Penn Engineering
UNIVERSITY of PENNSYLVANIA

Compositional Reinforcement Learning from Logical Specifications

Kishor Jothimurugan

Suguman Bansal

Osbert Bastani

Rajeev Alur

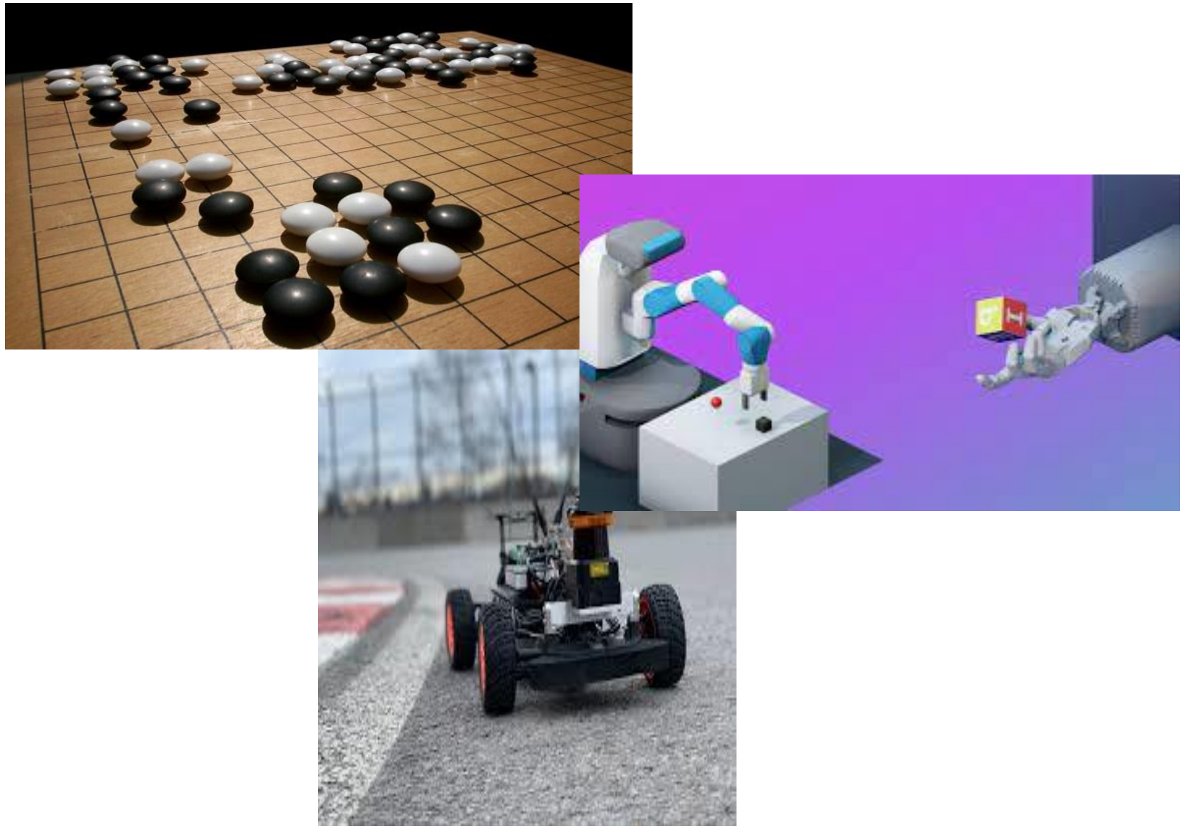
NeurIPS 2021

Reinforcement Learning (RL)

Environment

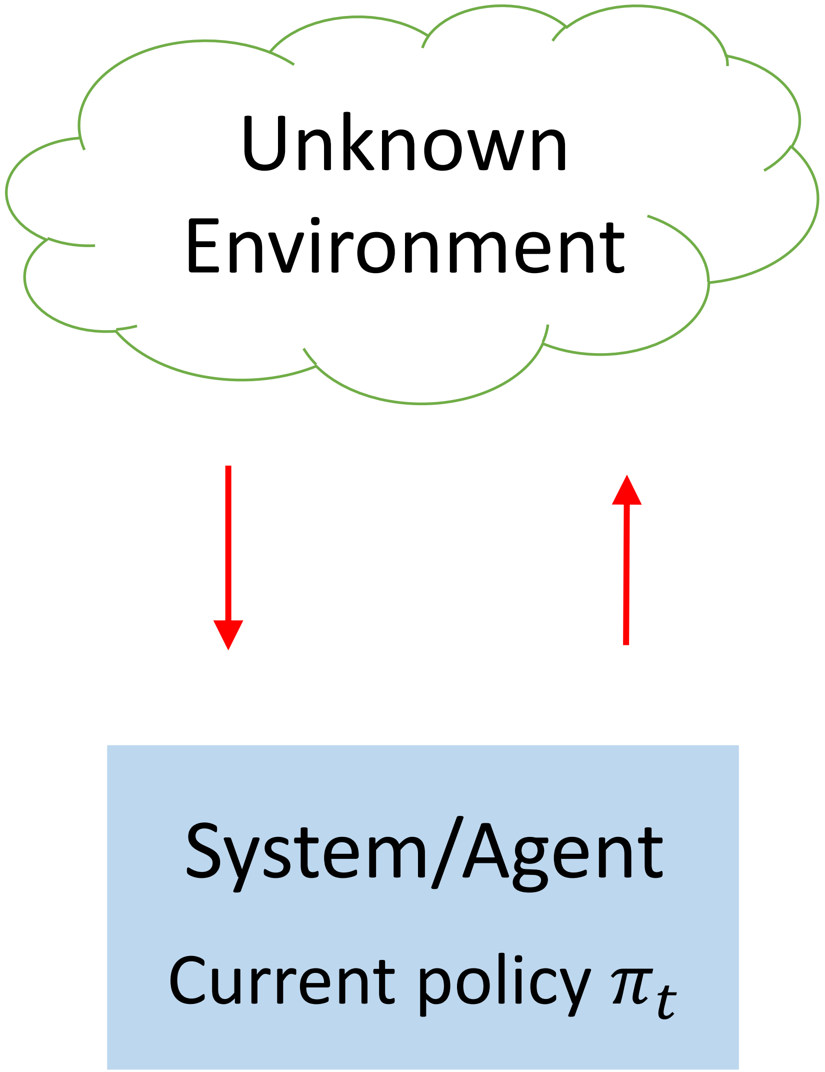
System/
Agent

Generate a **policy**
for system/agent



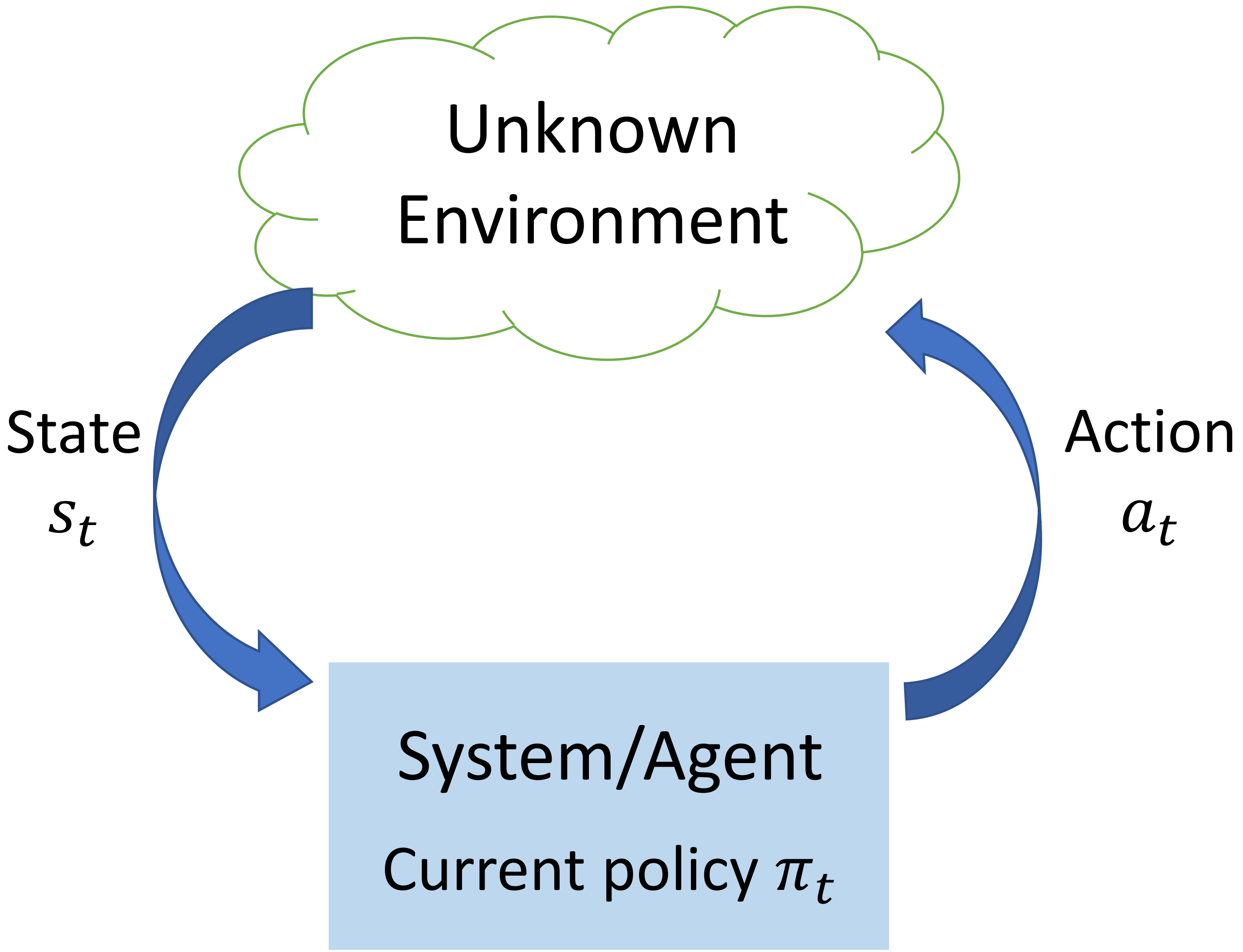
RL Algorithm

- Policy refinement loop
- Policy updated after **sampling** the environment



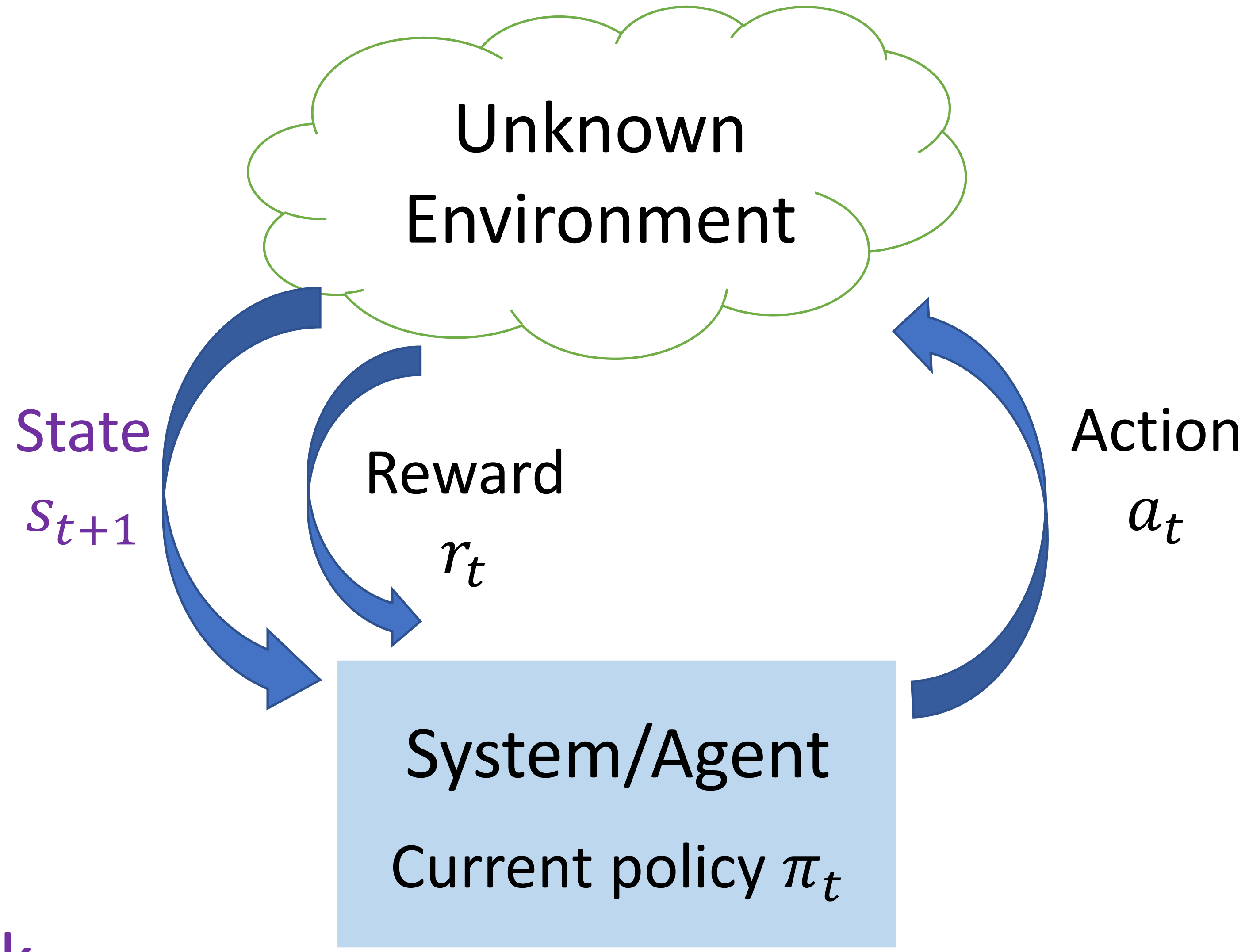
RL Algorithm

- Policy refinement loop
- Policy updated after **sampling** the environment



RL Algorithm

- Policy refinement loop
- Policy updated after **sampling** the environment
- Generate policy that optimizes **total reward**

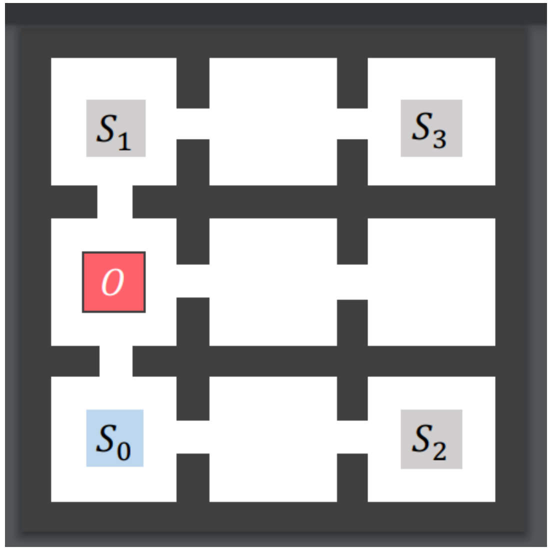


Rewards encode desired task

Hard to encode task with rewards

Environment: Continuous domain is $\mathbb{R}^2$, Initially in S_0

Task: Visit S_1 or S_2, then visit S_3. Always avoid O.




```
count = 0   # global variable

def get_rewards(s):
        if state.at(O) :
                return -10
        if count == 0 and state.at(S_1) :
                count = 1
        if count == 0 and state.at(S_2) :
                count = 1
        if count == 1 and state.at(S_3) :
                count = 0
                return 1
        return 0
```

Hard to encode task with rewards

Environment: Continuous domain is $\mathbb{R}^2$, Initially in S_0

Logical specifications to encode tasks?


```
if  count == 0  and  state.at(S_2) :
            count = 1
if  count == 1  and  state.at(S_3) :
            count = 0
            return 1
return 0
```

RL from Logical Specification

Learns policy that optimizes (probability of) satisfaction of specification

Weak Theoretical Guarantees

- No algorithm for optimal policy so far
- Non-existence of PAC algorithm for near-optimal

Practical Algorithms

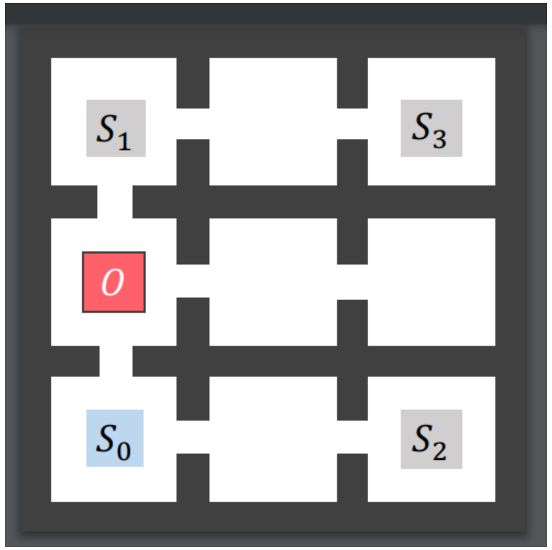
- **Compositional RL from logical specifications**
- Works on continuous environments

[1] A Framework for Transforming Specifications in Reinforcement Learning. Rajeev Alur, Suguman Bansal, Osbert Bastani, Kishor Jothimurugan. ArXiv 2021
[2] Compositional Reinforcement Learning from Logical Specifications. Kishor Jothimurugan, Suguman Bansal, Osbert Bastani and Rajeev Alur. NeurIPS 2021

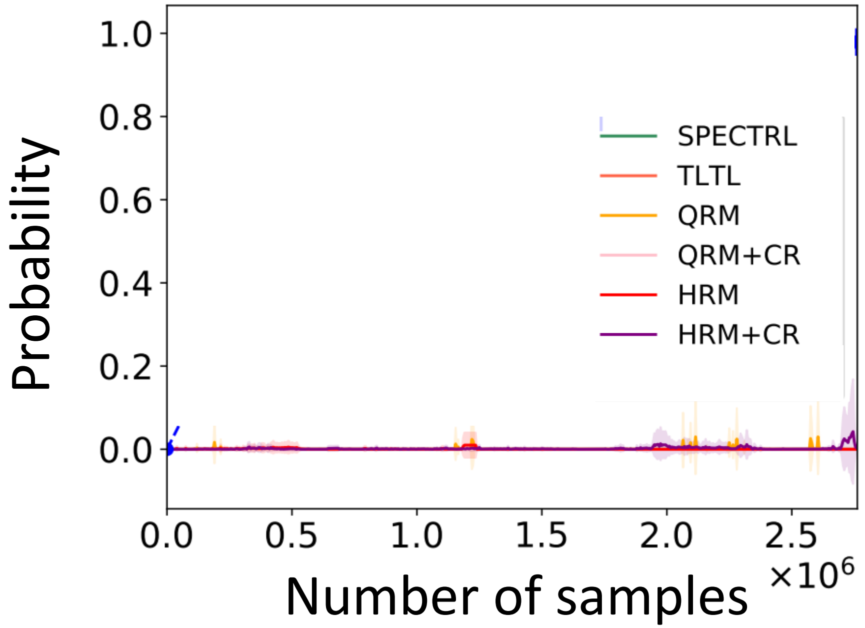
SOTA in Practical Algorithms

Environment: Continuous domain is $\mathbb{R}^2$, Initial state in S_0

Task: Visit S_1 or S_2, then visit S_3. Always avoid O.



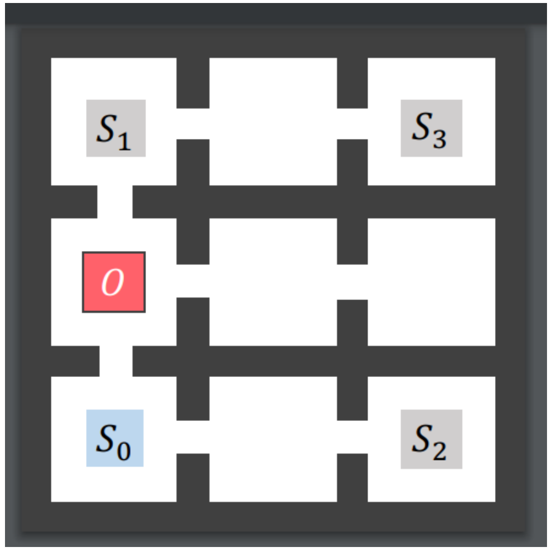
Poor Scalability



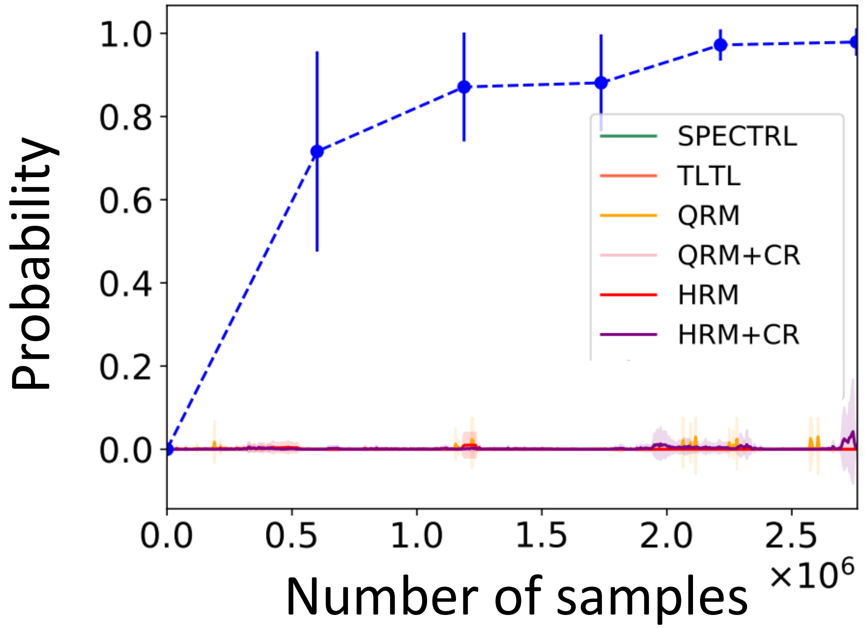
SOTA in Practical Algorithms

Environment: Continuous domain is $\mathbb{R}^2$, Initial state in S_0

Task: Visit S_1 or S_2, then visit S_3. Always avoid O.



~~Poor Scalability~~



DiRL (Ours)

Contributions

Leverage structure of logical specification to scale to long horizon tasks?

Novel compositional algorithm

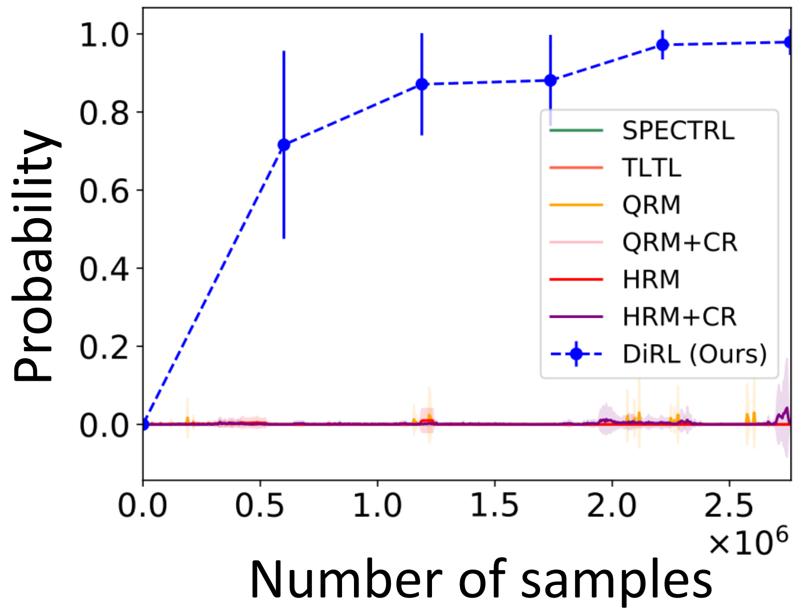
DiRL =

High-level planning on specification

+

Low-level RL on environment

Improved Scalability



Markov Decision Process (MDP)

Environment is an MDP $M = (S, A, P, \eta)$

- S is the set of states
- A is the set of actions
- $P : S \times A \times S \to [0,1]$ is the transition probability
  - $P(s, a, s')$ is the probability of transitioning to s' from s on action a
- $\eta : S \to [0,1]$ is the initial state distribution

SpectRL

[Jothimurugan, Bastani, Alur; NeurIPS 2019]

Logical specification language

- Temporal logic over predicates on the environment states
- Predicates map environment states to {True, False}

Syntax: $\varphi :=$ eventually b | φ ensuring b | φ ; φ | φ or φ

Example: "Visit S_1 or S_2 while avoiding O"

((eventually Visit S_1) or (eventually Visit S_2)) ensuring (Avoid O)

where, predicate Visit X is true in env. state s iff $s \in X$

predicate Avoid X is true in env. state s iff $s \notin X$

RL from Specifications

Given,

Environment **M** (MDP) with unknown transition probability

SpectRL specification $\boldsymbol{\varphi}$

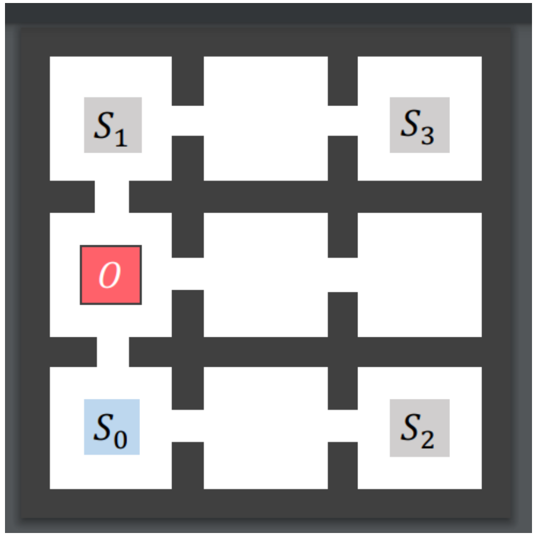
Generate,

Policy **P : (S $\times$ A)* $\times$ S -> D(A)** s.t.

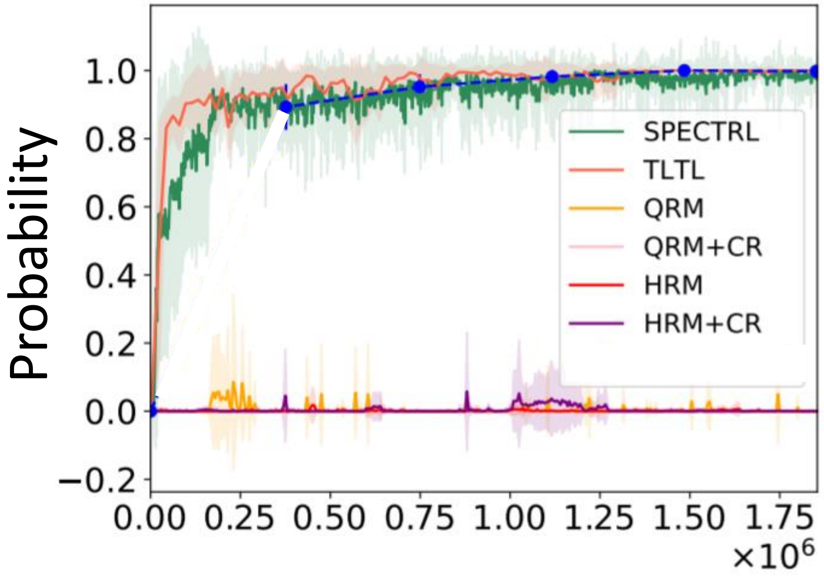
Probability that policy **P** satisfies $\boldsymbol{\varphi}$ is maximized in **M**

Challenge: Myopia in RL

RL is good at short-horizon tasks but poor at long-horizon tasks



Visit (S_1 or S_2)
while avoiding O

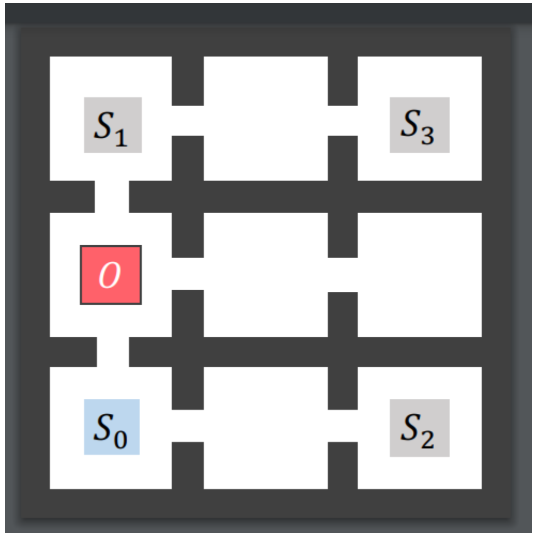


Number of samples

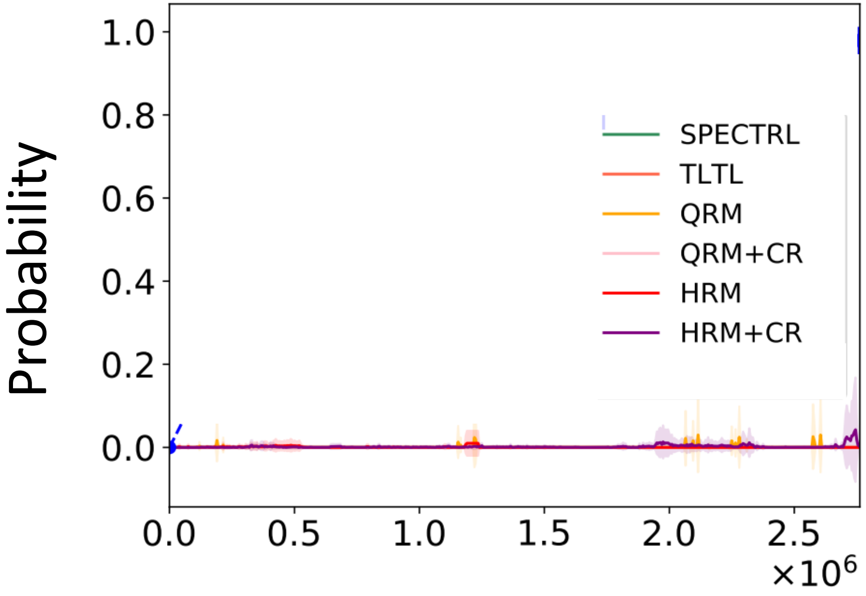
Learns to visit S_2 via obstacle-free path

Challenge: Myopia in RL

RL is good at short-horizon tasks but poor at long-horizon tasks



Visit (S_1 or S_2) then Visit S_3 while avoiding O



Number of samples

Futile to learn to visit S_2
Better to learn to visit S_1

DiRL = High-level planning + Low-level RL

Decompose specification to subtasks

↓

Learn policies for subtask
Use off-the-shelf RL

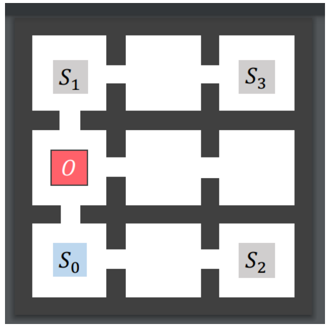
↓

Plan/Compose to compute best policy

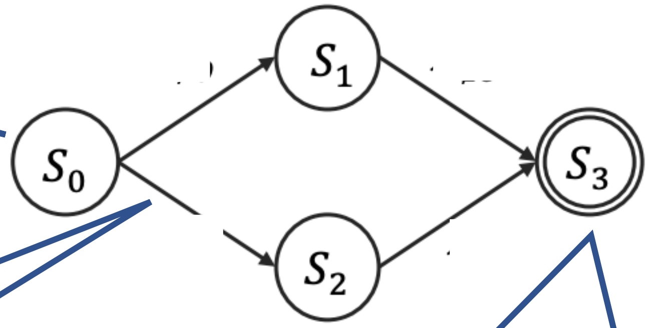
Decompose

SpectRL specifications are transformed to a DAG-like structure called **abstract graph**

Each vertex represents a set of states



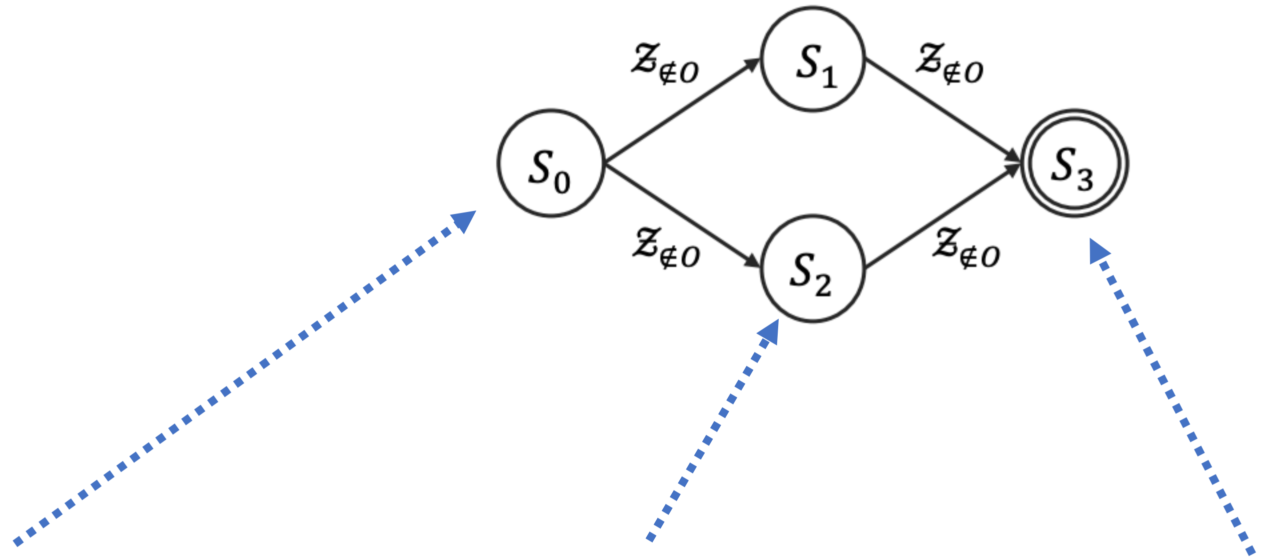
Visit (S_1 or S_2) then Visit S_3 while avoiding O $\longrightarrow$



Each edge represents a subtask

Final vertex

Satisfaction w.r.t. DAG-like structure

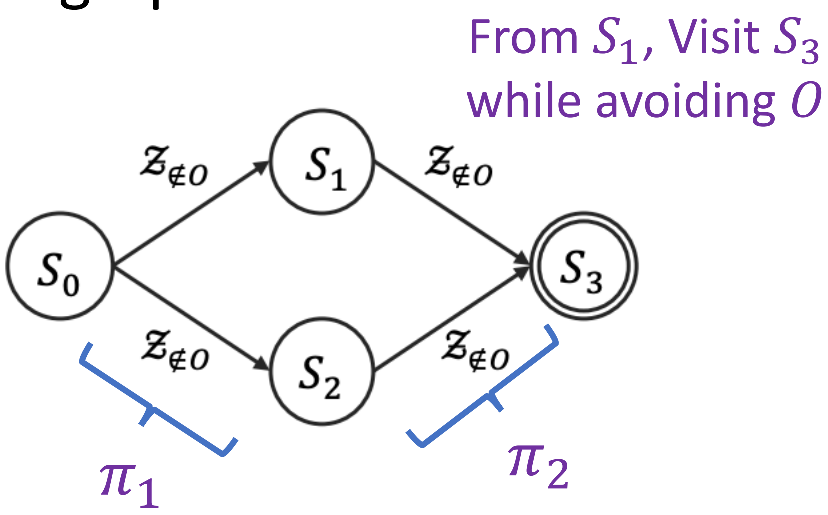


$\zeta = \ s_0 \quad s_{i_2} \quad s_{i_3} \quad \ldots\ldots$

$\zeta \vDash \varphi$ if and only if $\zeta \vDash G_\varphi$

Learn + Plan

Search for **path policies** to maximize probability to reach final vertex in abstract graph

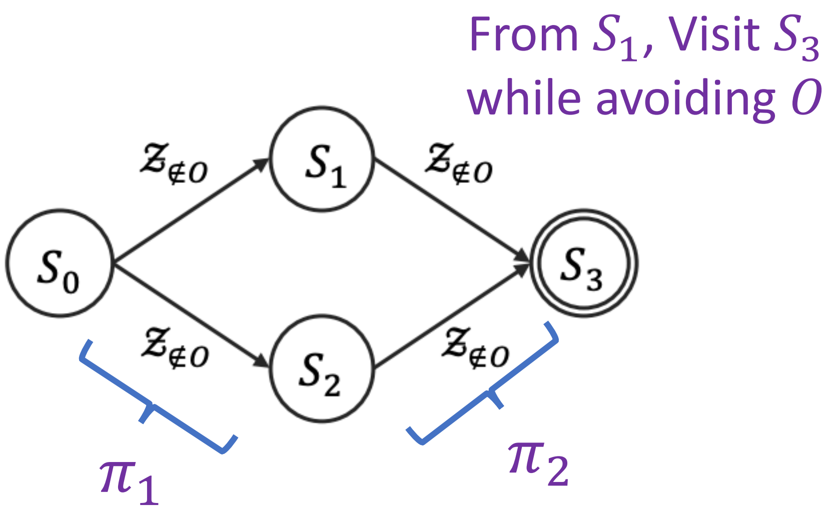


Path policy for $S_0 \rightarrow S_2 \rightarrow S_3$:

```
Execute π₁ until S₂ reached;
Execute π₂ until S₃ reached
```

Learn + Plan: Order of learning edges

Inefficient to learn S1-> S3 first. Explore states in topological order

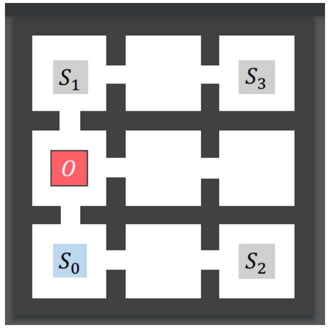


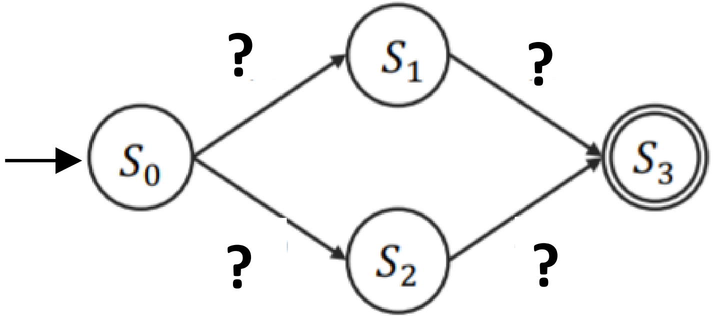
Our algorithm interleaves Dijkstra-style **planning** (searching for a path) and **learning policies** for edges in abstract graph

Learn + Plan

Obtain policies along path with max. probability to reach final state in DAG

- Learn policies for all edges (subtasks) in DAG
  - Probability of edge = Estimated probability of subtask satisfaction by policy

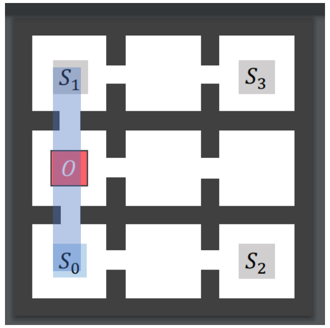


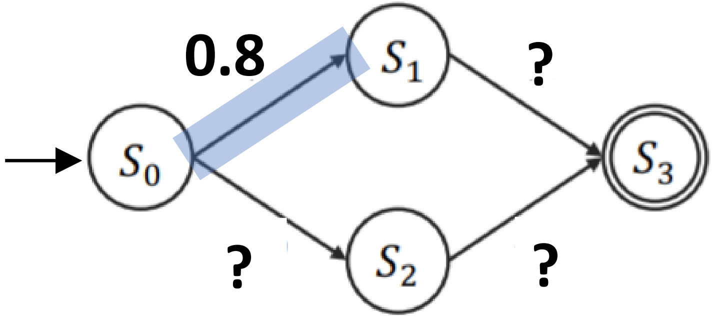


Learn + Plan

Obtain policies along path with max. probability to reach final state in DAG

- Learn policies for all edges (subtasks) in DAG
  - Probability of edge = Estimated probability of subtask satisfaction by policy

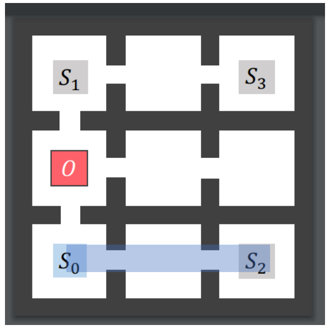


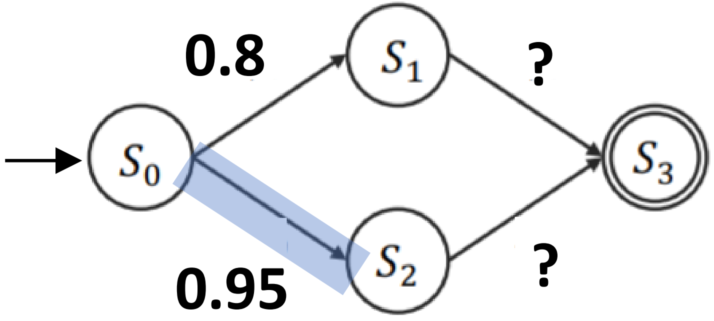


Learn + Plan

Obtain policies along path with max. probability to reach final state in DAG

- Learn policies for all edges (subtasks) in DAG
  - Probability of edge = Estimated probability of subtask satisfaction by policy

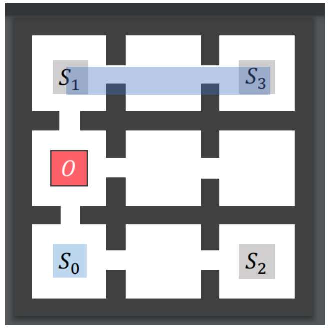


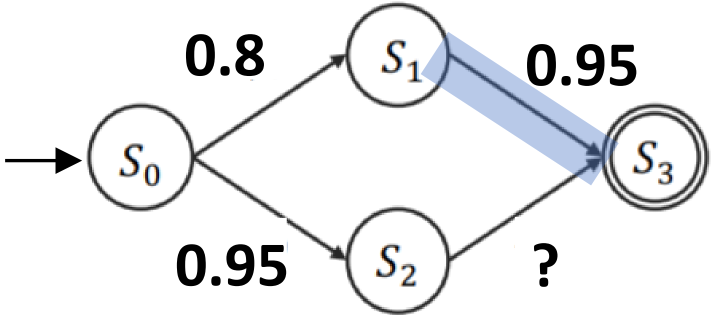


Learn + Plan

Obtain policies along path with max. probability to reach final state in DAG

- Learn policies for all edges (subtasks) in DAG
  - Probability of edge = Estimated probability of subtask satisfaction by policy

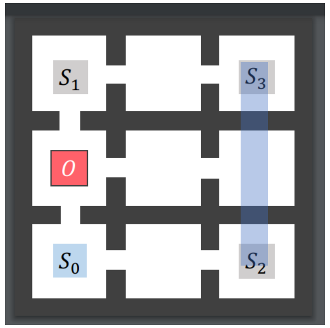


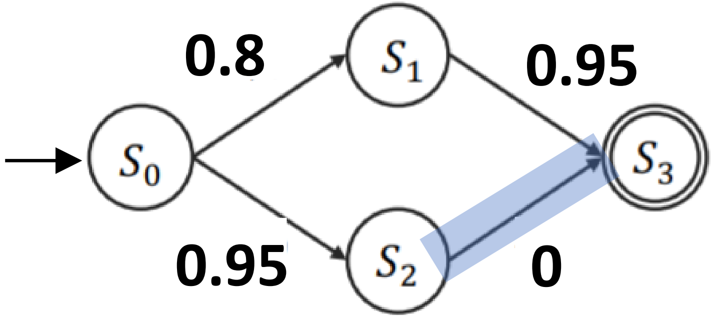


Learn + Plan

Obtain policies along path with max. probability to reach final state in DAG

- Learn policies for all edges (subtasks) in DAG
  - Probability of edge = Estimated probability of subtask satisfaction by policy

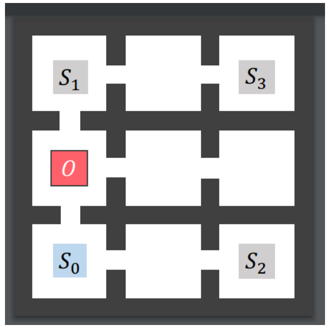


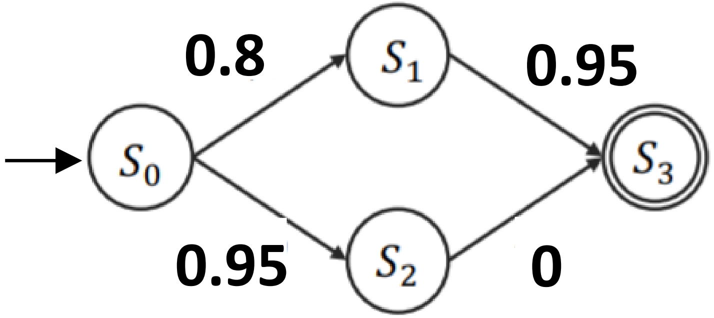


Learn + Plan

Obtain policies along path with max. probability to reach final state in DAG

- Learn policies for all edges (subtasks) in DAG
  - Probability of edge = Estimated probability of subtask satisfaction by policy

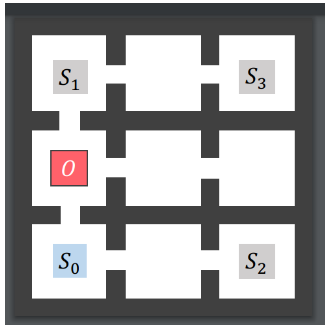


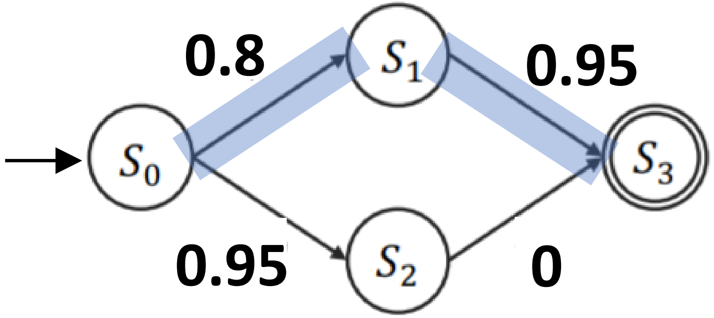


Learn + Plan

Obtain policies along path with max. probability to reach final state in DAG

- Learn policies for all edges (subtasks) in DAG
  - Probability of edge = Estimated probability of subtask satisfaction by policy





- Plan best path to final state
  - Final policy composes policies of edges on the best path

DiRL = High-level planning + Low-level RL

Decompose specification to subtasks
Leverage DAG structure of abstract graph

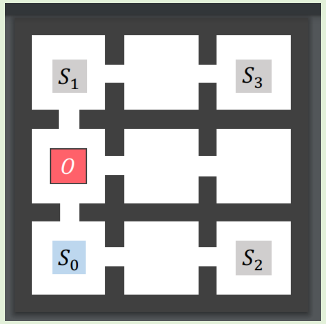
Learn policies for subtask
Use off-the-shelf RL

Plan/Compose to compute best policy
Use Dijkstra-style algorithm

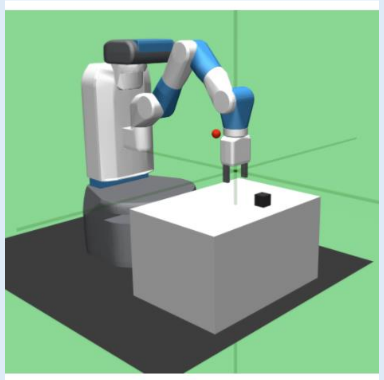
Empirical evaluation: Benchmark families

Environments with continuous states and continuous actions

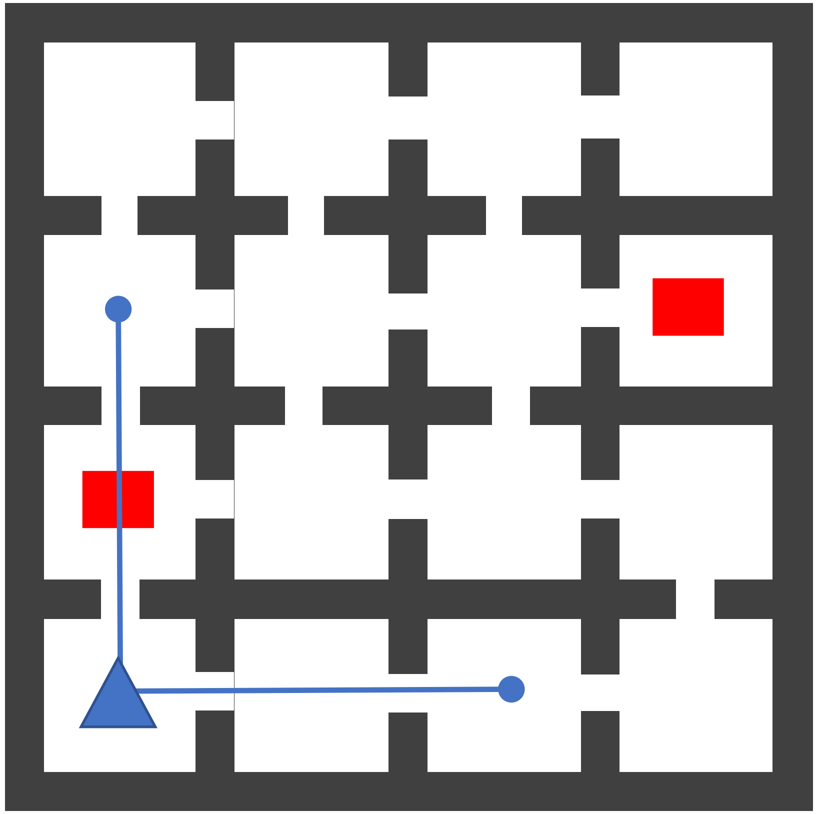
Rooms Environment

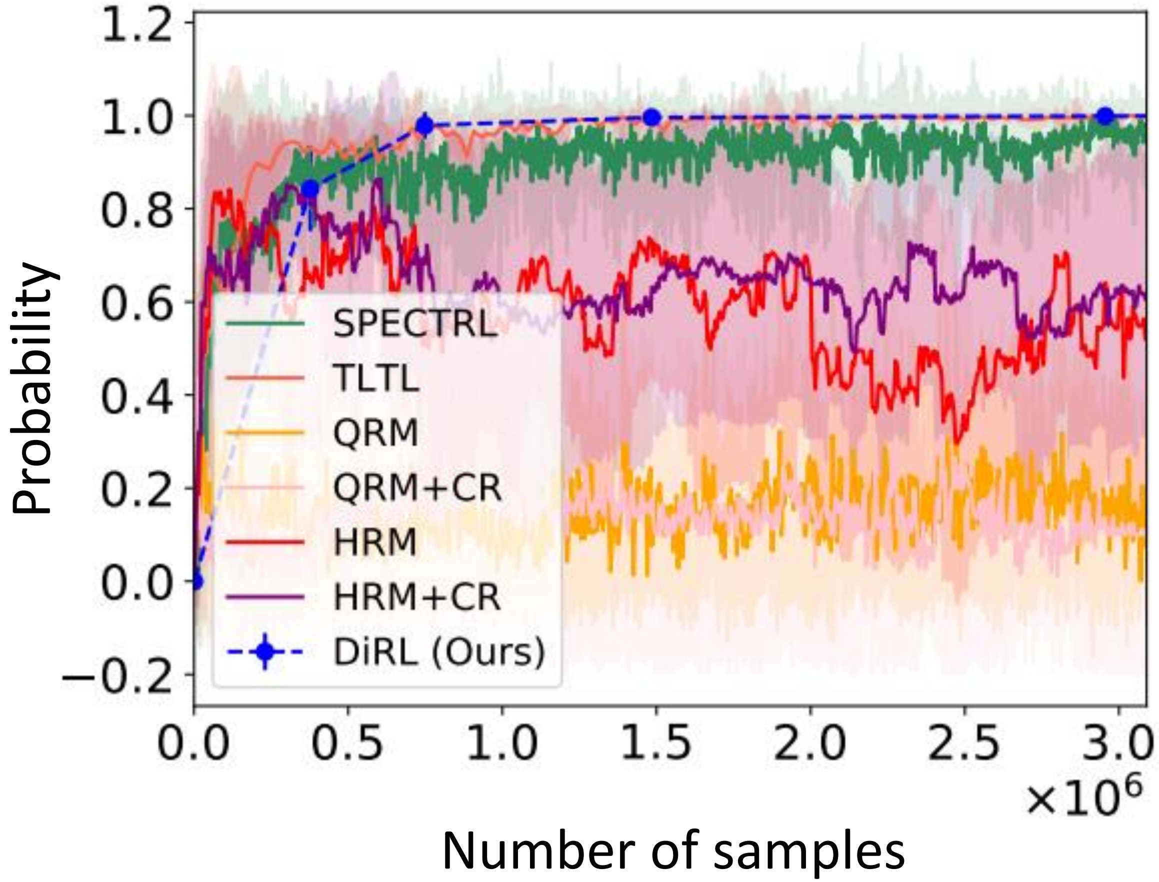


OpenAI Gym
Fetch-Pick-And-Place Environment

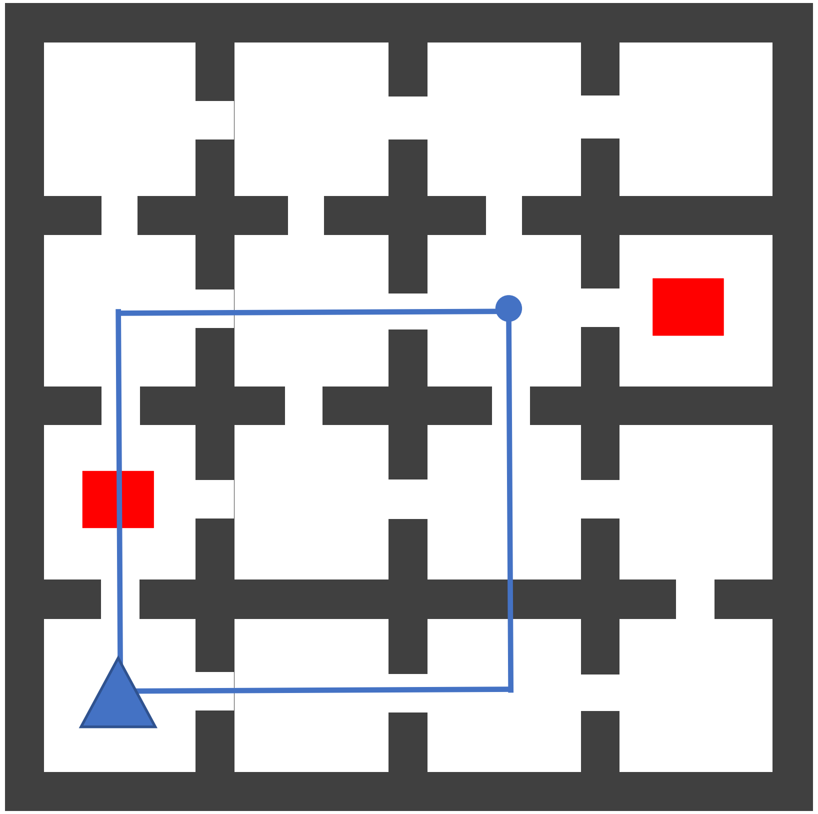


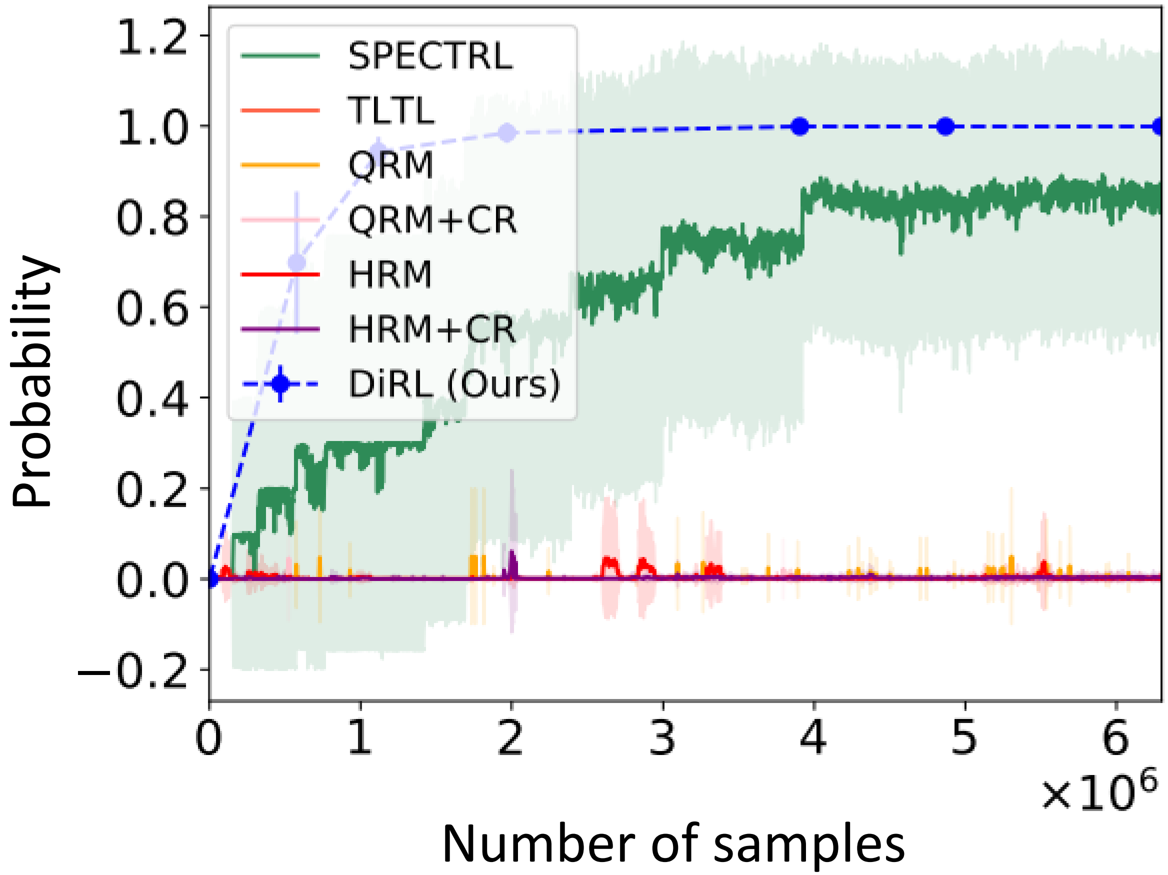
Rooms Environment



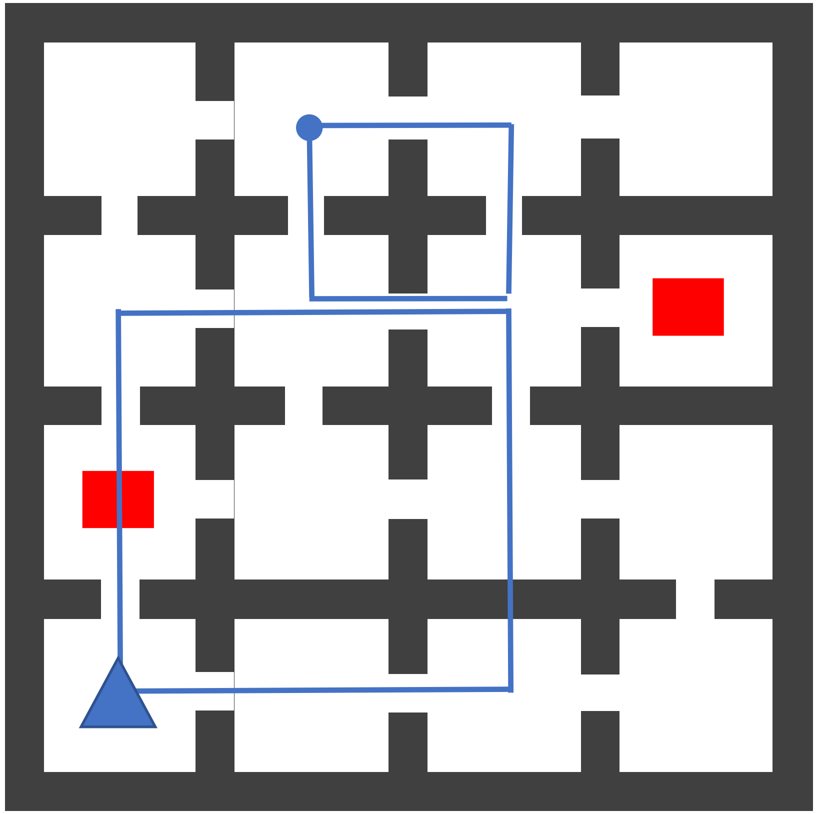


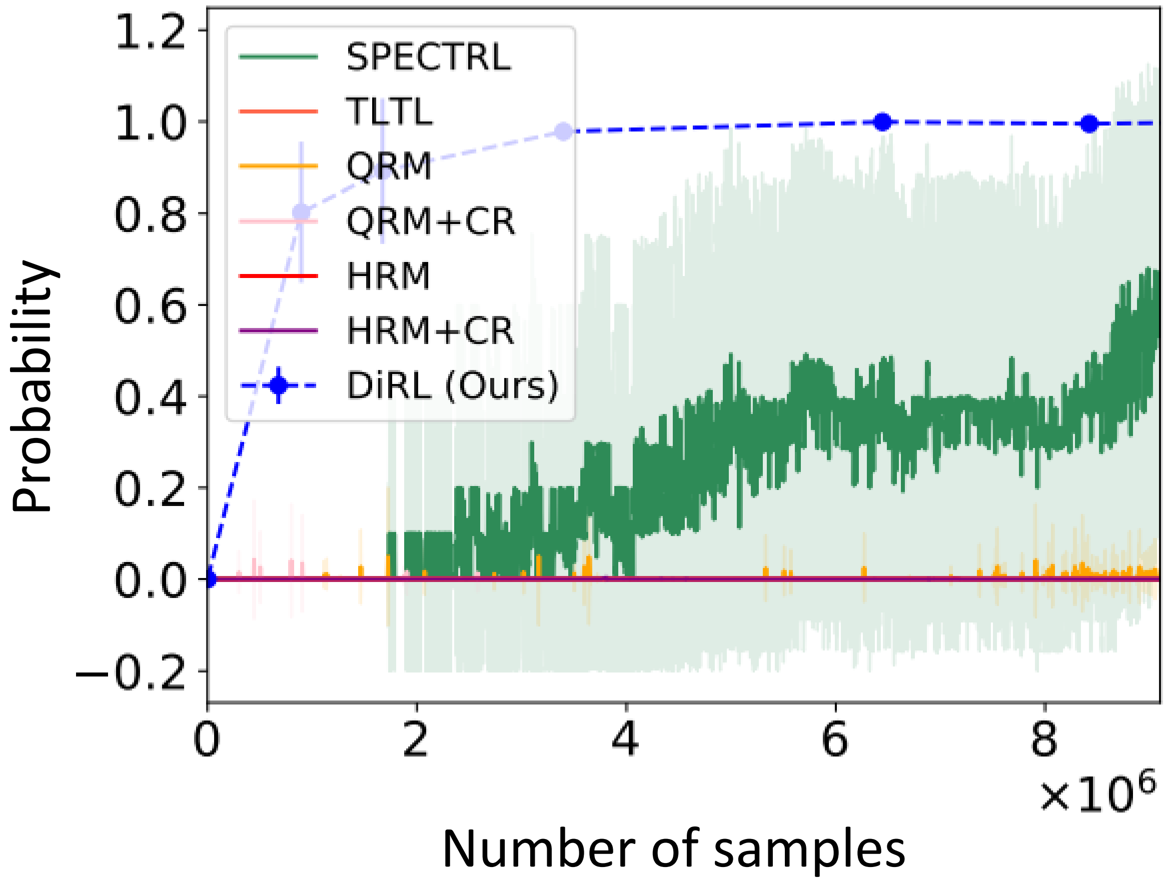
Rooms Environment



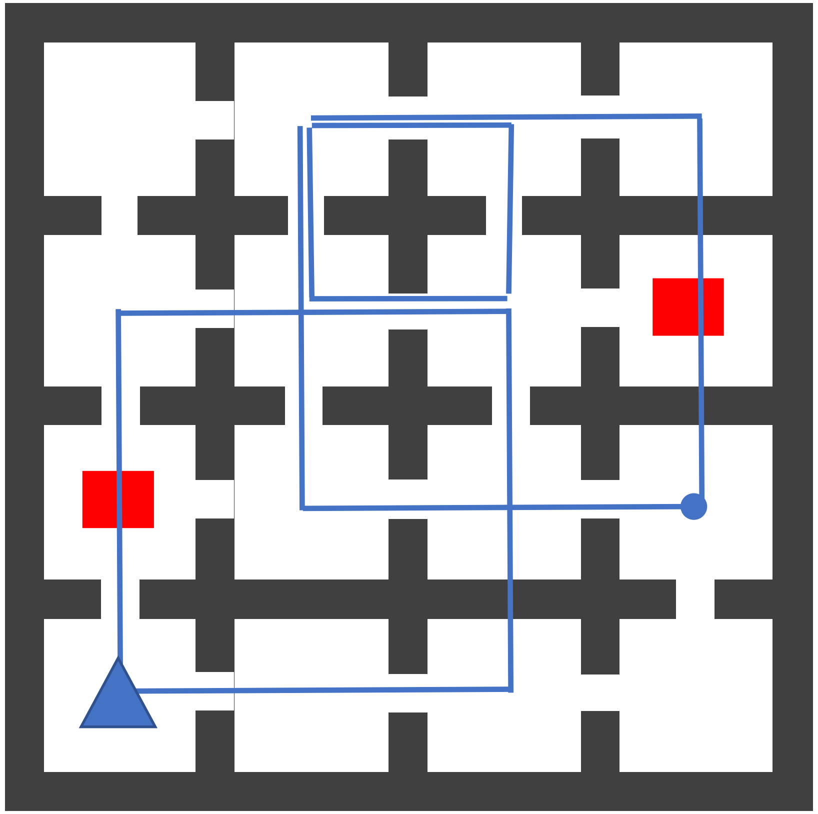


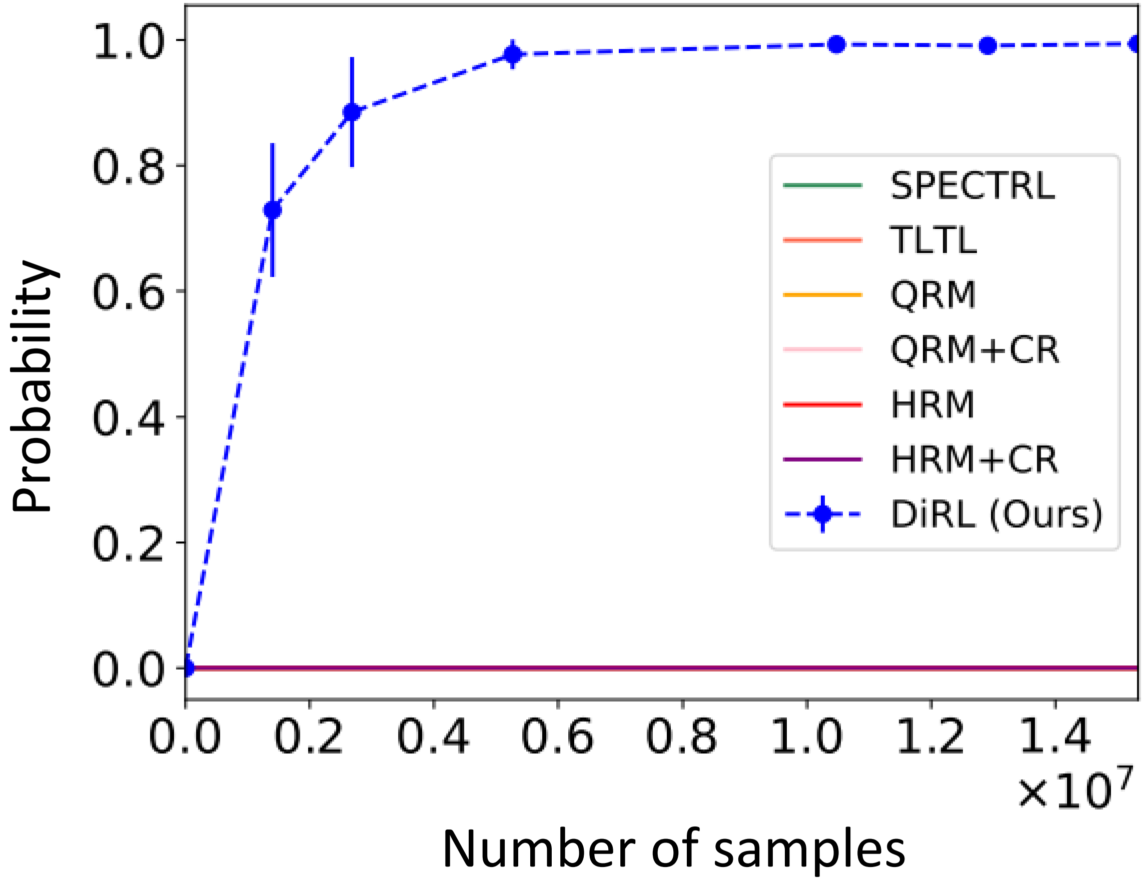
Rooms Environment



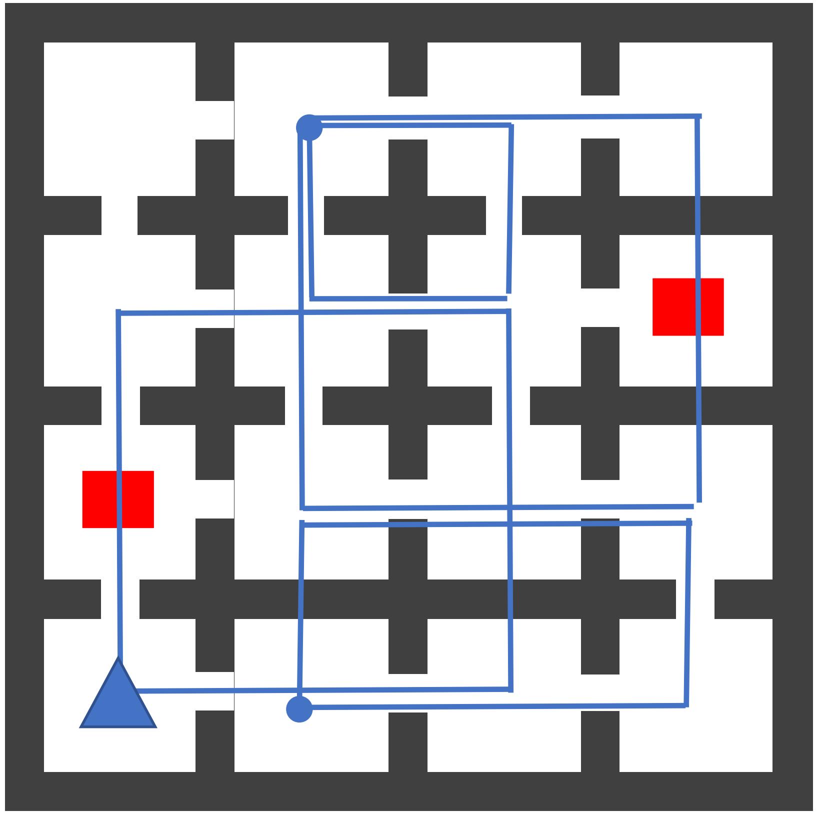


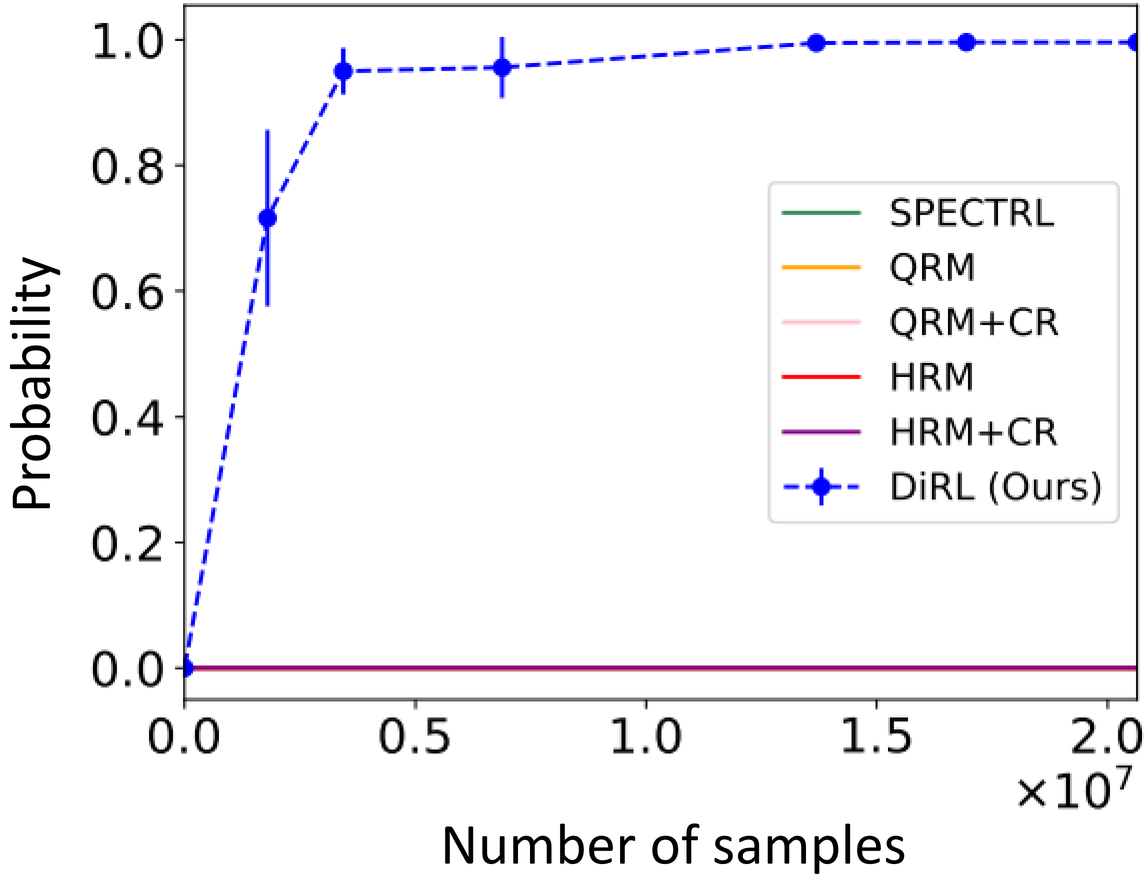
Rooms Environment



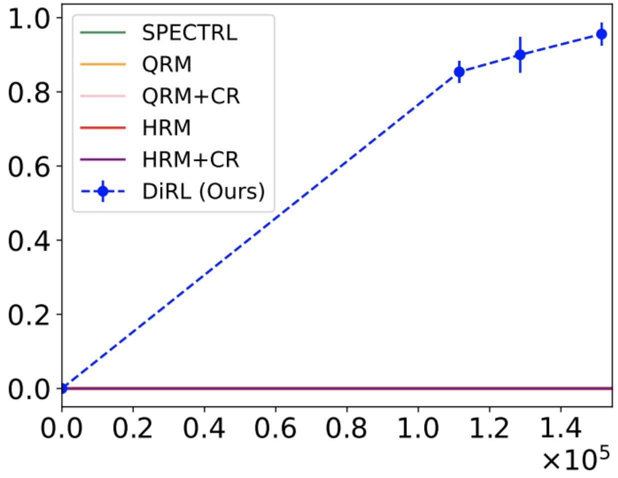


Rooms Environment

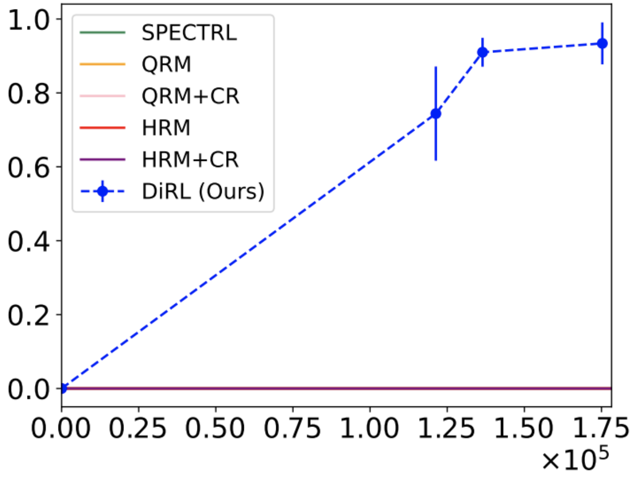




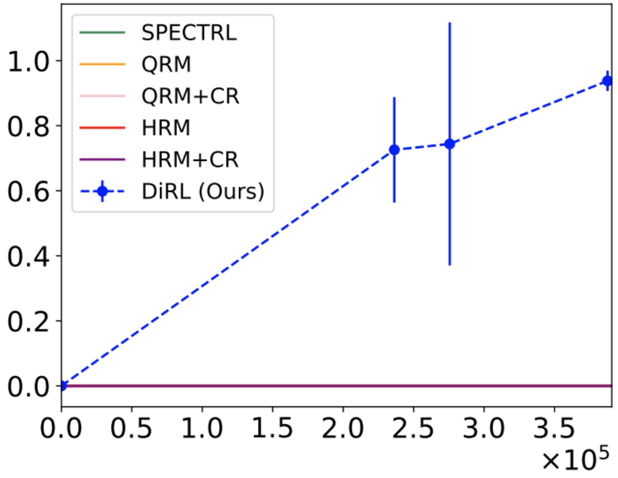
Fetch Environment



(a) PickAndPlace



(b) PickAndPlaceStatic



(c) PickAndPlaceChoice

Compositional RL from Logical Specifications @NeurIPS 2021

- Specifications are good at describing long-horizon tasks
- RL is good at learning short-horizon tasks

- DiRL = High-level planning + Low-level RL
  - Compositional algorithm
  - Scales to long-horizon tasks on continuous environments

- RL from specifications in adversarial games, multi-agent systems, etc
- Compositional verification

Compositional RL from Logical Specifications
@NeurIPS 2021

DiRL = High-level planning + Low-level RL

I. Leverages structure of specification

II. Compositional algorithm

III. Improves scalability significantly on continuous control tasks

DiRL is open-source!