2019 IEEE Symposium Series on Computational Intelligence (SSCI)
December 6-9 2019, Xiamen, China

Automatic Composite Action Discovery for Hierarchical Reinforcement Learning

Josiah Laivins and Minwoo Lee
Computer Science Department
University of North Carolina, Charlotte
Charlotte, NC 28223
Email: {jlaivins; minwoo.lee}@uncc.edu

***Abstract*—Even with recent advances in standard reinforcement learning, hierarchical reinforcement learning has been discussed as a promising approach to solve complex problems. From human-designed abstraction, planning or learning with composite actions are well-understood, but without human intervention, producing abstract (or composite) actions automatically is one of the remaining challenges. We separate this action discovery from reinforcement learning problem and investigate on searching impactful composite actions that can make meaningful changes in state space. We discuss the efficiency and flexibility of the suggested model by interpreting and analyzing the discovered composite actions with different deep reinforcement learning algorithms in different environments.**

***Keywords*—hierarchical reinforcement learning; composite actions; skill discovery; options; automatic discovery; abstraction; temporal composition**

I. Introduction

Design of autonomous agents to effectively traverse their environments has required a balance of exploration and exploitation. Early work involved detailed system designs [1], [2] were frameworks for humans to use as references for complex system design. Unfortunately, these early systems grow in complexity so much so that this prevents agents from truly scaling to human-level competence. This has encouraged research in reinforcement learning (RL) which promises reduced human involvement by simply asking the agent to determine the best self-designed methods of achieving a task.

[3] found success training a neural network to do this, and unprecedentedly was able to use the same architecture and parameters to win many different Atari games such as Video Pinball, Boxing, Breakout and 46 more, most above human-level playing abilities. [4] evaluated several RL algorithms such as Trust Region Policy Optimization (TRPO) [5], Proximal Policy Optimization (PPO) [6], and Deep Deterministic Policy Gradients (DDPG) [7] on physical robots. They evaluated actions such as reaching toward objects, moving, and docking with charging stations.

Unfortunately, as agents are asked to learn more complex tasks, they quickly suffer from sparser rewards and succumb to the curse of dimensionality in their state and action spaces. As a result, in earlier works, humans need to get involved to help them along: the designed actions enable the agent to reach a state difficult to reach if using primitive actions. This kind of human labor can manifest itself in different strategies such as by [8], [9] whose algorithms require human metadata input to better describe the environment that the agent is working in. [10] encourages the agent to be interested in novelty and will often become *bored* and thus maintain a constant balance of exploration and exploiting its environment similar to its biological counterparts. [11], [12], [13], [14] attempt to avoid laboriously adding metadata to environments by instead improving the effectiveness of the actions the agent takes. Instead of trying to reduce the state space, their common strategy lets a chunk of actions into hierarchies to make state space traversal and exploration more effective. These frameworks are working toward the concept of hierarchical reinforcement learning (HRL) of which was detailed by [15]. Throughout this paper, we will define *composite action* as a raw set of action values or policies of which could be hierarchical or flat. We summarize the main drawbacks contained in these frameworks are human involvement 1) in composite action design, 2) in defining the number of composite actions available, and 3) in composite action size as required by HAC algorithm developed by [14].

Automatic discovery of composite actions has received a lot of attention these days to overcome these drawbacks. Creating these composite actions with as little human intervention as possible is an important puzzle piece for HRL agents to be more effective. Nature seems to learn the same way. When a human wants to accomplish some new task such as cooking, they do not start learning to cook from scratch. Commonly, they would have knowledge of composite actions such as grasping, cutting, reaching, and stirring. All of these natural composite actions might have different lengths, numbers of nested composite actions, and most of all at some point in time are created from primitive movements (temporal abstraction [16]). Solutions were proposed to reduce the requirement for human design of *options* [12]. [17] allowed an agent to build more advanced hierarchies by chaining existing options into logical groups. [18] developed an algorithm for creating options via chaining from a root option. [19] used a clustering algorithm to break the MDP into regions, then assigned an option per region.

Our method separates the automatic composite action discovery problem from learning formulation in order to further alleviate the limitations and improve generality. Our dimen-

978-1-7281-2484-1/19/$31.00 ©2019 IEEE

sionality limit is higher and has been successfully run on Atari game RAM inputs. We also do not need any extensive MDP to start using our method for composite action extraction.

Our contributions are summarizes to

- no human involvement in composite action creation,
- individual problem design for composite action discovery, which can work with standard RL algorithms, and
- an effective composite action achievement by correlating it with notable changes in state space.

In this paper, we perform a proof of concept via evaluating several environments, using different agents, different models, different state and action space representations, all while using a single set of parameters. Our hypothesis is that changes in learned action sequences can be partitioned using entropy or distance methods. However, we also hypothesize that the most useful actions will cause some change in state space. Our method will compare the action distributions with the state distributions and keep only the composite actions that have been observed to cause a correlated effect on the state space. To the best of our knowledge, no previous research has attempted to "automatically" extract "effective" composite actions without extra metadata or human interference. We believe that our method can prove to significantly improve the usability of existing HRL algorithms, as well as future HRL algorithms.

II. Related Works

A. Planning and Learning with Composite Actions

Hierarchy-based composite actions have been of interest in the general planning [8], [9], [2], [1], and learning [20], [11], [13]. While these existing frameworks have their strengths, the common problem they share is the persistence of human intervention in the environment, agent, or action design.

Early work in hierarchical composite actions were defined by [1] via manually designing multiple layers of human designed *behaviors*. They organized these behaviors into hierarchical layers via a categorical system they termed 7 levels of competence. Similarly [2] postulate the existence of different *schemas* such as motor and perceptual Schemas. These methods shared the same weakness in that the intention was for the human designer to design each Behavior or Schema, and classify them under one of these levels.

STRIP-like logical descriptions of their environments and actions allow an agent to create action hierarchies. [8] accomplished this through the human creation of fluents and operators, and [9] did so using similar logical based constraints and agent to environment contact points. The amount of information required to describe the actions and environments would require a human to create an entire sub-ontology of their environment, and then debug that ontology.

Skills [11] assume the presence of multiple related tasks. They determine the number of skills as proportional to the number of tasks the agent is trying to accomplish. [12] covers a similar strategy called *options* and is further elaborated on in [21]. [13] developed *Boundary Localized Reinforcement Learning* (BLRL) which divides the state space into multiple *modes*. The mode boundaries evolve to develop sub-policies. Once again, there is a question of how options, modes, and skills are created in the first place. Unfortunately, the theory of options still requires a human designer to determine the number of options present in an environment, skills are directly tied to a task, and ultimately determined by the number of tasks, and modes require human-defined regions in the state space.

All of these algorithms in some way or another are trying to allow an agent to solve exceedingly complex tasks by reducing the state space, improving the complexity of actions, and adding extra information from the environment to make a valid decision. As we have found, behaviors and schemas are early attempts at improving agent robustness by designing abstractions of actions available to agents. Both of these are examples of how one would create a robust robotic system that could react to its world, and how to think about designing the system by considering the tasks that the robot might be trying to solve. While this can be seen in some form or another today, most agents are still given simple highly related tasks primarily due to the sheer complexity of these systems, and the amount of human design work involved. [11], [13], [12] understood this by leveraging the power of reinforcement learning. They could drastically reduce the amount of human design work. Their methods used the idea of telling the agent to use a specific number of human-designed composite actions (skills, modes, options), and augment those actions into more effective versions of themselves. While this has proven popular, many of the immediate drawbacks still involve a human correlating composite actions with the number of tasks, defining specific state space regions to develop in, or designing the composite actions manually. [9], [8] study alternative approaches to allowing agents to solve real world problems. And while both could be easily transferable to other tasks, the level of human design work is still infeasible for sufficiently complex environments. Moreover, modified formulation of semi-MDP [12] and Universal MDP [14] enables an agent to deal with multi-step composite actions along with primitive actions.

B. Automatic Composite Action Discovery

Understanding the needs of automated discovery of composite actions without human help lead the emergence of learning frameworks that construct or discover composite actions. [14] improve agent search through its state space by using an automatic form of sub-policy creating and training. Their method allows for generating hierarchies simply based on the agent's experience during training. However, these sub-policies are restricted in their size during the entire duration of the agent training. [18] create options via chaining. Based on a single root option, the algorithm works backward, creating options as a chain to the original one. [17] built upon options [12] by allowing an agent to build more advanced hierarchies by chaining options into logical groups. While this improves the capabilities of existing options, this does not address creating a single option from primitive actions. [19] proposed

the use of Perron Cluster Analysis (PCCA+) to discretize the state space regions based on the MDP. Once this was done, an option would be created per region. However, they note that PCCA+ suffers from high dimensional state spaces thus requiring a more complex pipeline. It also requires acceptably built MDPs before discretization.

We clearly cut connections between learning methods and composite action discovery so we can apply the general method for different algorithms for either standard RL or HRL. We design the algorithm to be invariant to a single composite action size, and so the agent could always have an opportunity to learn small composite actions or large ones. Also, our method does not need a root composite action to build from, nor will it restrict composite action creation to regions near other composite actions, which helps cut dependence to specific type of algorithm.

III. Methods

The input to our method is the environmental information such as action and state Markov Chains. We design the model computationally efficient so that we can run online, so during training of an RL agent, we can immediately start collecting candidates for our composite actions,

$$A_{T,n} = \begin{pmatrix} a_{0,1} & a_{0,2} & \cdots & a_{0,n} \\ a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{T,1} & a_{T,2} & \cdots & a_{T,n} \end{pmatrix}$$

where T is the max time step, and n is the action joint. Many simpler environments may only have $n = 1$, however, larger multi-joint agents will have more. Our method will output composite actions for each joint is they are found. Along with actions, we need the Markov Chains for state transitions. The state space can be anything from meta level environment output or raw image output. If images are provided, it is then assumed they are flattened to 1D vectors,

$$S_{T,n} = \begin{pmatrix} s_{0,1} & s_{0,2} & \cdots & s_{0,n} \\ s_{1,1} & s_{1,2} & \cdots & s_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ s_{T,1} & s_{T,2} & \cdots & s_{T,n}. \end{pmatrix}$$

Let b be defined as the number of bins to discretize action outputs. Let w be the size of the window to run over the Markov Chains. Our defaults will be $b = 64$ and $w = 5$. This method is largely inspired by [22] and [23]. [22] used Shannon entropy for anomaly detection in real time space shuttle systems, while [23] used entropy for drawing segmentation. Our method performs action segmentation via changes in both action and state space. Our algorithms for doing binning, calculating probabilities, and windowing are here:

$$X_{binned} = \{\arg\max(HIST(x_t, b, \min(X), \max(X))) \quad (1)$$
$$\mid x_t \in X\} \quad (2)$$

where $HIST$ is a generic histogram building function that takes in the number of bins b, the minimum and maximum of X, a an element x_t. X can be either S or A. The resulting X_{binned} is a discretized version of X,

$$X_p = \left\{ \frac{COUNTS(x_t, X_{binned})}{\sum COUNTS(x_t, X_{binned})} \mid x_t \in X \right\} \quad (3)$$

where $COUNTS$ is a generic value counting function. At present the probabilities for each value is strictly based on that value's occurrence frequency resulting in a set of probabilities X_p.

Once we bin our action and state readings similar to [22] and extract probabilities, we generate a window per time step. However, their final entropy extraction was done via graphing the readings of over 120 sensors of t time steps. Since a single action over T time steps is a single 1D vector, we convert both the actions and the states per time step into windows over w time steps given $w_l = w/2 + 1$ and $w_r = w/2$:

$$X_w = \{x_{t-w_l, t+w_r} \mid x_t \in X_p\}. \quad (4)$$

Our sliding windows are centered over each time step. Our implementation of Eq. (4) uses a padding procedure to maintain matching dimensions.

Eq. (5) takes three parameters for an entropy function f, the window size w, and X which is either a single action joint Markov Chain A_n or the state Markov Chain S:

$$E_w = \{f(x_{t \to t+\Delta \times w}) \mid x_t \in X\} \quad (5)$$

where we define Δ as:

$$\Delta = \begin{cases} 1, & t \leq T - w \\ -1, & \text{otherwise} \end{cases} \quad (6)$$

The resulting E_w is a mask of entropy values of the same length of X. Instead of padding the edges of E_w, we simply change the direction via Δ once we reach close to the end of the Markov Chain. Our method can next make use of any of the many different methods of comparing probability distributions. We divide the different methods we have experimented with into entropy and distance methods. As a note, E_w will need to be partially redefined via Eq. (7).

First and foremost, we started with Shannon entropy [24] and is of the form:

$$H = -\mathcal{K} \sum_{i-1}^{n} p_i \log p_i$$

where p_i is a single probability in a single window, and n is the window size. $\mathcal{K}$ is defaulted to 1.

If we are to simply consider a single 1D vector, typical Shannon entropy might not be desirable. We would be interested in methods of determining entropy over time series, and more specifically, find changes in entropy over those time series. [25] was also interested in this in finding irregularities in heartbeats, and so developed Approximate Entropy (ApEn).

It uses Kolmogorov-Sinai entropy as their measure with modifications where m and r are strictly fixed variables. Their algorithm is of the form:

$$ApEn = \Phi^m(r) - \Phi^{m+1}(r).$$

where m is a per comparison region size in the time series, and r is the behaves as a smoothing parameter. Some of the immediate drawbacks we found was pointed out by the authors in that their algorithm is "badly compromised by noise." ApEn is affected by series length and has relative inconsistency and bias. [26] designed their time series entropy analysis to solve this drawback:

$$SampEn = -\ln(A^m(r)/B^m(r)).$$

Compared to ApEn, SampEn has reduced bias, is invariant to series length, and is relatively consistent, which means that given a set of parameters m and r, one would expect a linear change in reported entropy.

Another form of time series based entropy analysis [27] was developed with the intent to have an entropy function that was both fast and did not require a high level of hyper parameter tuning. It is formulated similar to Shannon Entropy as:

$$H(n) - \sum p(\pi) \log p(\pi).$$

However, the variable π is found via

$$p(\pi) = \frac{\#\{t|0 \leq t \leq T-n, (x_{t+1}, \cdots, x_{t+n}) \text{ has type } \pi\}}{T-n+1}$$

where n is defined as the order of the function and defines the number of permutations to try in the time series.

KL-Divergence [28], [29] is one of the most popular strategies of comparing the regularity of two distributions. In order to do this, we can use two adjacent windows in a Markov Chain, and pass in their probability distributions here:

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \left(\frac{P(x)}{Q(x)} \right)$$

where P and Q represent different windows.

[30] developed Jensen-Shannon divergence to solve the trouble posed by KL-Divergence which as the authors stated "requires absolute continuity." Jensen-Shannon divergence makes use of Jensen's inequality to achieve this:

$$JS_\pi(p_{1,2}) = H(\pi_1 p_1 + \pi_2 p_2) - \pi_1 H(p_1) - \pi_2 H(p_2).$$

Most interestingly, JS-divergences allows us to optionally assign different importance to different probability distributions via π. In this work, we assign equal weights (0.5) for the given two probability distributions. In future work, one interesting experiment would be to test more than two windows with different weights.

[31] experimented with multiple distances and divergences training a Generative Adversarial Network. Along with KL-Divergence and JS-Divergence they tested Total Variation (TV) distance:

$$\delta(P_r, P_g) = \sup_{A \in \sum} |P_r(A) - P_g(A)|.$$

They also experimented with Earth-Mover (EM) distance:

$$W(P_r, P_g) = \inf_{\gamma \in \prod(P_r, P_g)} E_(x,y)\ \gamma[\|x-y\|].$$

One of the important benefits of EM is that it is differentiable, so they were able to use it in designing a loss function.

Regardless of the function we choose, our framework makes it easy to test all of them by simply plugging in. Given that b and w define our binning and window size, X is defined by our states and actions, and finally for any of the entropy functions we simply pass them as f to Eq. (5). However, given available divergence and distance functions, the formulation revised as:

$$E_w = \{\min(f_{j=1}^{w}(x_{t \to t+\Delta \times w}, x_{t+\Delta \times j \to t+\Delta \times (j+w)})) \mid x_t \in X\}. \quad (7)$$

We included $\min$ because given a window, we want to see if there is a similar pattern found in its neighbors. Δ changes the direction of which sequence of windows to look at. This is a padding avoidance measure.

For example, given multiple single element windows of some distribution X,

$$X = \begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.1 & 0.2 & 0.3 \end{bmatrix},$$

and given a window size of $w = 3$ at time step $t = 0$, we get

$$w_0 = \begin{bmatrix} 0.1 & 0.2 & 0.3 \end{bmatrix}.$$

If we test one of the functions such as $f = Jensen\_Shannon\_Divergence(p, q)$, we would get the expected output of

$$f_{j=1}^{w}(w_0, X_j) = \begin{bmatrix} 0.066 & 0.066 & 0.0 \end{bmatrix}.$$

The lowest value indicates the most similar neighbor to w_0. In this instance, the value was 0 indicating there is a pattern exactly similar to w_0. E_w will now have the the result of any of the entropy functions or divergence functions we have discussed above. The next step is normalizing the distributions as

$$E_n = (E_w - \min(E_w)) \ / \ (\max(E_w) - \min(E_w)). \quad (8)$$

Once normalized, we convert the distributions into binary distributions based on threshold λ (default is 0.5):

$$E_{bin} = E_n > \lambda. \quad (9)$$

Supposing that we have done above with distributions A and S (in place of X), we now have binary distributions $E_{bin(A)}$ and $E_{bin(S)}$, we correlate changes in the action A Markov Chain with changes in S Markov Chain. Our hypothesis is that a useful composite action should cause some noticeable change in state space. Formally, for each time step, we check if there is an intersection between $t+w$ state and action binary distribution truth values. This is modeled by:

$$A_{Correlated} = \{g(s_t, a_t) \mid s_t, a_t \in E_{bin(S)}, E_{bin(A)}\} \quad (10)$$

TABLE I. METHOD PARAMETERS

Gyms	w	b	λ	f
Cartpole	5	64	0.5	Earth-Mover
MountainCar	5	64	0.5	Earth-Mover
Pendulum	5	64	0.5	Earth-Mover
Acrobot	5	64	0.5	Earth-Mover
Boxing	5	64	0.5	Earth-Mover
Breakout	5	64	0.5	Earth-Mover
Pong	5	64	0.5	Earth-Mover
Tennis	5	64	0.5	Earth-Mover
Skiing	5	64	0.5	Earth-Mover

where,

$$g(s_t, a_t) = \begin{cases} True, & \text{if } \exists s_{t \to t+\Delta \times \omega} \cap a_{t \to t+\Delta \times \omega} \neq \emptyset \\ False, & \text{otherwise} \end{cases} \tag{11}$$

Our final step is breaking the action Markov Chain into composite actions based on our correlations between our state and action binary distributions. $PART$ is a function that takes in a binary distribution, and between each consecutive truth pair, a partition is created, so

$$A_{Composite} = \{A_{partition} \mid partition \in PART(A_{Correlated})\}.$$

The final result is a single function to extract composite actions simply by analyzing the action and state distributions. In total, the there are 4 basic parameters. We can denote the entire process as:

$$A_{Composite} = Extract(A, S, b, w, \lambda, f).^1$$

IV. EXPERIMENTS

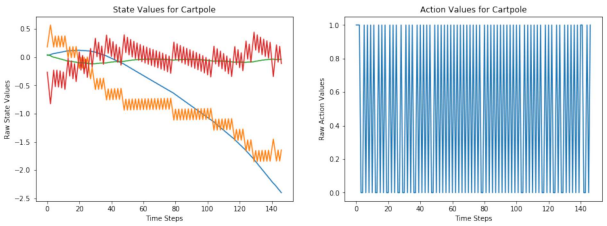
We validate our methods on several OpenAI environments [32] including simple classic benchmarks and Atari games. We used the same parameters for all environments as shown in Table I.

For discrete action environments we used Dueling Double DQNs (DDDQNs) [33], and for continuous action environments we used Deep Deterministic Policy Gradient (DDPG) [7]. We used Earth-Mover distance due to it generating the best results over our pilot tests.

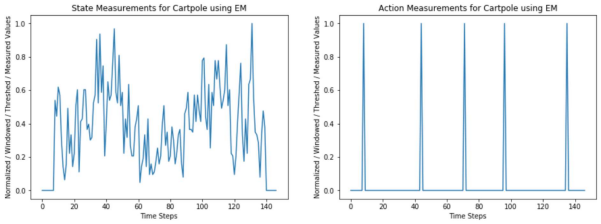
Fig. 1 illustrates the process of the automatic composite action discovery. Unlike other proposed methods, our method extracts the composite actions that have the greatest effect on the state space. This extraction is also rich in information, and the agent can first try composite actions that have been observed to cause the greatest changes in the state space.

We found that this did so successfully for the 8 other environments that we performed our pilot test on as referenced in Fig. 2 and Fig. 3. For classic control problems such as Cartpole and MountainCar were certainly easier for our method to extract actions which commonly fell between 1-5 composite actions. Cartpole composite actions typically were involved in stationary oscillation, while MoutainCar hard changes in

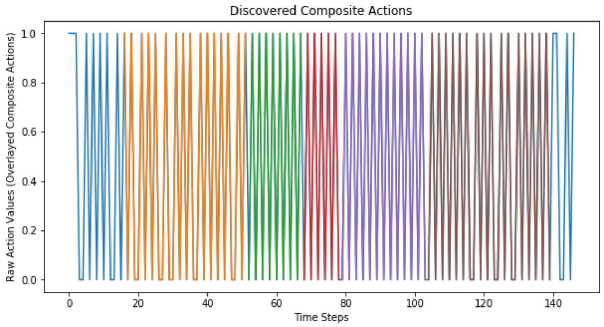
[1]The implementation is available at https://github.com/josiahls/CompositeActionExtractor.



(a) State and action values differ greatly in number of dimensions and values.



(b) Regardless of noise, we can break a Markov Chain into useful segments of changes between transitions.



(c) Composite action discovery via correlating changes in actions space with their effects in the state space. Each color represents different patterns of composite actions.

Fig. 1. Composite Action Extraction from Classic Environment.

direction. Based on our observations, Cartpole composite actions could be useful and transferable to agents with joints that require the same oscillating behavior. MoutainCar composite actions seem more task specific, so it may not be reusable but it can help learning fast in similar tasks.

Both Acrobot and Pendulum generated composite actions for handling momentum. Acrobot composite actions either involved swinging oscillation or as seen in Fig. 2 was a compact strategy for slinging the lower second arm in the counter clockwise direction. While that Acrobot's second composite action seems task specific, the first can be useful and transferable to other agents. Pendulum typically had 2 composite action phases: phase 1 for keeping the pendulum vertical and phase 2, as seen in the figure, hard torque in the counter clockwise direction. Phase 1 seems like a common behavior shared between pendulum and Cartpole, and interestingly phase 2 has high force, then quick slow down behavior that can be useful for other agents to employ.

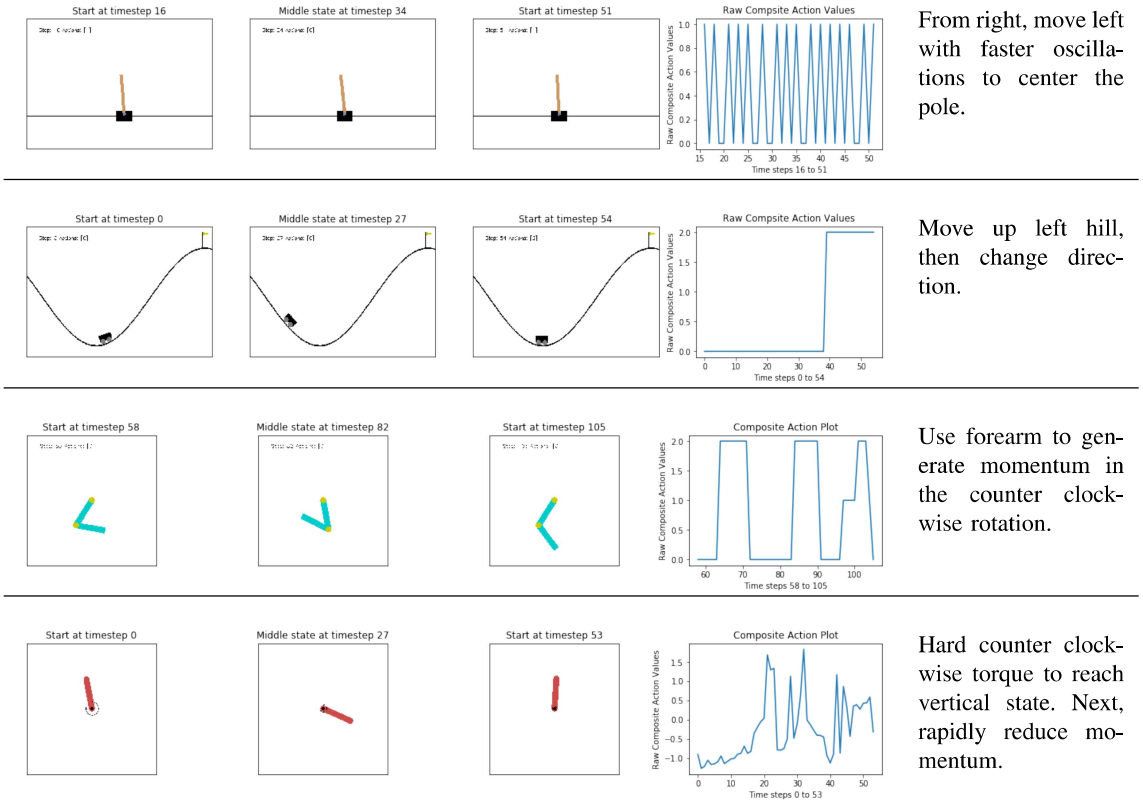


Fig. 2. Classic Environment Analysis and Human Interpretation of Composite Actions.

Boxing and Tennis in Fig. 3 demonstrate complex composite actions being employed against an adversary. Boxing action involved maintain distance, hitting, then flanking. Tennis found a serve and return composite action.

Pong, although also reacting to adversarial actions, produced composite actions of simple fast reactions to the ball. It is possible that Pendulum's high force, quick slow down composite action could be useful also to the Pong agent. Skiing and Breakout generated extremely simple actions such as repositioning after a previous action. Also based on our observations, Pong, Pendulum, and Breakout could possibly benefit from shared high force, quick slow down composite actions since those kinds of composite actions were generated from each of them separately.

Even when faced with both noisy action and state Markov Chains, by comparing them with each other we were able to extract useful slices of the action sequences, and interpret what each is doing. Not only were we able to accomplish this, but we were also able to do this across different agents, and different environments using the same set of parameters. Furthermore, since the extraction runs over the trajectories of states and actions, which are turned into binary distributions, we were able to run the algorithm regardless of the deep reinforcement learning algorithm choice. We have run the same method without modification for both DDDQN and DDPG.

V. Conclusions

We set out to develop a human intervention free approach to composite action discovery. Early preliminary results indicated that this is indeed feasible, and successfully discovered composite actions from multiple different environments and agents using a single set of parameters. We highlight that the discovered actions are not merely a temporal composition of a sequence of actions but as sequences that actually cause the agent's state to change.

Additionally, future works could extend our method to transfer learning between agents, composite action extraction from human demonstration, or to improve existing motion planning methods. Composite action projection for task transfer will be another interesting next step.

References

[1] R. Brooks, "A robust layered control system for a mobile robot," *IEEE journal on robotics and automation*, vol. 2, no. 1, pp. 14–23, 1986.

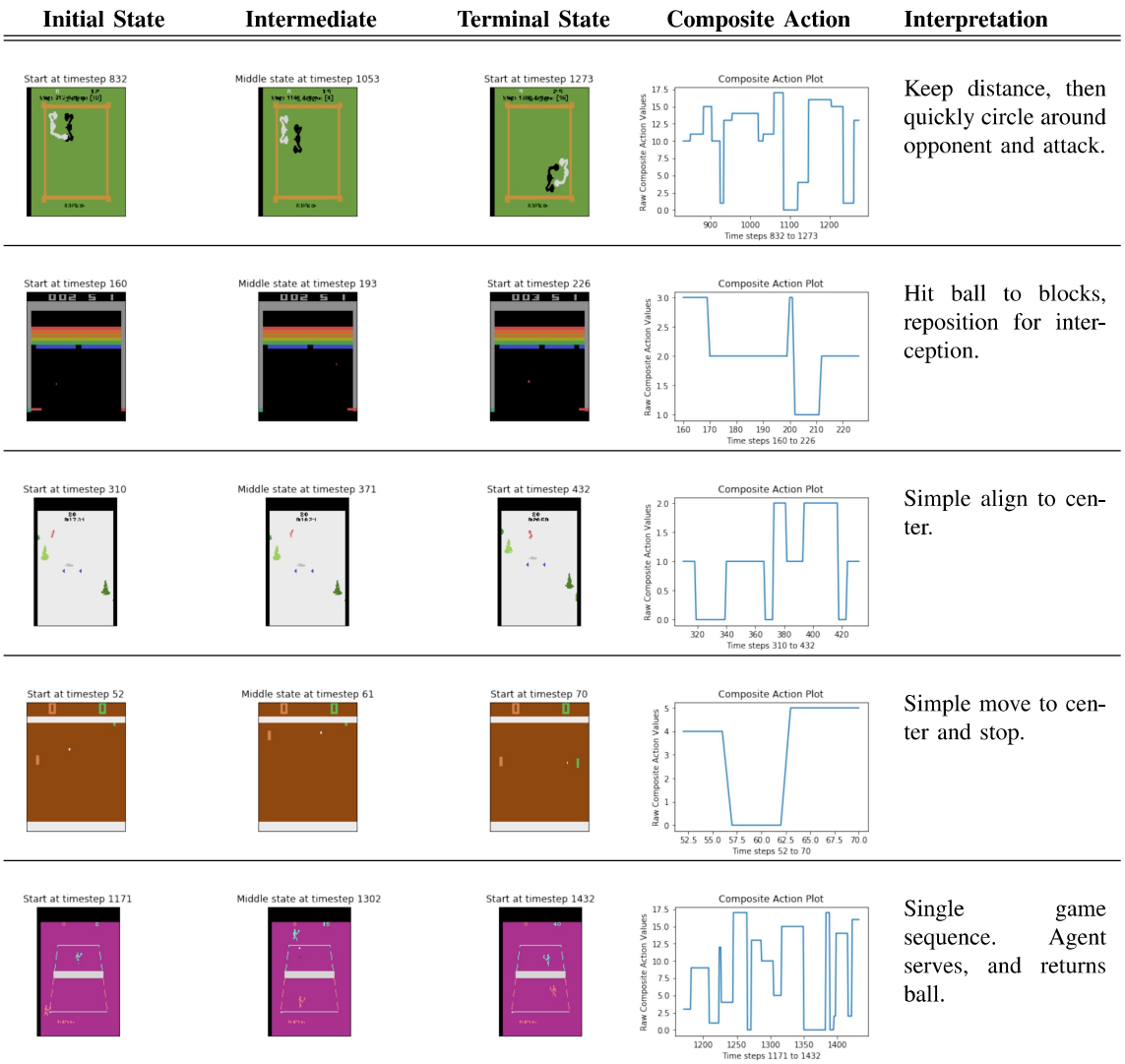


Fig. 3. Atari Game Environment Analysis and Human Interpretation of Composite Actions.

[2] M. A. Arbib, "Schema theory," *The Encyclopedia of Artificial Intelligence*, vol. 2, pp. 1427–1443, 1992.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[4] A. R. Mahmood, D. Korenkevych, G. Vasan, W. Ma, and J. Bergstra, "Benchmarking reinforcement learning algorithms on real-world robots," *arXiv preprint arXiv:1809.07731*, 2018.

[5] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, 2015, pp. 1889–1897.

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[7] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[8] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical planning in the now," in *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[9] K. Hauser and J.-C. Latombe, "Integrating task and prm motion planning: Dealing with many infeasible motion planning queries," in *ICAPS09 Workshop on Bridging the Gap between Task and Motion Planning*. Citeseer, 2009.

[10] N. Chentanez, A. G. Barto, and S. P. Singh, "Intrinsically motivated reinforcement learning," in *Advances in neural information processing systems*, 2005, pp. 1281–1288.

[11] S. Thrun and A. Schwartz, "Finding structure in reinforcement learning," in *Advances in neural information processing systems*, 1995, pp. 385–392.

[12] R. S. Sutton, D. Precup, and S. P. Singh, "Intra-option learning about temporally abstract actions." in *Proceedings of the 15th International Conference on Machine Learning ICML*, vol. 98, 1998, pp. 556–564.

[13] G. Z. Grudic and L. H. Ungar, "Localizing search in reinforcement learning," in *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, 2000, pp. 590–595.

[14] A. Levy, G. Konidaris, R. Platt, and K. Saenko, "Learning multi-level hierarchies with hindsight," in *The International Conference on Learning*

Representations, 2019.

[15] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete event dynamic systems*, vol. 13, no. 1-2, pp. 41–77, 2003.

[16] P.-L. Bacon, J. Harb, and D. Precup, "The option-critic architecture," in *Twelfth Conference on Innovative Applications of Artificial Intelligence*, 2017.

[17] H. Sahni, S. Kumar, F. Tejani, and C. Isbell, "Learning to compose skills," *arXiv preprint arXiv:1711.11289*, 2017.

[18] G. Konidaris and A. G. Barto, "Skill discovery in continuous reinforcement learning domains using skill chaining," in *Advances in neural information processing systems*, 2009, pp. 1015–1023.

[19] A. S. Lakshminarayanan, R. Krishnamurthy, P. Kumar, and B. Ravindran, "Option discovery in hierarchical reinforcement learning using spatio-temporal clustering," *arXiv preprint arXiv:1605.05359*, 2016.

[20] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.

[21] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[22] A. Agogino and K. Tumer, "Entropy based anomaly detection applied to space shuttle main engines," in *2006 IEEE Aerospace Conference*. IEEE, 2006, pp. 7–pp.

[23] Z. Sun, C. Wang, L. Zhang, and L. Zhang, "Free hand-drawn sketch segmentation," in *European Conference on Computer Vision*. Springer, 2012, pp. 626–639.

[24] C. E. Shannon and W. Weaver, *The mathematical theory of communication*. University of Illinois press, 1998.

[25] S. M. Pincus, I. M. Gladstone, and R. A. Ehrenkranz, "A regularity statistic for medical data analysis," *Journal of Clinical Monitoring*, vol. 7, no. 4, pp. 335–345, Oct 1991.

[26] J. S. Richman and J. R. Moorman, "Physiological time-series analysis using approximate entropy and sample entropy," *American Journal of Physiology-Heart and Circulatory Physiology*, vol. 278, no. 6, pp. H2039–H2049, 2000.

[27] C. Bandt and B. Pompe, "Permutation entropy: a natural complexity measure for time series," *Physical review letters*, vol. 88, no. 17, p. 174102, 2002.

[28] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, no. 1, pp. 79–86, 03.

[29] D. J. MacKay, *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[30] J. Lin, "Divergence measures based on the shannon entropy," *IEEE Transactions on Information theory*, vol. 37, no. 1, pp. 145–151, 1991.

[31] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[32] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[33] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.