Hierarchical Abstraction for Combinatorial Generalization in Object Rearrangement

Anonymous Author(s)
Affiliation
Address
email

Abstract

Object rearrangement is a challenge for embodied agents because solving these tasks requires generalizing across a combinatorially large set of underlying entities that take the value of object states. Worse, these entities are often unknown and must be inferred from sensory percepts. We present a hierarchical abstraction approach to uncover these underlying entities and achieve combinatorial generalization from unstructured inputs. By constructing a factorized transition graph over clusters of object representations inferred from pixels, we show how to learn a correspondence between intervening on states of entities in the agent's model and acting on objects in the environment. We use this correspondence to develop a method for control that generalizes to different numbers and configurations of objects, which outperforms current offline deep RL methods when evaluated on a set of simulated rearrangement and stacking tasks.

1 Introduction

A core property of intelligence is the ability to re-purpose previously acquired knowledge for solving new problems, but how to identify symmetries for factorizing modeling and control into independent components has been a challenge for AI systems, which either assume human-defined abstractions to begin with [11, 13] or learn monolithic representations with no clear mechanism for reuse [19, 30].

The problem of *object rearrangement* offers an intuitive setting for studying this problem of knowledge reuse. Because the space of object configurations is combinatorially large, solving novel rearrangement problems requires the agent to recognize that the same action for moving an object from one location to another can be reused for different objects in different contexts. We can formulate this problem as an offline goal-conditioned reinforcement learning (RL) problem, where the agent is trained on a dataset of sensorimotor interactions and is evaluated on rearranging objects to satisfy constraints depicted in a goal image. The research questions are, therefore, how can we enable the agent to represent abstractions of objects in a way that is amenable for such reuse in control, and how can we enable it to discover these abstractions on its own directly from the sensorimotor interface?

Our main contribution is a method called Hierarchical Abstraction (HA) that offers an answer to the above questions. The key idea is to represent objects as independent and symmetric latents, which we call entities, and factorize each entity h^k into two parts, its identity z^k and its state s^k. The identity is the part that is not affected by action and the state is the part that is. The modeling component of our approach abstracts the dataset of sensorimotor interactions into a graph over state transitions of individual entities. By construction, a state transition is agnostic to the identity of the entity or the context entities present in the sensorimotor interaction from which it was learned, enabling it to be reused across multiple entities and contexts. The control component of our approach then re-composes sequences of state transitions to solve new rearrangement problems by inferring what state transition can be taken given only the current and goal image observations.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

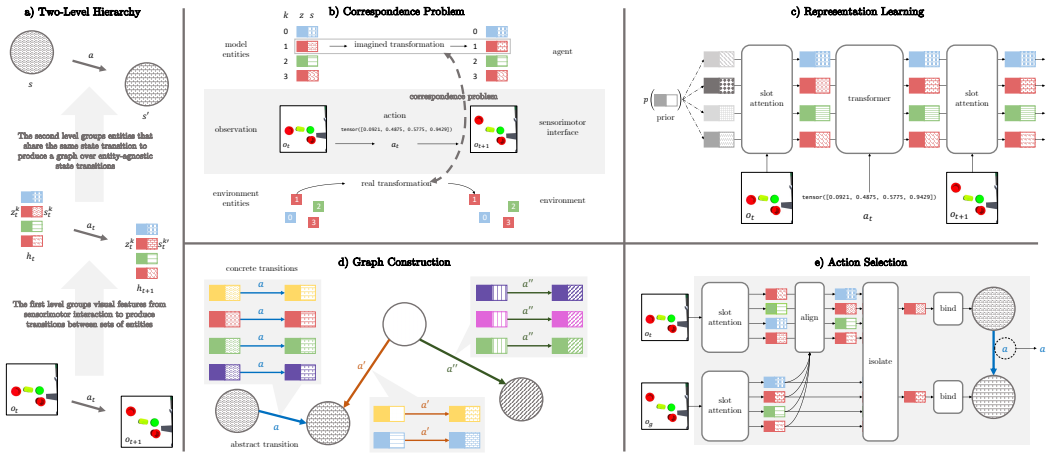

Figure 1: **Overview** (a) HA abstracts video interactions into a graph over state transitions of individual entities with a two-level hierarchy. (b) To do so requires learning representations of entities in a way that exhibits a correspondence between how model entities and how real objects transform under action. (c) The first level of the hierarchy abstracts visual features into transitions over sets of entities. (d) The second level abstracts these transitions into a graph of transitions over individual states. (e) HA decomposes the rearrangement task into a subtask per entity and chooses actions by returning the action tagged to the edge between nodes in the graph.

In contrast to other works in object-centric learning [14–16, 21, 24, 31, 34, 40] which evaluate the quality of inferred entities via segmentation metrics, we evaluate our inferred entities by *how well they can reused for solving tasks*, which more directly assesses what we want an agent's object representations to do. Kulkarni et al. [23], Veerapaneni et al. [36] also evaluate on control tasks. Whereas Kulkarni et al. [23] considers how object representations improve exploration, we consider the offline setting which requires zero-shot generalization. The shooting-based planning method in Veerapaneni et al. [36] suffers from compounding errors as other learned single-step models do [20], but our non-parametric approach of composing previously seen transitions enables us to plan for longer horizons. Indeed, our results show that HA outperforms both state-of-the-art offline RL methods and slot-based planning methods across three simulated object rearrangement problems.

2 Hierarchical Abstraction

We present our method for solving object rearrangement problems from pixels. Given a transition $o, a \rightarrow o'$ from the training set in which a single object k has been moved, the modeling problem is to represent the state transition $s^k \rightarrow s^{k\prime}$ as decoupled from the identity z^k of the affected entity and as decoupled from the other context entities $h^{\neq k}$ in the scene. The control problem is to compose previously seen state transitions to satisfy goal constraints on new object configurations. Though we do not assume the number of action primitives is finite, we assume the number of observed locations can be clustered into a set of finite groups.

2.1 Modeling

Our modeling approach abstracts the training dataset of action-conditioned videos into a factorized graph over state transitions that can be reused across different rearrangement problems. Constructing such a graph decomposes into two representation learning problems: the first concerns how to represent objects in a way that exposes their commonalities and the second concerns how to represent actions in a way that enables them to be reused across these commonalities. Our key insight is that these two problems are both different instances of the same fundamental problem of partitioning a space into discrete groups, but at different levels of abstraction: learning how to cluster visual features into entities (Fig. 1b) and learning how to cluster transitions over entity-sets into transitions over individual states (Fig. 1c). Our model thus implements a two-level abstraction hierarchy (Fig. 1a).

Level 1: abstracting visual features into sets of entities The goal of the first level is to map a transition over two video frames $o_1, a_1 \rightarrow o_2$ into a transition over entity-sets $(h_1^1, ..., h_1^K) \rightarrow (h_2^1, ..., h_2^K)$ (Fig. 1c). The criteria our entity-set transitions should satisfy are (1) entities represent

different objects in the scene, i.e. they are *independent* and *symmetric* and (2) only the state s^k of entity k changes, not its identity z^k, i.e. the entity is *factorized*. Because each video transition depicts only a single object being moved, we also want (3) the transition to be *sparse*, i.e. only one entity h^k changes in the transition and the rest $h^{\neq k}$ remain unchanged.

Our approach manually enforces independence, symmetry, and factorization but lets sparsity emerge. Specifically, we formulate the inference of entity-sets as the application of an equivariant action-conditioned sequential Bayesian filter with a mixture model as the latent state, where entity representations are the parameters of the mixture components. Given the connection between the slots produced by slot attention (SA) [25] and mixture components Chang et al. [6], we concretely implement this filter by temporally evolving the slots produced by the slot attention module of the state-of-the-art SLATE architecture [32] with a transformer decoder (TD) [28, 35]. With $\mathbf{h} = (h^1, ..., h^K)$,

$$\mathbf{h}_{t+1} = SA(\hat{\mathbf{h}}_t, o_{t+1}) \qquad \hat{\mathbf{s}}_{t+1} = TD\,(\text{queries} = \mathbf{s}_t, \text{keys/values} = [\mathbf{s}_t, a_t])\,,$$

where $\hat{\mathbf{h}}_{t+1} = [\mathbf{z}_t, \hat{\mathbf{s}}_{t+1}]$ and $\hat{\mathbf{h}}_0$ are independent samples from a Gaussian. This approach enforces independence because $\hat{\mathbf{h}}_0$ are independent, symmetry because the transformer is equivariant, and factorization because $\hat{\mathbf{z}}_{t+1}$ is a copied from $\mathbf{z}_t$. Empirically, we find that this architecture models sparse changes in the slots (Fig. 3). We call this implementation **dynamic SLATE** (dSLATE), but our contribution of how structurally we enforce the above three criteria remains invariant to *how* the sequential Bayesian filter is implemented, which we expect will advance beyond SA and TD.

Level 2: abstracting transitions over sets of entities into individual state transitions The goal of the second level is to construct a state transition graph from a buffer of entity-set transitions $\{(\mathbf{h}_1, ...\mathbf{h}_T)\}_{n=1}^N$ produced from the first level (Alg. 1, Fig. 1d). Specifically, we seek each edge of this graph to represent a state transition that has been shared across multiple entities in different contexts in the training dataset, such that we can reuse such state transitions when choosing actions for solving new rearrangement problems. Given our assumption that observed locations can be grouped into a finite set of clusters, this goal translates into a problem of clustering state transitions, and the problem of clustering state transitions reduces to clustering the states before and after the transition. We treat each cluster centroid as a node in the state transition graph, and an edge between nodes is tagged with the single action that transforms one node's state to another's.

Algorithm 1 Building the Graph

input `model`, `buffer`
for $\{(o_t, a_t, o_{t+1})\}_n$ in `buffer` **do**
  # infer entities from transition
  $\{(\mathbf{h}_t, a_t, \mathbf{h}_{t+1})\}_n \leftarrow$ `model` $(\{o_t, a_t, o_{t+1}\}_n)$.
  # identify which entity changed in transition
  $\{(h_t^k, a_t, h_{t+1}^k)\}_n \leftarrow$ `isolate` $(\{(\mathbf{h}_t, a_t, \mathbf{h}_{t+1})\}_n)$
end for
# partition transitions by clustering entities
$\{s_*\}_{m=1}^M \leftarrow$ `cluster` $\left(\{(s_t^k, a_t, s_{t+1}^k)\}_{n=1}^N\right)$
transitions between clusters are edges
initialize `graph` with nodes $s_*^{[m]}$, for $m \in [1:M]$
for each $\{(h_t^k, a_t, h_{t+1}^k)\}_n$ **do**
  # infer cluster assignments
  $[i], [j] \leftarrow$ `bind` $\left(h_t^k\right)$, `bind` $\left(h_{t+1}^k\right)$
  # tag edge with action a_t
  `graph.edges`$[i, j] \leftarrow$ `create-edge` $\left(s_*^{[i]} \overset{a_t}{\rightarrow} s_*^{[j]}\right)$
end for
return `graph`

To create the nodes, the first step is to identify the entity h^k that dSLATE predicted was affected by a_t in each transition $(\mathbf{h}_t, a_t, \mathbf{h}_{t+1})$. Our `isolate` procedure achieve this by solving $k = \arg\max_{k' \in \{1,...,K\}} d(s_t^{k'}, s_{t+1}^{k'})$ to identify the index of the entity whose state has most changed during the transition, where $d(\cdot, \cdot)$ is a distance function, detailed in the Appendix. Having the buffer of transitions over entity sets $\{(\mathbf{h}_1, ...\mathbf{h}_T)\}_{n=1}^N$ to a buffer of individual entity transitions $\{(h_t^k, a_t, h_{t+1}^k)\}_{n=1}^N$, and thus a buffer of individual state transitions $\{(s_t^k, a_t, s_{t+1}^k)\}_{n=1}^N$, the second step is to cluster similar transitions together by clustering the states before and after each transition. Our `cluster` procedure achieves this by clustering over all states together, which is sufficient since the end state of transition is the starting state of another. A proper clustering should return centroids $\{s_*\}_{m=1}^M$ that all correspond to actual states of entities, which we hypothesize will correspond to actual locations in the training set based on how dSLATE is trained.

The purpose of edges between the nodes is to record an action that transforms an entity from one state to another. For each entity transition (h_t^k, a_t, h_{t+1}^k) we `bind` h_t^k and h_{t+1}^k to their associated nodes $s_*^{[i]}$ and $s_*^{[j]}$ and create an edge between $s_*^{[i]}$ and $s_*^{[j]}$ tagged with action a, overwriting previous

edges based on the assumption that with a proper clustering there should only be one action primitive per pair of nodes. Applying the `bind` procedure to an entity h^k returns the index of the cluster of the centroid s_* that is nearest to the entity's state s^k. For our experiments `cluster` is implemented with K-means and `bind` is implemented with nearest neighbors using the same distance metric used for K-means (see Appendix), but other clustering algorithms and distance metrics can also be used.

2.2 Control

Having constructed a graph of state transitions, our approach for control re-composes sequences of state transitions to solve new rearrangement problems. Specifically, the agent decomposes the rearrangement problem into a set of per-entity subproblems, searches the transition graph for a transition that transforms the current entity's state to its goal state, and executes the action tagged with this transition in the environment. This problem decomposition is possible because the transitions in our graph are constructed to be agnostic to identity and context, enabling different rearrangement problems to share solutions to the same subproblems. The core challenge in deciding which transitions to compose is in determining which transitions are *possible* to compose. That is, the agent must determine which nodes in the graph correspond to the given goal constraints and which nodes correspond to the entities in the current observation, but the current entities $\mathbf{h}_t$ and goal constraints $\mathbf{h}_g$ must themselves be inferred from the current and goal observations o_t and o_g, requiring the agent to infer both what to do and how to do it purely from its sensorimotor interface.

Algorithm 2 Action Selection

given `model`, `graph`
input goal o_g, observation o_t
infer goal constraints and current entities
$\mathbf{h_g}, \mathbf{h_t} \leftarrow \texttt{model}\left(o_g\right), \texttt{model}\left(o_t\right)$
align entity indices of $\mathbf{h_t}$ with those of $\mathbf{h_g}$
$\pi \leftarrow \texttt{align}\left(\mathbf{h_t}, \mathbf{h_g}\right)$
permute indices of $\mathbf{h_t}$ according to π
$\mathbf{h_t} \leftarrow \left(h_t^{\pi[1]}, ..., h_t^{\pi[K]}\right)$
identify kth goal constraint to satisfy next
$k \leftarrow \texttt{select-constraint}\left(\mathbf{h_t}, \mathbf{h_g}\right)$
infer cluster assignments
$[i], [j] \leftarrow \texttt{bind}\left(h_t^k\right), \texttt{bind}\left(h_g^k\right)$
action that transforms node $[i]$ to node $[j]$
return $\texttt{graph.edges}[i, j]\texttt{.action}$

Our approach takes four steps (Alg. 2, Fig. 3). The first step applies dSLATE to infer $\mathbf{h}_t$ and $\mathbf{h}_g$ from o_t and o_g. The second step permutes the indices of $\mathbf{h}_t$ and $\mathbf{h}_g$ to `align` the identities of h_t^k with h_g^k. This step is necessary because the permutation invariance of the entities does not guarantee that the entities with the same identities will have the same index k, i.e. z_t^k might not correspond to z_g^k. We implement this using the Hungarian algorithm that minimizes the matching cost between $\mathbf{z}^t$ and $\mathbf{z}^g$. The third step selects which goal constraint h_g^k to satisfy next. In the case where objects can move independently, we implement this `select-constraint` procedure by determining which constraint h_g^k has the highest difference in state with its counterpart h_t^k, which reduces to solving the same argxmax problem as in `isolate`. We discuss how we handle more complex dependencies in the Appendix. Lastly, we `bind` h_t^k and h_t^g to the graph and return the action tagged to the edge between their respective nodes. If an edge does not exist between the nodes, we simply take a random action.

3 Experiments

The crucial test for knowledge reuse is to solve a set of tasks from a part of the combinatorial space of object rearrangement problems that are disjoint from the part of the space given for training, i.e. generalizing to different numbers of objects. We measure the "fractional success rate:" the mean change in the number of satisfied constraints divided by the number of initially unsatisfied constraints for each episode. In the *complete* setting, all objects have an associated constraint, whereas in the *partial* setting, only a subset of objects have associated constraints, and this is reflected in the goal image (Fig. 2b). We consider three challenging environments: *block-rearrange*, *robogym-rearrange*, and *block-stacking*. We compare against five baselines. We implement a version of MPC [36] by replacing our factorized graph search with CEM. We investigate an ablation of HA that constructs a monolithic graph over transitions between sets of slot states rather than individual slot states (MGS). We also compare with state-of-the-art pixel-based behavior cloning (BC) and implicit Q-learning (IQL) implementations based off of [22]. Our last baseline just takes random actions (Rand).

Results Figure 4 shows that HA performs significantly better than the baselines in combinatorial generalization (about a 5-10x improvement). Notably, the MPC baseline performs poorly because its rollouts are poor, and it is significantly more computationally expensive to run (11 hours instead of

20 minutes). Fig. 3a shows a t-SNE plot of the factors over 100 batches in the training set, where different colors label different clusters, with several clusters annotated with the average slot attention mask for the entities in the cluster. The right panel of Fig. 3 walks through the steps our algorithm takes to select an action, visualizing the `align`, `isolate`, and `bind` procedures, showing that the entities inferred with dSLATE intuitively capture objects that can be independently acted upon.

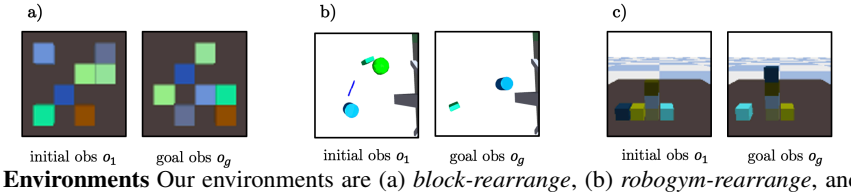

Figure 2: **Environments** Our environments are (a) *block-rearrange*, (b) *robogym-rearrange*, and (c) *block-stacking*. (a) shows the complete specification of goal constraints for all objects; (b) shows a partial specification of goal constraints. (c) shows the case where some goal constraints are already satisfied in the initial observation.

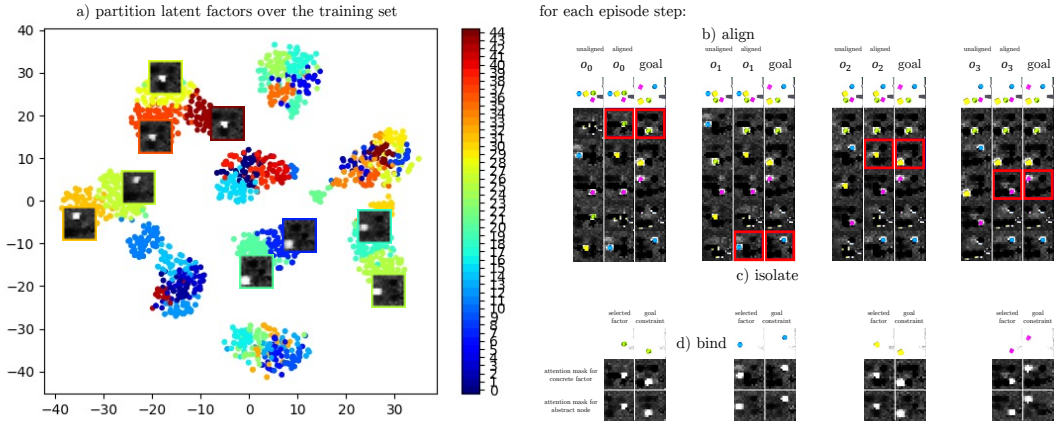

Figure 3: **Qualitative walk-through.** (a) shows the slot attention masks of the centroids from clustering states in *robogym-rearrange* (b) shows a walk-through of how HA selects the next action.

Figure 4: Generalizing from an offline ataset of 4 objects to solving rearrangement tasks with 4-7 objects, evaluated for 100 episodes across 10 seeds, showing standard error.

Table 1: *block-rearrange*, complete specification.

Method	4	5	6	7
Ours	**0.94** $\pm$ 0.01	**0.93** $\pm$ 0.00	**0.93** $\pm$ 0.00	**0.89** $\pm$ 0.00
Rand	0.06 $\pm$ 0.02	0.07 $\pm$ 0.03	0.07 $\pm$ 0.03	0.08 $\pm$ 0.03
MPC	0.16 $\pm$ 0.06	0.12 $\pm$ 0.04	0.11 $\pm$ 0.04	0.10 $\pm$ 0.03
MGS	0.07 $\pm$ 0.03	0.06 $\pm$ 0.02	0.07 $\pm$ 0.02	0.08 $\pm$ 0.03
IQL	0.07 $\pm$ 0.01	0.03 $\pm$ 0.00	0.02 $\pm$ 0.00	0.02 $\pm$ 0.00
BC	0.03 $\pm$ 0.00	0.02 $\pm$ 0.00	0.01 $\pm$ 0.00	0.01 $\pm$ 0.00

Table 2: *block-stacking*, complete specification.

Method	4	5	6	7
Ours	**0.58** $\pm$ 0.02	**0.48** $\pm$ 0.02	**0.35** $\pm$ 0.01	**0.24** $\pm$ 0.01
Rand	0.02 $\pm$ 0.01	0.01 $\pm$ 0.01	0.02 $\pm$ 0.01	0.02 $\pm$ 0.01
MPC	0.03 $\pm$ 0.01	0.01 $\pm$ 0.01	0.02 $\pm$ 0.01	0.01 $\pm$ 0.01
MGS	0.13 $\pm$ 0.00	0.01 $\pm$ 0.01	0.02 $\pm$ 0.01	0.01 $\pm$ 0.01
IQL	0.19 $\pm$0.02	0.12 $\pm$0.01	0.13 $\pm$0.02	0.07 $\pm$0.01
BC	0.21 $\pm$0.01	0.13 $\pm$0.01	0.13 $\pm$0.01	0.08 $\pm$0.01

Table 3: *robogym-rearrange*, complete specification.

Method	4	5	6	7
Ours	**0.64** $\pm$ 0.01	**0.47** $\pm$ 0.01	**0.49** $\pm$ 0.01	**0.41** $\pm$ 0.01
Rand	0.01 $\pm$ 0.0	0.01 $\pm$ 0.0	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0
MPC	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0
MGS	0.01 $\pm$ 0.0	0.01 $\pm$ 0.0	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0
IQL	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0
BC	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0

Table 4: *block-stacking*, partial specification.

Method	4	5	6	7
Ours	**0.34** $\pm$ 0.01	**0.27** $\pm$ 0.01	**0.21** $\pm$ 0.01	**0.15** $\pm$ 0.01
Rand	0.03 $\pm$ 0.01	0.03 $\pm$ 0.01	0.03 $\pm$ 0.01	0.03 $\pm$ 0.01
MPC	0.04 $\pm$ 0.02	0.03 $\pm$ 0.01	0.02 $\pm$ 0.01	0.04 $\pm$ 0.01
MGS	0.03 $\pm$ 0.01	0.03 $\pm$ 0.01	0.02 $\pm$ 0.01	0.02 $\pm$ 0.01
IQL	0.14 $\pm$0.01	0.06 $\pm$0.01	0.04 $\pm$0.00	0.03 $\pm$0.01
BC	0.17 $\pm$0.01	0.07 $\pm$0.01	0.04 $\pm$0.00	0.03 $\pm$0.00

4 Discussion

We have demonstrated that HA outperforms various offline RL methods across three simulated object rearrangement problems. We hope HA will inspire future work on how abstractions of independent and symmetric entities can be learned and re-composed.

References

[1] David Abel, D. Ellis Hershkowitz, Gabriel Barth-Maron, Stephen Brawner, Kevin O'Farrell, James MacGlashan, and Stefanie Tellex. Goal-based action priors. In *ICAPS*, pages 306–314. AAAI Press, 2015.

[2] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[3] Craig Boutilier, Richard Dearden, and Moises Goldszmidt. Exploiting structure in policy construction. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, pages 1104–1111, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[4] Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. Stochastic dynamic programming with factored representations. *Artif. Intell.*, 121(1-2):49–107, 2000.

[5] Berk Calli, Arjun Singh, Aaron Walsman, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. The ycb object and model set: Towards common benchmarks for manipulation research. In *2015 international conference on advanced robotics (ICAR)*, pages 510–517. IEEE, 2015.

[6] Michael Chang, Thomas L Griffiths, and Sergey Levine. Object representations as fixed points: Training iterative refinement algorithms with implicit differentiation. *arXiv preprint arXiv:2207.00787*, 2022.

[7] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. *arXiv preprint arXiv:1612.00341*, 2016.

[8] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. 2016.

[9] Carlos Diuk, Andre Cohen, and Michael L. Littman. An object-oriented representation for efficient reinforcement learning. In *ICML*, volume 307 of *ACM International Conference Proceeding Series*, pages 240–247. ACM, 2008.

[10] Ben Eysenbach, Russ R Salakhutdinov, and Sergey Levine. Search on the Replay Buffer: Bridging Planning and Reinforcement Learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/5c48ff18e0a47baaf81d8b8ea51eec92-Paper.pdf.

[11] Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3):189–208, 1971. ISSN 0004-3702. doi: https://doi.org/10.1016/0004-3702(71)90010-5. URL https://www.sciencedirect.com/science/article/pii/0004370271900105.

[12] Natalia Gardiol and Leslie Kaelbling. Envelope-based Planning in Relational MDPs. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems*, volume 16. MIT Press, 2003. URL https://proceedings.neurips.cc/paper/2003/file/4a06d868d044c50af0cf9bc82d2fc19f-Paper.pdf.

[13] M. Ghallab, A. Howe, C. Knoblock, D. Mcdermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL—The Planning Domain Definition Language, 1998. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.212.

[14] Klaus Greff, Sjoerd Van Steenkiste, and Jürgen Schmidhuber. Neural expectation maximization. *arXiv preprint arXiv:1708.03498*, 2017.

[15] Klaus Greff, Raphaël Lopez Kaufman, Rishabh Kabra, Nick Watters, Christopher Burgess, Daniel Zoran, Loic Matthey, Matthew Botvinick, and Alexander Lerchner. Multi-object representation learning with iterative variational inference. In *International Conference on Machine Learning*, pages 2424–2433. PMLR, 2019.

[16] Klaus Greff, Sjoerd van Steenkiste, and Jürgen Schmidhuber. On the binding problem in artificial neural networks. *arXiv preprint arXiv:2012.05208*, 2020.

[17] C. Guestrin, D. Koller, R. Parr, and S. Venkataraman. Efficient Solution Algorithms for Factored MDPs. *Journal of Artificial Intelligence Research*, 19:399–468, October 2003. ISSN 1076-9757. doi: 10.1613/jair.1000. URL https://jair.org/index.php/jair/article/view/10341. tex.ids: guestrin2003EfficientSolutionAlgorithms.

[18] Carlos Guestrin, Daphne Koller, Chris Gearhart, and Neal Kanodia. Generalizing plans to new environments in relational mdps. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, IJCAI'03, page 1003–1010, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers Inc.

[19] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[20] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

[21] Thomas Kipf, Gamaleldin F Elsayed, Aravindh Mahendran, Austin Stone, Sara Sabour, Georg Heigold, Rico Jonschkowski, Alexey Dosovitskiy, and Klaus Greff. Conditional object-centric learning from video. *arXiv preprint arXiv:2111.12594*, 2021.

[22] Ilya Kostrikov. JAXRL: Implementations of Reinforcement Learning algorithms in JAX, 10 2021. URL https://github.com/ikostrikov/jaxrl.

[23] Tejas D Kulkarni, Ankush Gupta, Catalin Ionescu, Sebastian Borgeaud, Malcolm Reynolds, Andrew Zisserman, and Volodymyr Mnih. Unsupervised learning of object keypoints for perception and control. *NeurIPS*, 2019.

[24] Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. *arXiv preprint arXiv:2006.15055*, 2020.

[25] Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. In *NeurIPS*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/8511df98c02ab60aea1b2356c013bc0f-Abstract.html.

[26] OpenAI. Robogym. https://github.com/openai/robogym, 2020.

[27] Luis Pineda, Brandon Amos, Amy Zhang, Nathan O. Lambert, and Roberto Calandra. Mbrl-lib: A modular library for model-based reinforcement learning. *Arxiv*, 2021. URL https://arxiv.org/abs/2104.10159.

[28] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[29] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. *arXiv preprint arXiv:2102.12092*, 2021.

[30] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[31] Gautam Singh, Fei Deng, and Sungjin Ahn. Illiterate DALL·E learns to compose. *arXiv preprint arXiv:2110.11405*, 2021.

[32] Gautam Singh, Fei Deng, and Sungjin Ahn. Illiterate DALL-e learns to compose. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=h0OYV0We3oh.

[33] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[34] Sjoerd Van Steenkiste, Michael Chang, Klaus Greff, and Jürgen Schmidhuber. Relational neural expectation maximization: Unsupervised discovery of objects and their interactions. *arXiv preprint arXiv:1802.10353*, 2018.

[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[36] Rishi Veerapaneni, John D Co-Reyes, Michael Chang, Michael Janner, Chelsea Finn, Jiajun Wu, Joshua Tenenbaum, and Sergey Levine. Entity abstraction in visual model-based reinforcement learning. In *Conference on Robot Learning*, pages 1439–1456. PMLR, 2020.

[37] C. Wang, S. Joshi, and R. Khardon. First order decision diagrams for relational mdps. *Journal of Artificial Intelligence Research*, 31:431–472, Mar 2008. ISSN 1076-9757. doi: 10.1613/jair.2489. URL http://dx.doi.org/10.1613/jair.2489.

[38] Ge Yang, Amy Zhang, Ari S. Morcos, Joelle Pineau, Pieter Abbeel, and Roberto Calandra. Plan2vec: Unsupervised Representation Learning by Latent Plans. In *Proceedings of The 2nd Annual Conference on Learning for Dynamics and Control*, volume 120 of *Proceedings of Machine Learning Research*, pages 1–12, 2020.

[39] Amy Zhang, Adam Lerer, Sainbayar Sukhbaatar, Rob Fergus, and Arthur Szlam. Composable Planning with Attributes. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80, pages 5837–5846. JMLR.org, 2018.

[40] Daniel Zoran, Rishabh Kabra, Alexander Lerchner, and Danilo J Rezende. Parts: Unsupervised segmentation with slots, attention and independence maximization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10439–10447, 2021.