BIPLEX: Creative Problem-Solving by Planning for Experimentation

Vasanth Sarathy[1*] and **Matthias Scheutz**[2]
[1]SIFT, LLC
[2]Tufts University
vsarathy@sift.net, matthias.scheutz@tufts.edu

Abstract

Creative problem-solving in humans often involves real-world experimentation and observation of outcomes that then leads to the discovery of solutions or possibly further experiments. Yet, most work on creative problem-solving in AI has focused on solely mental processes like variants of search and reasoning for finding solutions. In this position paper, we propose a novel algorithmic framework called BIPLEX that is closer to how humans solve problems creatively in that it involves hypothesis generation, experimentation, and outcome observation as part of the search for solutions. We introduce BIPLEX through various examples in a baking domain that demonstrate important features of the framework, including its representation of objects in terms of properties, as well its ability to interleave planning for experimentation and outcome evaluation when execution impasses are detected, which can lead to novel solution paths. We argue that these features are essentially required for solving problems that cannot be solved by search alone and thus most existing creative problem-solving approaches.

Introduction

Suppose you need to tighten a screw but do not have a screwdriver. Among the available objects you can use are a coin and pliers. After short reflection, you grip the coin with the pliers turning the "pliers-cum-coin" combination into a makeshift screwdriver. Psychologists call this "creative problem-solving" (Maier 1930).

While this solution might have been easy for you, it would have been much more difficult for a novice or a child, and practically impossible for current state-of-the-art AI systems. There several reasons for this difficulty. For one, an agent would need to determine the relevant properties of a screwdriver that makes it the appropriate tool to operate on screws: that it has a flat tip that fits into the screw's slot, and that the fit is tight enough so that if the screw driver were rotated, the screw would rotate with it, in addition to noting that a rotational downward movement is needed with a certain amount of force and that the screwdriver provide a handle that makes gripping it and applying the force easier. This analysis of the properties of the screwdriver can then guide the search for objects and their properties that could be used as a substitute. While one important *insight* is to notice that the coin has the property of fitting tightly into the slot, it does not have the property of providing enough of a grip to apply the necessary forces to generate the required rotational movement. The additional *insight* then is that the pliers which has a much better handle can be used to grip the coin tightly and thus establish a rigid connection so that when the pliers is turned and push downward, the coin is equally turned and pushed downward, turning the screw.

From the example it should be apparent why current systems cannot perform such feats: they would have to integrate detailed perception and affordance inferences with common sense knowledge, property-based analysis and planning, hypothetical reasoning and simulation, and potentially experimentation and on-the-fly learning (e.g., how to best grip the coin). While providing a system that can do all of the above is unrealistic at this point, we can nevertheless make progress by focusing on important constituents of such as system. This is what we set out to do for this position paper, namely take a fresh look a planning for crafting objects like make-shift tools based on object properties.

We will introduce a novel planner called BIPLEX which has several important properties for this type of creative problem-solving: (1) it represents objects in terms of their properties and affordances, and can thus (2) handle unknown objects to the extent that it can capture them in terms of known properties, and most importantly, it can plan to craft novel objects based on property requirements. We demonstrate how BIPLEX can handle goals involving the crafting of novel objects based on property specifications. While for most planners a direct comparison to BIPLEX is not possible (because they cannot handle properties or unknown objects), we sketch out how even for goals that regular planners like Fast-Forward (FF) (Hoffmann 2001) can handle, BIPLEX can lead to significant speedups. Finally, we conclude by discussing some limitations of BIPLEX as well as future research questions in creative problem-solving.

Background and Related Work

Problem solving in humans typically involves deliberate and conscious processing that advances a solution step by step. Insight, on the other hand, is believed to involve a "sudden" and unexpected emergence of an obvious solution or strategy sometimes accompanied by an affective "Aha!" experience which is why solvers often find it difficult to consciously explain how they generated a solution in a sequential manner. MacGregor et al. proposed the

Criterion for Satisfactory Progress Theory (CSPT), which is based on Newell and Simon's original notion of problem solving as being a heuristic search through the problem space (MacGregor, Ormerod, and Chronicle 2001). The key aspect of CSPT is that the solver is continually monitoring their progress with some set of criteria. Impasses arise when there is a criterion failure, at which point the solver tries non-maximal but promising states. Ohlsson et al.'s *Representational Change Theory* (RCT), on the other hand, suggests that impasses occur when the goal state is not reachable from an initial problem representation (which may have been generated through unconscious spreading activation) (Ohlsson 1992). To overcome an impasse, the solver needs to restructure the problem representation through (1) elaboration (noticing new features of a problem), (2) reencoding (fixing mistaken or incomplete representations of the problem), and (3) changing constraints (believed to involve two sub-processes of constraint relaxation and chunk-decomposition). Ollinger's extended RCT is a dynamic and iterative or recursive process that involves repeated instances of search, impasse and representational change (Oellinger, Jones, and Knoblich 2014; Oellinger et al. 2017): a solver first forms a problem representation and begins searching for solutions; when an impasse is encountered because no solution can be found, the solver must restructure or change the problem representation and once again search for a solution, thus combining heuristic searches, hill climbing and progress monitoring with creative mechanisms of constraint relaxation and chunk decomposition to enable restructuring.

Another related theory of creative problem solving views insight as the retrieval of an analogy from long-term memory using spreading activation (Langley and Jones 1988). This view depends on having sufficient prior experience and thus knowledge encoded such that an analogical mapping can be established.

Different from the above proposals, we are making the core claim that creative problem solving is not just a mental exercise, one that can be approached with "searching" some problem space alone, but that is essentially involves experimentation when impasses due to knowledge limitations are reached (e.g., when no plan for goal accomplishment can be found or when the execution of a plan fails with no explanation for what went wrong). Only through formulating hypotheses that lead to new experiments and observation of the outcome of these experiments is it possible for an agent to augment its knowledge base with novel facts about objects, their properties, and their affordances, which, in turn can be used by the planner to find different routes to the goal states.

The view that creative problem solving requires the extension of one's concepts has be echoed recently by (Gizzi et al. 2020) who define creative problem solving as "the process by which the agent discovers new concepts that were not in the initial conceptual space of the agent, allowing it to accomplish a previously impossible goal." Yet, their proposal only involves combinatorial methods (combining existing concepts), transformational methods (changing conceptual representations), and exploratory methods (searching the concept space). As we will show below, these methods are insufficient for discovering solutions to even simple problems without experimentation.

(Freedman et al. 2020) use analogical reasoning to find a substitution for a missing resource that is needed in a plan that accomplishes the goal. However, all knowledge about potential resource substitution candidates needs to be provided to the planner in order to accomplish the reasoning, there is no discovery of potential resource candidates involved. Analogical reasoning is solely used to find candidates that are close in property structure. In contrast, our approach hypothesizes potential substitutions based on common properties (which could be based on analogical reasoning as well) and devises experiments to test whether they, in fact, have the desired properties. Consequently, it does not require advance knowledge about all possible substitutions but it can discover them.

The approach closest to our approach of planning experiments to learn more about the domain is (Lamanna et al. 2021) which attempts to learn the actions and thus operators in a planning domain under certain assumptions about the structure of pre- and post-conditions in a deterministic setting. The algorithm starts with a set of given operators but without any knowledge about their pre- and post-conditions and systematically performs experiments to determine the exact conditions. This is done by initially starting with a set of pre-conditions consisting of all atoms that can be formed using a set of variables and predicates and an empty set of post-conditions for all operators, observing the outcome of performing an applicable operator in the current state and updating the pre- and post-conditions based on what is true in the predecessor and the successor states. However, their learning algorithm does not deal with transformative actions that change the objects in the planning domain, nor does it deal with property representations of objects in pre- or post-conditions, and hence cannot learn object substitutions in actions as it requires constants denoting all objects ahead of time. Moreover, since the goal is to learn the complete domain model, it does so without any particular preference for a given operator or particular state; all that matters is that the planner pick a state to explore that could potentially refine the pre- and post-conditions and as long as such states exist that the planner can get to using a finite state machine model of the domain it has learned so far that has a state size exponential in the number of facts, i.e., grounded predicates – clearly that latter makes this approach intractable for domains with a large number of objects. In contrast, our approach explores only those actions that could advance its goal to produce objects that are not available and thus does is not impacted by large numbers of objects in the domain.

Another related experimentation system implemented on a robot focuses on finding matching objects that can be used for creating tools based on a set of given reference tools (Nair et al. 2019). While this approach does not discover the tool per se or infers properties of tools needed, it demonstrates how an embodied agent could use its perceptual apparatus to determine properties of objects that are similar enough to desired properties, use those objects to assemble tools and evaluate those tools on actions that use them.

As such, our planner would be synergistic with the robot system in that it would provide the robot with tool properties that the robot can then assemble and evaluate, i.e., the robot would function as the experimenter carrying out the actions proposed by our planner in order to determine whether the resulting objects have the desired properties. More recently, there have been advances, in the robotics domain, for finding object substitutions based on properties (Fitzgerald, Goel, and Thomaz 2021). This work proposes the use of constraints (tool shape, segments, and visual features) that will be useful in exploring and evaluating other potential tool candidates. The approach suggests the discovery of new tools, however, what is missing then is building on this knowledge to compose and construct new tools using available resources.

The general idea of learning domain knowledge through experimentation is not new, and certainly not being claimed as such in this paper. Early work in the symbolic AI literature explored how agents can adjust and improve their symbolic models through experimentation. Gill proposed a method for learning by experimentation in which the agent can improve its domain knowledge by finding missing operators (Gil 1994). The agent is able design experiments at the symbolic level based on observing the symbolic fluent states and comparing against an operator's preconditions and effects. Other approaches (to name a few: (Shen 1989; Shen and Simon 1989; Mitchell, Keller, and Kedar-Cabelli 1986; Yang, Wu, and Jiang 2007; Aineto, Jiménez, and Onaindia 2019; Cresswell, McCluskey, and West 2013; Hogg, Kuter, and Munoz-Avila 2010; Sussman 1973)), comprising a significant body of literature, have explored learning from experimentation, recovering from planning failures, refining domain models, and open-world planning. Recently, (Musliner et al. 2021) proposed a planning system capable of hypothesis generation, model-modification and evaluation using a library of domain-independent heuristics useful to help agents accommodate novelties in the environment. What is missing from these approaches is a solution for handling transformative actions (like object construction or crafting) where new object types are generated during execution, which are then needed to solve the problem at hand. As we saw in the coin-plier example presented earlier, and as we will demonstrate in the rest of the paper, selecting object substitutions, composing them together to generate entirely new types of objects and reasoning about these compositions to re-plan and revise execution is needed for creative problem solving.

Introducing BIPLEX

In this section we introduce our novel BIPLEX (**bi**nd types, **pl**an and **ex**ecute) approach to creative problem-solving.[1] Rather than starting with abstract principles, we will motivate its core ideas and ways to solve problems by working through examples in a baking domain. We selected the baking domain because it is a familiar domain with a combination of navigation, manipulation and transformative actions involving mixing items to produce new products, thus allowing ample room for creativity through creating novel products and resourceful substitution of unavailable items. Tracing through the operation of BIPLEX then will show the synergistic effects of (1) open-world property-based planning and (2) experimentation through plan execution with subsequent (3) rule learning, plan refinement and replanning based on the outcome of hypothesized experiments. In addition to highlighting the involved principles, we will also point the technically interested reader to place in the pseudocode that accomplish the various steps in the process.

We start with a departure from the classical planning framework in how we represent objects by allowing object definitions in terms of functional and material properties (the reason for this will become clear in a bit). In the baking domain this means that we will describe the various baking ingredients such as eggs, egg whites, sugar, flour, etc. as well as utensils such as bowls, spoons, frying pans, etc. in terms of their defining properties (but critically without attempting to give sufficient conditions, only necessary ones for the baking domain). For example, we might describe "egg yolks" as "yellow and liquid and containing fatty acids" and "egg whites" as "transparent and liquid and containing protein" (leaving out the color as they are transparent).[2] Similarly, we might describe actions in terms of pre-conditions ensuring applicability, action signature (i.e., the action together with its arguments), and the expected post-conditions when execution succeeds.[3]

Now consider a goal for the agent to make *egg batter*, an important step in pancake-making. Egg batter, in our environment is a class of objects of type **`yftl`**.[4] Unlike most classical planners, BIPLEX allows for specifying lifted-goals[5] of the form (`have ?x-`**`yftl`**), with variable and type declaration. Such a goal can be satisfied if there exists a literal in the current state that evaluates to true, where the literal's name unifies to the name of the goal and the literals argument is a constant that has the same type or subtype[6] as that specified in the goal. Most classical planners require a grounded goal, which means, the agent would need to instantiate the constant associated with a desired type, which, in turn, means the agent would need to, apriori, instantiate all the types it might be able to craft, in case, it will later need to plan to craft one of those. This becomes intractable very quickly, once one realizes that in many real-world settings, there are many instances of each object type – many tomatoes, many teaspoons of sugar, etc. We will explore various cases, each of increasing difficulty to describe some of the capabilities of BIPLEX .

[1] BIPLEX has been fully implemented and the code can be found at: https://github.com/vasanthsarathy/biplex

[2] We can represent each property with a single character, e.g., "y" for yellow and "t" for protein and "f" for fatty acids

[3] This is generally how domains are represented in classical AI planning.

[4] A "type" in our formulation is merely a sequence of characters, each specifying a property.

[5] "lifted" here means goals that do not have grounded constants in their terms, but instead have variables

[6] Subtypes might be represented as a type hierarchy provided to planners to facilitate variable binding

Problem-Solving with Transformative Actions

Consider the goal, (have *?x*-**yftl**) corresponding to egg batter. Let us assume that the BIPLEX agent has all the resources it needs to make egg batter **yftl**. That is, it has the domain specification[7] that contains several navigation and manipulation actions along with several transformative actions like the following (variables start with a "?" and italicized, types are in bold, action names are in bold and italic):

```
(:action mixeggbatter1
:parameters (?y - yftl ?x1 - yfl ?x2 - tl ?x3 - wl
    ?z - rmc)
:precondition (and (have ?x1) (have ?x2) (have ?x3)
    (have ?z))
:effect (and  (have ?y)  (not (have ?x1))
    (not (have ?x2)) (not (have ?x3)) ))
```

This action ***mixeggbatter1*** requires the agent to have ingredients of type **yfl** (egg yolk), **tl** (egg white), **wl** (milk), and an object of **rmc** (bowl) to mix the ingredients. At the completion of then action, the ingredients are consumed (except the bowl) and the agent is left with an egg batter object **yftl**. This action is a "transformative action" in the sense that a new object type is created and some ingredients are consumed and cease to exist as objects in the problem space. The agent may need to sequence a plan to find and collect these ingredients and therefore must interleave navigation and manipulation with mixing or crafting. To achieve this goal with a state-of-the-art classical planner (like FF (Hoffmann 2001)), we would need to instantiate all constructable objects, which in this case includes generating a constant symbol for the egg batter. An FF agent would need to also ground the goal to this constant object, and then proceed to generating a complete plan before it can begin any execution. The FF agent, thus begins with an domain specification and a problem specification[8] with all possible constants of all possible types and a grounded goal. In most classical offline planning scenarios, execution does not begin until the FF agent has ground all the variables from all the types, completed planning and produced a plan.

Like the FF agent, the BIPLEX agent is also given a domain (action schema) specification containing navigation, manipulation and transformative actions. However, unlike the FF agent, the BIPLEX agent is not given a complete problem specification. Instead, we embed the agent in an environment allowing it to plan and execute in an interleaved manner. The agent scans the environment and observes objects present in the current initial state along with their respective types. Note, the agent cannot observe objects with types that have not yet been generated by transformative actions. So, initially, the BIPLEX agent does not observe `eggbatter1` as it does not yet exist. In addition to a domain specification, the BIPLEX agent is also provided a lifted goal (have *?x*-**yftl**).

From the domain specification, the BIPLEX agent generates (1) a stripped-down domain specification, and (2) a tree with nodes representing inputs and outputs of the transformative action and nodes representing the name of the transformative actions. The stripped-down action schema contains all the transformative action schemas, but stripped-down to only contain their output types, any preconditions, and any types that the agent believes will not be transformed by the action. For example, the ***mixeggbatter1*** will be stripped-down to ***hyp-mixeggbatter1***[9] :

```
(:action hyp-mixeggbatter1
:parameters (?y - yftl ?z - rmc)
:precondition (and (have ?z))
:effect (and  (have ?y) )
```

The BIPLEX agent first grounds the lifted-goal with a gensym, a hypothetical constant symbol (e.g., (have hyp-yftl-a9b88541)), and then generates a problem specification based on information it can observe from the initial state. It then attempts to generate a "plan sketch" using the stripped-down domain specification and problem specification (line 12, Algorithm 1). The agent can do this at very low computational cost as the stripped transformative actions have far fewer parameters and no preconditions. Moreover, using the stripped-down domain allows BIPLEX to reason about transformative actions that have ingredients that themselves are products of other transformative actions. Without any preconditions, BIPLEX can essentially generate a list of resources and intermediate products it will need, a shopping list of sorts, if you will. If the plan sketch does not contain any stripped-down actions or any hypothetical constant symbol names, the BIPLEX agent simply executes the plan (lines 14-16, Algorithm 1). This occurs when the actions are primarily navigational or involve manipulation. If, however, the plan contains hypothetical symbols or transformative actions, it will turn to the crafting tree to construct the types it needs (lines 25-35, Algorithm 1). Instead of having to *apriori* generate all the constants for all possible types as is required for FF, BIPLEX only generates symbols for types as and when it needs them. At the crafting tree, it finds the type that it needs to construct, then traverses the crafting tree to find the associated transformative action and its linked input ingredients.

BIPLEX uses two operations – **PROVE** (Algorithm 2) and **GROUND** (Algorithm 3) – to bind types to ground constants as it traverses the crafting tree. In our running example, the BIPLEX agent will find a plan-sketch (Algorithm 1): (***pickup*** bowl1)[10] and then (***hyp-mixeggbatter1*** hyp-yftl-a9b88541 bowl1).[11] This two-step plan-sketch cannot be executed as there are hypothetical constants as well as stripped-down actions. The agent will attempt to "prove" the existence of an object of type **yftl** by "grounding" any action that is a predecessor to type **yftl** in the crafting tree, namely any action that constructs **yftl** (line 2, Algorithm2). In this example, we

[7] This is a PDDL 1.2 representation of an action schema usually contained in a `domain.pddl` file that is provided to a planner.

[8] Also provided to the planner as a `problem.pddl` file

[9] We adopt the convention of representing stripped-down action names with the prefix "hyp-"

[10] We use LISP notation as is customary in AI planning for representing actions with action name followed by parameters within a list

[11] As part of Stanford's MOLGEN project, Mark Stefik echoed the idea of generating relevant "skeletal plans and then refining them" (Stefik 1981).

only have one action ***mixeggbatter1***, and so the agent will attempt to "ground" it. Grounding an action in the crafting tree involves (recursively) proving all of its predecessor types (lines 2-10, Algorithm 3). Here, the action has several predecessor types (which are the input ingredients to the mix action) including **yfl** (egg yolk), **wl** (milk) and **tl** (egg white). For each of these resources, BIPLEX will interleave planning and execution to first acquire milk by planning and executing (***gofromto*** `bowl1 milk2`), (***pickup*** `milk2`), and then planning and executing (***gofromto*** `milk2 yolk2`), (***pickup*** `yolk2`), and then (***gofromto*** `yolk2 eggwhite2`), (***pickup*** `eggwhite2`). As it acquires each ingredient it proves the corresponding type node in the crafting tree. Upon proving all the predecessor nodes of the action, the BIPLEX agent then is ready to perform the transformative mix action itself (lines 11-26, Algorithm 3). To do so, it generates a new domain specification containing the navigation and manipulation actions along with a single, fully specified transformative action, which in this case is ***mixeggbatter1***. The agent generates a new planning problem specification with the goal of (`have ?x-`**yftl**). Remember, the agent still needs a bowl, which it will acquire next. Upon completion of this plan, a new constant appears in the agent's observations (`eggbatter1`) and several other constants disappear. We perform some bookkeeping to replace the hypothetical symbol for eggbatter with the one from the environment, as well as removing objects that were consumed. We maintain a strict separation of the agent and the environment, so the agent does not know about an object unless it is perceivable in the environment.

The approach of breaking down the problem is more intuitive, allowing for not only easier debugging, but also significantly faster performance. This is the case, because each planning instance during grounding only contains a single transformative action along with other navigational and manipulation actions. Moreover, only those types are instantiated that are needed per the crafting tree.

Next we will explore the benefit of representing classes of objects or types as a conjunction of properties. We have alluded to this earlier that creative problem-solving involves reasoning about object properties themselves. We will next discuss how this reasoning process can be computationalized.

Creative Problem-Solving with Hypothesis Generation and Testing

Thus far, we have shown that BIPLEX can solve problems that classical planners can also solve, but BIPLEX solves them faster and is able to handle transformative actions as well as lifted specifications more naturally. We now turn to discussing how BIPLEX is also creative. Continuing with our running example, consider what happens if there is no egg yolk. Classical planning systems like FF would fail as no plan exists given the domain and problem – without yolk, the agent cannot make egg batter. Using findings from human problem-solving we propose that the agent should consider other objects with similar, possibly overlapping properties. We operationalize this creative mechanism as follows.

In addition to **PROVE** and **GROUND**, BIPLEX agents have the ability to **PLAY** that is, "hypothesize" and "test" resource substitution ideas. At the core of this capability is the ability to track object properties. The key idea here is that the BIPLEX agent is able to compose available types to generate ideas for new novel types. The agent does not know, apriori, if these combinations will work or even if relevant properties will persist after combination. The best an agent can do is assume that certain properties exist and experiment and make discoveries. Based on the result of the experiment, the agent is able to gain additional knowledge with which to derive improved future hypotheses. We will now walk through an example of how the BIPLEX agent accomplishes this task.

First, consider simpler goal of having a egg yolk (`have ?x-`**yfl**) when there isn't any available in the environment. Moreover, there is no transformative action available to make egg yolk. As noted earlier, BIPLEX will first try to generate a plan-sketch using the stripped-down domain specification (Algorithm 1). However, the planner will fail, as we might expect. Upon failure, BIPLEX will enter "creative mode" and first attempt to hypothesize different ways it can compose together objects (lines 18-24, Algorithm 1). BIPLEX does this by generating a set of available interesting types. This is a set of types, where each type (which is represented as a conjunction of properties) intersects with the target type **yftl** in terms of properties. Thus, if the environment contained the following objects: milk **wl**, water **l**, yogurt **wftl**, applesauce **yf** then these would all be intersecting types as they contain one or more properties that are also properties in our target type **yftl**. BIPLEX then generates a power set of these intersecting types (line 1, Algorithm 4). Each element of the power set represents a possible composition of multiple types that, in theory, could be combined to generate the target type. BIPLEX also filters this power set to only include those elements that when combined possess all the properties of the target type. Thus, yogurt and applesauce together have at least the properties y-f-l, and so are included. However, yogurt and water together do not have the property "y", so this composition is not included in the power set. This filtered power set is a complete set of hypotheses available to the BIPLEX agent to experiment with.

For each hypothesis in the set of candidate hypotheses, BIPLEX generates a novel (generic) action (let's call it ***mix1***) that is intended to serve as a generic template with which to allow the agent to try to mix objects. not sure how to do this generally. The agent uses this template to generate a domain and problem specifications to allow it to plan and execute the particular experiment (lines 10-12, Algorithm 4). Upon completion, BIPLEX reviews the new state and compares it against the previous state. It declares the experiment a success if the new state contains an object of a novel type that did not exist in the prior state. To compare states, the agent inspects the type signature (conjunction of properties) of this novel type to ensure that it possesses all the desired properties of its target type **yfl**.[12] If so, the hypothesis-testing

[12]It is worth noting that we are assuming that the agent can observe all the types be it directly or indirectly via inference or measurement instruments.

phase is declared complete, and the agent returns to generating plan-sketches followed by planning, crafting and execution as described earlier. When the agent discovers the new type, it does not yet know if this will be all it needs for the rest of the plan. If it encounters another impasse (say a missing resource), it will revisit hypothesis-testing then to troubleshoot and make new discoveries. Now, if after performing the experiment, the agent does not observe a state change (i.e., no new types were constructed), the agent will try the next hypothesis. If none of the hypotheses work, the agent will give up and end problem solving. It is conceivable, although we have not implemented this capability, this may be when and where the agent will need to discover or consider a new property of the environment. Human creative problem-solving involves this capability, and often solutions to insight problems are found when a property is found to be relevant, but previously thought to be irrelevant (Sarathy 2018; Sarathy and Scheutz 2018). The classic example of this is the three-bulb insight problem in which the solver must discover an unexpected property of the light bulb to solve it.

Thus, in our example of having egg yolk, the agent found 79 hypotheses, tested two of them before it found one that works. First it considered combining yogurt **wftl** and oil **ly**. While these two when combined possess all the properties of egg yolk **yfl**, when tested, there is no observed state change. The reason for this is that in our testing environment, we did not encode any new type for combining these two elements. The agent was not told this apriori, but instead discovered it during hypothesis testing. The next hypothesis the agent tried was to combine yogurt **wftl** with applesauce **yf**. When tested, the BIPLEX agent discovered that it produces a new type, **ylft**, which is really a wet batter without any eggs. The agent declares success here because **ylft** contains the properties "y", "f" and "l", which was what defined egg yolk, the target type. Now, clearly this is not the same as egg yolk and it might seem a bit strange to call this a success, however, it is worth noting that in this example, we did not equip the environment with any dynamics for making egg-yolk. So, the agent found the best answer with the resources it had on hand. What is particularly interesting about this new type **ylft** is that it contains the same properties (in a different order) as egg batter. The agent might have discovered this as a substitute for egg batter if it were planning to make pancakes and not just egg yolks, is that this new type allows it to skip the step of making egg batter and move to the other steps of pancake making. This would simplify the agent's planning and execution as it would not need to seek out and acquire all the other ingredients for making egg batter as it already has what it needs, a discovery it made while making something else, namely egg-yolk. We next discuss how the agent might creatively make egg batter with no egg-yolk if it did not make this discovery.

Consider the goal of having egg batter (`have ?x-`**yftl**), as we discussed previously. Unlike when we trying to make egg yolk, here, the initial attempt at generating plan-sketch will not fail. The agent will generate a plan-sketch: (***pickup*** `bowl1`), (***hyp-mixeggbatter1*** `hyp-yftl-ac7adcdf`). Realizing it needs to construct an object of type **yftl**, it will search the crafting tree. The crafting tree indeed has a node for egg batter, which it will traverse and identify the ingredients needed, the first of which is milk **wl**. The agent will then plan and execute to go and get milk. Once it does that the next ingredient for the agent is to acquire egg yolk **yfl**. As we mentioned above, since there is no egg yolk, the agent will enter creative mode, generate hypotheses (79), try and experiment until it finds one that works. Let's say the one it finds is combining water **l** and applesauce **yf** to thereby generate diluted applesauce **lyf**. At this point, we know that diluted applesauce is not the same as egg yolk and is only potentially a viable substitute for it in egg batter. But, the agent marches on and continues the process of making egg batter and plans and acquires egg whites **tl**. Once it has "proved" all the ingredients or predecessors to the action node ***mixeggbatter1***, it is ready to "ground" the action, and "prove" the type **yftl** for egg batter. However, this plan will fail because it cannot make egg batter with diluted apple sauce. At this point, the BIPLEX agent once again enters creative mode to find a substitute for egg batter. Amongst the many hypotheses, one that works is one that requires the use of the newly created diluted apple sauce to make non-egg egg batter **ylft**. The agent then declares success in making a substitute and ends problem solving. Again, this may not be a viable substitute for downstream use, but in this particular instance of making no-egg-yolk-pancakes, this substitute works just fine.

Thus far we have discussed creative resource substitution, when an agent is attempting to acquire a known type, say egg batter or egg yolk. These are types listed in the agent's own domain, and types for which the agent has domain knowledge about the types of actions that can performed with them and the fluents that apply to them. However, the approach underlying BIPLEX can be used to pursue previously unknown goals. For example, if the agent is tasked with making diluted applesauce **lyf** or even making no-egg-yolk egg batter **ylft** or no-egg-yolk egg pancake **vaftuy**. The planning, execution, constructing, hypothesis generation and testing proceeds just as described before.

It should now be clear why we need to use properties to represent object types instead of using those types themselves, or at the very least properties in additions to types; for without explicit property representation we would not be able to formulate the kinds of hypotheses we need in order to handle novel situation and learn substitutions through experimentation. It will also not be able to find plans that ultimately require the creation of novel objects or substances.

Discussion and Limitations

As we discussed earlier, most existing work, particularly in AI, approaches creativity and creative problem-solving as a mental search problem for possibilities and associations. This view, however, neglects the critical role of real-world experimentation and its potential for gaining insight from observations during the experimentation process. As (Sarathy 2018) has noted, there is evidence in human neuroscientific studies that the environment, and the human's interaction with the environment is crucial to the creative

Algorithm 1: SKETCH()

Input: $goals$ {global variable}
Input: $completed$ {global variable}
while $\exists$ $goals$ **do**
  $goal \leftarrow goals$.pop()
  if $goal \in s_0$ **then**
    $completed$.append($goal$)
    return True, s_0
  end if
  if $goal \in completed$ **then**
    return True, s_0
  end if
  $objects \leftarrow$ observe-objects()
  $\mathcal{P} \leftarrow$ construct-problem($goal, s_0, objects$)
  $\pi \leftarrow$ plan($\Sigma^*, \mathcal{P}$)
  $\mathcal{O}^* \leftarrow$ is-executable(π)
  if $\pi \neq \emptyset$ and $\mathcal{O}^* = \emptyset$ **then**
    completed.append($goal$)
    return execute(π)
  end if
  if $\pi = \emptyset$ **then**
    $status, s_1 \leftarrow$ play($goal$)
    if $status =$ False **then**
      return False, s_0
    end if
    return True, s_1
  end if
  if $\mathcal{O}^* \neq \emptyset$ **then**
    $objects \leftarrow$ get-objects-to-construct($\mathcal{O}^*$)
    for $object \in objects$ **do**
      $type \leftarrow$ get-type($object$)
      $status, s_1 \leftarrow$ **PROVE**($type$)
      if $status =$ False **then**
        return False, s_0
      end if
    end for
    return True, s_1
  end if
  return False, s_0
end while
return True

Algorithm 2: PROVE($type$)

Input: $type$ {An object type}
Input: $tree$ {global variable}
$s_0 \leftarrow$ observe()
$actions \leftarrow$ get-predecessors($tree$, $type$)
if $actions = \emptyset$ **then**
  return False, s_0
end if
while $actions \neq \emptyset$ **do**
  $action \leftarrow actions$.pop()
  $status, s_1 \leftarrow$ **GROUND**($action$)
  if $status =$ True **then**
    return True, s_1
  end if
end while
return False, s_0

Algorithm 3: GROUND($action$)

Input: $action$ {An action name}
Input: $tree$ {global variable}
Input: $grounded$ {global variable}
$types \leftarrow$ get-predecessors($tree, action$)
for $type$ in $types$ **do**
  $s_0 \leftarrow$ observe()
  $goal \leftarrow$ construct-goal-literal($type$)
  $goals$.push($goal$)
  $status, s_1 \leftarrow$ **SKETCH**()
  if $status =$ False **then**
    return False, s_0
  end if
end for
if $action \notin grounded$ **then**
  $grounded$.append($action$)
  $\Sigma \leftarrow$ create-domain($\Sigma^-, action, tree$)
  $type \leftarrow$ get-successor$action, tree$
  $goal \leftarrow$ construct-goal-literal($type$)
  $status, s_1 \leftarrow$ plan-and-execute($goal, \Sigma$)
  if $status =$ True **then**
    $goals$.push($goal$)
    return **SKETCH**()
  end if
  $status, s_1 \leftarrow$ **PLAY**($goal$)
  if $status =$ True **then**
    return True, s_1
  end if
  return False, s_0
end if
return True, s_1

process. For new knowledge needs to enter the mind somewhere and we claim that this interaction with environment must be interleaved with mental search and reasoning processes. Moreover, there is a mutually-supportive aspect of this relationship between environmental experimentation and mental search, whereby when the agent notices something interesting in the environment, it may trigger new mental search and reasoning processes that can lead to new experiments to further understand the observation, and so on.

The BIPLEX approach to planning is thus different from most existing planning approaches that rely solely on grounding action schemas to every object provided in a problem instance and finding a path in the search space to the goal. Although BIPLEX calls a planner as part of its operation, the planning is limited to small instances enabling BIPLEX to handle large problem instances with a large number of objects. Creativity in the real-world involves the ability to reason about large number of possible objects, ill-defined problem statements and partial knowledge. These real-world conditions prevent standard planners from being easily applied in creative settings. Here, we suggest that BIPLEX can serve as a "wrapper" over fast state-of-the-art planners, one that can handle these real-world constraints more effectively. Moreover, it is unclear how BIPLEX should decide which properties are relevant to a particular problem instance. BIPLEX was designed with "fluidity" in mind, which can be seen in how it dynamically defines and redefines new problem instances as it approaches its goal. This

Algorithm 4: PLAY($goal$)

Input: $goal$

$combinations \leftarrow$ get-intersecting-properties($goal$)
$hypotheses \leftarrow \emptyset$
for $comb \in combinations$ **do**
  if $goal$.type $\subseteq comb$ **then**
    $tree \leftarrow$ make-tree($comb$, $goal$)
    $hypotheses$.append($tree$)
  end if
end for
$s_0 \leftarrow$ observe()
for $hypo$ in $hypotheses$ **do**
  $\Sigma \leftarrow$ create-domain(Σ^{-},generic,$hypo$)
  $status, s_1 \leftarrow$ plan-and-execute($goal, \Sigma$)
  if $status =$ True **then**
    if $s_0 \overset{\text{type}}{=} s_1$ **then**
      continue
    else
      $completed$.append($goal$)
      return True, s_1
    end if
  end if
end for
return False, s_1

fluidity, however, has many dimensions beyond what we have already shown: from selection of properties, granularity of representations, and selection of relevant objects to consider. In future work, we intend to study other dimensions of fluidity as we believe it can help lower complexity by reducing the particular problem instances for an underlying planner.

A limitation of the current approach is that the agent tries each hypothesis independent of any prior attempts. It is reasonable to expect that the agent should "learn" from each experiment and only try hypotheses that are likely to produce new information. (Lamanna et al. 2021) discuss the value of this idea in their online planner that learns a planning domain during execution. One avenue for future work is to consider how the agent could learn from an experience and use this knowledge in a different problem. We are not advocating that creative problem-solving be fully-unguided exploration. Instead, BIPLEX relies on a predefined set of relevant properties over which the object types are defined, and the agent itself is goal-directed, in that we have provided a planning goal that they agent must reach. We intend to explore, in future work, different strategies for guided experimentation, hypothesis generation, and representations for learned and acquired knowledge.

Another limitation is that BIPLEX executes each viable hypothesis until it finds one that works. However, there may be other constraints (such as cost, time, availability of materials, normative and ethical ones) that limit what is doable and acceptable to experiment with for the agent. In some cases it may thus be prudent to put a human in the loop and allow BIPLEX to suggest new ideas and give the human final decision-making authority as to whether to test a hypothesis or not, especially if it is likely that the human or even another agent might already know what will happen. In collaborative settings, the agent may essentially be able to eliminate hypotheses without even trying them.

BIPLEX generates hypotheses when it must construct novel types or when it needs to find a substitute to a known type and it declares success in its hypothesis testing when it finds a type that has all the desired properties of its target type. At this point, BIPLEX does not know if the newly found type will work for the rest of its plan. It is thus possible that somewhere downstream an action fails as a result of not having found the right substitute here. In such a case, BIPLEX should backtrack and revisit other successful hypotheses, which is currently not implemented.

Finally, BIPLEX assumes, during hypothesis generation, that a target type must be formed with a combination of other types that have intersecting properties. However, it is possible that non-intersecting types produce novel types. Take for example, combining salt and ice: salt is of type solid-granular-white and ice is of type solid-cold. When combined, salt melts the ice to produce water which is not a solid, does not contain granules, is not white and presumably is less cold than ice. So, if the target object is water of type clear-liquid, BIPLEX would not combine these two. Clearly, the possibility of combining known types to yield potentially novel types is an important direction for future work as it will allow the agent to expand its conceptual basis (and, of course, is it yet another interesting question of how the agent would be able to recognize the new type).

Conclusion

In this position we introduced the BIPLEX framework, a novel approach for creative problem-solving that uses property-based representations of objects to enable planning with transformative actions and object substitution in a tightly integrated hypothesis generation, experimentation, observation, and adaptation loop. By planning experiments and performing them to utilizing observations of their outcomes in the planning process, BIPLEX offers an approach for problem solving that goes beyond mere search-based methods and more closely mimics the process of scientific discovery which essentially involves experiments in the real world to try out theoretical predictions and confirm or reject hypotheses. In a next steps, we plan to evaluate the performance of BIPLEX and compare it to state-of-the-art planners to demonstrate that it can better handle large numbers of objects due to its property-based representations and that it can solve problems that other planners cannot solve due its ability to perform experiments.

Author Contributions

Vasanth Sarathy and Matthias Scheutz jointly developed and wrote the ideas and algorithms in this paper.

Acknowledgements

This work was funded by DARPA grant W911NF-20-2-0006.

References

Aineto, D.; Jiménez, S.; and Onaindia, E. 2019. Learning strips action models with classical planning. *arXiv preprint arXiv:1903.01153*.

Cresswell, S. N.; McCluskey, T. L.; and West, M. M. 2013. Acquiring planning domain models using locm. *Knowledge Engineering Review* 28(2):195–213.

Fitzgerald, T.; Goel, A.; and Thomaz, A. 2021. Modeling and learning constraints for creative tool use. *Frontiers in Robotics and AI* 8.

Freedman, R.; Friedman, S.; Musliner, D.; and Pelican, M. 2020. Creative problem solving through automated planning and analogy. In *AAAI Workshop on Generalization in Planning*.

Gil, Y. 1994. Learning by experimentation: Incremental refinement of incomplete planning domains. In *Machine Learning Proceedings 1994*. Elsevier. 87–95.

Gizzi, E.; Nair, L.; Sinapov, J.; and Chernova, S. 2020. From computational creativity to creative problem solving agents. In *Proceedings of the 11th International Conference on Computational Creativity (ICCC'20)*.

Hoffmann, J. 2001. Ff: The fast-forward planning system. *AI magazine* 22(3):57–57.

Hogg, C.; Kuter, U.; and Munoz-Avila, H. 2010. Learning methods to generate good plans: Integrating htn learning and reinforcement learning. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*. Citeseer.

Lamanna, L.; Saetti, A.; Serafini, L.; Gerevini, A.; and Traverso, P. 2021. Online learning of action models for pddl planning. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*.

Langley, P., and Jones, R. 1988. A computational model of scientific insight. In Sternberg, R. J., ed., *The Nature of Creativity: Contemporary Psychological Perspectives*. New York, NY: Cambridge University Press. 177–201.

MacGregor, J. N.; Ormerod, T. C.; and Chronicle, E. P. 2001. Information processing and insight: a process model of performance on the nine-dot and related problems. *Journal of Experimental Psychology: Learning, Memory, and Cognition*.

Maier, N. R. 1930. Reasoning in humans. i. on direction. *Journal of comparative Psychology* 10(2):115.

Mitchell, T. M.; Keller, R. M.; and Kedar-Cabelli, S. T. 1986. Explanation-based generalization: A unifying view. *Machine learning* 1(1):47–80.

Musliner, D. J.; Pelican, M. J.; McLure, M.; Johnston, S.; Freedman, R. G.; and Knutson, C. 2021. Openmind: Planning and adapting in domains with novelty. In *Proceedings of the Ninth Annual Conference on Advances in Cognitive Systems*.

Nair, L.; Shrivastav, N.; Erickson, Z.; and Chernova, S. 2019. Autonomous tool construction using part shape and attachment prediction. In *Proceedings of Robotics: Science and Systems*.

Oellinger, M.; Fedor, A.; Brodt, S.; and Szathmary, E. 2017. Insight into the ten-penny problem: guiding search by constraints and maximization. *Psychological Research* 81(5):925–938.

Oellinger, M.; Jones, G.; and Knoblich, G. 2014. The dynamics of search, impasse, and representational change provide a coherent explanation of difficulty in the nine-dot problem. *Psychological research* 78(2):266–275.

Ohlsson, S. 1992. Information-processing explanations of insight and related phenomena. *Advances in the Psychology of Thinking* 1–44.

Sarathy, V., and Scheutz, M. 2018. Macgyver problems: Ai challenges for testing resourcefulness and creativity. *Advances in Cognitive Systems* 6:31–44.

Sarathy, V. 2018. Real world problem-solving. *Frontiers in human neuroscience* 12:261.

Shen, W.-M., and Simon, H. A. 1989. Rule creation and rule learning through environmental exploration. In *IJCAI*, 675–680. Citeseer.

Shen, W.-M. 1989. *Learning from the environment based on percepts and actions*. Ph.D. Dissertation, Carnegie Mellon University.

Stefik, M. 1981. Planning with constraints (molgen: Part 1). *Artificial intelligence* 16(2):111–139.

Sussman, G. J. 1973. A computational model of skill acquisition. Technical report.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted max-sat. *Artificial Intelligence* 171(2-3):107–143.