Representations for Hierarchical Reinforcement Learning

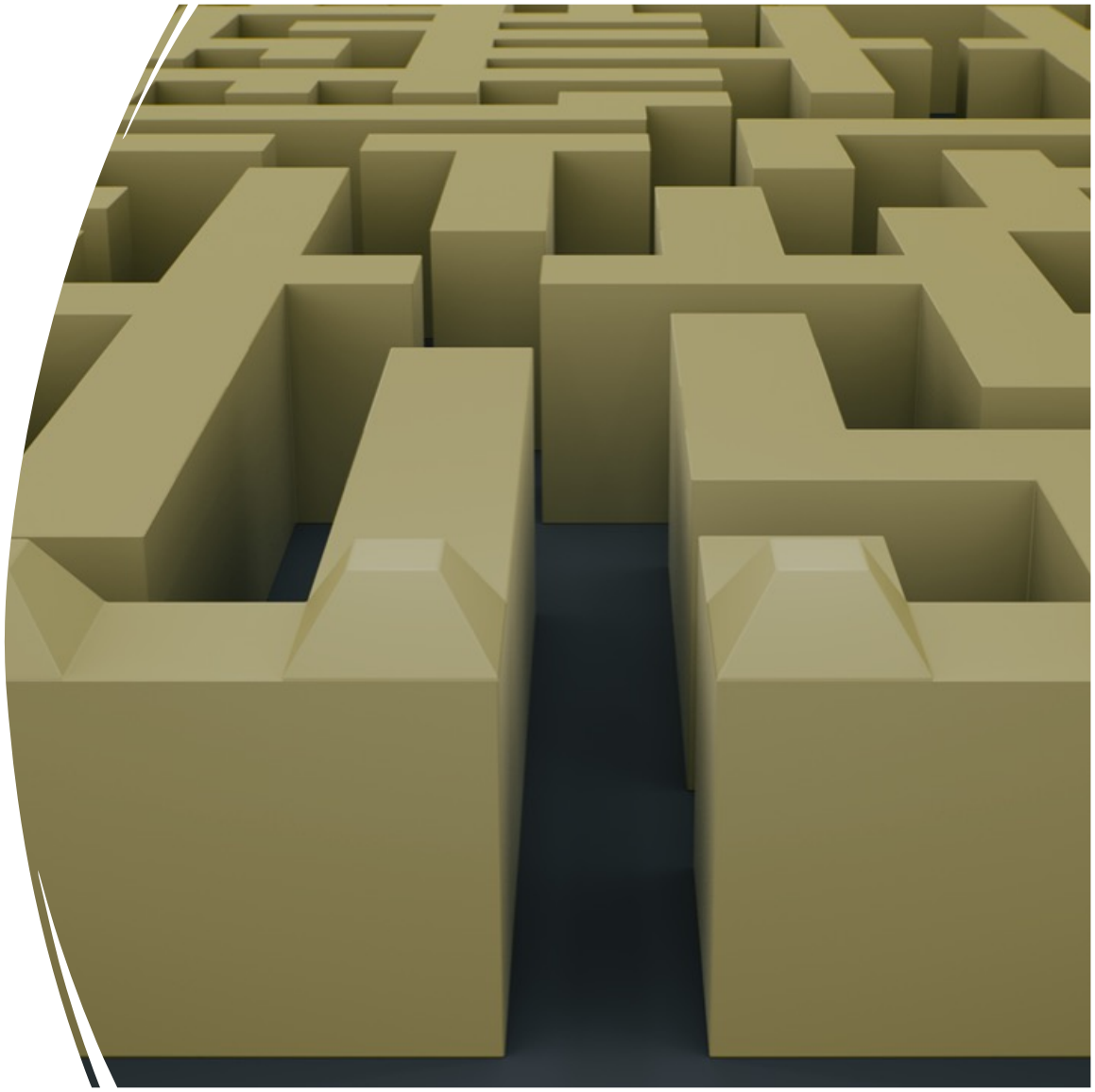
Amy Zhang

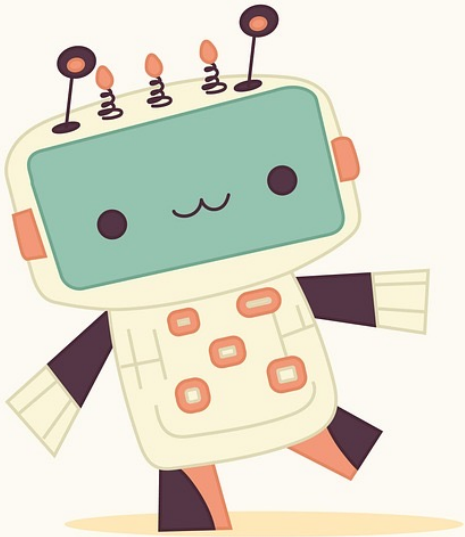
Roadmap

1. Why Hierarchy?
2. Existing hierarchical methods and their problems
3. Compositionality in Relational MDPs: A class of problems where hierarchy is well defined
4. Leverage factorized structure in relational MDPs with a hand designed relational abstraction
5. How can we learn a factored abstraction and leverage it?
6. What can we do if we can’t assume factorized structure?

Why Hierarchy?

- Long horizon tasks are hard to solve
- There may be “skills” that can get reused
- Task is easier to solve when split up into different levels of abstraction





Skill Hierarchies

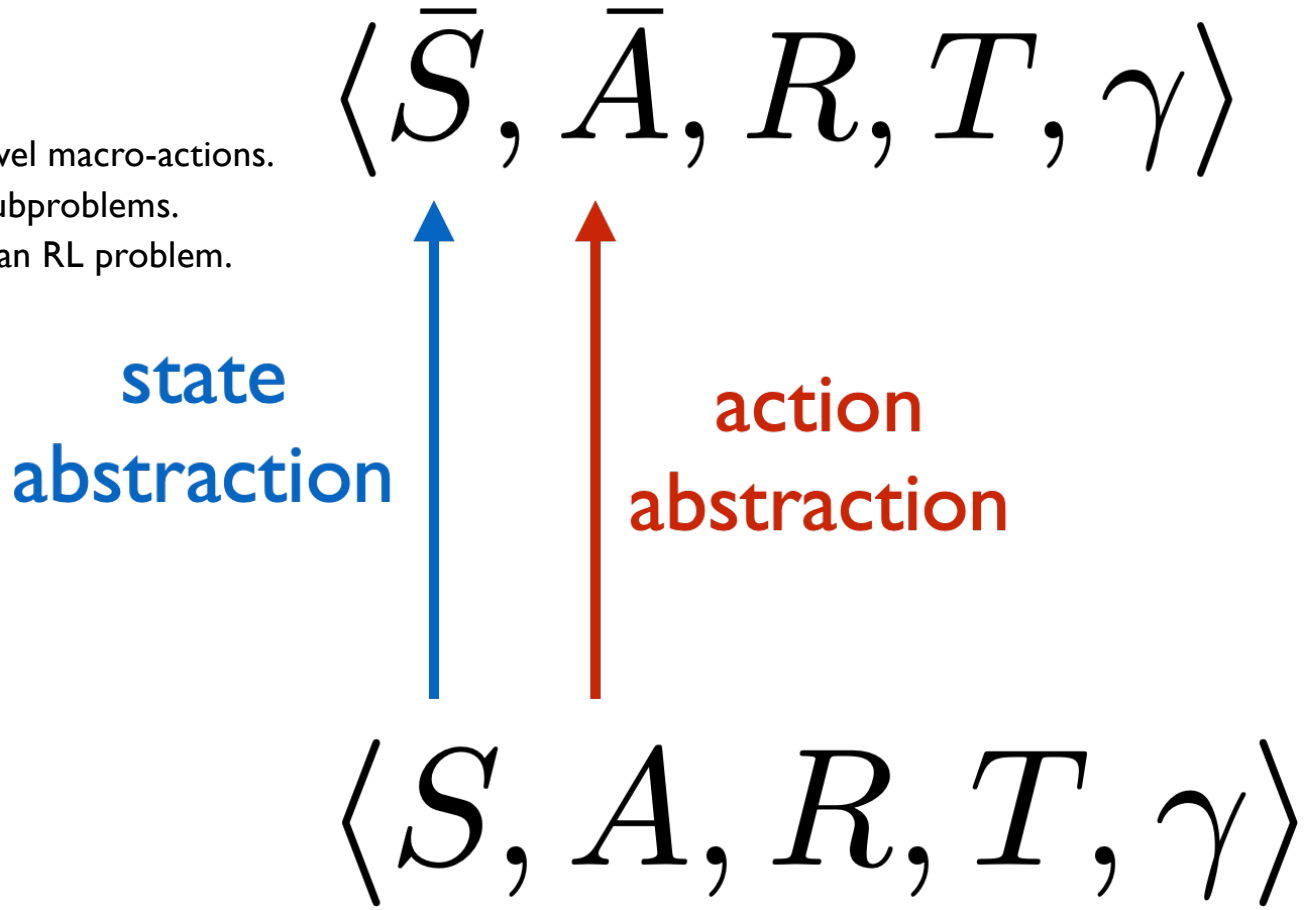
- Base hierarchical control on *skills*.
  - Component of behavior.
  - Performs continuous, low-level control.
  - Can treat as discrete action.

Behavior is modular and compositional

Forms of Abstraction

Action abstraction:

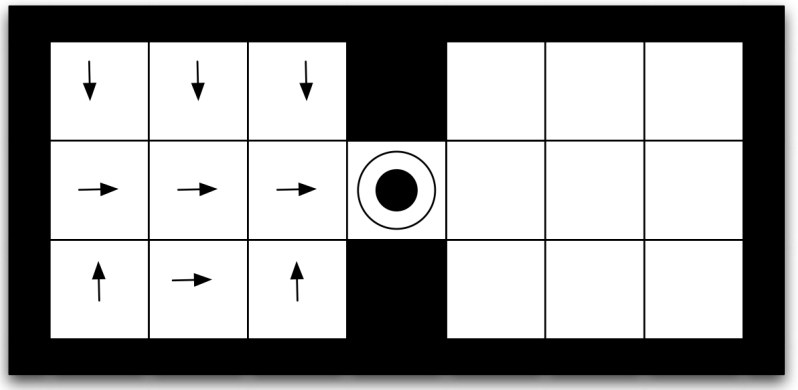
- Create and use higher-level macro-actions.
- Problem now contains subproblems.
- Each subproblem is also an RL problem.



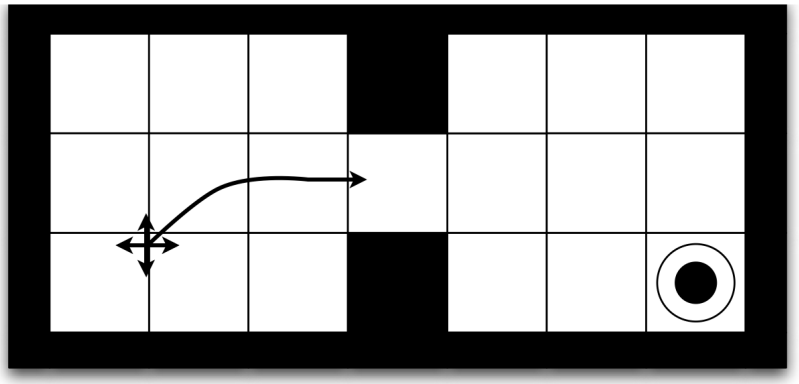
Options Framework (Sutton et al., 1999)

- Key Idea: Introduces options — temporally extended actions that consist of a policy, a termination condition, and an initiation set.

Skill



Problem

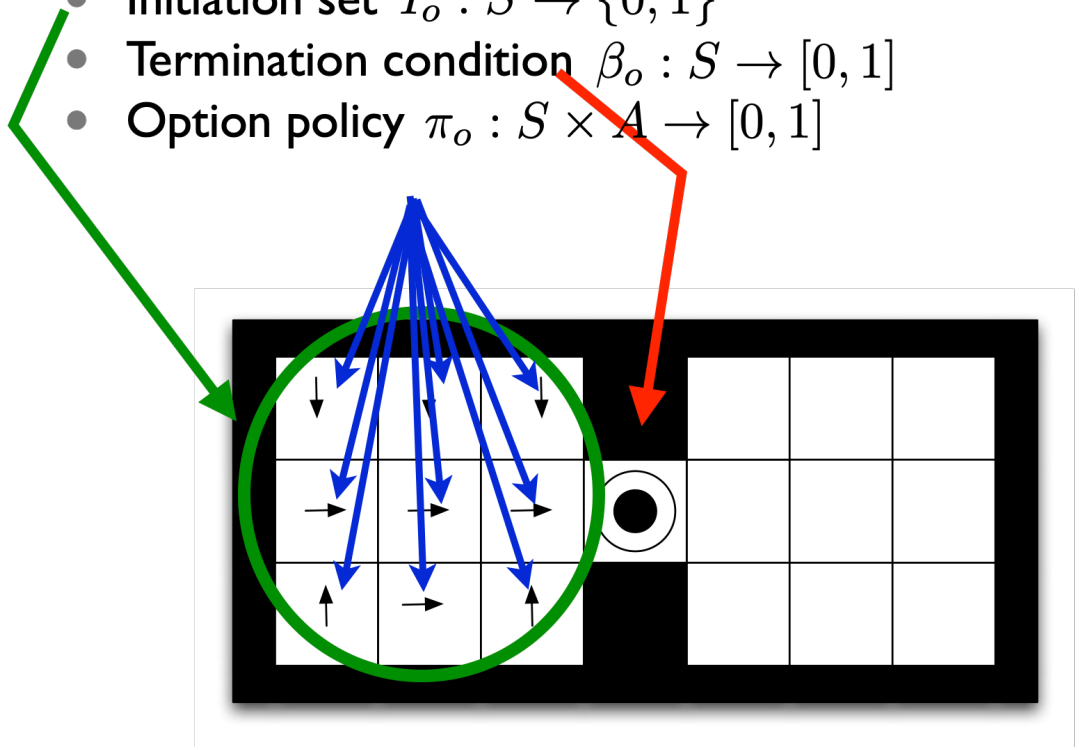


Options Framework

An option is one formal model of a skill.

An option *o* is a policy unit:

- Initiation set $I_o : S \to \{0, 1\}$
- Termination condition $\beta_o : S \to [0, 1]$
- Option policy $\pi_o : S \times A \to [0, 1]$



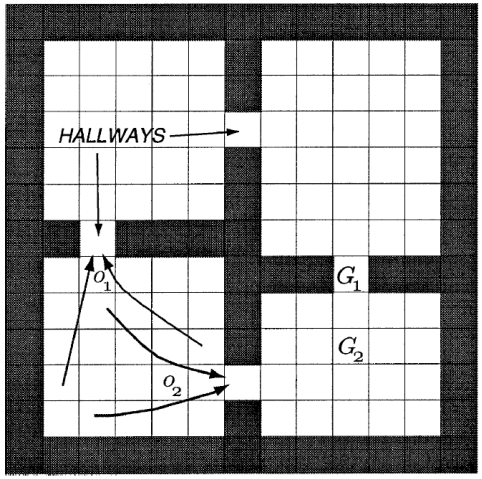
Actions as Options

A primitive action *a* can be represented by an option:

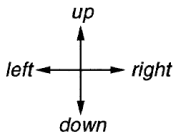
- $I_a(s) = 1, \forall s \in S$
- $\beta_a(s) = 1, \forall s \in S$
- $\pi_a(s, b) = \begin{cases} 1 & a = b \\ 0 & \text{otherwise} \end{cases}$

A primitive action can be executed anywhere, lasts exactly one time step, and always chooses action *a*.

Example



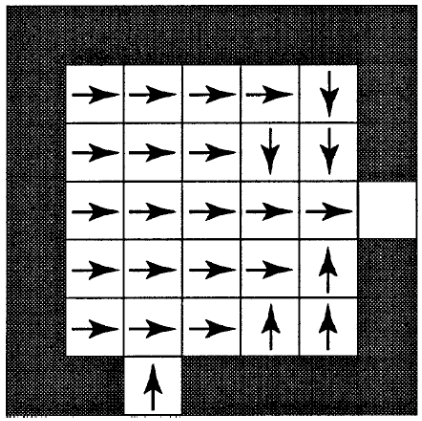
4 stochastic primitive actions



Fail 33% of the time

8 multi-step options

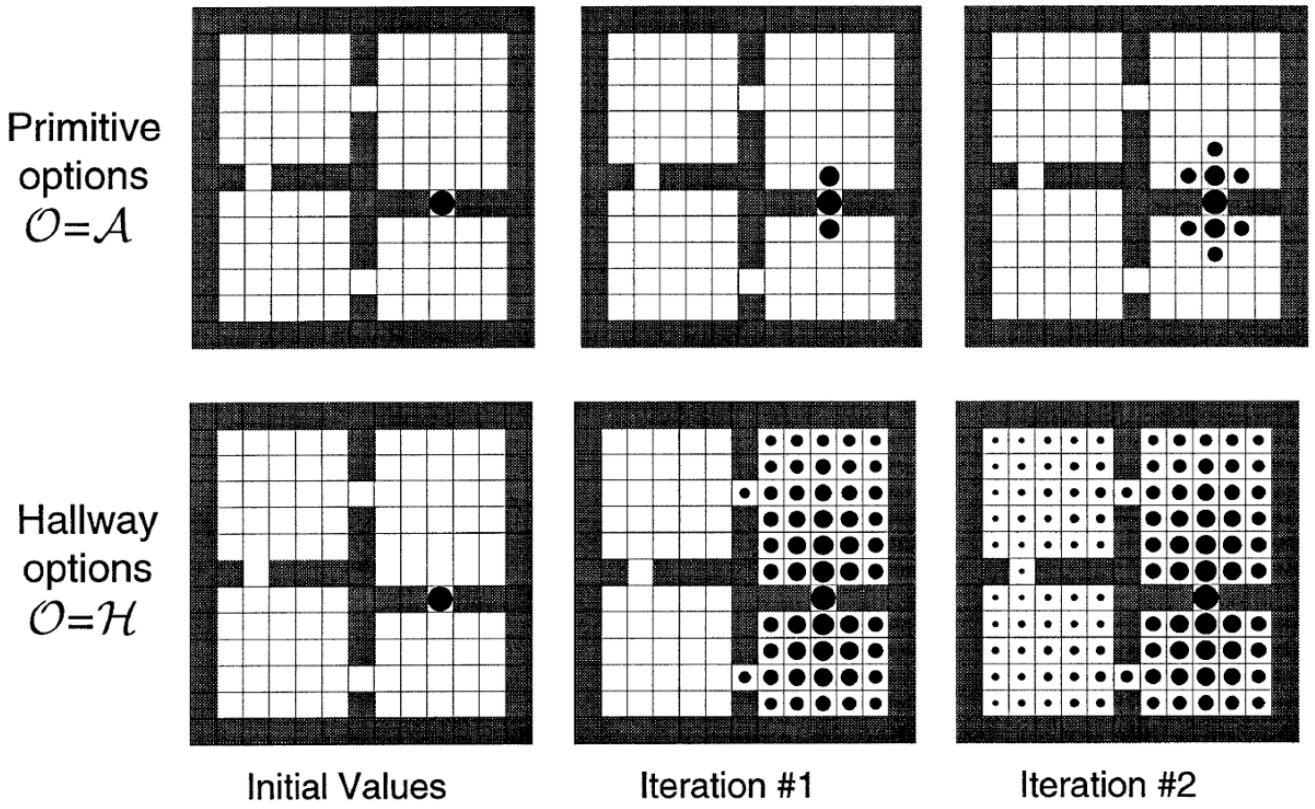
(to each room's 2 hallways)



Target Hallway

(Sutton, Precup and Singh, AIJ 1999)

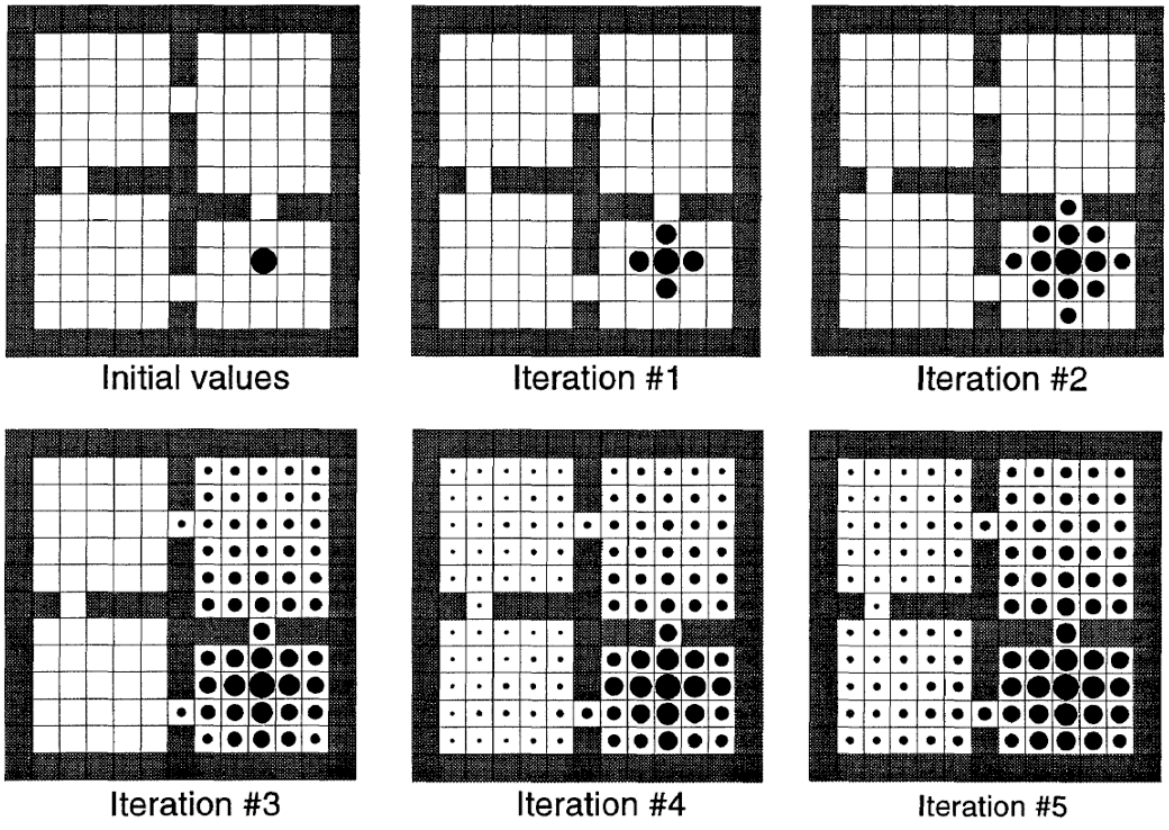
Example



(Sutton, Precup and Singh, AIJ 1999)

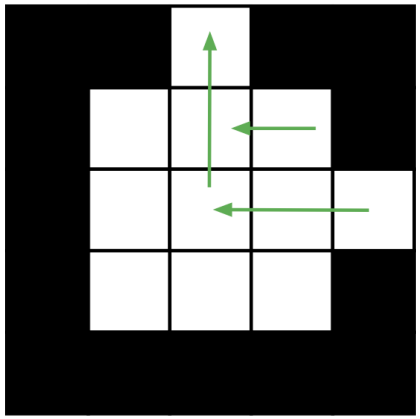
Example

Primitive and hallway options $\mathcal{O}=\mathcal{A}\cup\mathcal{H}$

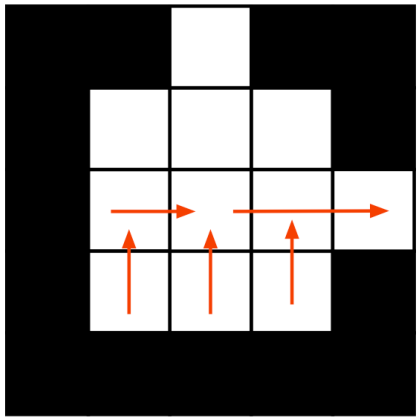


(Sutton, Precup and Singh, AIJ 1999)

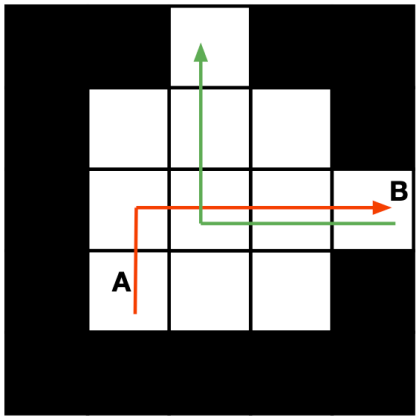
Example



Option A



Option B



Policy

“Modern” Hierarchical RL Methods

- Option-Critic (Bacon et al., 2017): Learns both intra-option policies and termination conditions.
  - Unstable, tends to collapse to single action options.

- Hierarchical DQN (h-DQN) (Kulkarni et al., 2016): Combines the options framework with Deep Q-Networks.
  - A high-level policy chooses subgoals.
  - A low-level policy tries to achieve those subgoals.

What's Missing?

- These methods still use the environment reward to learn options
- Long horizon problems are *hard!*

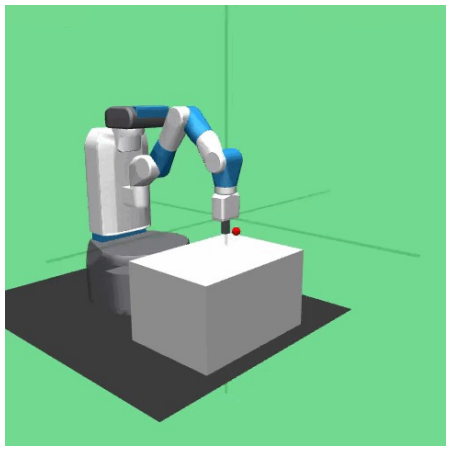
- Problem: These methods learn both high-level and low-level policies simultaneously. But high-level policy *depends* on low-level performance.

- Solution: Goal-conditioned low-level policies can be trained separately.

Goal Conditioned RL

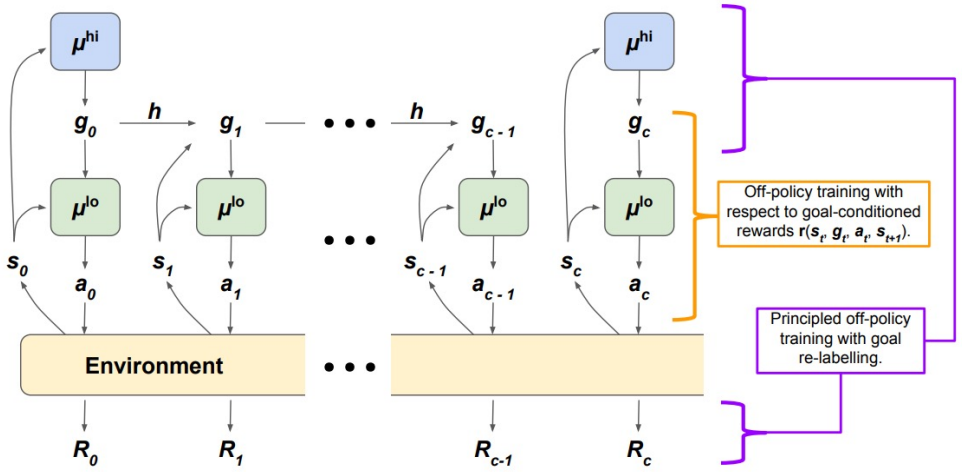
- The agent is trained to solve a number of tasks (reach goals) within the same environment.
- A goal is sampled at the beginning of episode.
- The reward function:

$$r(s; g) = \begin{cases} 0, & s = g \\ -1, & s \neq g \end{cases}$$



“Modern” Hierarchical RL Methods

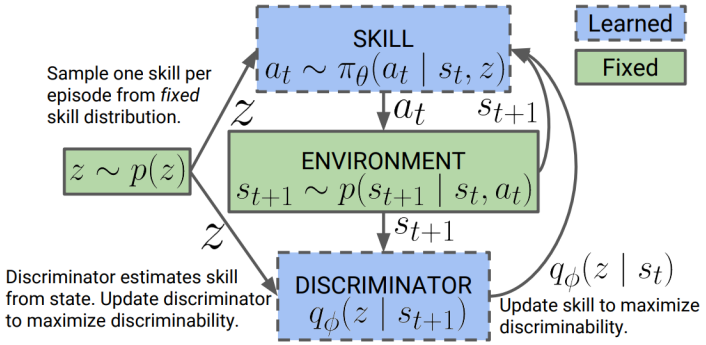
- **HIRO (HIerarchical Reinforcement learning with Off-policy correction) (Nachum et al., 2018):** Uses off-policy correction for effective training of the lower-level policy.
  - High-level policy selects goals in state space.
  - Lower-level policy is trained off-policy with hindsight-like correction.



1. Collect experience $s_t, g_t, a_t, R_t, \ldots$.
2. Train μ^{lo} with experience transitions $(s_t, g_t, a_t, r_t, s_{t+1}, g_{t+1})$ using g_t as additional state observation and reward given by goal-conditioned function $r_t = r(s_t, g_t, a_t, s_{t+1}) = -||s_t + g_t - s_{t+1}||_2$.
3. Train μ^{hi} on temporally-extended experience $(s_t, \tilde{g}_t, \sum R_{t:t+c-1}, s_{t+c})$, where $\tilde{g}_t$ is re-labelled high-level action to maximize probability of past low-level actions $a_{t:t+c-1}$.
4. Repeat.

“Modern” Hierarchical RL Methods

- DIAYN (Diversity is All You Need, Eysenbach et al., 2018): **Unsupervised skill discovery**: Low-level policies are learned without a task-specific high-level controller, by maximizing mutual information between latent variables (skills) and state transitions.
  - The high-level controller can later be trained to **select or condition** on these skills.
  - Useful for **offline training** of diverse, general-purpose low-level policies.



Algorithm 1: DIAYN

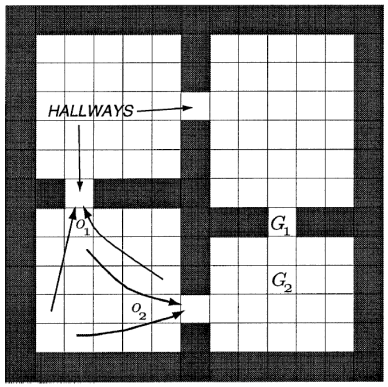
while *not converged* **do**
  Sample skill $z \sim p(z)$ and initial state $s_0 \sim p_0(s)$
  for $t \leftarrow 1$ **to** *steps_per_episode* **do**
    Sample action $a_t \sim \pi_\theta(a_t \mid s_t, z)$ from skill.
    Step environment: $s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$.
    Compute $q_\phi(z \mid s_{t+1})$ with discriminator.
    Set skill reward $r_t = \log q_\phi(z \mid s_{t+1}) - \log p(z)$
    Update policy (θ) to maximize r_t with SAC.
    Update discriminator (ϕ) with SGD.

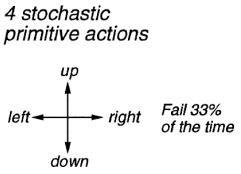
Task-Agnostic Options

- Train a k-step goal-conditioned policy independent of the high-level policy (HIRO)
- Use some other self-supervised objective to train low-level policies (DIAYN)

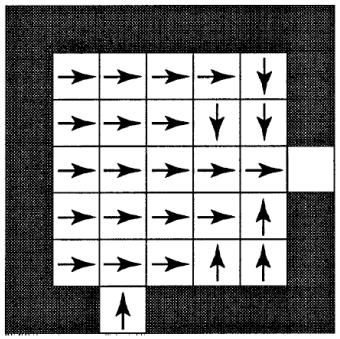
How can we do better?

- These methods do not promote skill *reuse*





8 multi-step options
(to each room's 2 hallways)

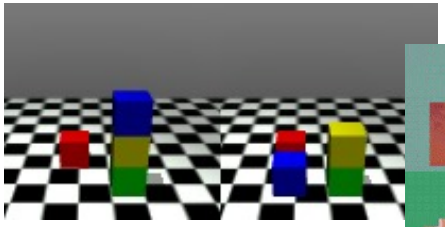


Navigating within each room or through hallways is the same!

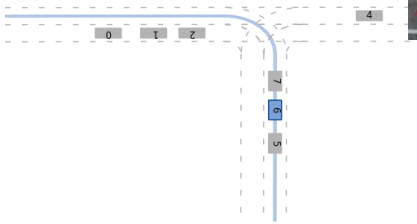
(Sutton, Precup and Singh, AIJ 1999)

What’s a setting where reuse is *expected?*

Open Problems: Compositionality



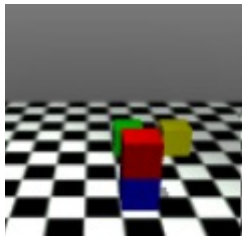
ProcGen



Bark Simulator

Useful Assumptions?

Factorization: A way to achieve compositional generalization?



Forms of Compositional Generalization

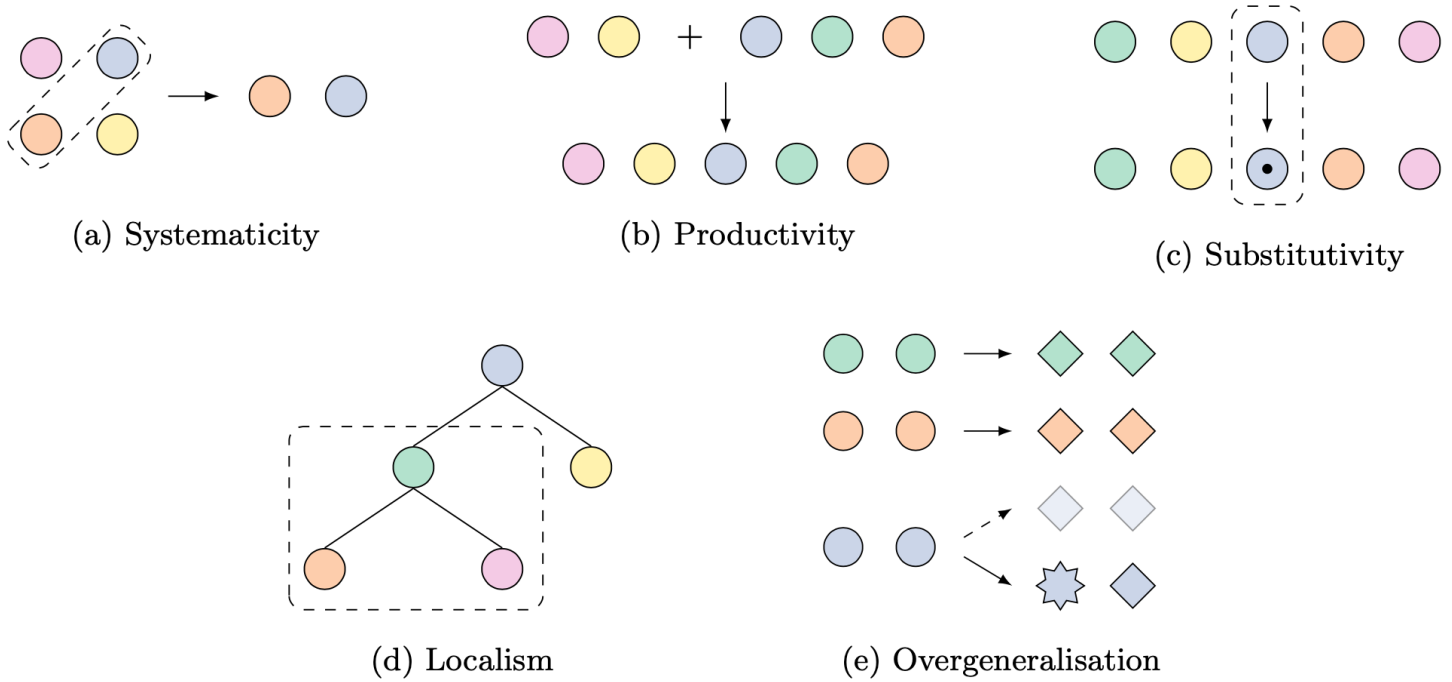
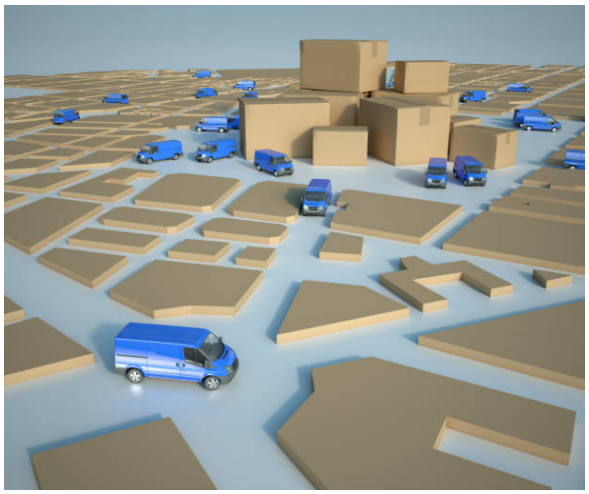


Figure from: Compositionality decomposed: how do neural networks generalize? Hupkes et al. 2

Relational MDP



A Relational MDP family can be described by

- C: Set of classes denoting different types of object
  e.g., {Box, Truck, City}
- F: Set of function schemata that take objects as input
  e.g., {Bin(Box, City), On(Box, Truck)}
- A: Set of action schemata that operate on objects
  e.g., {Unload(Box,Truck,City), Load(Box, City, Truck), Drive(Truck,City,City)}
- D: Set of domain objects, each associated with a single type from C
- T: Transition function
- R: Reward model

C, F, and A are sets of relational schemata.

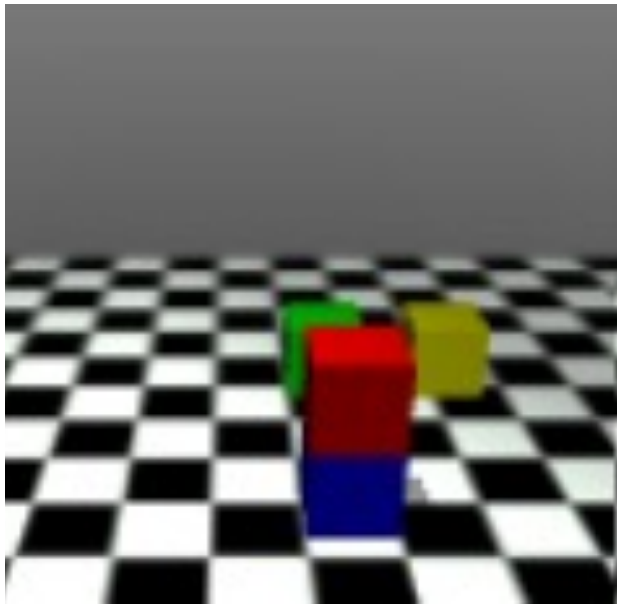
Boutilier, Reiter, & Price 2001

Composable Planning with Attributes

with Adam Lerer, Sainbayar Sukhbaatar, Rob Fergus, Arthur Szlam
ICML 2018

Facebook AI Research, NYU

Inspiration: Stacking Blocks



Learning to Reach Any Goal

- In many domains, there can be combinatorially many possible goals
  - Building Lego structure
  - Performing tasks in an RTS (real time strategy game)
- You may not know at training time which goals are important

Learning to Reach Any Goal

- **An RL Approach:** Provide a *representation of the goal space* and train a policy $\pi(s, \rho_g)$ to reach goal ρ_g using RL
  - e.g. universal value function approximators [Schaul et al, 2015]
  - Sparse rewards
  - Ignores goal space structure

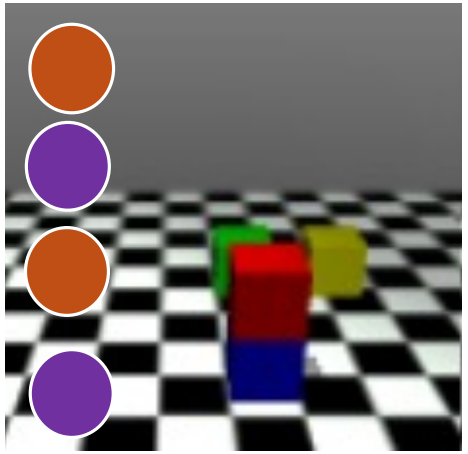
- **Our Approach:** Learn how to make local transitions in goal space, plan over the transition graph
  - Allows for zero-shot generalization from short to long tasks

Approach of This Work

- Provide the **attributes of the environment we care about**
  - This state -> attribute mapping is the *only* supervision
- The agent uses **unsupervised exploration** to learn how to modify attributes of the environment
- At test time, the agent uses attributes to specify a goal *and* as a space for **planning**

What Are Attributes?

- A set of high level features/properties of the state users care about
  - e.g. which block is on which
- Each property is a binary function of the state



Red stacked on blue

Green NOT stacked on blue

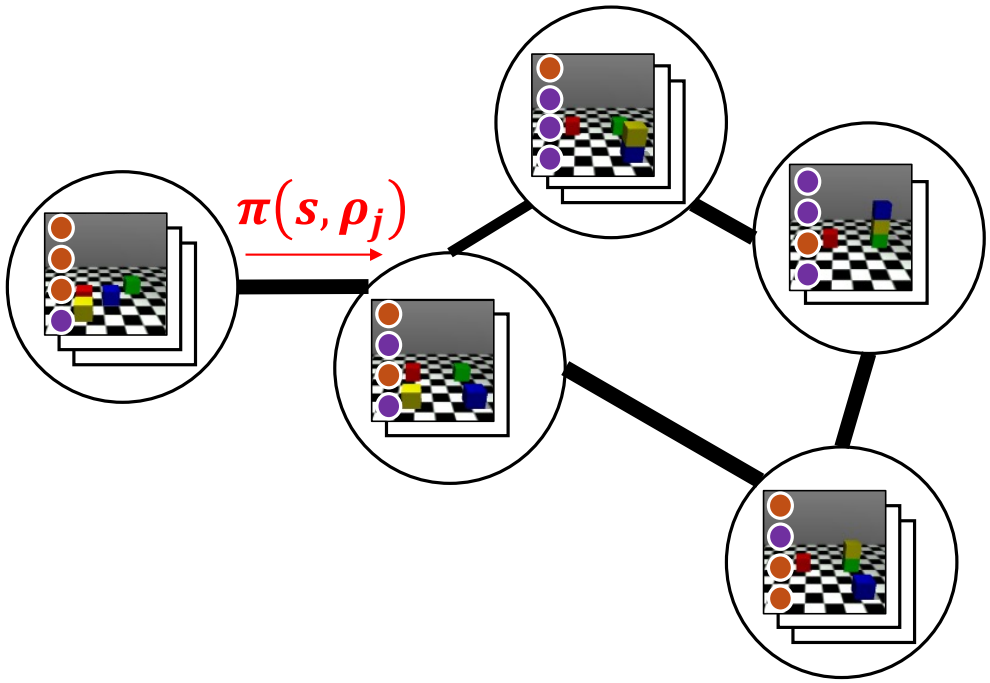
Yellow right of blue

Green NOT right of blue

...

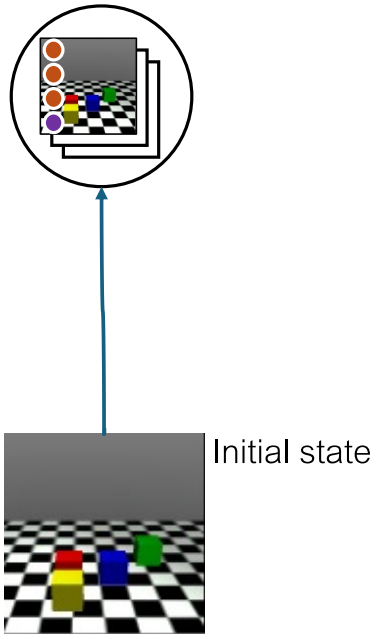
The Attribute Planner (AP) Model

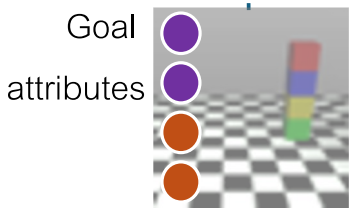
1. a neural-net based **attribute detector** f
2. a **graph G** of reachable attributes and edges between them
3. a neural net-based low level **policy** $\pi(s, \rho_j)$
   - Only knows how to reach *nearby* attribute sets



Solving Tasks With The AP Model

1. Given current state s and goal attributes ρ_g
2. Predict the current attributes ρ_0
3. Find shortest path $[\rho_0, \rho_1, \ldots, \rho_g]$ on the graph G
4. Execute first step in the path with the low-level policy.
5. Goto 1

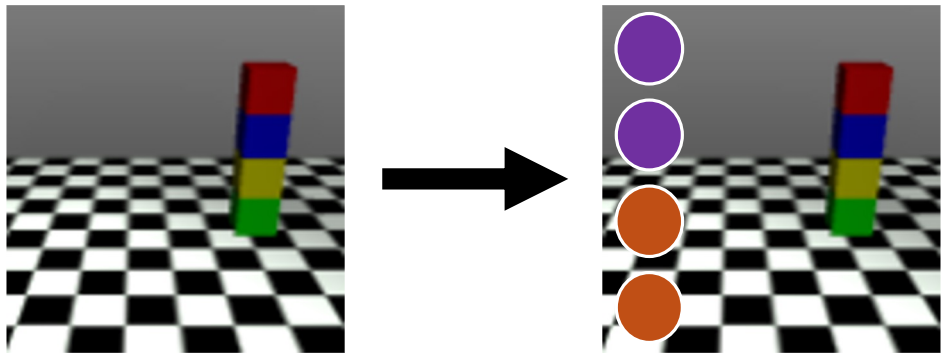




Training the AP Model

Attribute Detector f

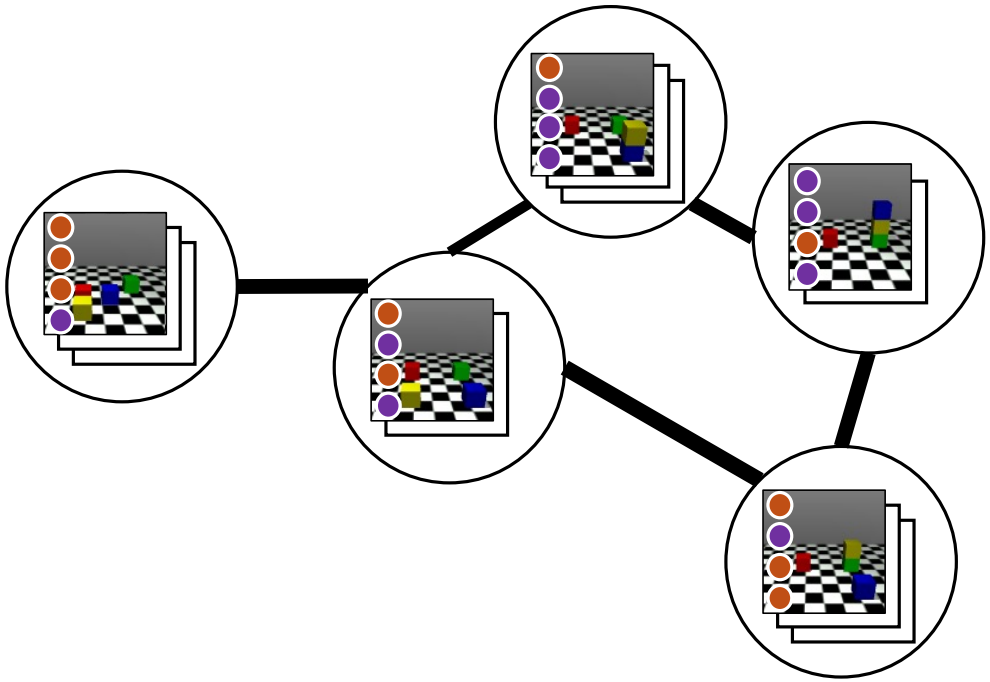
- Train the attribute detector on a small number of labeled examples.



Training the AP Model

Attribute Graph G

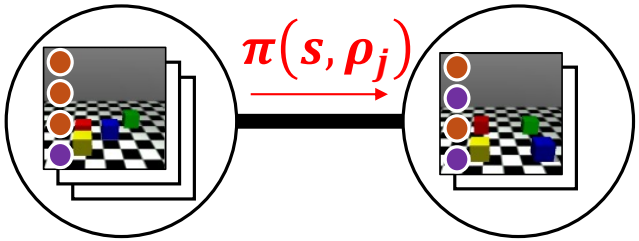
- Take actions with an exploratory policy (e.g. random) until you reach a new set of attributes ρ_j. Add (ρ_i, ρ_j) to the graph.
- Repeat from ρ_j or reset. Only store attributes that actually occur.



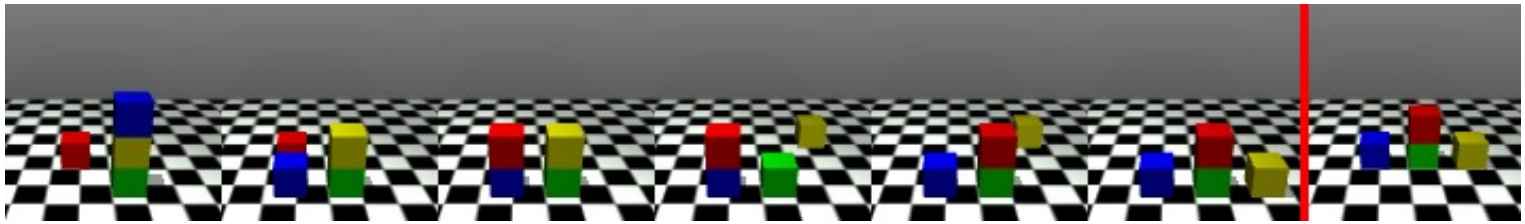
Training the AP Model

Low-Level Policy π

Inverse Training: Train π as an inverse model to predict the actions that led to a transition during the exploration phase.

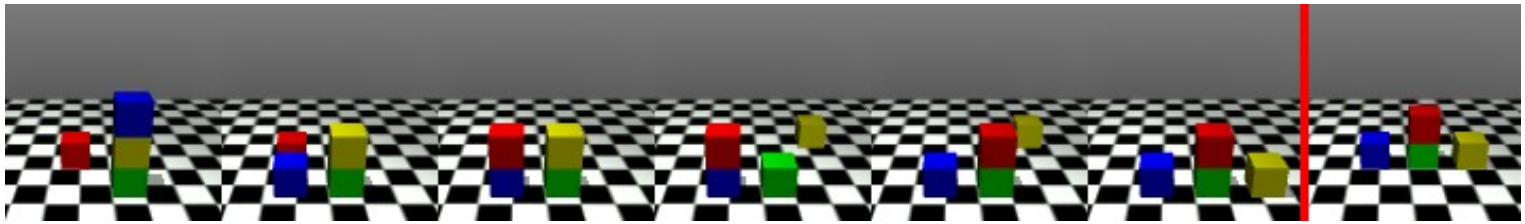


Block Stacking Experiments



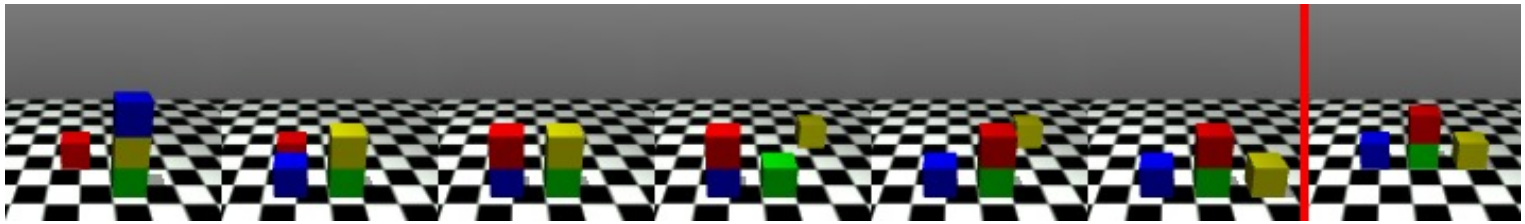
Model	Training Data	multi-step
A3C	one-step	8.1%
A3C	multi-step	0%
A3C	curriculum	17%
AP	one-step	**66.7%**

Block Stacking Experiments



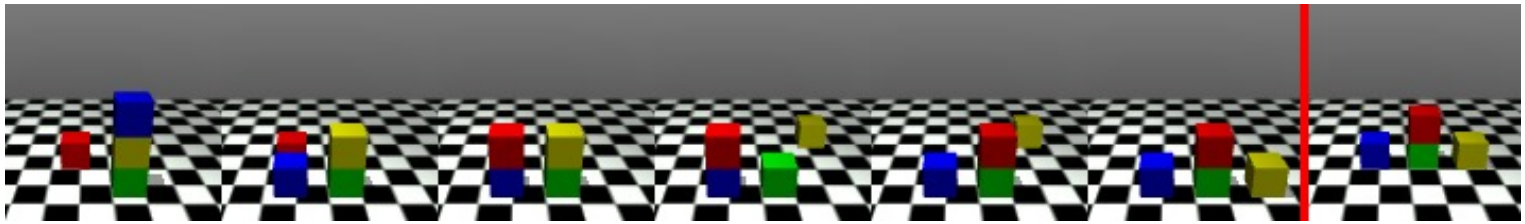
Model	Training Data	multi-step
A3C	one-step	8.1%
A3C	multi-step	0%
A3C	curriculum	17%
Inverse	one-step	9.1%
Inverse	multi-step	13.7%
Option-Critic *	one-step	0.6%
Option-Critic *	multi-step	0.2%
Option-Critic *	curriculum	0.4%
AP	one-step	**66.7%**

Block Stacking Experiments



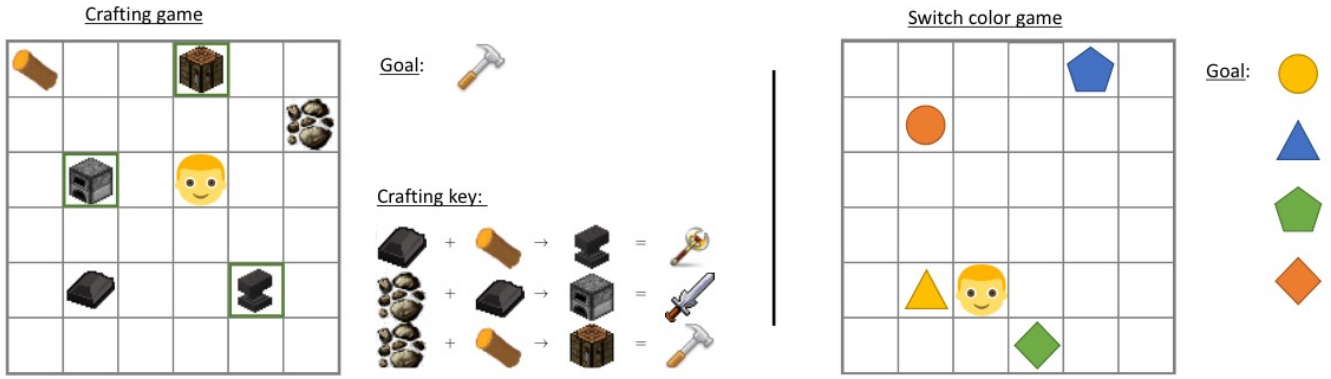
Model	Training Data	multi-step
A3C	one-step	8.1%
A3C	multi-step	0%
A3C	curriculum	17%
Inverse	one-step	9.1%
Inverse	multi-step	13.7%
Option-Critic *	one-step	0.6%
Option-Critic *	multi-step	0.2%
Option-Critic *	curriculum	0.4%
AP (no c_π)	one-step	29.7%
AP	one-step	**66.7%**

Block Stacking Experiments



Model	Training Data	multi-step	4-stack	underspecified
A3C	one-step	8.1%	1.9%	6.6%
A3C	multi-step	0%	0%	0%
A3C	curriculum	17%	2.9%	0.2%
Inverse	one-step	9.1%	0.5%	18.8%
Inverse	multi-step	13.7%	4.6%	9.6%
Option-Critic *	one-step	0.6%	1.0%	1.2%
Option-Critic *	multi-step	0.2%	0.5%	1.7%
Option-Critic *	curriculum	0.4%	0.9%	1.0%
AP (no c_π)	one-step	29.7%	62.2%	28.1%
AP	one-step	**66.7%**	**98.5%**	**63.5%**

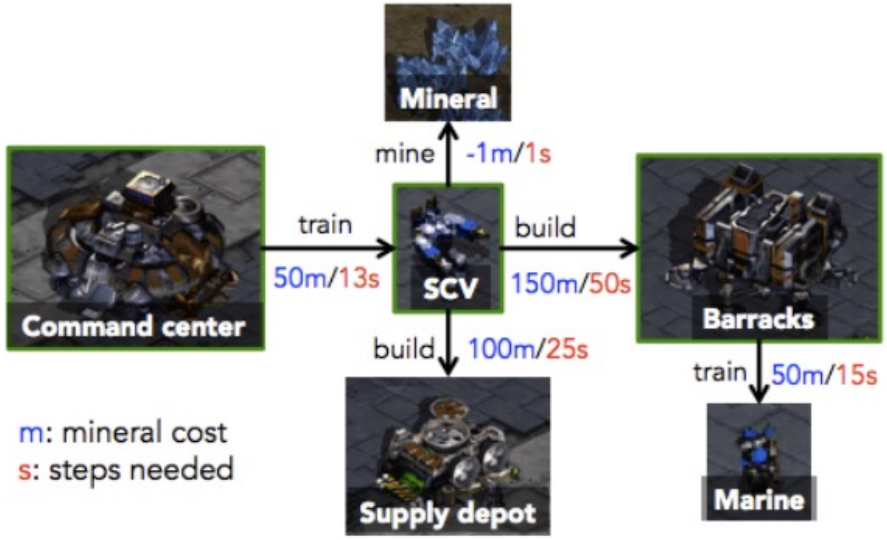
Grid World Experiments



Method	Training Data	Switches	Crafting
Reinforce	one-step	15.4%	49.0%
Reinforce	multi-step	0.0%	28.6%
Reinforce	multi-step + curriculum	33.3%	83.9%
AP	one-step	**83.1%**	**96.2%**

StarCraft Experiments

Method	Training Data	Small	Large
Reinforce	test tasks	12.6%	2.3%
Reinforce	curriculum	18.9%	1.9%
AP	-	**31.7%**	**35.2%**



- Unlike RL, the AP does not see test-time tasks during training
- AP performs count-based exploration in attribute space to more efficiently learn the attribute graph.
- AP scales better as the graph becomes larger

Recap

- The right abstraction leads to compositional generalization
- We define a relational abstraction that naturally breaks the block stacking problem into a hierarchy
- What if we don’t have these relational attributes predefined, or can’t predefine them?

Combinatorial Generalization in Object Rearrangement by Hierarchical Latent Grouping

With Michael Chang, Alyssa L. Dayan, Franziska Meier, Thomas L. Griffiths, Sergey Levine

ICLR 2023

How do we leverage factorization?

- Problems to solve:
  - Extracting factors from the environment
  - Correspondence problem: How do we assign identities to factors that persist over trajectories?
  - Combinatorial problem: How do we leverage factored structure to generalize to new states and different numbers and types of factors?

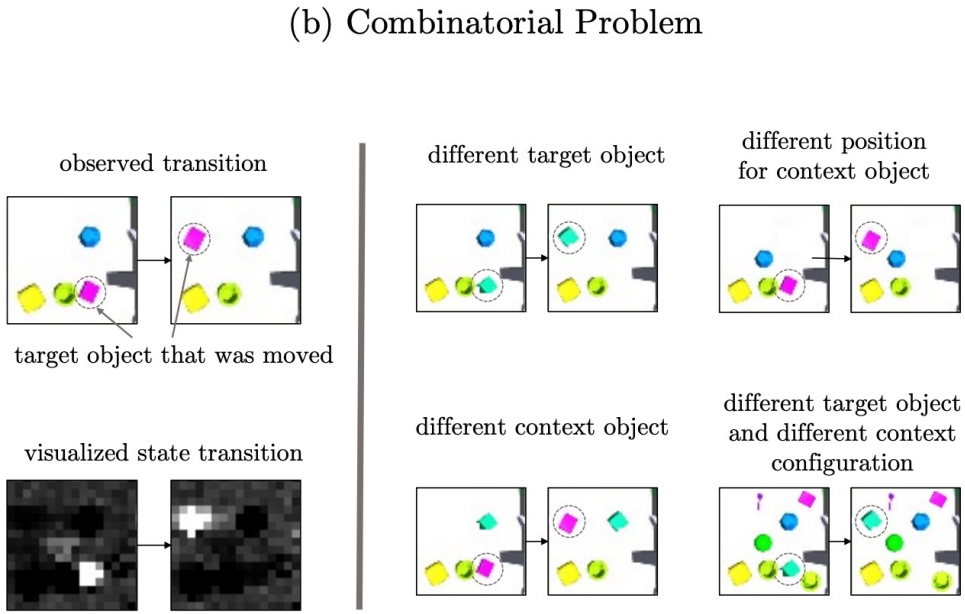
Hierarchical Abstraction for Combinatorial Generalization in Object Rearrangement. M. Chang, A. Dayan, F. Meier, T. Griffiths, S. Levine, AZ. ICLR 2023.

A contribution for solving the factorization problem

- Split a factor entity into:
  - action-invariant features (its **type**)
  - Action-dependent features (its **state**)

- Why?
- Makes it possible during planning and control to reuse action representations for different objects in different contexts

How do we leverage factorization?



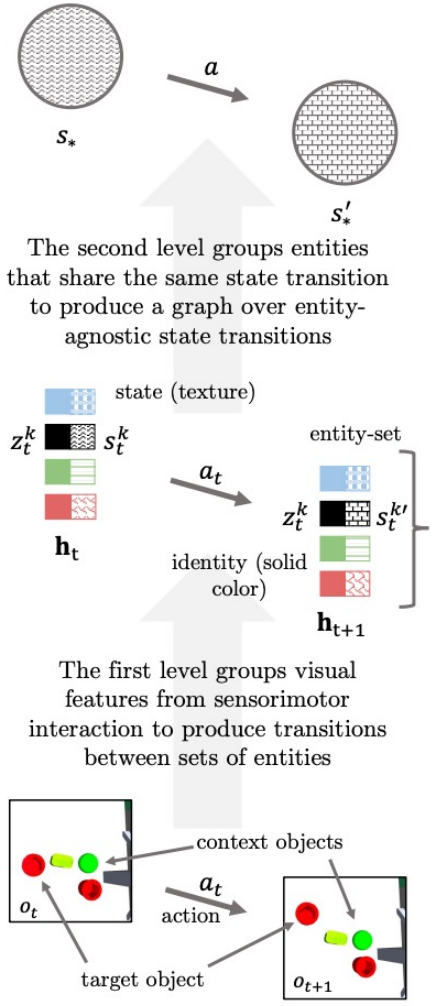
The same state transition can manifest for different objects and in different contexts

Hierarchical Abstraction for Combinatorial Generalization in Object Rearrangement. M. Chang, A. Dayan, F. Meier, T. Griffiths, S. Levine, AZ. ICLR 2023.

How do we implement factorization?

Neural Constraint Satisfaction

- Two-level hierarchy:
  - abstract the experience buffer into a graph over state transitions of individual entities, separated from other contextual entities
  - solve new rearrangement problems: NCS infers what state transitions can be taken given the current and goal image observations, re-composes sequences of state transitions from the graph, and translates these transitions into actions.

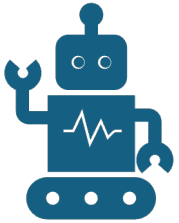


Implementing Neural Constraint Satisfaction (NCS)

Modeling

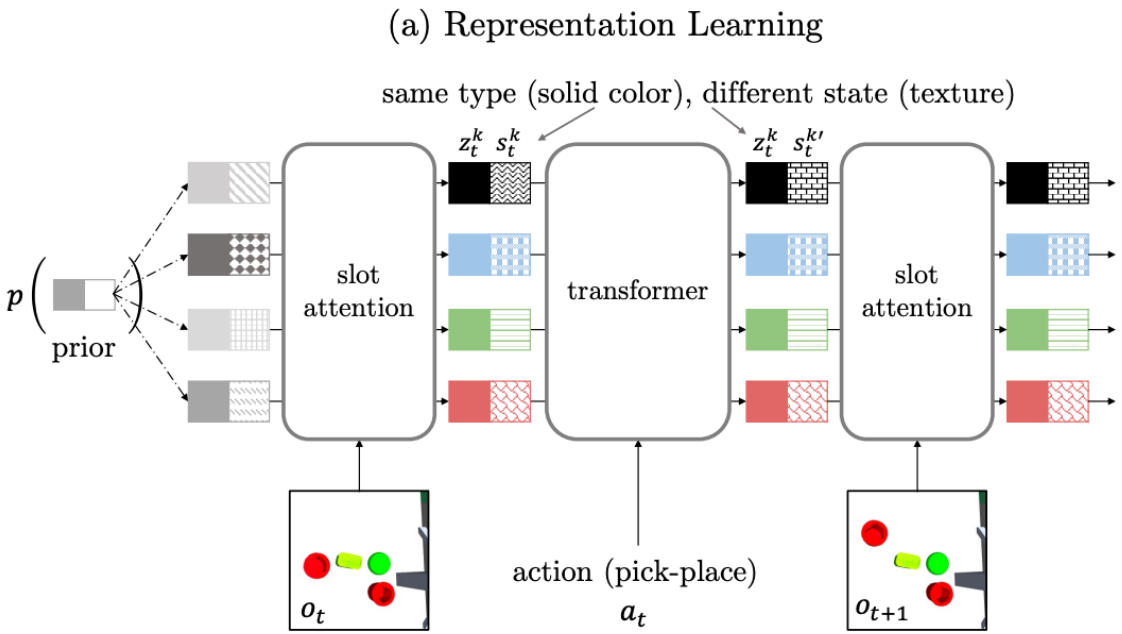
Representation learning

Graph construction



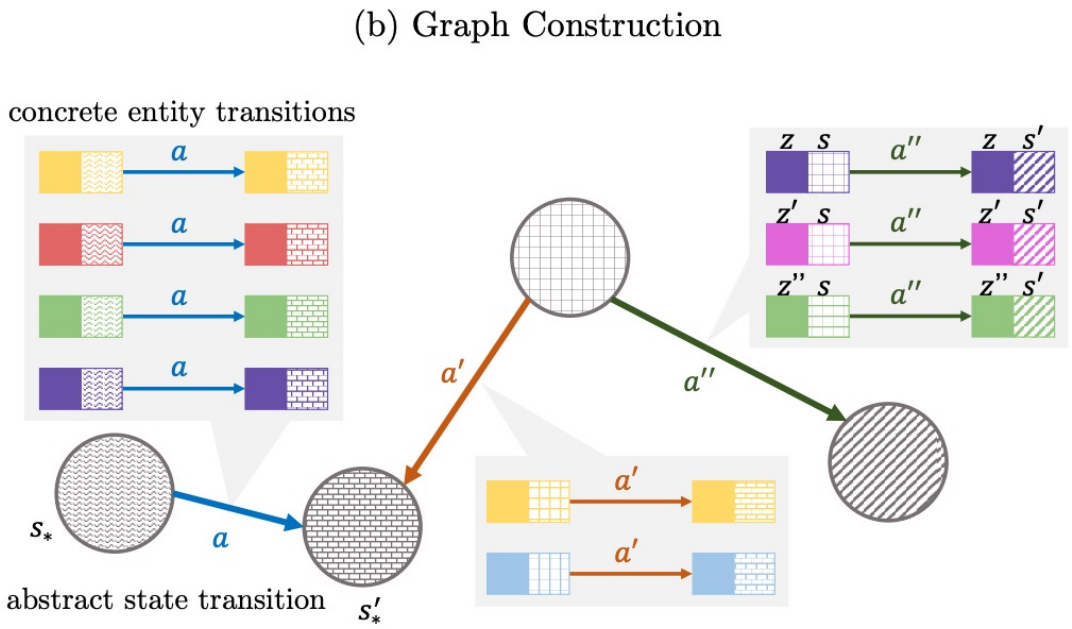
Control

Modeling: Representation Learning



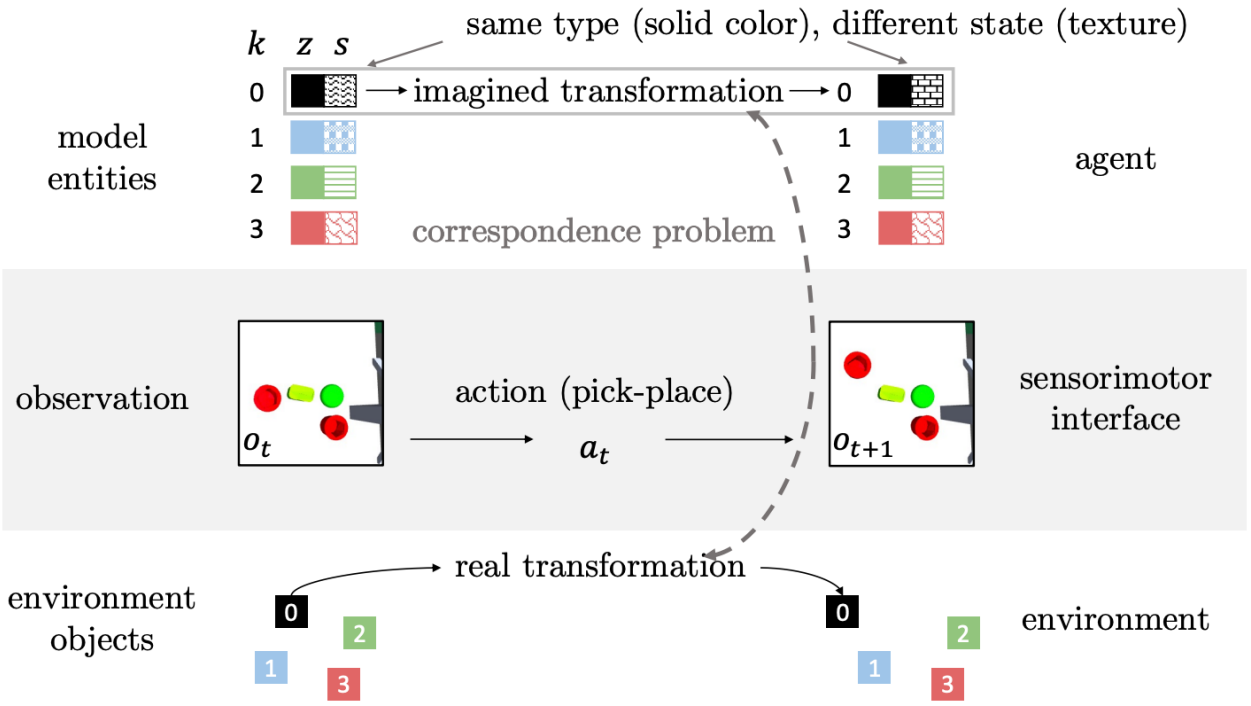
Hierarchical Abstraction for Combinatorial Generalization in Object Rearrangement. M. Chang, A. Dayan, F. Meier, T. Griffiths, S. Levine, AZ. ICLR 2023.

Modeling: Graph Construction

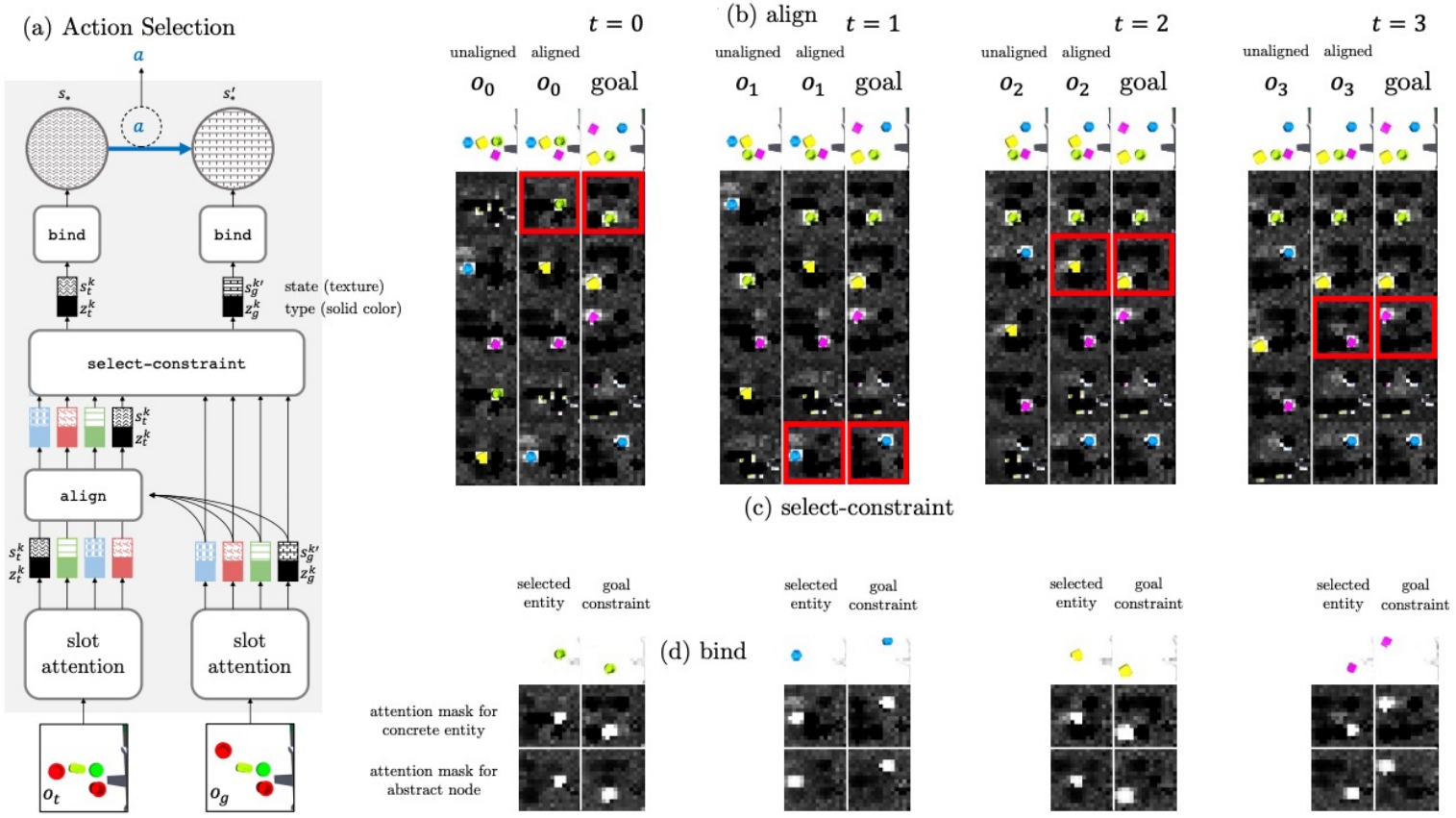


Hierarchical Abstraction for Combinatorial Generalization in Object Rearrangement. M. Chang, A. Dayan, F. Meier, T. Griffiths, S. Levine, AZ. ICLR 2023.

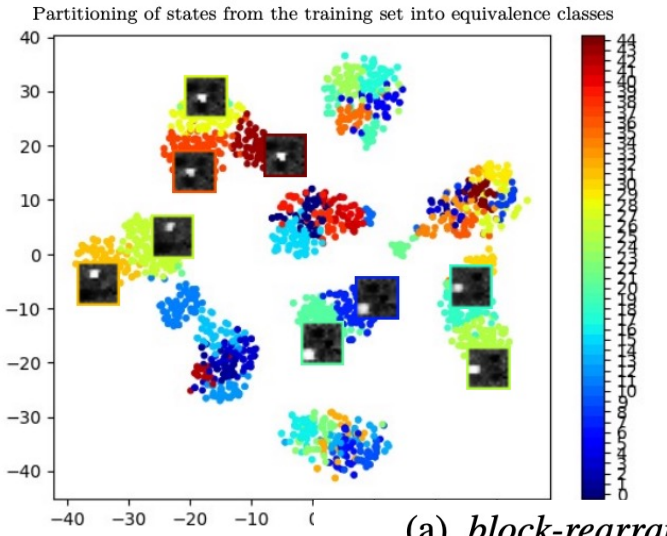
Correspondence Problem



Planning and Control



Hierarchical Abstraction for Combinatorial Generalization in Object Rearrangement. M. Chang, A. Dayan, F. Meier, T. Griffiths, S. Levine, AZ. ICLR 2023.



Results

NF: non-factorized graph

(a) *block-rearrange*, complete specification.

Method	4	5	6	7
NCS (ours)	**0.94** ± 0.01	**0.93** ± 0.00	**0.93** ± 0.00	**0.89** ± 0.00
Rand	0.06 ± 0.02	0.07 ± 0.03	0.07 ± 0.03	0.08 ± 0.03
MPC	0.16 ± 0.06	0.12 ± 0.04	0.11 ± 0.04	0.10 ± 0.03
NF	0.07 ± 0.03	0.06 ± 0.02	0.07 ± 0.02	0.08 ± 0.03
IQL	0.07 ± 0.01	0.03 ± 0.00	0.02 ± 0.00	0.02 ± 0.00
BC	0.03 ± 0.00	0.02 ± 0.00	0.01 ± 0.00	0.01 ± 0.00

(b) *block-rearrange*, complete specification.

Method	4	5	6	7
NCS (ours)	**0.89** ± 0.01	**0.86** ± 0.01	**0.78** ± 0.01	**0.70** ± 0.01
Rand	0.06 ± 0.02	0.08 ± 0.03	0.08 ± 0.03	0.08 ± 0.03
MPC	0.13 ± 0.05	0.11 ± 0.04	0.10 ± 0.04	0.08 ± 0.03
NF	0.06 ± 0.03	0.07 ± 0.03	0.08 ± 0.03	0.07 ± 0.03
IQL	0.01 ± 0.01	0.07 ± 0.01	0.05 ± 0.01	0.05 ± 0.00
BC	0.05 ± 0.01	0.04 ± 0.00	0.03 ± 0.00	0.03 ± 0.00

(c) *robogym-rearrange*, complete specification.

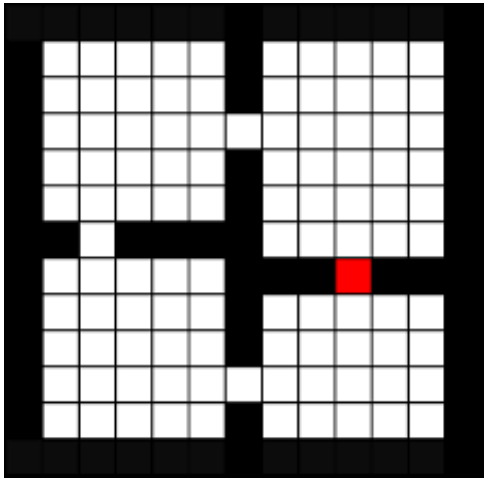
Method	4	5	6	7
NCS (ours)	**0.64** ± 0.01	**0.47** ± 0.01	**0.49** ± 0.01	**0.41** ± 0.01
Rand	0.01 ± 0.00	0.01 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
MPC	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
NF	0.01 ± 0.00	0.01 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
IQL	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
BC	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00

(d) *robogym-rearrange*, partial specification.

Method	4	5	6	7
NCS (ours)	**0.47** ± 0.01	**0.33** ± 0.01	**0.27** ± 0.01	**0.22** ± 0.01
Rand	0.005 ± 0.001	0.001 ± 0.00	0.002 ± 0.001	0.001 ± 0.00
MPC	0.00 ± 0.00	0.001 ± 0.001	0.00 ± 0.00	0.00 ± 0.00
NF	0.005 ± 0.001	0.001 ± 0.00	0.002 ± 0.001	0.001 ± 0.00
IQL	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
BC	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00

Going beyond the factored setting: What do we want in an abstraction?

1. Capture controllability
2. Only capture locally controllable components



Noisy TV problem

Capturing Controllability

- Multistep-inverse prediction with a latent forward model is enough to learn a lossy representation that filters out uncontrollable information (noisy TV)

Multistep Inverse Is Not All You Need
Alexander Levine, Peter Stone, **AZ**

Learning a Fast Mixing Exogenous Block MDP using a Single Trajectory
Alexander Levine, Peter Stone, **AZ**

Summary

1. Why Hierarchy?
2. Existing hierarchical methods and
   their problems
3. Compositionality in Relational
   MDPs: A class of problems where hierarchy is well defined
4. Leverage factorized structure in relational MDPs with a hand designed relational abstraction
5. How can we learn a factored abstraction and leverage it?
6. What can we do if we can’t assume factorized structure?