A Framework for Creative Problem Solving Through Action Discovery

Evana Gizzi, Mateo Guaman Castro, Wo Wei Lin, Jivko Sinapov
Department of Computer Science
Tufts University
Medford, Massachusetts 02155
{Evana.Gizzi, Mateo.Guaman, Wo_Wei.Lin, Jivko.Sinapov}@tufts.edu

***Abstract*—Creative problem solving (CPS) is a process through which an agent discovers previously unknown information about itself and its environment in order to achieve an unsolvable task. In this paper, we introduce a unified framework for CPS through action discovery. We describe two methods which enable action discovery at a declarative and neurosymbolic level, namely through action primitive segmentation, and behavior babbling, respectively. We review experimental evaluations of our framework, and end with a discussion on limitations and future work considerations for CPS.**

I. Creative Problem Solving Framework

Recent advances in the learning community have enabled complex robotic behaviors to be learned from scratch, such as manipulating textiles [10] and solving a Rubik's cube [7]. Yet, the ability for a robot to creatively act and improvise in unstructured environments, much like humans and animals do on a daily basis [1, 2], remains one of the key challenges of modern robotics. Moreover, learning these behavior policies tabula-rasa usually results in learned actions that are neither easily explainable nor flexible. We propose a framework for creative problem that addresses the issues of explainability and policy inflexibility by combining declarative representations of the world with methods for exploring the agent's environment and updating its knowledge base.

We begin with a description of a creative problem solving task, describing its differences to a general problem solving (GPS) task. In the context of GPS, a planning agent, given an accurate depiction of its world and its own capabilities, is able to act in its environment to accomplish goals. In an encounter with novelty, however, the agent may no longer contain an accurate description of its world, e.g., the underlying dynamics of the world may have changed, or the agent has incurred an unforeseen scenario. For example, a robot may encounter a door knob which requires twisting, as opposed to its previously known "pull-down" action. In either case, the agent is unable to use its world representation to plan and act successfully as in GPS. This presents the case of CPS, where the agent must somehow update its world model at both a declarative and neurosymbolic level, for planning and execution respectively, in order to successfully problem solve. Thus, a key distinguishing factor between a GPS problem, and one which necessitates CPS is that the latter requires knowledge base updating in order to accomplish a goal task [5].

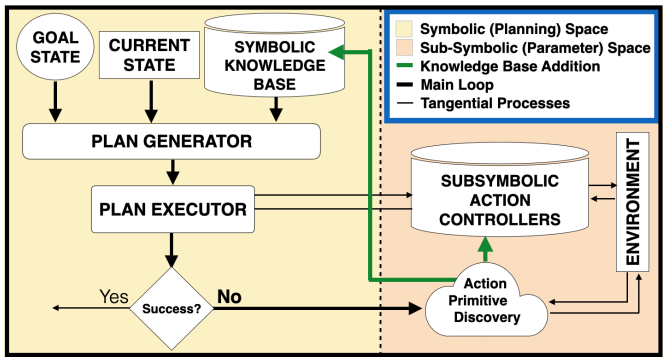


Fig. 1. System diagram of the Action Primitive Discovery Framework

We depict a high level description of the CPS framework in Figure 1, where the agent first reasons about a problem at a fully symbolic level. Given a desired goal state, the agent is able to generate a plan for reaching the desired goal state. At the time of plan execution, if the agent encounters either an execution failure *during* plan execution, or a failure to accomplish the desired goal *after* plan execution, it enters into a sub-symbolic action primitive discovery mode. Any novel actions discovered in this mode are then added back into its knowledge base, where the cycle continues until the agent is successful in accomplishing its goal.

We describe two methods for autonomous knowledge acquisition for CPS in the context of action discovery at both a symbolic and sub-symbolic level. In both methods, candidate actions are generated, and subsequently evaluated based on their effects on the environment. In the action primitive segmentation method, the trajectories of known actions are segmented using change point detection, and used as candidate actions. In the behavior babbling method, candidate actions are generated by varying the low level action parameters of known actions, employing a behavior babbling technique. In both cases, those candidate actions which produce novel effects, *or* those which succeed in obtaining the desired effects of the original action are added back into the knowledge base of the agent, and used in re-planning.

A. Notation

We describe the two knowledge abstraction levels of our framework, namely those corresponding to high level symbolic

planning, and the low level sub-symbolic action execution. Following, we describe two action primitive discovery methods which bridge our two knowledge abstraction spaces.

1) Symbolic Knowledge Base: A symbolic knowledge base Σ is defined as $\Sigma = \langle \mathcal{E}, \mathcal{F}, \mathcal{S}, \mathcal{A} \rangle$, where:

- $\mathcal{E}$ is a finite set of known entities in the environment such that $\mathcal{E} = \{e_1, \ldots, e_{|\mathcal{E}|}\}$
- $\mathcal{F}$ is a finite set of known predicate descriptors (and their negations) about the world which operate over entities in the world such that $\mathcal{F} = \{f_1(\odot), \ldots, f_{|\mathcal{F}|}(\odot)\}, \odot \subset \mathcal{E}$. It follows that for every predicate descriptor $f_i(\odot_i)$, there exists a negation of that predicate descriptor $\neg f_i(\odot)$, where $f_i(\odot), \neg f_i(\odot) \in \mathcal{F}$.
- $\mathcal{S}$ is a set of possible world states such that $\mathcal{S} = \{s_i \ldots s_{|\mathcal{S}|}\}$. Each $s_i \in \mathcal{S}$ is composed of a finite set of predicate values $\mathcal{F}_i \subset \mathcal{F}$ which hold true in the given world state s_i.
- $\mathcal{A}$ is a set of known actions such that $\mathcal{A} = \{a_1, \ldots a_{|\mathcal{A}|}\}$ where each action a_i operates over a finite entity list $\mathcal{E}_i \subset \mathcal{E}$, such that each action takes the form $a_i(e_i^1, \ldots, e_i^{|\mathcal{E}_i|})$ = $a_i(\mathcal{E}_i)$.

We assume a *Planning Domain Definition Language (PDDL)* representation of actions, where for a given action a_i, we refer to the set $\mathcal{E}_i$ as the *arguments* of a_i, and the sets χ_i, $\psi_i \in \mathcal{F}$ as the *preconditions* and *effects* of a_i, respectively. Following, a problem in Σ is defined as $\phi = (\Sigma, s_0, s_g)$, where s_0 is an initial state, s_g is a goal state, and $s_0, s_g \in \mathcal{S}$. A plan $\pi = [a_1, \ldots a_{|\pi|}]$ is a solution to a problem ϕ.

2) Sub-Symbolic Knowledge Base: We assume that our agent has a set of action controllers $\mathcal{M}_\Sigma = [m_1, \ldots, m_{|\mathcal{A}_\Sigma|}]$ containing sub-symbolic information about the actions in Σ. We denote the action and its associated controller as a_i and m_i, respectively. Each controller encodes information about the underlying parameter settings of its associated action. The parameters could describe information about the agents' end effectors relative to the entities being operated over in actions, along with the velocities/force and magnitude of motion execution (referred to explicitly as *execution parameters*). Additionally, the parameters could describe specific trajectory points for starting and stopping action execution (referred to explicitly as *trajectory parameters*). Thus, a given action controller m_i encodes a finite set of continuous action parameters $\mathcal{P}_i = p_i^1, \ldots p_i^{|\mathcal{P}_i|}$, such that each controller takes the form $m_i(p_i^1, \ldots, p^{|\mathcal{P}_i|_i})$ = $m_i(\mathcal{P}_i)$.

B. Problem Formulation

An agent starts with a knowledge base $\mathcal{K} = (\Sigma, \mathcal{M}_\Sigma)$, and given a problem ϕ, a planner generates a plan π to accomplish a goal state s_g. The planning agent, containing an accurate representation of the world in its symbolic knowledge base Σ, is able to successfully execute π, thereby achieving its goal state s_g. We refer to this case as the *original scenario*. Suppose that in the context of novelty, something about the world changes such that Σ is no longer sufficient, but needs to be updated with new information such that Σ becomes Σ'. The agent also must learn a new set of corresponding action controllers $\mathcal{M}_{\Sigma'}$. We refer to this scenario as the *novel scenario*. In this novel context, the planner initially uses Σ to plan for solving ϕ, once again generating π. Upon executing π, a plan failure occurs for some action $a_f \in \pi$.

At this point, the agent needs to explore its world to learn a new knowledge base Σ', providing it with an updated and accurate representation of the new world, along with its corresponding set of action controllers $\mathcal{M}_{\Sigma'}$. We define the learning process $\mathcal{L}$ as the process in which an agent can learn a new knowledge base $\mathcal{K}'$ using exploration method ω, such that $\mathcal{L}(\mathcal{K}, \omega) \mapsto \mathcal{K}'$.

C. Action Discovery

The learning process is composed of two-part incubation stage, where the agent attempts to find a viable candidate action for CPS. In the first phase of incubation (*passive incubation*), the agent generates a list of candidate actions to vary. We refer to these variation candidates as action-entity-parameter (AEP) combinations. This list of candidates is generated as a subset of an exhaustive list of variation combination options of $\mathcal{K} = (\Sigma, \mathcal{M}_\Sigma)$ denoted:

$$C_\mathcal{K} = C(a \times e \times p), \forall a \in \mathcal{A}_\Sigma, e \in \mathcal{E}_a, p \in \mathcal{P}_a$$

where a given subset list of $C_\mathcal{K}$, generated under an exploration method ω, is denoted $C_{\mathcal{K},\omega}$. The list $C_\mathcal{K}$ denotes all possible actions, paired with all possible entity arguments and *focus parameter* selections to each given action, where the focus parameter is the parameter which will be varied in the exploration phase of a particular AEP execution. Next, the agent scopes its AEP exploration space to prepare for action discovery. We present two modes of de-scoping in passive incubation, namely *focused passive incubation* and *defocused passive incubation*. In the focused case, the agent prunes its AEP generation to the specific plan involved in the failure. In the defocused case, the agent instead relaxes its constraints to allow for entities other than those included as arguments to actions in the initial plan π. In this way, the agent is able to explore the possibility of other entities as a means for problem solving.

Next, the agent enters the second phase of incubation (*active incubation*), the agent then begins evaluating candidate actions, using a chosen discovery method ω, to determine their utility toward solving ϕ. We introduce two action discovery methods ω which enable action discovery, namely through action segmentation and action parameter variation, unified under the learning process $\mathcal{L}$.

1) Segmentation: The agent attempts to discover novel actions by segmenting the trajectories of known action primitives into reusable and significant substructure, such that the newly discovered actions are useful in a stand-alone manner. As such, the agent executes each action combination in $c \in C_\mathcal{K}$ which has a trajectory parameter as its focus parameter. From each execution, the agent generates set of change points $\Delta_i = [t_1, \ldots t_k], t \in \mathbb{Z}$, where Δ is a time series list of ordered time stamps corresponding to change points. The input to the change point detection algorithm is a set of time series

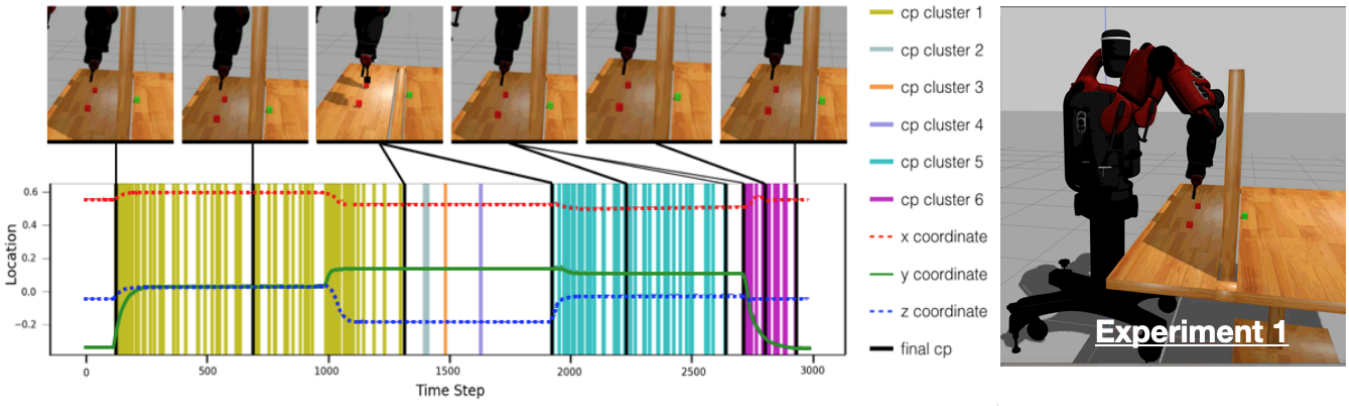


Fig. 2. **(Left)** Change point detection for press_button(right_gripper, left_button). The plot shows the x-y-z location of the left gripper over time steps during action execution, separated into three separate traces. All vertical lines represent change points in the motion data, grouped based on clustering, with a total of 6 clusters. The black vertical lines show the change points that got used for segmentation, consisting of the min/max/mean of clusters that include more than 10 change points. **(Right)** Setup for Experiment 1.

trajectories that represent motor data (e.g., joint efforts, or end-effector position) for the arm that executed the action. That is, for an action a_i, a set of n trajectories $d_1, \ldots d_n$ are represented by a matrix $M \in \mathbb{R}^{n \times d_i}$. These change points are used to generate candidate action segmentations $\hat{a_1}, \ldots \hat{a_{k-1}}$ where each $\hat{a_j}$ corresponds to the limb trajectory from time step t_j to t_{j+1}. Each candidate $\hat{a_j}$ is executed, generating effects $\psi_{\hat{a_j}}$. Finally, the preconditions of the failure action a_f are refined to include the effects of the action that succeeded to accomplish s_g.

2) Parameter Variation: In this stage, the agent executes the actions listed in the AEP list $C_{\mathcal{K},\omega}$, generated by one of the two strategies mentioned. As such, the agent executes each action combination in $c \in C_{\mathcal{K}}$ which has a execution parameter as its focus parameter. Execution of these combinations are less straightforward, however, than the segmentation case. In the case of parameter variation, the chosen parameter to vary p_{a_i} is first set to values ranging from its lower and upper bound, with a chosen Δ_n interval partition value, such that $(\overline{p_{a_i}} - \underline{p_{a_i}}) \div \Delta_n = I$, where I is the interval of execution. That is, for any $c(a_i, \mathcal{E}_{a_i}, p_{a_i})$, the action $a_i((E)_{a_i})$ (a_i for short) is executed for the focus parameter p_{a_i} from its lower bound to its upper bound value, incrementing by the interval of execution. This generates a secondary list of candidate variants, $\mathcal{P}'_{a_i}$, shown below:

$$\mathcal{P}'_{a_i} = (\underline{p_{a_i}}), (\underline{p_{a_i}} + I), (\underline{p_{a_i}} + 2I), \ldots (\overline{p_{a_i}})$$

Therefore, for each AEP element, the elements of a corresponding candidate variation list $\mathcal{P}'_{a_i}$ are each executed.

3) Candidate Action Evaluation: For each parameter variation in the candidate list ($p^j \in \mathcal{P}'_{a_i}$), the agent evaluates its execution to determine whether it should be added into the knowledge base for future use. There are two conditions by which a given variant $c(a_i, \mathcal{E}_{a_i}, p_{a_i})$ (call it c) with $p_{a_i} = p^j$ (call it $\hat{c}$) can be added to Σ – if $\psi_{\hat{c}} = \psi c$, or if $\psi c \subset \psi_{\hat{c}}$. That is, a candidate can be added if it accomplishes precisely the same end effects as its parent action, or if it accomplishes the same end effects as its parent action, with additional novel effects. We call these conditions the *inheritance* and *novelty* condition, respectively. The inheritance condition holds in focused incubation, and the novelty condition holds in defocused incubation.

It should be noted that some algorithmic components are agnostic to our framework, including the choice of planner used to find π, the change point detection algorithm used to find Δ, and motion data collected in M.

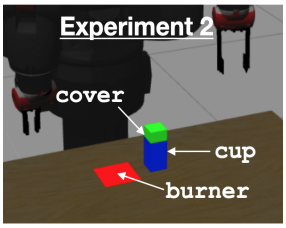
II. Experimental Evaluations

We ran a total of 4 experiments on a Baxter robot in a Gazebo simulation environment. Each experiment presents a manipulation task, where the Baxter is given objects sitting on a table in front of its grippers. For our motion trajectory data, we recorded the 3-dimensional coordinate (*x*-*y*-*z*) position of the gripper performing the action. Using the data collected during exploration, change points were generated using our BCP/clustering algorithm, with motion trajectory matrix M as input. We illustratively describe 2 of our 4 experiments below. Additional information and full experimentation results can be found at [4, 6].

A. Experiment 1

The environment of Experiment 1 is shown in Figure 2(right). Here, the agent has the goal of obtaining the green block, located at the other side of the sliding wall. In the original scenario, the wall is very short, and the agent is able to reach over it to obtain the object. In the novel scenario, the wall is much higher (depicted in Figure 2). Thus, upon generating and executing $\pi = [\texttt{obtain\_object(l\_gripper, block)}]$, a failure is incurred due to the wall obstructing the path of the grippers to the object. At this point, the agent enters focused passive incubation, where it generates an AEP list comprised on argument variants of `obtain_object`. It is consequently unable to find a solution action from segmenting variants

Experiment 2	
s_0	at(cup, loc_a) at(cover, loc_b) at(burner, loc_c) is_visible(cup) is_visible(cover) is_visible(burner) covered(cup)
s_g	cook(cup)
π	push(cover) uncover_obj(cup) shake(cup) prep_food(cup) put_on_burner(cup) cover_obj(cup) cook(cup)



(Left) Initial state, goal state, and resulting plan in both the original and novel scenarios of Experiment 2 shown.
(Top) Setup shows the Baxter robot in Experiment 2 manipulation task.

TABLE I
EXPERIMENT 2 SETUP

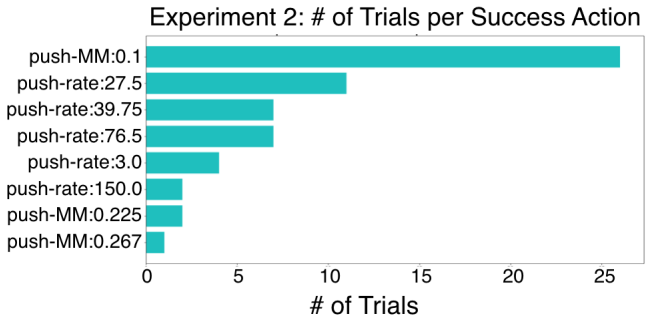


Fig. 3. Success Actions discovered in Experiment 2.

of the obtain_object action. Next, it enters a defocused passive incubation stage, where it is able to successfully segment the press_button action to find a solution plan. This process is illustrated in Figure 2, shown over the execution of a = press_button(l_gripper, l_button), where nine change points were detected $[t_1, \ldots t_9]$, resulting in eight trajectories $\hat{a_1}, \ldots \hat{a_8}$. Additionally, it can be seen that the only sub-trajectory with non-empty state effects list is $\hat{a_2}$, with $\psi_{\hat{a_2}}$ = is_visible(block),pressed(l_button), corresponding to the sub-trajectory from t_2 to t_3. In this particular example, none of the other sub-trajectories produced a change in state-based predicates in their effects lists. Thus, the robot was able to use $\hat{a_2}$ to successfully obtain the object by first holding the left button (via $\hat{a_2}$), and then using the obtain_object action primitive to move the object with its right gripper, generating a new plan, $\pi' = [\ \hat{a_2}$(l_gripper, l_button),obtain_object(r_gripper, block)$]$.

A detailed description of an algorithm for discovering novel actions based on segmentation is presented in [4]. Repeated evaluations showed that the method was able to successfully discover useful actions for accomplishing the task in the novel scenario. The robot also discovered actions with consistent end effects that were not used as part of the solution, suggesting that CPS can be formulated as a life-long learning problem, where actions discovered in one situation may become useful in another, later scenario.

B. Experiment 2

The environment of Experiment 2 is shown in the figure within Table I, describing the original scenario, where the agent has the goal of "cooking" food. In the original scenario, the agent is able to successfully execute the plan π shown in Table I. In the novel scenario, the cover is lighter in weight than the original scenario, resulting in a plan execution failure when attempting cover_obj, since the light weight cover is pushed off of the table when executing push, and is therefore out of reach of the robot. This presents a peculiar scenario, where the failure action a_f = cover_obj is not responsible for the failed plan. Here, the agent executes an full passive incubation cycle to discover that substituting push with push_rate:3.0 will result in a successful scenario. In this way, the agent has learned the equivalent of a nudge action.

We examine the diversity of the discovered actions, where we found interesting and unexpected results, shown in Figure 3. We had originally hypothesized that a decrease in the rate parameter would be the only sufficient success action (push-rate:3.0 did get utilized 4 times). We were surprised to find that the most frequently utilized success action was push-MM:1.0, which resulted in a decreased movement magnitude of the push action. In this way, the agent was able to "nudge" the cover off of the cup without fully pushing it off of the table. Another surprise was that an *increased* rate also resulted in the cover staying on the table. We believe this is because of the trajectory of the robot's arm, which would hover higher up when moving across the table to push the cover. As a result, the gripper simply "skimmed" the cover just enough to knock it off the cup, without the leverage of a full contact push, as was the case in the original scenario. These results show that our framework can enable a robot to find multiple solutions to a problem, some that are unforeseen by the designers of the environment.

III. Conclusion and Future Work

In this paper, we have presented a unified framework for creative problem solving, with action discovery examples. While recent work has considered using reinforcement learning for action controllers [3, 8], future work should consider extending this approach to continuous robotic state spaces, and integrating it into the framework. Additionally, future work should implement methods for lifelong CPS, wherein the agent is able to improve in its CPS over time.

We propose two research avenues for future work. The first avenue is the development of action discovery methods that minimize the destructiveness of the environment during the incubation processes. This would allow agents to perform CPS in situations in which the cost of exploration in the real world is high, such as in space missions. The second avenue is exploring the combination of a symbolic knowledge base with a learned model of the world [9], which could be learned prior to the CPS process or concurrently, to direct the CPS process based on the model predictions.

REFERENCES

[1] Sarah R Beck, Ian A Apperly, Jackie Chappell, Carlie Guthrie, and Nicola Cutting. Making tools isn't child's play. *Cognition*, 119(2):301–306, 2011.

[2] Christophe Boesch and Hedwige Boesch. Tool use and tool making in wild chimpanzees. *Folia primatologica*, 54(1-2):86–99, 1990.

[3] Rohan Chitnis, Tom Silver, Joshua Tenenbaum, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Glib: Efficient exploration for relational model-based reinforcement learning via goal-literal babbling. In *Proc. AAAI*, 2021.

[4] Evana Gizzi, Mateo Guaman Castro, and Jivko Sinapov. Creative problem solving by robots using action primitive discovery. In *2019 Joint IEEE 9th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 228–233. IEEE, 2019.

[5] Evana Gizzi, Lakshmi Nair, Jivko Sinapov, and Sonia Chernova. From computational creativity to creative problem solving agents. In *International Conference on Computational Creativity (ICCC)*, 2020.

[6] Evana Gizzi, Amel Hassan, Wo-Wei Lin, Keenan Rhea, and Jivko Sinapov. Toward creative problem solving agents: Action discovery through behavior babbling. In *Accepted: The International Conference on Development and Learning (ICDL)*. IEEE, 2021.

[7] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving rubik's cube with a robot hand. *arXiv preprint*, 2019.

[8] Vasanth Sarathy, Daniel Kasenberg, Shivam Goel, Jivko Sinapov, and Matthias Scheutz. Spotter: Extending symbolic planning operators through targeted reinforcement learning. *arXiv preprint arXiv:2012.13037*, 2020.

[9] Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *International Conference on Machine Learning*, pages 8583–8592. PMLR, 2020.

[10] Yilin Wu, Wilson Yan, Thanard Kurutach, Lerrel Pinto, and Pieter Abbeel. Learning to manipulate deformable objects without demonstrations. *RSS*, 2019.