Toward Creative Problem Solving Agents: Action Discovery through Behavior Babbling

Evana Gizzi, Amel Hassan, Wo Wei Lin, Keenan Rhea, Jivko Sinapov
Department of Computer Science
Tufts University
Medford, MA
{Evana.Gizzi,Amel.Hassan,Wo_Wei.Lin,Keenan.Rhea,Jivko.Sinapov}@tufts.edu

***Abstract*—Creative problem solving (CPS) is the process by which an agent discovers unknown information about itself and its environment, allowing it to accomplish a previously impossible goal. We propose a framework for CPS by robots for discovering novel actions via behavior babbling, capable of learning a representation of novel actions at both a symbolic planning level, and a sub-symbolic action controller level. Our framework employs two modes of discovery – a focused incubation method that scopes its search to the actions and entities composing the failed plan, and a defocused incubation method which enables exploration of actions and entities outside of the failed plan. We implemented and tested our framework using a Baxter robot in a 3D physics-based simulation environment, where we ran three proof-of-concept object manipulation scenarios. Results suggest that it is possible to use behavior babbling as a method for the autonomous discovery of flexible and reusable actions.**

***Index Terms*—Cognitive Robotics, Creative Problem Solving, Novelty**

I. Introduction

Creative problem solving (CPS) has often been described as the hallmark of intelligence and cognition [1]. When faced with challenging problems, both humans and non-human species have demonstrated the use of CPS for deriving novel and/or innovative solutions. In one interesting study in 2012, CPS abilities were shown to be present in great apes in a simple improvisation experiment using a puzzle box problem [2]. The apes, when presented with a clear horizontal tube containing a food reward, first attempted to retrieve the out-of-reach reward using their fingers. Through successful improvisation, the apes were eventually able to figure out how to push a stick that was pre-inserted inside the tube to displace the food outside of the tube, thereby attaining the reward.

This ability to adapt to unforeseen situations has been referred to as *MacGyvering* in recent cognitive robotics literature, in which an agent synthesizes a solution to a seemingly unsolvable problem by using a non-trivial combination of resources from its environment [3]. These resources can be physical, as in the case of *Tool MacGyvering* [4], or non-physical, where the agent learns new skills or previously unknown information about its environment to derive a problem solution. In this work, we explore the synthesis of non-physical resources through action learning. Problem solving improvisation through action learning remains a challenge in robotics, where problem solving often takes place through planning, by sequencing over symbolically represented actions. To this end, we propose a framework which enables the discovery of new actions at both a sub-symbolic action execution level, and a symbolic planning level, generating representations for use in CPS planning tasks. We test our framework across 3 scenarios in a robotic simulation environment, and provide evaluation toward our proof-of-concept[1].

II. Related Work

Behavior babbling has been researched in robotics as a method for action discovery. In many cases, behavior babbling takes place in the form of motor babbling, where the goal is to learn a model of self (sometimes referred to as emergent behavior or self-organization). Examples of such models are forward models and kinematic models, which use body dynamics to learn possibilities and constraints of robot movement [5]–[7]. Model of self techniques vary in their babbling validation approaches. For example, babbling can be reinforced proprioceptively on a sensorimotor level [8], [9], or by demonstrative measures [10], [11]. Other examples of validation techniques include "motionese," where the start and end states of actions are emphasized by adult teachers as a way to delineate important action stages [12], [13]. Bottom-up attention models use the detection of significant environmental changes as a way to learn new actions [14].

In addition to general purpose action discovery, models for computational creative problem solving have been developed, implemented and tested. Colin *et al.* formalize and implement a computational solution for the CPS process which utilizes hierarchical reinforcement learning (HRL), demonstrating that successful CPS is more likely to occur with prior relevant experiences [15]. Lieto *et al.* propose a logic-based method for CPS through conceptual blending, where features of objects are evaluated and combined in order to re-frame and re-formulate CPS problems to converge on a solution [16]. Kralik *et al.* develops a model of CPS using a Q-Tree learning algorithm to yield a hierarchical structure for problem representations, which was tested on empirical data from a reverse-reward problem run on rhesus monkeys [17]. Other CPS approaches have investigated methods for learning action

[1]Code and results available at https://github.com/tufts-ai-robotics-group/RAPDR_babble

transition models (via action operators) through model-based reinforcement learning [18] and bottom up relational learning in a task and motion planning (TAMP) domain [19].

Recent work in cognitive robotics has demonstrated success in creative problem solving implementations in robots and intelligent agent systems [20]. Nair *et al.* [4] proposed a framework for tool construction for CPS. Using a reference tool, and a set of available parts, the robot is able to reason about geometric properties to construct substitution tools in object manipulation tasks. More recently, autonomous tool construction for problem solving is explored by Yang *et al.* [21], which use gated graph neural networks to model relationships between tool parts in order to intelligently construct tools with appropriate contextual considerations. While related to these recent works which explore CPS in the context of tool construction, we focus our research on CPS in the context of general action discovery.

This work develops a method for action discovery through behavior babbling. The motivation for our approach is in inspired from human action discovery, where seemingly disparate high level actions can actually be considered coarse grained behavior parameterizations of one another. For example, consider the actions `shake` and `pour`. These two actions, when performed with a side grasp on a cup, have drastically different outcomes. Where shaking a cup of fluids will result in mixing its contents into a consistent mixture, pour the same cup will result in emptying the cup of its contents. Yet even so, these actions can be seen as parameter variations of one another in terms of the parameter `movementMagnitude`. This research is an extension of our previous work in action discovery through segmentation [22]. Both works use a common overarching framework for CPS, but differ in their method for action discovery.

III. Theoretical Framework

Next, we define the symbolic level of abstraction that describes information known to the planning agent, and the sub-symbolic level which describes information encoded for action execution, responsible for action babbling and parameter variation. We then define the problem formulation, and the action discovery method employed by the agent.

A. Symbolic Planning Representation

Knowledge Base: We assume that the robot has a symbolic knowledge base Σ, defined as $\Sigma = \langle \mathcal{E}, \mathcal{F}, \mathcal{S}, \mathcal{A}, \gamma \rangle$, where:

- $\mathcal{E}$ is a finite set of known entities in the environment such that $\mathcal{E} = \{e_1, \dots e_{|\mathcal{E}|}\}$
- $\mathcal{F}$ is a finite set of known predicate descriptors (and their negations) about the world which operate over entities in the world such that $\mathcal{F} = \{f_1(\odot), \dots f_{|\mathcal{F}|}(\odot)\}, \odot \subset \mathcal{E}$. 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 $\mathcal{S} = \{s_i \dots s_{|\mathcal{S}|}\}$. 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 $\mathcal{A} = \{a_1, \dots a_{|\mathcal{A}|}\}$ where each action a_i operates over a finite entity list $\mathcal{E}_i \subset \mathcal{E}$, such that each action takes the form $a_i(e_i^1, \dots, e_i^{|\mathcal{E}_i|})$ = $a_i(\mathcal{E}_i)$.
- γ is a transition function which contains known transitions between a finite set of states of the system $\mathcal{S}$ using $\mathcal{A}$, such that $\gamma(\mathcal{S}, \mathcal{A}) \mapsto \mathcal{S}'$

In addition to the action notation described above, we assume a *Planning Domain Definition Language (PDDL)* representation of actions. We refer to the set $\mathcal{E}_i$ of a particular action a_i as the *arguments* of a_i. Each action is assumed to have a set of *preconditions* and *effects*, denoted χ_i, $\psi_i \in \mathcal{F}$. The preconditions χ_i of a_i indicate the predicate descriptors which must hold true before executing a_i, and the effects ψ_i of a_i indicate the predicate descriptors which will be assumed to hold true after successful execution of a_i. The predicate descriptors composing both χ_i and ψ_i may also include negations, indicating predicate descriptors which must be false before and after execution, respectively.

Problem: 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 $\pi = [a_1, \dots a_{|\pi|}]$ is a solution to a problem ϕ.

B. Sub-Symbolic Action Controllers

In addition to the symbolic knowledge base Σ, we assume that our agent has a set of action controllers $\mathcal{M}_\Sigma = [m_1, \dots, m_{|\mathcal{A}_\Sigma|}]$ 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. Therefore, we define the symbolic and sub-symbolic information of the agent as the tuple $\mathcal{K}$ = $(\Sigma, \mathcal{M}_\Sigma)$, where $|\mathcal{A}_\Sigma|$ = $|\mathcal{M}_\Sigma|$. A given action controller m_i encodes a finite set of continuous action parameters $\mathcal{P}_i = p_i^1, \dots p_i^{|\mathcal{P}_i|}$, such that each controller takes the form $m_i(p_i^1, \dots, p^{|\mathcal{P}_i|_i})$ = $m_i(\mathcal{P}_i)$. Each individual parameter p_i^j encodes three values - a set default value, a set upper bound value, and a set lower bound value, denoted $*p_i^j$, $\overline{p_i}^j$, and $\underline{p_i^j}$, respectively. Presumably, the parameters could describe information about the agents' end effectors relative to the entities being operated over in actions. Additionally, the parameters could describe the velocities/force and magnitude of motion execution. While focused on continuous parameters, this methodology can be modified to consider discrete and/or non-numerical parameterizations, discussed later in the context of our framework.

Example: Consider the action a = `push`, with a controller m_a and parameters, p_a^1 = `rate`, p_a^2 = `movementMagnitude`, and p_a^3 = `orientation`. Each parameter has a set value, an upper bound, and a lower bound for a, encoded in m_a. For example, the `rate` parameter, which controls the rate of motion when pushing an object, may have the following encoding: $\underline{p_a^1}$ = 3, $\overline{p_a^1}$ = 100, and a set value of $*p_a^1$ = 20. In this case, the `push` symbolic action will move at the rate of 20, as is specified by

$*p_a^1$ in its associated controller m_a. This particular parameter instantiation of m_a is what connects the symbolic and sub-symbolic levels of knowledge representation.

C. 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 $a_f \in \pi$. We assume that for each action in the domain, there is an action executor model which can determine whether its action has succeeded or failed. These failures can happen throughout execution, or by checking if the end effects of the action have been fulfilled.

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 $\mathcal{L}(\mathcal{K}, \omega) \mapsto \mathcal{K}'$.

D. Action Discovery

1) Passive Incubation: Focused and Defocused Generation: In the first stage of incubation, *passive incubation* (Algorithm 2), the agent generates a list of candidate actions to vary through behavior babbling. 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. We denote any given AEP combination of $C_\mathcal{K}$ as $c(a, \mathcal{E}_a, p)$ where $a \in \mathcal{A}_\Sigma, \mathcal{E}_a \subset \mathcal{E}, p \in \mathcal{P}_a$.

We propose two modes of AEP generation, *focused* and *defocused*, inspired by concept development proposed by Sarathy [23]. In this work, focused mode is described as one where the agent utilizes goal-directed problem solving, and defocused mode as one where the agent utilizes exploration-directed problem solving. We modify this notion to instead consider both focused and defocused passive incubation stages as a part of the creative problem solving process [24]. We refer to

Algorithm 1 Action Primitive Discovery Framework

procedure PLANEXECUTOR($s_g, \mathcal{K}, \Delta_n, s_C$)
    s_g : predicates needed to accomplish goal; $\mathcal{K}$: knowledge base; Δ_n : interval partition parameter; s_C : predicates which are currently true
    $e \leftarrow 1$: episode number
    $\omega \leftarrow$ `focused` : AEP generation mode
    $\Psi \leftarrow \emptyset$: newly added actions
    $C_\mathcal{K} \leftarrow \emptyset$: AEP list
    while $s_g \not\subset s_C$ **do**
        $\pi \leftarrow generatePlan(s_c, s_g, \mathcal{K}, \Psi)$
        $\Psi \leftarrow \Psi \setminus \pi$
        for $a \in \pi$ **do**
            if $execute(a) = FAIL$ **then**
                $a_f \leftarrow a$; **break**
        if $s_g \subset s_C$ **then**
            break
        if $C_\mathcal{K} = \emptyset$ **then**
            $C_\mathcal{K} \leftarrow genAEPlist(\pi, a_f, \omega, \mathcal{E}_\Sigma)$
            if $e > 1$ **then**
                $\omega \leftarrow$ `defocused`
        if $\Psi = \emptyset$ **then**
            $c \leftarrow pop(C_\Sigma)$
            $P_c \leftarrow getParamVariants(c, \Delta_n)$
            for $\hat{c} \in P_c$ **do**
                $execute(\hat{c})$
                $\mathcal{E}_{\hat{c}} \leftarrow s_C$
                switch ω **do**
                    case ω = `focused`
                        if $\mathcal{E}_{\hat{c}} = \mathcal{E}_c$ **then**
                            $\Psi \leftarrow \Psi \cup \{\hat{c}\}$
                    case ω = `defocused`
                        if $\mathcal{E}_{\hat{c}} \neq \emptyset$ **and** $\mathcal{E}_{\hat{c}} \neq \mathcal{E}_c$ **then**
                            $\Psi \leftarrow \Psi \cup \{\hat{c}\}$
                $A_\Sigma \leftarrow A_\Sigma \cup \Psi$
        $e \leftarrow e + 1$
    return $\mathcal{K}$

these stages as focused incubation and defocused incubation, respectively. Unlike Sarathy [23], the distinction between these two phases is not drawn from goal vs. exploration affixed behavior. Instead, goal affixed behavior is utilized in *both* focused and defocused incubation. The distinction between the two modes, in this case, is between the scope of the problem solving search space, described in the following section. Additionally, unlike traditional ϵ-greedy algorithms, there is no notion of policy exploitation. Both focused and defocused incubation are exploration based, wherein focused incubation exploration is scoped toward actions involved in the initial plan, and defocused incubation exploration allows for a broadened search beyond those actions.

In *focused passive incubation* (shown on lines 8 - 11 of Algorithm 2), the agent scopes its AEP generation to the specific plan involved in the failure. We employ an *Informed*

Probabilistic Strategy (IPS), where AEP's are stochasticly chosen from a Half-normal distribution of the reverse of the actions in a plan π starting from a_f and ending at a_0 (Algorithm 2, line 6). For example, given $\pi = [a_1, a_2, a_3, a_4]$ and $a_f = a_3$, the chosen AEP will be sampled from a_1, a_2, a_3, with a_3 having the highest likelihood of being selected, and a_1 having the lowest likelihood of being selected. We assume that in most cases, the action that needs to be modified is more likely to occur closer to a_f, as opposed to other actions in π.

In *defocused passive incubation* (lines 12-15 of Algorithm 2), the agent again uses IPS where π is used for AEP generation, but 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. For example, given an environment with entity list $\mathcal{E} = [$`cup1, cup2, cover`$]$ and $\pi = [$`push(cup1), drop_object(cover)`$]$, an agent may explore `push(cup2)`.

2) Active Incubation: Parameter Variation: In the second stage of incubation, which we refer to as *active incubation*, the agent then begins evaluating candidate actions to determine their utility toward solving ϕ (line 20 of Algorithm 1). 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. That is, for any given AEP element $c(a_i, \mathcal{E}_{a_i}, p_{a_i})$, the agent executes the element in a *parameter variation mode* (Algorithm 3), in which action a_i is executed with specified entity arguments E_{a_i}, with p_{a_i} as its chosen focus parameter to vary. As a first step, the chosen parameter to vary p_{a_i} is 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 list of candidate variants, $\mathcal{P}'_{a_i}$, shown below:

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

In the case of discrete parameters (denoted $\hat{p}$), this strategy can be modified to consider two cases – in the case that the finite set of variant choices for a discrete parameter (denoted $\{\hat{p}\}$) is *less than* the interval partition value Δ_n, then the candidate variants list assumes the form $\hat{\mathcal{P}'}_{a_i} = \{\hat{p}_{a_i}\}$. In the case that $\Delta_n < |\{\hat{p}\}|$, the parameter variants list instead assumes the following form:

$$\hat{\mathcal{P}'}_{a_i} = \binom{\{\hat{p}_{a_i}\}}{\Delta_n}$$

That is, Δ_n randomly chosen elements of the discrete set are chosen for variation. In this way, both numerical and non-numerical discrete parameters can be utilized.

Candidate AEP 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 (Algorithm 1, lines 22 - 32). 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.

Algorithm 2 AEP Combination Generation

1: **procedure** GENAEPLIST($\pi, a_f, \omega, \mathcal{E}_\Sigma$)
2: $\quad\pi$: plan to be explored; a_f : failure action; ω : exploration strategy; $\mathcal{E}_\Sigma$: entity list
3: $\quad\zeta \leftarrow \emptyset$
4: $\quad\pi' \leftarrow \pi[0 : a_f]$
5: $\quad$**while** $\pi' \neq \emptyset$ **do**
6: $\quad\quad i \sim \mathcal{H}(\sigma)$: Half-normal distribution
7: $\quad\quad$**switch** ω **do**
8: $\quad\quad\quad$**case** $\omega =$ `focused`
9: $\quad\quad\quad\quad a \leftarrow \pi'[i]$
10: $\quad\quad\quad\quad \zeta \leftarrow \zeta \cup \{a\}$
11: $\quad\quad\quad\quad \pi' \leftarrow \pi' \setminus a$
12: $\quad\quad\quad$**case** $\omega =$ `defocused`
13: $\quad\quad\quad\quad \mathcal{E}' \leftarrow shuffle(\mathcal{E}_\Sigma)$
14: $\quad\quad\quad\quad$**for** $\mathcal{E}_a \subset \mathcal{E}'$ **do**
15: $\quad\quad\quad\quad\quad \zeta \leftarrow \zeta \cup \{a(\mathcal{E}_a)\}$
16: $\quad$**return** ζ

Algorithm 3 Generate Parameter Variations

procedure GETPARAMVARIANTS(c, Δ_n)
$\quad c$: AEP combination to vary
$\quad\Delta_n$: interval partition parameter
$\quad\delta \leftarrow \emptyset$: variants
$\quad$**for** $p \in \mathcal{P}_a$ **do**
$\quad\quad I \leftarrow (p_a^{upper} - p_a^{lower}) \div \Delta_n$
$\quad\quad i \leftarrow 0$
$\quad\quad$**while** $i \leq \Delta_n$ **do**
$\quad\quad\quad p' \leftarrow p_a^{lower} + (i \cdot I)$
$\quad\quad\quad a' \leftarrow a(p = p')$
$\quad\quad\quad \delta \leftarrow \delta \cup \{a'\}$
$\quad\quad\quad i = i + 1$
$\quad$**return** δ

IV. EXPERIMENTAL RESULTS

We ran our experiments using a Baxter robot simulated in Gazebo. We used a PDDL planner which required specifying an initial state (s_i) and goal state (s_g), where it could find a plan π for attaining s_g. Actions in the knowledge base were either parameterizable, or not (see Table III). For any parameterizable action a, the corresponding parameter set $\mathcal{P}_a$ of its controller m_a consists of the following parameters: $p_a^1 =$ `rate` (dictating the rate of motion of the agent during action execution), $p_a^2 =$ `orientation` (denoted `orient` dictating the orientation of the end effector of the robot in single

Experiment 1	
s_0	at(cup, loc_a) at(cover, loc_b) is_visible(cup) is_visible(cover) covered(cup)
s_g	not(at(cover, loc_a))
π	push(cover)

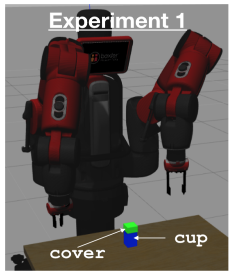

TABLE I: **(left)** Initial state s_0, goal state s_g, and resulting plan π in both the original and novel scenarios of Experiment 1 shown. **(right)** Setup shows the Baxter robot in Experiment 1 manipulation task.

degree rotation along a chosen plane, relative to the entity argument of the action), and p_a^3 = movementMagnitude (denoted MM, dictating the range of movement in the action). Any variant added $\mathcal{K}$ is named in the following format – actionName-parameterVaried:defaultSetting, e.g., push-rate:10.

A. *Performance Measures*

Execution phases refer to any event in which the robot is executing an action as a part of a plan to attain a goal. *Exploration phases* refer to any event in which the robot is executing actions toward behavior babbling. An *episode* refers to any attempt of the agent to accomplish its goal. Some episodes include an exploratory phase (where it must generate more novel actions to experiment with), whereas others do not (where the agent is using a stored candidate action to attempt to accomplish its goal). A *trial* refers to a full run of the CPS framework from start to finish, where the agent is able to accomplish its initial goal. We considered 3 performance metrics: The *action cost* per episode is the sum of the total simulation time of the execution phase and exploration phase of an episode. The *total action cost (TAC)* per trial is the sum of the total action costs of all episodes in a trial. The total *number of episodes (NOE)* for each trial indicates how many re-plan/behavior babbling cycles the robot need to perform to accomplish its goal. This accounted for all simulation time that the robot was in motion executing actions, but explicitly excluded planning time, and simulation reset time, which introduces an additional cost to the framework.

B. *Experiment 1: Simple Scenario*

The environment and problem setup of our first experiment is shown in Table I. The initial state s_0 is composed of a cup and a cover, both sitting on the table, where the cover is placed on the cup. The goal of the planning agent is to remove the cover from the cup. Given this goal state s_g = [not(at(cover loc_a))], the agent generates a plan π = [push(cover)]. The action controller of this action is shown as m_{push} in Table III. In this original scenario of Experiment 1, the agent is able to successfully accomplish its goal using π. In the novel scenario, the cover is heavier than it was in the original scenario. In this case, although the agent is able to plan toward the same goal, it fails upon execution, because the rate of motion of the push action controller (rate = 7.0) is no longer sufficient toward moving the cover.

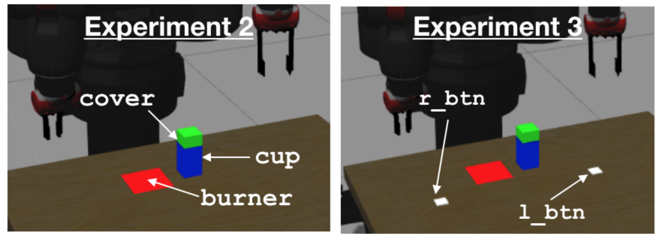

Fig. 1: Baxter robot is shown prior to manipulation of entities on the table. The blue block represents cup entity, the green block represents a cover entity, the red flat square represents burner1 entity, and the white squares represent l_btn (left button) and r_btn (right button) entities (relative to the robot).

	Experiment 2	Experiment 3
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)	at(cup, loc_a) at(cover, loc_b) at(burner, loc_c) at(l_btn, loc_d) at(r_btn, loc_e) is_visible(cup) is_visible(cover) is_visible(burner) is_visible(l_btn) is_visible(r_btn) covered(cup)
s_g	cook(cup)	cook(cup)
π	push(cover) uncover_obj(cup) shake(cup) prep_food(cup) put_on_burner(cup) cover_obj(cup) cook(cup)	push(cover) uncover_obj(cup) shake(cup) prep_food(cup) put_on_burner(cup) cover_obj(cup) cook(cup)

TABLE II: Initial state s_0, goal state s_g, and resulting plan π in both the original and novel scenarios of Experiment 1(left) and Experiment 2(right) shown.

The agent enters passive incubation, where it generates AEP list C = [c(push, [cover], rate), c(push, [cover], orient), c(push, [cover], MM)], using the focused generation method. During active incubation, the agent varies the focus parameter of each $c \in C$, attempting to find variants to add to its knowledge base, where it discovers that executing push-rate:75.0 will result in attaining s_g

C. *Experiment 2: Focused Passive Incubation*

The environment of Experiment 2 is shown in Figure 1, with its corresponding problem setup in Table II, describing the original scenario. In the novel scenario of Experiment 2, 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 incubation cycle (similar to that described in Experiment 1) 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.

D. *Experiment 3: Defocused Passive Incubation*

The environment of Experiment 3 is shown in Figure 1, with its corresponding problem setup in Table II, which describes

Parameterizable Actions				
action	param	min	max	$\mathrm{m_{action}}$
`push`	`rate`	1.0	150.0	7.0
	`orientation`	0	180	0
	`movementMagnitude`	0.01	0.3	0.13
`shake`	`rate`	1.0	150.0	7.0
	`orientation`	0	180	90
	`movementMagnitude`	0.01	0.3	0.13
Non-Parameterizable Actions				
`uncover_obj`, `cover_obj`, `prep_food`, `put_on_burner`, `cook`				

TABLE III: Actions comprising the initial knowledge base of the agent in all experiments

Exp #	# trails	aNOE	aTAC (s)
1	60	1.23	83.09
2	60	3.32	578.63
3	10	13.90	2098.71

TABLE IV: Quantitative summary data of all 3 experiments

the original scenario. In the novel scenario, unknown to the planning agent, the burner is required to be turned on by pressing the right button, in order for the `cooking(cup)` predicate to hold. In this case, a focused incubation strategy is not sufficient since the right button is not an entity present in π. Once the planning agent exhausts all AEP's generated by the focused passive incubation strategy, it attempts to find a solution through a defocused strategy. Through this exploration, the agent discovers that executing `push-orient:90` successfully actuates the button (equivalent of a `press` action), turning on the burner and thus attaining s_g. This experiment demonstrates the case described earlier where two seemingly distinct actions can be derived from one another using parameter variations.

E. Results

Summary results (Table IV) suggest that the size of the AEP list ($|C_{\mathcal{K}}|$) and length of the action plan ($|\pi|$) affect the average number of episodes (aNOE) and average total action cost (aTAC), respectively. The aNOE of Experiment 2 is 2.7 times that of Experiment 1, and the aNOE of Experiment 3 is 4.2 times that of Experiment 2. Both of these values are roughly equivalent to 1.37 times the ratio of the respective $|C_{\mathcal{K}}|$ values. The aTAC of Experiment 2 was 7 times that of Experiment 1, and the aTAC of Experiment 3 was only 4.2 times that of Experiment 2. We believe this is due to the difference in their corresponding $|\pi|$ values, since, in any episode which included an exploration phase, π must be fully executed. Experiment 1 had a significantly smaller $|\pi|$ than Experiment 2 and 3, which had equal $|\pi|$ values.

Interval Partition Value ($\mathbf{\Delta_n}$): We evaluated the effect of the Δ_n parameter on both Experiment 1 and 2. We ran 3 sets of 20 trials on each scenario, at Δ_n = 3, 5, 7, with results shown in Figure 2. It can be seen that the overall variance of the simple scenario trials are less than the complex scenario trials. This is due to their differing $|C_{\mathcal{K}}|$ sizes. With Δ_n intervals, Experiment 1 could find a success scenario within 1 to Δ_n episodes, whereas Experiment 2 could take anywhere from 1 to 6 times Δ_n episodes. This phenomenon is largely responsible for the variations seen across Δ_n values for each experiment.

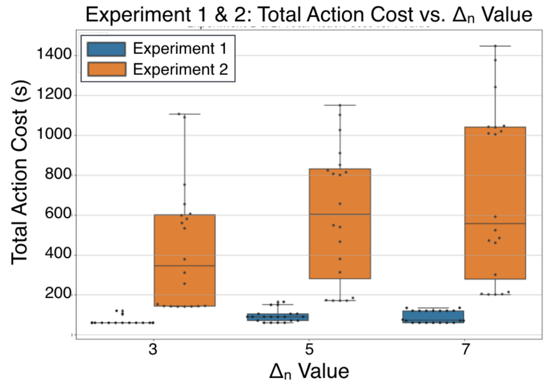

Fig. 2: Total Action Cost vs. Δ_n value for Experiment 1 & 2.

While Experiment 1 shows only a slight increase in variance, Experiment 2 shows a much more significant increase.

Success Action Discovery: Next, 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. Lastly, in a preliminary trial for Experiment 1, there was one case where the agent was unable to find a solution in its first round of focused incubation, thereby entering a second round of exploration. Here, the agent found a two-parameter variation solution, where the agent both increased its rate and movement magnitude in order to successfully push the heavy block off of the table (`push-MM:1.0-rate:3.0`). The implications of this discovery will be discussed in Section V. 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.

Defocused Incubation Case: Experiment 3 was meant to demonstrate the defocused passive incubation strategy. We ran 10 trials with a Δ_n value of 3. Experiment 3 had a significantly longer total action cost than Experiment 2, requiring more episodes for success action discovery (See Table IV). We had originally expected that the `push-orient:90` variant (equivalent to a `press` action) would be the sole success action, due to its ability to press the button to turn on the burner. Results show that in addition to `push-orient:90`, there were 4 variants of `shake` that were able to successfully actuate the button in the 10 trials. Actuation occurs in `shake` variant cases in the following manner – when the agent

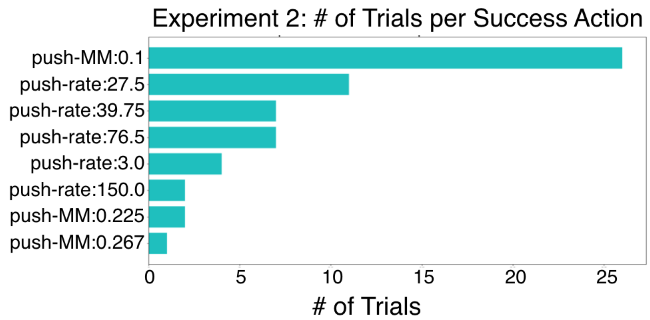

Fig. 3: Success Actions discovered in Experiment 2.

attempted to pick up the object to shake it, its grippers closed around nothing, lifted nothing, and then placed the ungrasped button back down in its place. In doing so, the agent performed the equivalent `press` action described above, pressing down on the button with closed grippers.

V. Conclusions and Future Work

We presented a novel framework for action discovery through behavior babbling, which we have demonstrated on 3 creative problem solving scenarios of varying complexities. Our framework utilizes 2 novel CPS incubation strategies, *focused* and *defocused*, which we tested in a robotics environment to demonstrate their usefulness in CPS tasks.

A limitation of this work is that it does not guarantee convergence on the action needed for a given CPS solution. It is possible that, given a selected interval partition value, a correct parameterization selection for the needed success action may be missed. Future work should consider exploration strategies for parameter selection which guarantee this convergence. An additional limitation of this work is that it relies on previously learned action controllers to execute novel actions, and thus, if CPS solutions necessitate new controllers, this method would fall short. In this case, future work should consider integrating developed methods in reinforcement learning to handle these cases [25]. Additionally, we encourage investigation into methods which differentiate context-dependant use of discovered action variants. Intelligent methods for action sequencing may include object predicate discovery, which could be used as a precondition to discovered actions. Lastly, future work should consider tractability through informed incubation search.

References

[1] A.-M. Olteţeanu, "From simple machines to eureka in four not-so-easy steps: Towards creative visuospatial intelligence," in *Fundamental Issues of Artificial Intelligence*. Springer, 2016, pp. 161–182.

[2] C. J. Völter and J. Call, "Problem solving in great apes (pan paniscus, pan troglodytes, gorilla gorilla, and pongo abelii): the effect of visual feedback," *Animal Cognition*, vol. 15, no. 5, pp. 923–936, 2012.

[3] V. Sarathy and M. Scheutz, "MacGyver problems: AI challenges for testing resourcefulness and creativity," *Advances in Cognitive Systems*, vol. 6, pp. 31–44, 2018.

[4] L. Nair, J. Balloch, and S. Chernova, "Tool MacGyvering: Tool construction using geometric reasoning," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 5837–5843.

[5] K. Takahashi, T. Ogata, S. Sugano, and G. Cheng, "Dynamic motion learning for a flexible-joint robot using active-passive motor babbling," in *The 33rd Annual Conference of the Robotics Society of Japan*, 2015, pp. 2G1–07.

[6] K. Takahashi, T. Ogata, H. Yamada, H. Tjandra, and S. Sugano, "Effective motion learning for a flexible-joint robot using motor babbling," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 2723–2728.

[7] F. Gama, M. Shcherban, M. Rolf, and M. Hoffmann, "Active exploration for body model learning through self-touch on a humanoid robot with artificial skin," in *2020 Joint IEEE 10th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*. IEEE, 2020, pp. 1–8.

[8] R. Der and G. Martius, "From motor babbling to purposive actions: Emerging self-exploration in a dynamical systems approach to early robot development," in *International Conference on Simulation of Adaptive Behavior*. Springer, 2006, pp. 406–421.

[9] R. Saegusa, G. Metta, G. Sandini, and S. Sakka, "Active motor babbling for sensorimotor learning," in *2008 IEEE International Conference on Robotics and Biomimetics*. IEEE, 2009, pp. 794–799.

[10] Y. Demiris and A. Dearden, "From motor babbling to hierarchical learning by imitation: a robot developmental pathway," in *5th International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*. Lund University Cognitive Studies, 2005, pp. 31–37.

[11] Y. Demiris and B. Khadhouri, "Hierarchical attentive multiple models for execution and recognition of actions," *Robotics and autonomous systems*, vol. 54, no. 5, pp. 361–369, 2006.

[12] Y. Nagai and K. J. Rohlfing, "Computational analysis of motionese toward scaffolding robot action learning," *IEEE Transactions on Autonomous Mental Development*, vol. 1, no. 1, pp. 44–54, 2009.

[13] E. Ugur, Y. Nagai, E. Sahin, and E. Oztop, "Staged development of robot skills: Behavior formation, affordance learning and imitation with motionese," *IEEE Transactions on Autonomous Mental Development*, vol. 7, no. 2, pp. 119–139, 2015.

[14] Y. Nagai, "From bottom-up visual attention to robot action learning," in *2009 IEEE 8th International Conference on Development and Learning*. IEEE, 2009, pp. 1–6.

[15] T. R. Colin, T. Belpaeme, A. Cangelosi, and N. Hemion, "Hierarchical reinforcement learning as creative problem solving," *Robotics and Autonomous Systems*, vol. 86, pp. 196–206, 2016.

[16] A. Lieto, F. Perrone, G. L. Pozzato, and E. Chiodino, "Beyond subgoaling: A dynamic knowledge generation framework for creative problem solving in cognitive architectures," *Cognitive Systems Research*, vol. 58, pp. 305–316, 2019.

[17] J. D. Kralik, T. Mao, Z. Cheng, and L. E. Ray, "Modeling incubation and restructuring for creative problem solving in robots," *Robotics and Autonomous Systems*, vol. 86, pp. 162–173, 2016.

[18] R. Chitnis, T. Silver, J. Tenenbaum, L. P. Kaelbling, and T. Lozano-Perez, "Glib: Exploration via goal-literal babbling for lifted operator learning," *arXiv preprint arXiv:2001.08299*, 2020.

[19] T. Silver, R. Chitnis, J. Tenenbaum, L. P. Kaelbling, and T. Lozano-Pérez, "Learning symbolic operators for task and motion planning," *arXiv preprint arXiv:2103.00589*, 2021.

[20] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, "Emergent tool use from multi-agent autocurricula," *arXiv preprint arXiv:1909.07528*, 2019.

[21] C. Yang, X. Lan, H. Zhang, and N. Zheng, "Autonomous tool construction with gated graph neural network," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 9708–9714.

[22] E. Gizzi, M. G. Castro, and J. 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)*. IEEE, 2019, pp. 228–233.

[23] V. Sarathy, "Real world problem-solving," *Frontiers in human neuroscience*, vol. 12, p. 261, 2018.

[24] E. Gizzi, L. Nair, J. Sinapov, and S. Chernova, "From computational creativity to creative problem solving agents," in *11th International Conference on Computational Creativity*, 2020, pp. 370–373.

[25] M. Eppe, P. D. Nguyen, and S. Wermter, "From semantics to execution: Integrating action planning with reinforcement learning for robotic causal problem-solving," *Frontiers in Robotics and AI*, vol. 6, p. 123, 2019.