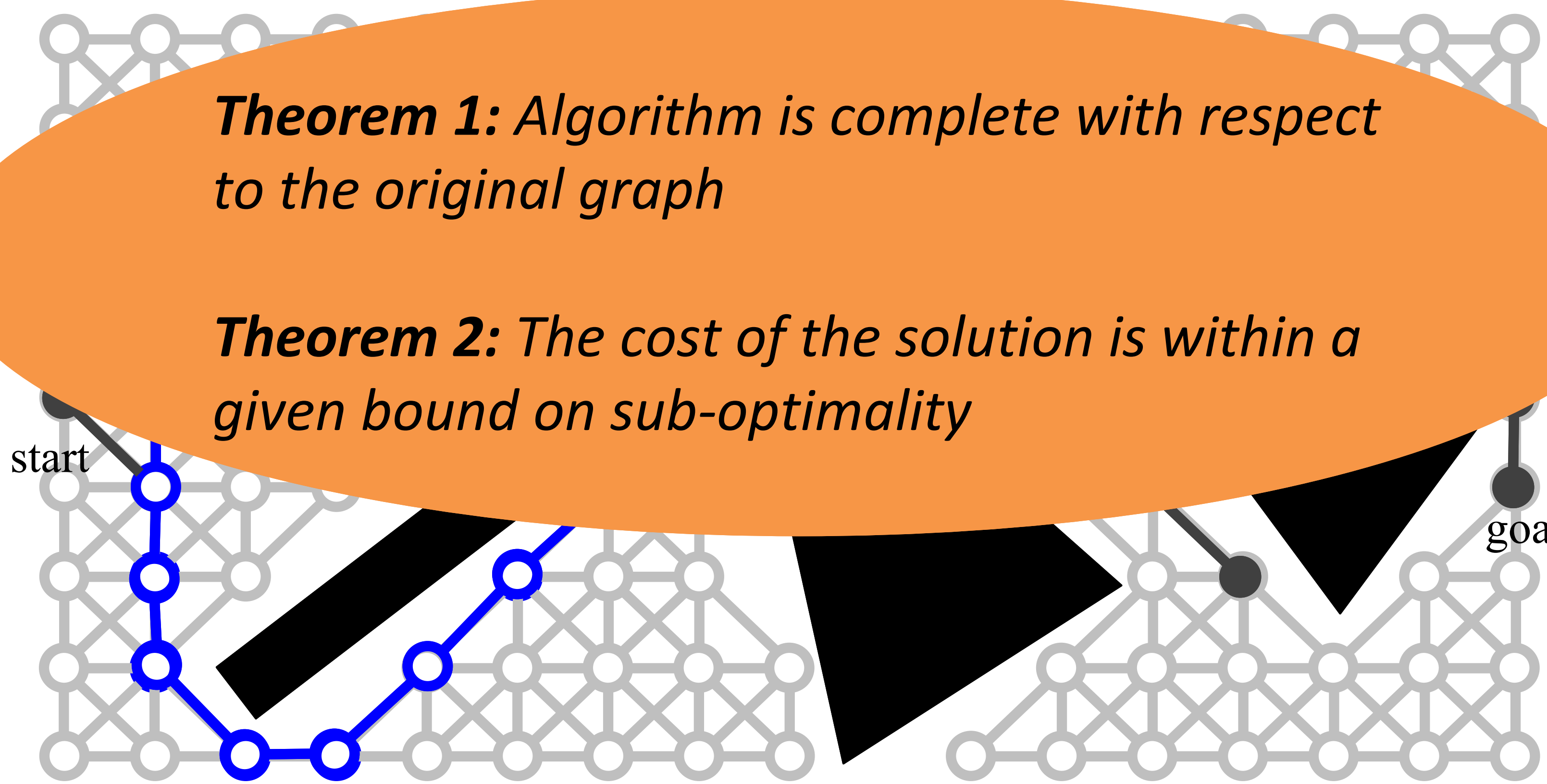
heuristics $h^{\varepsilon}(s)$ is guaranteed to be ε-consistent

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*

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



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*

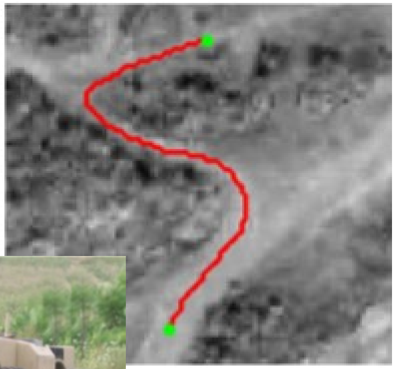
Experience Graphs [Phillips et al., RSS'12]

Learning in Search-based Planning

Speeding up planning

Learning cost function

Going beyond the prior model



Crusher (from Ratliff et a., '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], ...

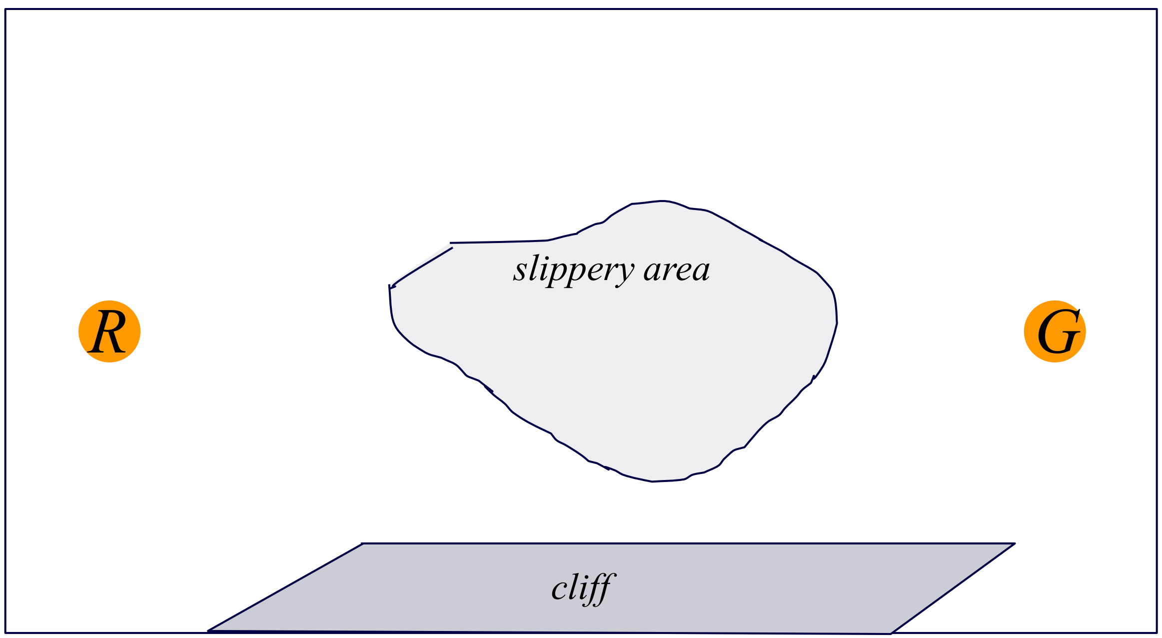
Inverse Reinforcement Learning (IRL), Inverse Optimal Control

Learning a cost function

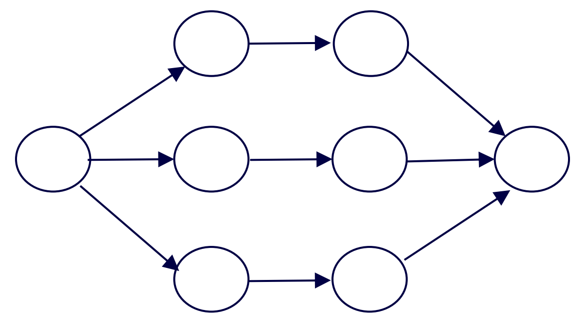
- **Recover a cost function that makes given demonstrations optimal plans** [Ratliff, Silver & Bagnell, '09]

Example

- Consider a (simple) outdoor navigation example



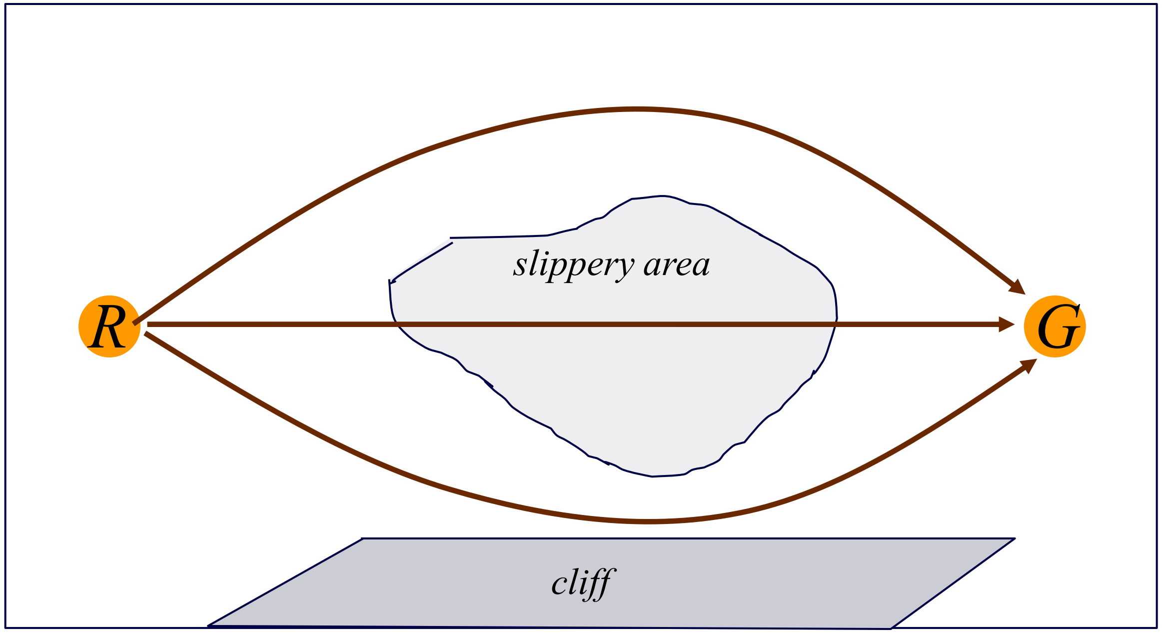
Modeled as graph search



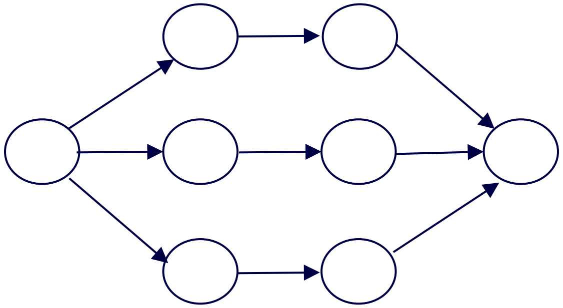
Example

- 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)?



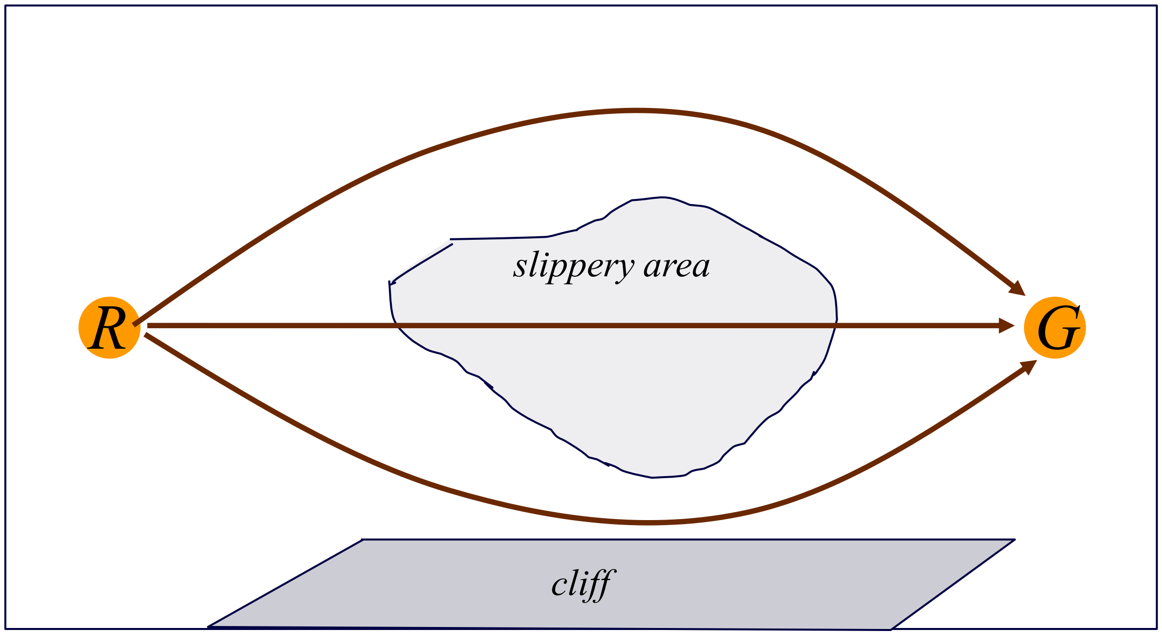
Modeled as graph search



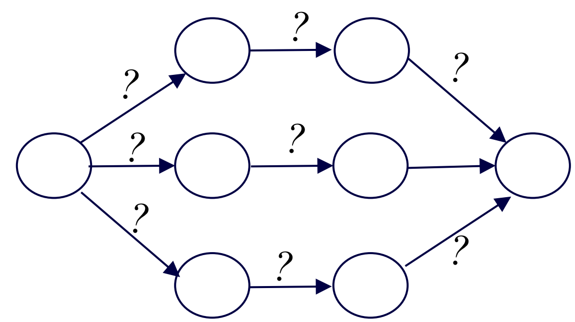
Example

- 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

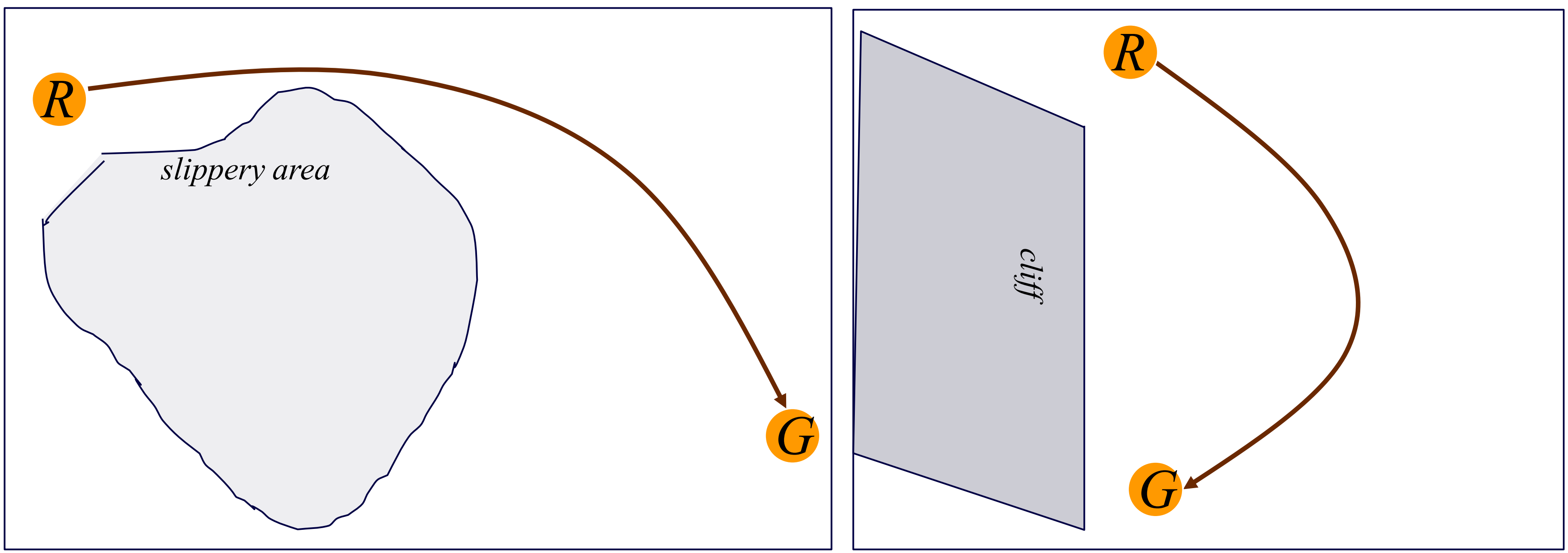


Example

- 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)?

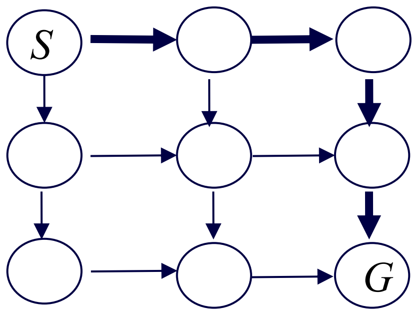
A user gives N demonstrations of what paths are good.
We want a cost function for which these demonstrated trajectories are least-cost plans



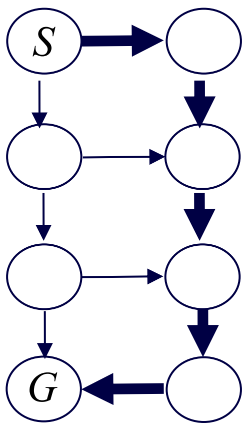
Example

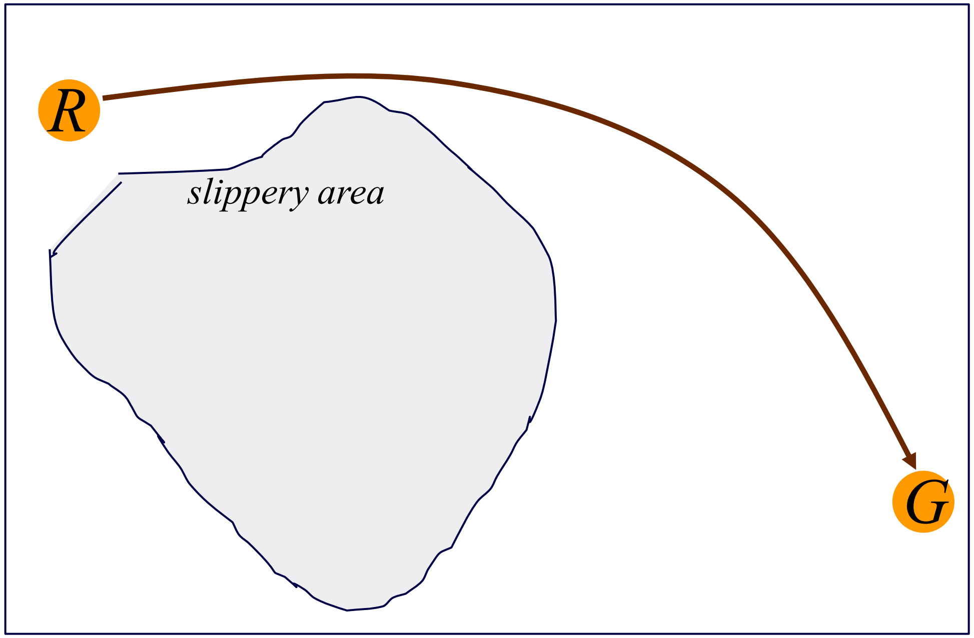
- Consider a (simple) outdoor navigation example

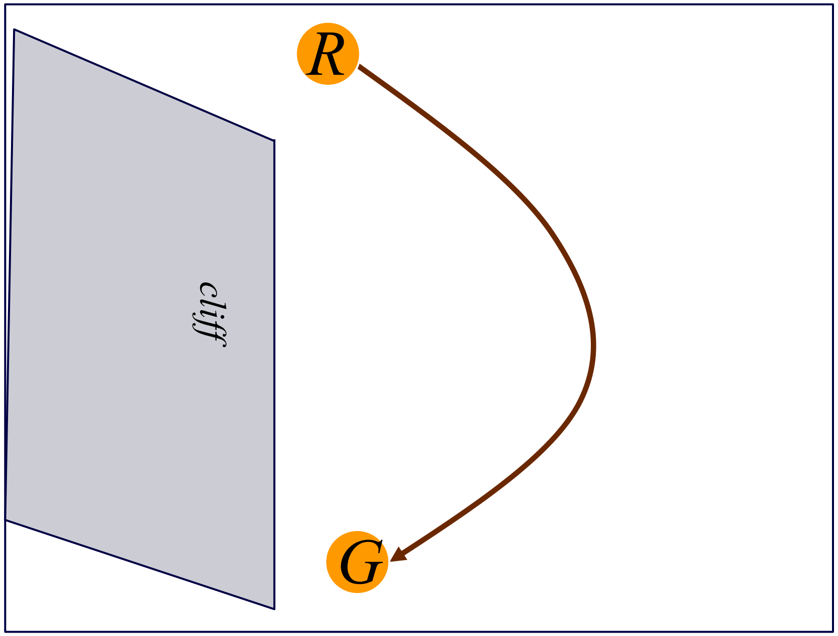
Demonstration d_1 on graph G_1



Demonstration d_2 on graph G_2



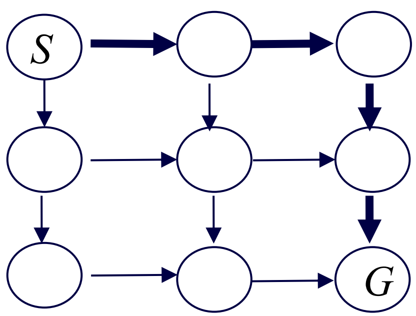




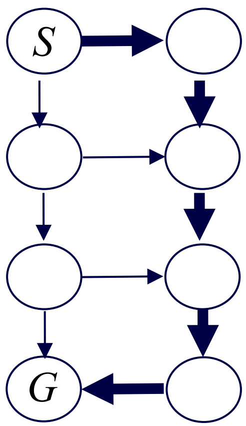
Example

- Consider a (simple) outdoor navigation example

Demonstration d_1 on graph G_1



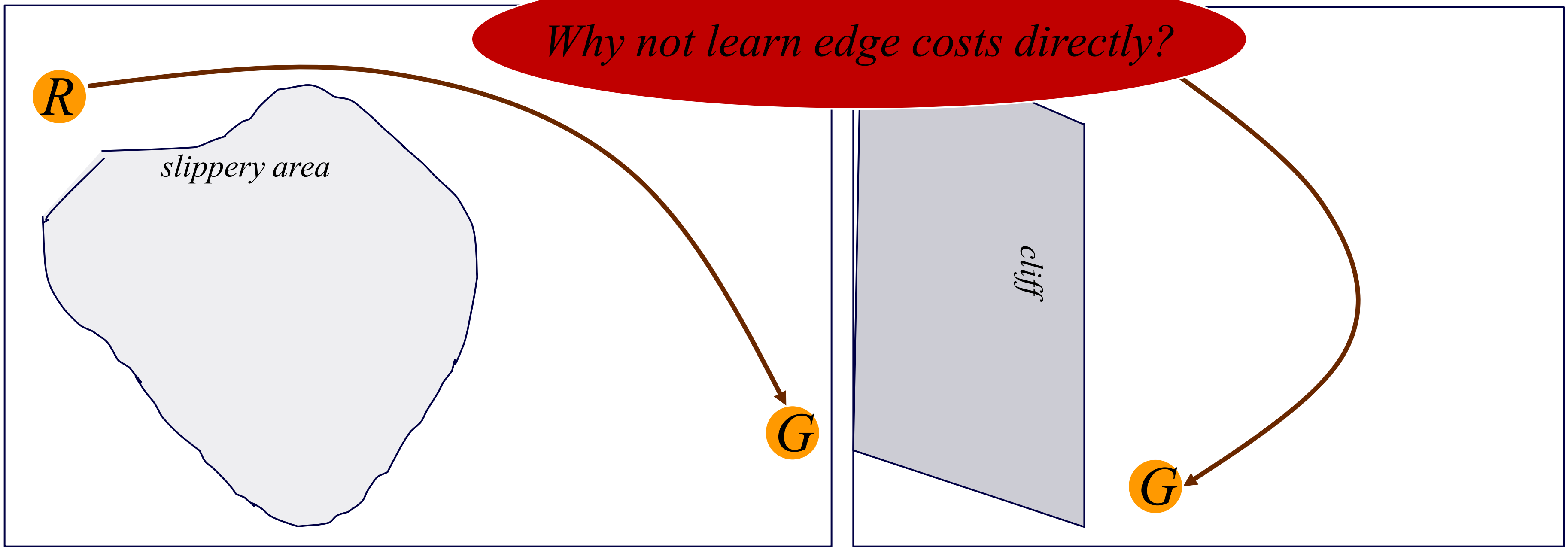
Demonstration d_2 on graph G_2



Compute cost function that makes these demonstrations optimal paths

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

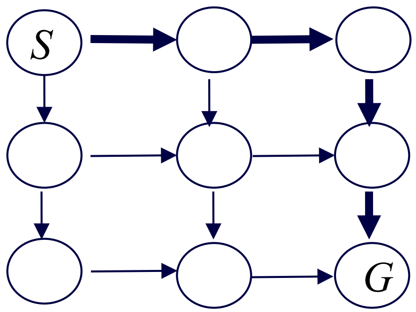
Why not learn edge costs directly?



Example

- Consider a (simple) outdoor navigation example

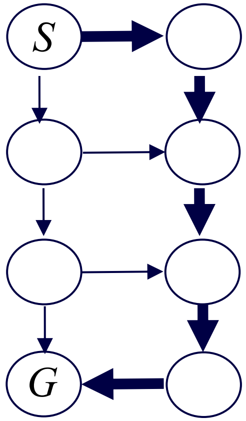
Demonstration d_1 on graph G_1



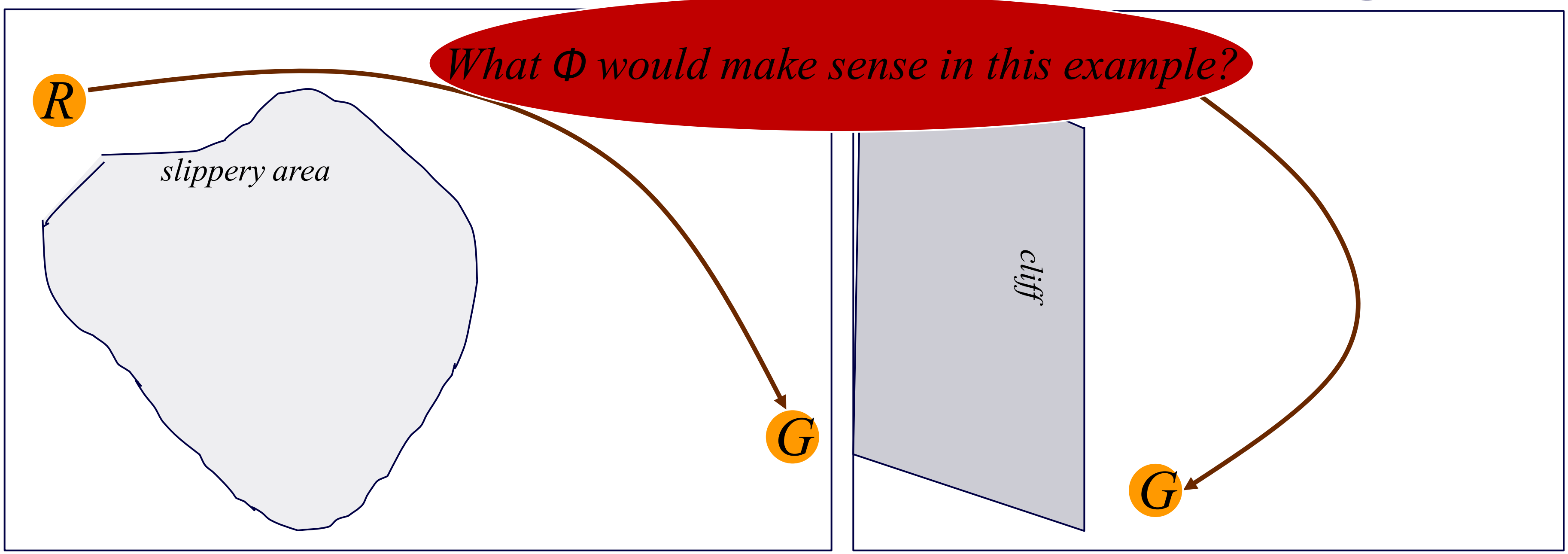
Compute cost function that makes these demonstrations optimal paths

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

Demonstration d_2 on graph G_2



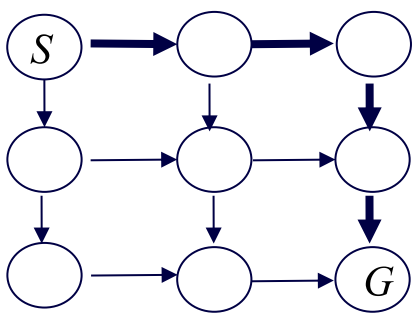
What Φ would make sense in this example?



Example

- Consider a (simple) outdoor navigation example

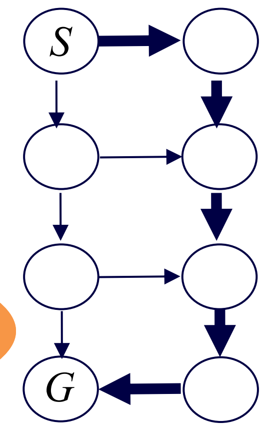
Demonstration d_1 on graph G_1



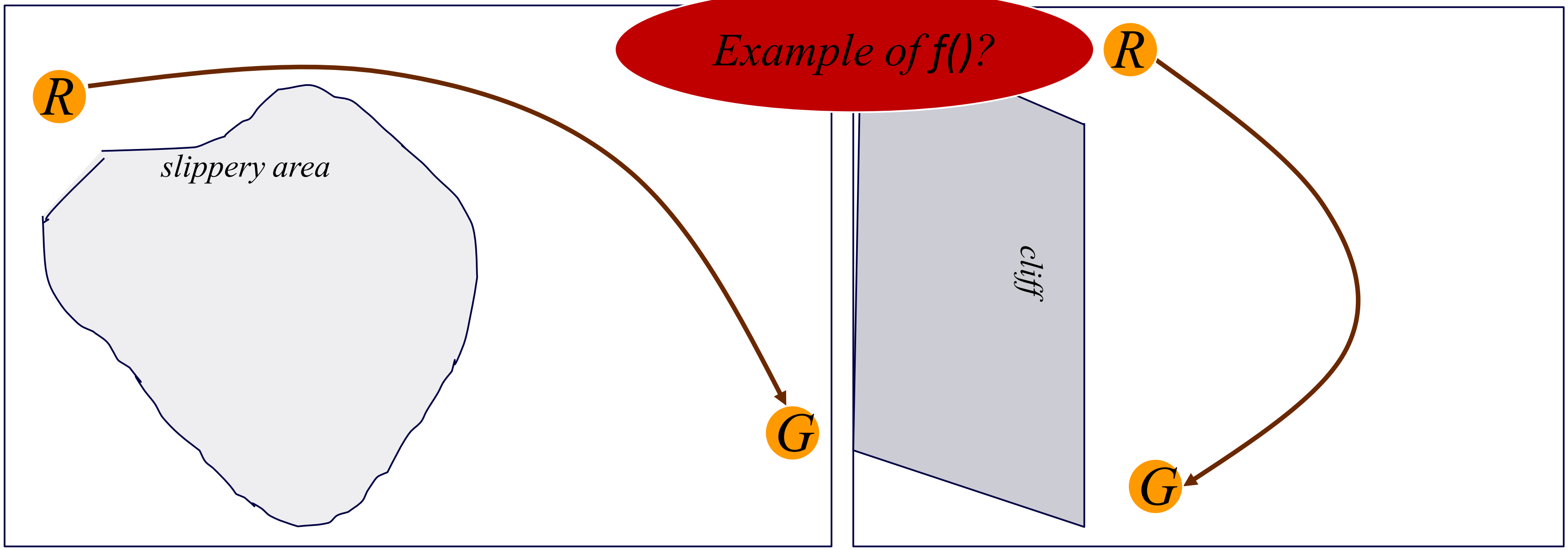
Compute cost function that makes these demonstrations optimal paths

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

Demonstration d_2 on graph G_2



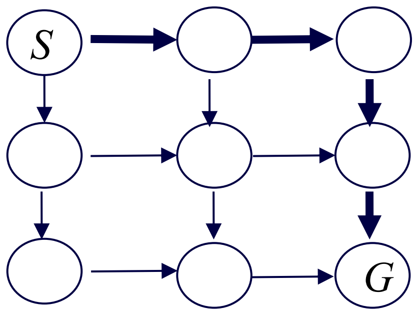
Example of f()?



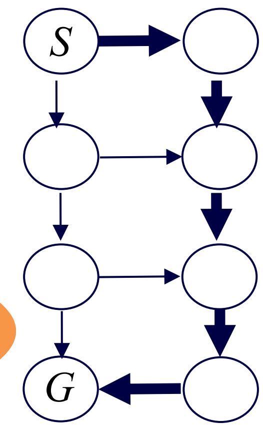
Example

- Consider a (simple) outdoor navigation example

Demonstration d_1 on graph G_1



Demonstration d_2 on graph G_2

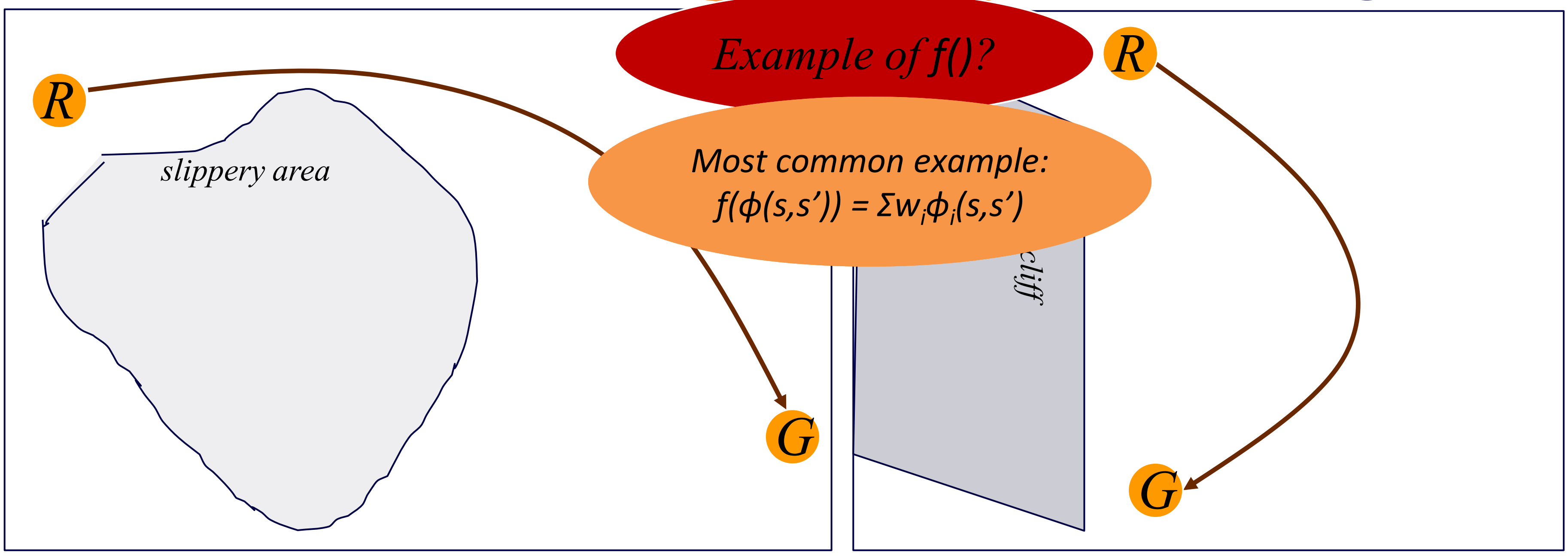


Compute cost function that makes these demonstrations optimal paths

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

Example of f()?

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



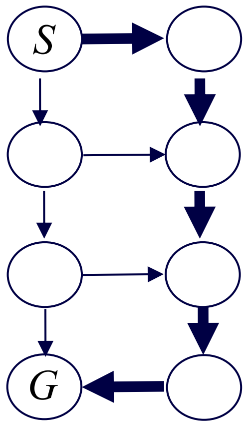
Example

- Consider a (simple) outdoor navigation example

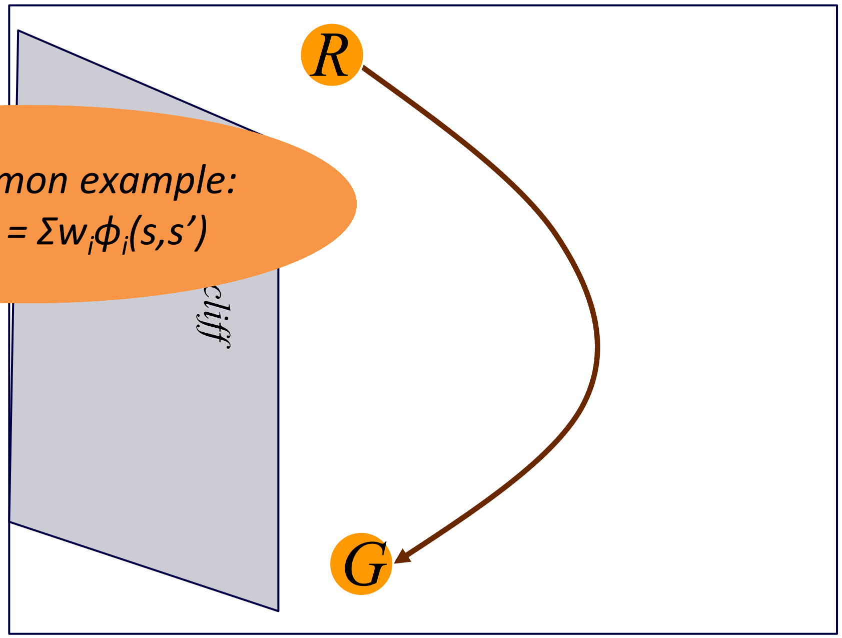
For example:
ϕ_0 : 1/(distance to slippery area)
ϕ_1 : 1/(distance to cliff)
ϕ_2 : length of the transition

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

Demonstration d_2 on graph G_2



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



LEARCH (LEArning to searCH)

[Ratliff, Silver, Bagnell, 09]

Given demonstrations $\{d_1, \ldots d_N\}$ *on graphs* $\{G_1, \ldots, G_N\}$ *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…N

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

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

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

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

LEARCH (LEArning to searCH)

[Ratliff, Silver, Bagnell, 09]

Given demonstrations $\{d_1, \ldots d_N\}$ *on graphs* $\{G_1, \ldots, G_N\}$ *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…N
update edge costs in graph G_i *using the current function* $f(\phi(,))$
plan an optimal path $\pi_i^* = \arg\min_{\pi_i} \sum_{k=0}^{length(\pi_i)-1} c(s_k, s_{k+1})$
increase $f(\phi(,))$ *for edges (u,v) s.t.* $\{(u,v)$ *in* π_i^* *AND (u,v) not in* $d_i\}$
decrease $f(\phi(,))$ *for edges (u,v) s.t.* $\{(u,v)$ *not in* π_i^* *AND (u,v) in* $d_i\}$

Is π_i^* *always guaranteed to converge to* d_i*?*

LEARCH (LEArning to searCH)

[Ratliff, Silver, Bagnell, 09]

Given demonstrations $\{d_1, \ldots d_N\}$ *on graphs* $\{G_1, \ldots, G_N\}$ *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…N

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

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

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

decrease $f(\phi(,))$ *for edges (u,v) s.t. {(u,v) not in* π_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 $\{d_1, \ldots d_N\}$ *on graphs* $\{G_1, \ldots, G_N\}$ *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…N

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

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

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

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

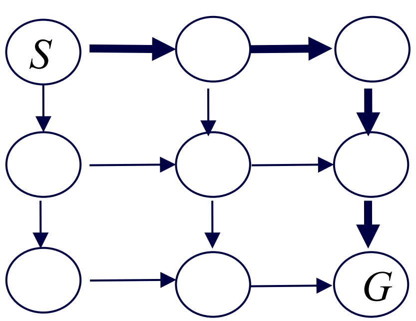
Example

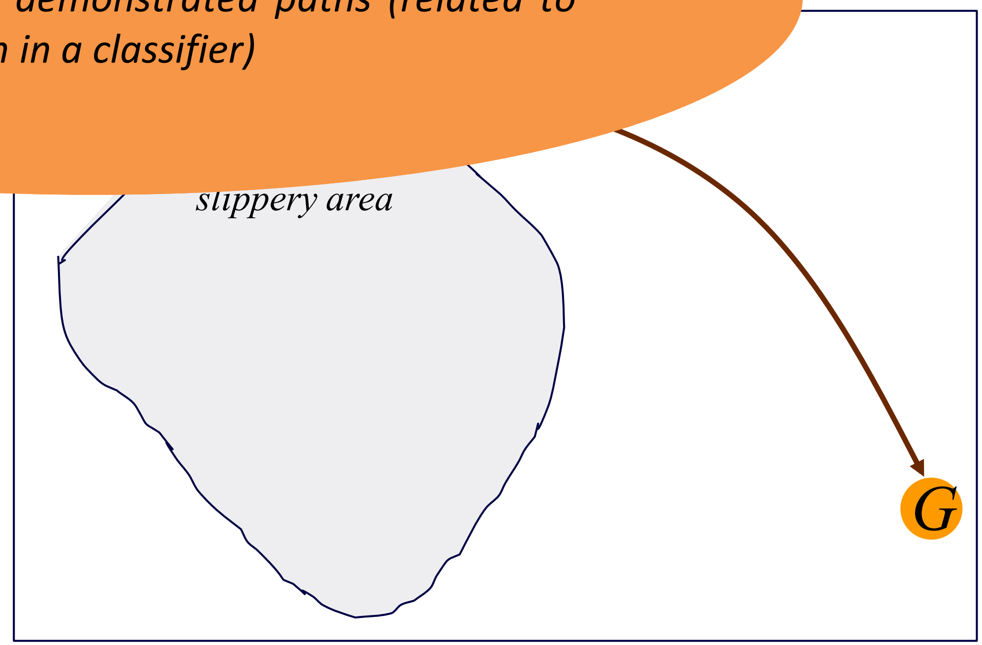
- Consider a (simple) outdoor navigation example

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 $\{d_1, \ldots d_N\}$ *on graphs* $\{G_1, \ldots, G_N\}$ *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…N
update edge costs in graph G_i *using the current function* $f(\phi(,))$
plan an optimal path $\pi_i^* = \arg\min_{\pi_i} \sum_{k=0}^{length(\pi_i)-1} \{c(s_k, s_{k+1}) - \boldsymbol{l}(\mathbf{s_k}, \mathbf{s_{k+1}})\}$
increase log $f(\phi(,))$ *for edges (u,v) s.t.* $\{$*(u,v) in* π_i^* *AND (u,v) not in* $d_i\}$
decrease log $f(\phi(,))$ *for edges (u,v) s.t.* $\{$*(u,v) not in* π_i^* *AND (u,v) 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 $\{d_1,\ldots d_N\}$ *on graphs* $\{G_1,\ldots,G_N\}$ *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…N

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

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

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

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

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

LEARCH (LEArning to searCH)

[Ratliff, Silver, Bagnell, 09]

Given demonstrations {d_1,…d_N} on graphs {G_1,…,G_N} and features function Φ

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

While (Not Converg…

for i=1…N

update edge costs in …

plan …

increase log f(φ(,)) f…

decrease l…

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

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(φ(,)) to make a step in the direction of dC

For example, if f(φ(s,s')) = Σ$w_i φ_i$(s,s')=ΦW, then:

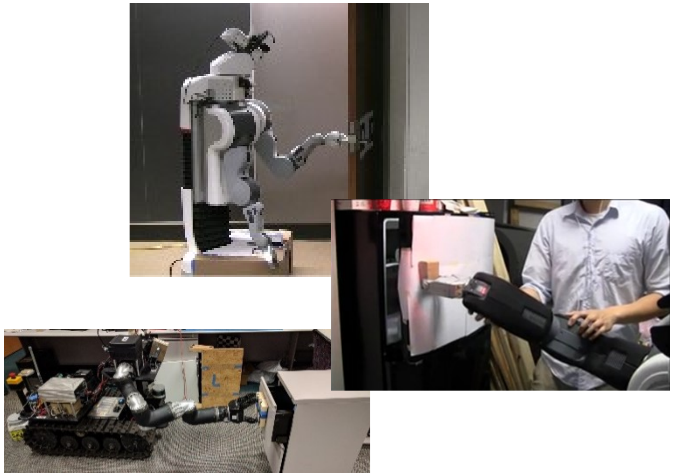
1. *Solve for vector dW from ΦdW = dC (e.g., dW = $(\Phi^T\Phi)^{-1}\Phi^T dC$)*
2. *Update W: W = W + ηdW*

Learning in Search-based Planning

Speeding up planning

Learning cost function

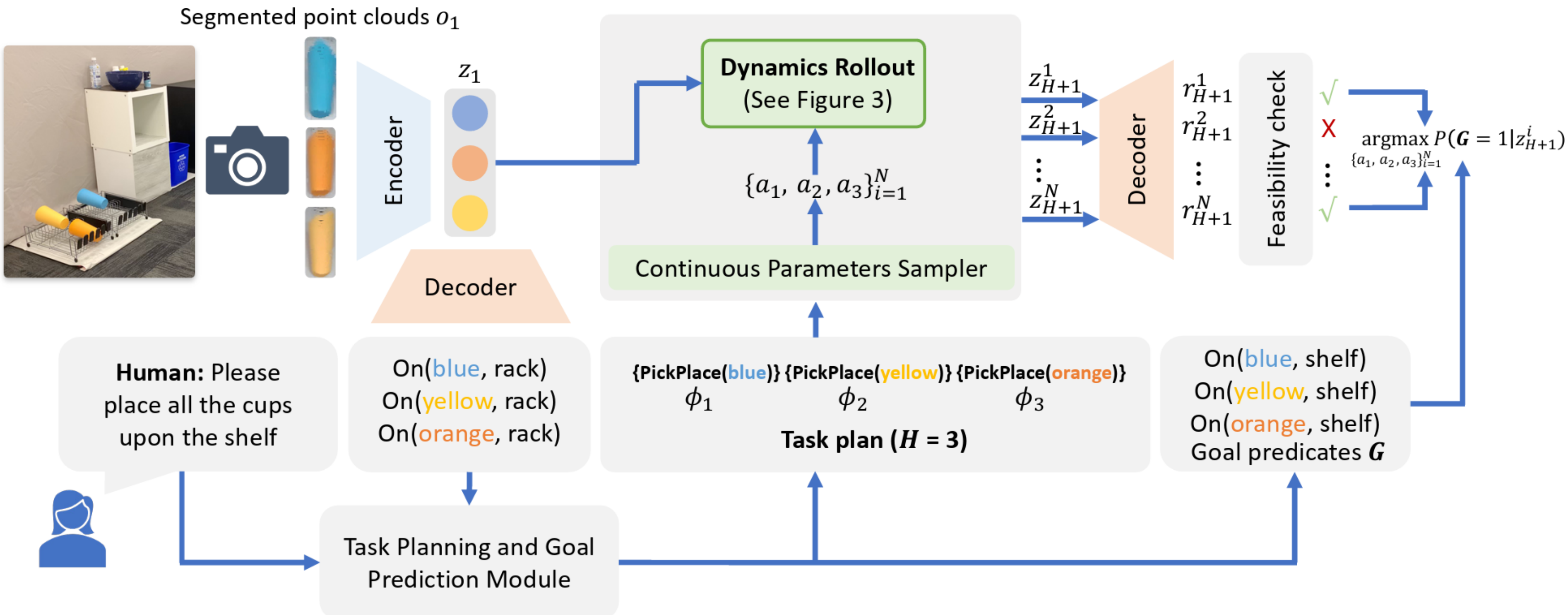
Going beyond the given model



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)
Planning directly in sensor space (Huan et al., ‘25)

Points2Plans

[Huang, Agia, Wu, Herman, and Bohg, '25]



Input:

- *instruction l*
- *segmented partial-view point clouds* o_1,

Compute plan $\tau = [\psi_1, \ldots, \psi_H]$ *that maximizes the probability of the goal implied by instruction l:*

$$\tau = \operatorname{argmax}_{G,\psi} p(l \mid G, o_1)\, p(G \mid \psi_{1:H}, o_1)$$

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