Semantics emerge from solving problems given abstract prior

Abstract

Evidences from cognitive and developmental psychology support that people solve unseen problems with the help of high-level strategies according to prior knowledge that is agnostic to problem context. Specifically, we argue that people construct such strategies by assigning semantics to the elements in the problem to make connection with prior experience. Such assignment can be viewed as Bayesian inference given goals and constraints as prior, which implies the diversity and the convergence of strategies used by people. To understand and model the capability of people, we propose the ProbSol Worlds (ProbSol) environment—the world is driven by the same dynamics but can be configured as different tasks, such as tool use, causal inference, and sketching. Equipped with magnetism-based dynamics, ProbSol alleviates the confounding variables of prior semantics brought by conventional physically-grounded problem solving tasks. Hence, the environment-agnostic prior of goals and constraints can be disentangled from problem solving. We show the potential of ProbSol for carrying out large-scale behavioral studies and benchmarking computational models to probe people's and machine's diverse understanding of goals and constraints in problem solving.

Introduction

People, even young children, come up with ideas to solve unseen problems before they start to solve them. Consider an experimental observation in the developmental psychology literature (Schulz et al., 2008; Chu and Schulz, 2020): two groups of the 5-year-old are asked to explore the property of a bag of unseen objects, after demonstrated with the magnetic property of seen objects called *dax* by attaching them to a steel crossbar, the two groups are told the name of the unseen objects as *tufa* and *dax* respectively—though both started by trying attaching them to the crossbar yet failed, most kids in the the *tufa* group soon changed their way to interact with the objects, such as stacking them vertically; while by contrast, most kids in the *dax* group are stuck with the same way of trying. How does such capability of *deciding how to try* come?

If we view the behavioral patterns *flexibly changing the way to explore* and *persisting on one way to explore* as two distinctive *high-level strategies*, *exploring the property of unseen objects* as the common goal, these strategies are not likely generated directly by planning over the goal—(1) in the same deterministic environment, planning should lead to one optimal trajectory; (2) the environment of the new task should be fully observable in advance because reward can only be obtained during interacting with the environment. As the only distinction between the two groups is *the name of unseen objects*, which can be viewed as the constraint over the goal, we interpret this observation as the result of generating the *high-level strategy* given the abstract representation of the goal and constraint of a task, just before starting to solve the problem. This intuition for *generating the ways to generate strategies* may come from the understanding of goals and constraints based on prior experience, which is much more accessible than the specific prior knowledge of the environment. Hence, we argue that people represent problems in an abstract space of goals and constraints without knowing the exact environmental dynamics. Given such representation, people plan in an unknown environment through top-down generative inference from the goal and the constraint, generating trial-and-error plans by assigning estimated semantics to the environment. Our perspective reflects the theory that people represent problems in a rational and conditional way in the literature (Ho et al., 2022), but takes a step beyond: people may plan over an environment-agnostic problem representation and decide how to interact with the environment according to goals and constraints.

To probe human's prior knowledge of goals and constraints in planning and to benchmark the computational

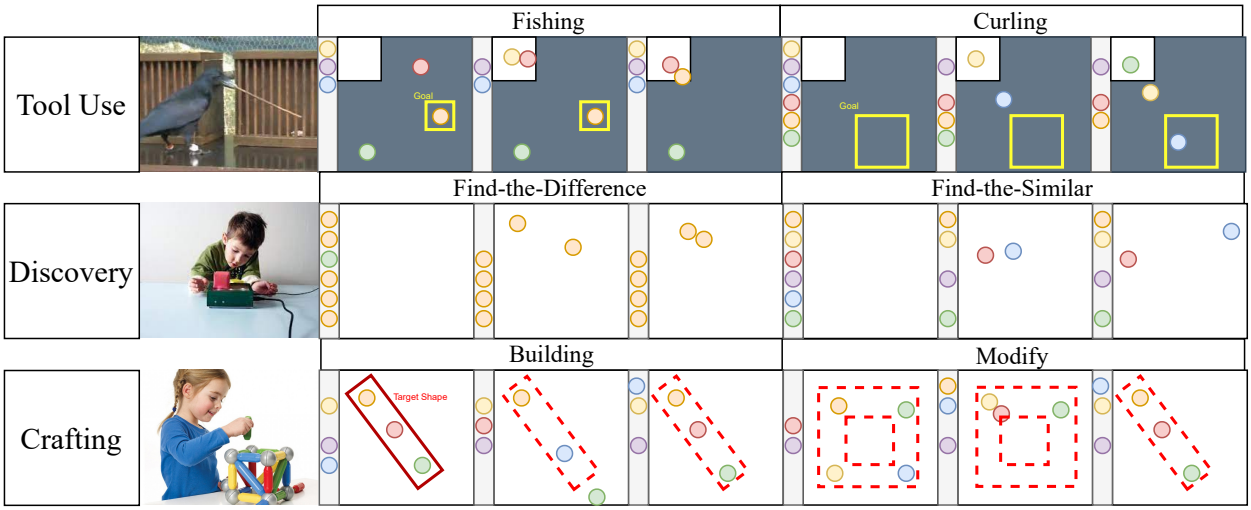

Figure 1: **Overview of the ProbSol Worlds.** The simulated environment mimics physically-grounded problems in daily life. Circles denote objects, and areas with dark backgrounds indicate the inaccessible. These environments include three problem families: (1) tool use, such as fetching targets from inaccessible areas to manipulable areas with the help of other objects, reflecting the capability of meta-tool use and planning in both human and intelligent animals (Bird and Emery, 2009); (2) discovery, such as finding the only pair of objects that share the same in property, given a set of objects, reflecting the people's capability of causal inference in human (Schulz, 2012); (3) crafting, such as arranging objects to resemble a given shape or transform a shape to another, reflecting the capability of concept abstracting and sketching in human (Fan et al., 2020). In these scenarios, magnetism plays totally different roles—helpful tool in tool use, clues and experimental materials in discovery, and harmful destroyer in crafting.

prior model of goals and constraints, we propose the **ProbSol Worlds (ProbSol)**[1] environment. ProbSol is a physically-grounded 2-D world with magnets—ones with different polarities attract each other, while ones with same polarity repulse each other—but the player never know the physical properties (polarity, mass, and maybe boundary of different polar) of the magnets until she starts to intervene and observe the world. Hence, ProbSol helps alleviate several confounding variables brought by the environment, such as semantic prior in Atari games like Montezuma's Revenge (Mnih et al., 2015), *e.g.*, fires and monsters that may be harmful, or doors and ladders that may lead to more space for exploration—such prior knowledge specifies the problem representation by making the properties of objects visible to the solvers; or spatial prior in physical problem solving environments like Phyre (Bakhtin et al., 2019), *e.g.*, the effect of gravity and elasticity that are determined by relative positions between objects—such prior knowledge specifies the problem representation by simulating the dynamics via intuitive physics (Allen et al., 2020). Instead, ProbSol provides a sandbox-game-like open-ended world environment that supports arbitrary assignment of both semantic and spatial information—different tasks are generally controlled by the same dynamics, but become diverse for manipulating the problem context. For example, one may try to exploit magnets as tools to reach goals that can never achieve by herself, *e.g.*, fishing and curling; one may play with a set of magnets to discover the differences and similarities of their properties; one may create new shapes with magnets and try to keep the shapes from being destroyed by magnetic forces (see Fig. 1 for details). Sometimes magnetism may be the tool while in other times it may be the destroyer—these semantics are top-down inferred from the goals. There are constraints over the goals: sometimes one has only a few trial times, or one must explain some properties of the objects. Building upon simple magnetism dynamics, ProbSol Worlds provides an environment that can be specified to tasks with goals and constraints.

This work aims to (1) provide the first pieces of evidence for the existence of the prior of goal and constraint for human problem solving with its richness and limitation and (2) to model the prior computationally from the first principle. Hence, we propose three hypotheses that shape the study.

The emergence of semantics for generating high-level strategies People represent unknown problems in an environment-agnostic space of the goal and the constraint as the prior model. Behavioral studies should

[1]Visit `probsol.yzhu.io` to interact with the web-based user interface.

significantly show that subjects generate *high-level strategy* only given the goal and the constraint before starting to interact with the environment, and such *high-level strategies* should to some extent converge given the same goal and the same constraints but different environmental layouts.

When do the strategies people used for solving problems become diverse, and when they converge? When people plan through top-down generative inference from the prior model of goals and constraints, they rationally trade-off between task utility and inner preference. Behavioral studies should significantly show that people tend to generate case-by-case *high-level strategies* for different goals and constraints, and the plan should converge when the constraint is critical to the goal; but people tend to generate *high-level strategies* based on their inner preferences when the constraint becomes irrelevant to the goal, and the plausible plans become diverse—the mode-shift between task utility and inner preference should reflect the rationality of planning over the prior model of goals and constraints.

Computational model for the prior of problem solving Machines equipped with the prior model of goals and constraints should behave more closely to humans (*i.e.*, predict human plans with higher accuracy) than their pure reward-based counterparts. The prior model can be latent energy-based models for describing the world compactly or large language models for predicting policy, which are prompt by goal and constraint descriptions. We should train the agents to be sensitive about environment-agnostic goals and constraints through a generative task-free process, and look into how latent variables influence agents' trade-offs over plausible strategies. To evaluate these capabilities, we should introduce a new metric that indicates behavioral consistency over goals and constraints.

Once our evidence is strong enough to accept these hypotheses, we have taken the first step towards modeling the prior knowledge of human problem solving. The contribution of our work comes with two aspects: 1) we introduce a new prior-focused perspective on problem-solving to the AI community; 2) our model limitations lead to new experiment directions for cognitive science to study human planning.

Problem Formulation

We model a *task* as an abstract version of Constrained Markov Decision Process (CMDP) (Altman, 1999) $T = (\mathcal{S}, \mathcal{A}, \mathcal{P}, g, c) \in \mathcal{T}$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ is the transition probability function which reflects the dynamics of our physical world where $s_{t+1} \sim \mathcal{P}(\cdot|s_t, a_t)$, $g = (\rho, r) \in \mathcal{G}$ is the goal of a task consisting its initial state ρ where $s_0 \sim \rho$ and reward r indicating in what condition the task is solved, and $c = \{c_{trial} \in \mathcal{C}_{trial}, c_{world} \in \mathcal{C}_{world}\}$ is the set of abstract constraint, with two kinds of constrains, chances for trials $\mathcal{C}_{trial} \in \mathbb{R}$ (measured by times of an agent restarting new episodes in a single evaluation task), and understanding of the environment $\mathcal{C}_{world} \in \mathbb{R}$ (measured by the mean accuracy in a post-hoc quiz of a single evaluation task).

Under the notation of Reinforcement Learning (RL) (Sutton and Barto, 2018), a policy π is a distribution over solutions $\tau = [s_1, a_1, s_2 \cdots, s_{-1}]$, indicating $a_t \sim \pi(\cdot|s_t)$ given an environment and a task. Here we introduce the term over-policy ξ that controls the intrinsically motivated behavior towards a solution in an abstract level, *e.g.*, the curiosity for exploration, the empowerment for controlling objects in the environment, or hacking for shortcut of the task. A policy for solving a task both depends on the task itself and the given over-policy. We denote the set of all stationary policies as Π, and denote the set of all over-policies as Ξ.

Solution rate To evaluate the solution rate Υ for a single trial, we calculate the relative distance between the last state s_{-1} of the trial and the goal g.

$$\Upsilon_{[T]} = 1 - d(s_{-1}, g) \tag{1}$$

See details for distance metric $d(\cdot, \cdot)$ in different problem classes in A.1. Please note that the solution rate does not serve as the reward in a single trial.

Constraint satisfaction rate Similarly, to evaluate the constraint satisfaction rate Ω, we calculate the relative distance between the current constraint value c^* and the target value c. Please note that only when the current solution violates the constraint, such calculation is meaningful.

$$\Omega_{[T]} = 1 - d(c^*, c) \tag{2}$$

See details for distance metric $d(\cdot, \cdot)$ in different kinds of constraints in A.2.

Formulating as a multi-objective optimization The objective for an agent to solve a task given an over-policy is to find a plausible policy $\pi \in \Pi$ that optimizes the problem

$$\begin{aligned} \pi^*|\xi = \arg\max_{\pi \in \Pi} \ & \Upsilon_{[T]}(\pi|\xi) \\ s.t. \ & \Omega_{[T]}(\pi|\xi) \leq \epsilon \end{aligned} \tag{3}$$

where ϵ is a small constant that relaxes the constraint from $\Omega_{[\mathcal{T}]}(\pi) = 0$. We transform this constrained optimization problem to a multi-objective optimization problem and solve it through finding the Pareto optimal solutions on the Pareto front (Oster and Wilson, 1978; Vieira et al., 2004). There are four considerable reasons: 1) As we are interested in modeling the prior for goals and constraints in problem-solving, explicitly optimizing constraints means providing the agent with additional in-process guidance, which is sort of deviated from our initial motivation; 2) Constrained optimization is resource-consuming, while humans solve problems with very limited computational resources and we quest machines with such merit, thus transforming the problem to one with an efficient approach to solve is plausible; 3) Also of interest is the diversity of human behavior when given a goal and a constraint. However, constrained optimization is hard to explore the solution space globally (Achiam et al., 2017). Solutions on the Pareto front model the diversity of trade-off over plausible policies well; 4) We are modeling humans' intuition on goals and constraints which should be flexible and inaccurate, thus obtaining portability by sacrificing algorithmic strictness is acceptable. Also, the monotonic optimality in reinforcement learning satisfies the prerequisite for the existence of Pareto front (Shoval et al., 2012). Hence, we have the transformed multi-objective optimization problem as

$$\pi^*|\xi = \arg\max_{\pi \in \Pi} \ \big(\Upsilon_{[T]}(\pi|\xi), \Omega_{[T]}(\pi|\xi)\big) \tag{4}$$

and we obtain two different optimal policies $\pi_1^*, \pi_2^* \in \Pi^*$ by applying different preferences for solving the objective function—one considers objective as principal and the other treats the constraint with higher priority—both the policies are Pareto Optimal with $\Upsilon_{[T]}(\pi_1^*|\xi) \geq \Upsilon_{[T]}(\pi_2^*|\xi)$ and $\Omega_{[T]}(\pi_2^*|\xi) \geq \Omega_{[T]}(\pi_1^*|\xi)$. A preference is a partial order over the objectives. When the objective and the constraint are balanced, *i.e.*, they are almost equally hard to optimize, diversity of human solutions and the trade-off over plausible solutions come from the preference order over all objective functions. Sometimes the objective and the constraint are not balanced, *e.g.*, the constraint is very relaxed or very rigid, then cognitive pragmatism drives the decision maker to put a higher priority on solving the goal or the constraint respectively. Hence, the multi-objective optimization formulation captures the two properties of human's prior for goals and constraints: 1) optimality for a specific task; 2) and trade-off over all tasks. Without loss of generality, we formulate the problem of solving all tasks as

$$\xi^* = \arg\max_{\xi \in \Xi} \ \big(\Upsilon_{[T_1]}(\xi), \Omega_{[T_1]}(\xi), \Upsilon_{[T_2]}(\xi), \Omega_{[T_2]}(\xi), \cdots, \Upsilon_{[T_{|\mathcal{T}|}]}(\xi), \Omega_{[T_{|\mathcal{T}|}]}(\xi)\big) \tag{5}$$

where $T_1 \succeq T_2 \succeq \cdots \succeq T_{|\mathcal{T}|}$ is the partial order over tasks, indicating preference over solving multiple objectives. Let $\lambda \in \Lambda : \mathcal{T} \times \{\mathbb{1}(c \succeq g)\} \mapsto \mathbb{R}^{\mathcal{T} \times 2}$ be the preference indicator that generates the partial order. The optimal over-policy set Ξ^* is the Pareto front. This also provides us with a potential interpretation from the aspect of evolution (*i.e.*, bias from tasks experienced), which may reflect the resource of humans' prior for problem-solving. Hence, we obtain two ideal properties from such setting: 1) if we could (approximately) describe $\Xi^*|\lambda$, we have the chance to generate ξ^* given a task description in advance of starting to solve the task; 2) if we could setup the space for Λ, we can model the diversity of behavior obeying the first principle. Let $f(\text{proj}_f(\lambda), \xi, g, c) : \Lambda \times \Xi \times \mathcal{G} \times \mathcal{C} \mapsto \mathbb{R}$ denote arbitrary one of the objectives, the prior space for goals

and constraints is

$$\Phi = \{f(\text{proj}_f(\lambda), \xi, g, c) | \forall \lambda \in \Lambda, \forall \xi \in \Xi, \forall g \in \mathcal{G}, \forall c \in \mathcal{C}\} \tag{6}$$

and the aim of this work is modeling this prior and study its property.

Probabilistic modeling Here we describe the computational modeling of the prior for goals and constraints. We also compare our modeling with the explicit constraint optimization counterpart. Let $\lambda, \xi, g, c \sim \Phi$ be the prior distribution for problem-solving, $\lambda, \xi \sim P(\cdot, \cdot | g, c)$ be the intuitive problem-solver that generates near-optimal policies given goals and constraints while maintaining the diversity of policies, we have

$$\begin{aligned} P(\lambda, \xi | g, c) \propto & P(\lambda, \xi) P(g, c | \lambda, \xi) \\ \propto & P(\lambda) P(\xi | \lambda) P(g, c | \lambda, \xi) \\ \propto & P(\lambda) P(\xi | \lambda) P(c | g) P(g | \lambda, \xi) \\ \text{or } \propto & P(\lambda) P(\xi | \lambda) P(g | c) P(c | \lambda, \xi) \end{aligned} \tag{7}$$

where $P(\lambda)$ is the prior model indicating preference, $P(\xi|\lambda)$ is a generator that generates the over-policy from the prior model. Likelihood $P(g, c|\lambda, \xi)$ can be factorized into a term indicating the difficulty of the task $P(g|c)$ or $P(c|g)$ and a likelihood term considering the empirical knowledge about how a prior model satisfies a goal or a constraint.

References

Achiam, J., Held, D., Tamar, A., and Abbeel, P. (2017). Constrained policy optimization. In *International conference on machine learning*, pages 22–31. PMLR. 4

Allen, K. R., Smith, K. A., and Tenenbaum, J. B. (2020). Rapid trial-and-error learning with simulation supports flexible tool use and physical reasoning. *Proceedings of the National Academy of Sciences*, 117(47):29302–29310. 2

Altman, E. (1999). *Constrained Markov decision processes: stochastic modeling*. Routledge. 3

Bakhtin, A., van der Maaten, L., Johnson, J., Gustafson, L., and Girshick, R. (2019). Phyre: A new benchmark for physical reasoning. *Advances in Neural Information Processing Systems*, 32. 2

Bird, C. D. and Emery, N. J. (2009). Insightful problem solving and creative tool modification by captive nontool-using rooks. *Proceedings of the National Academy of Sciences*, 106(25):10370–10375. 2

Chu, J. and Schulz, L. E. (2020). Play, curiosity, and cognition. *Annual Review of Developmental Psychology*, 2:317–343. 1

Fan, J. E., Hawkins, R. D., Wu, M., and Goodman, N. D. (2020). Pragmatic inference and visual abstraction enable contextual flexibility during visual communication. *Computational Brain & Behavior*, 3(1):86–101. 2

Ho, M. K., Abel, D., Correa, C. G., Littman, M. L., Cohen, J. D., and Griffiths, T. L. (2022). People construct simplified mental representations to plan. *Nature*, 606(7912):129–136. 1

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533. 2

Oster, G. F. and Wilson, E. O. (1978). *Caste and ecology in the social insects*. Princeton University Press. 4

Schulz, L. (2012). The origins of inquiry: Inductive inference and exploration in early childhood. *Trends in Cognitive Sciences*, 16(7):382–389. 2

Schulz, L. E., Standing, H. R., and Bonawitz, E. B. (2008). Word, thought, and deed: the role of object categories in children's inductive inferences and exploratory play. *Developmental psychology*, 44(5):1266. 1

Shoval, O., Sheftel, H., Shinar, G., Hart, Y., Ramote, O., Mayo, A., Dekel, E., Kavanagh, K., and Alon, U. (2012). Evolutionary trade-offs, pareto optimality, and the geometry of phenotype space. *Science*, 336(6085):1157–1160. 4

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT Press. 3

Vieira, D. A., Adriano, R. L., Vasconcelos, J. A., and Krahenbuhl, L. (2004). Treating constraints as objectives in multiobjective optimization problems using niched pareto genetic algorithm. *IEEE Transactions on Magnetics*, 40(2):1188–1191. 4

A Tasks and evaluation protocols

In this section we describe the evaluation protocols both for behavioral studies and algorithmic experiments.

A.1 Solution Rate

In this subsection we describe the distance metrics in different problem classes in detail.

- ***Fishing* in *Tool Use*.** The distance is the minimum normalized Euclidean distance between the target objects $t(o) \in t(O)$ in the environment and all the target areas $t(a) \in t(A)$ when the object target is static.

$$d(s_{-1}, g) = \begin{cases} 1, & s_{-1} = \rho \text{ or } v[t(o)] > \epsilon, \\ \dfrac{1}{|t(O)|} \displaystyle\sum_{t(o)\in t(O)} \min_{t(a)\in t(A)} \widetilde{L_2}(t(o), t(a)), & otherwise \end{cases} \tag{8}$$

  where $\epsilon > 0$ is the threshold for maximum pace of all static objects.

- ***Curling* in *Tool Use*.** The distance is the minimum normalized Euclidean distance between all static objects $o \in O$ in the environment and all the target areas $t(a) \in t(A)$.

$$d(s_{-1}, g) = \begin{cases} 1, & s_{-1} = \rho, \\ \dfrac{1}{|t(A)|} \displaystyle\sum_{t(a)\in t(A)} \min_{o\in O, t(a)\in t(A)} \max\{\widetilde{L_2}(o, t(a)), \mathbb{1}(v[o] > \epsilon)\}, & otherwise \end{cases} \tag{9}$$

  where $\epsilon > 0$ is the threshold for maximum pace of all static objects.

- ***Find-the-Diff* and *Find-the-Same* in *Discovery*.** The distance is the minimum normalized Damerau-Levenshtein distance between the id strings of the objects $\langle o|s_{-1} \in O^* \rangle$ in the environment of the last state in the environment and the answer to the task $\langle y|\mathcal{T} \rangle$.

$$d(s_{-1}, g) = \begin{cases} 1, & s_{-1} = \rho \\ \dfrac{1}{\max\limits_{x\in O^*, y\in O^*} lev(x, y)} lev(\langle o|s_{-1} \in O^* \rangle, \langle y|\mathcal{T} \rangle), \ \text{OP} = \{\text{add}, \text{sub}, \text{del}\} & otherwise \end{cases} \tag{10}$$

  Please note that this distance is defined on discrete space. The weights over the three operators add, sub (substitute), and del (delete) are 2, 1, and 2 respectively.

- ***Building* and *Modifying* in *Crafting*.** The distance is the minimum normalized Euclidean distance between all static objects $o \in O$ in the environment and all the block areas $b(a)$ in the target shape $t(S)$.

$$d(s_{-1}, g) = \begin{cases} 1, & s_{-1} = \rho, \\ \dfrac{1}{|t(S)|} \displaystyle\sum_{t(a)\in t(S)} \min_{o\in O, t(a)\in t(S)} \max\{\widetilde{L_2}(o, t(a)), \mathbb{1}(v[o] > \epsilon)\}, & otherwise \end{cases} \tag{11}$$

  where $\epsilon > 0$ is the threshold for maximum pace of all static objects.

A.2 Constraint Satisfaction Rate

In this subsection we describe the distance metrics for different kinds of constraints in detail. Please note that once the constraint is satisfied, $d = 0$. The following metrics only apply when the constraint is violated.

- ***Constraint on chances for trials.*** We normalize d by rescaling the numerical space of the constraints and calculate the current trial times' offset out of the constraint.

$$d(c^*, c) = \frac{|c^*_{trial} - c_{trial}|}{\max_c \mathcal{C}_{trial} - \min_c \mathcal{C}_{trial}} \tag{12}$$

- ***Constraint on understanding of the environment.*** We consider the post-hoc quiz as a descriminative prediction task or a generative prediction task respectively. The distance is the absolute accuracy difference $|c^*_{world} - c_{world}|$ for descrimination setting and is related to the cosine similarity between the generated dynamics and target $1 - \cos(c^*_{world}, c_{world})$ for generation setting.

B User interface of ProbSol Worlds

In this section we describe the implementation details of the ProbSol Worlds for behavioral studies.

B.1 Web version of the ProbSol Worlds

The web front end is implemented using html, css and JavaScript. We mainly uses various interfaces provided by the gamejs library to complete the basic simulation of the physical world of the ProbSol Worlds. We use Webpack to convert various static resources into a static file, reducing page requests. The core of the ProbSol web version environment is in file folder ProbSol-Worlds/ProbSol_Web/src, which cyclically operates the main loop of the game (the step(), render(), stats_updater() functions) .

Web front-end of the ProbSol Worlds provides introduction and basic gameplay. If further adjustments is required, you should modify the settings as follow. If game is running as a demo without experimenter information recording, we should disable the back-end program or log in. So, we should set variable ̈login_needd̈o false in file ProbSol-Worlds/ProbSol_Web/dist/index.html. If you need more information of solution_rate in time on the console in the browser, you should set variable under ẅhether print solution_rate or notd̈o false in file ProbSol-Worlds/ProbSol_Web/src/index.js. To collect and record the player's game information, we should enable the front-end login function and change the back-end domain name to your running back-end server in files, including ProbSol-Worlds/ProbSol_Web/dist/index.html, ProbSol-Worlds/ProbSol_Web/dist/game_entrance.html, ProbSol-Worlds/ProbSol_Web/dist/game/game.html, and ProbSol-Worlds/ProbSol_Web/src/probsol_env.js . Start Web front-end again to complete the setup.

It is noted that the current back-end of the ProbSol Worlds uses the Flask framework, a light python back-end framework, which may become a bottleneck in website maintenance and causes problems in dealing with high-concurrency scenarios. Sqlite3 is used as the database management system. The specific database design can be found in the code comments.

The backend records the user's username, email and all game information. We can view the data of all players and every episode by visiting "the front-end domain name +/info.html". We can use similar methods as info.html to obtain and then analyze data. We can simulate the player's game data in the local environment on the python version of the ProbSol Worlds to reproduce the game process.

B.2 Python version of the ProbSol Worlds

The python version of ProbSol Worlds make full use of pygame library to implement the gym standard environment suitable for various scenarios. For more specific implementation methods, you can read our code in git repository.

The python version mainly includes the gym environment of the ProbSol game, a packaged multi-task environment interface for in-depth research, a random agent, and a behavior simulation program. Gym env we build can be used for reinforcement learning training, game behavior simulation and other aspects. And

we use Adapter Pattern to implement probsol_dir_env, which is a simple packaged multi-task environment interface. We can run ProbSol with player's data. Start bhv_exp.py, copy a game data record, paste them in running console, and then we can observe a episode of the player in detail. We can update that code from the following perspective, such as the information input method and whether the intermediate frame information can be saved. Wait. We can also run ProbSol with a random agent which we provide.

B.3 Implementation Details and Showcases

In this subsection, we want to explain in detail the various executable actions and possible constraints of the ProbSol environment with specific showcases(see Fig. 2 for details). What we have discussed here applies not only to Web version but also to Python version of the ProbSol Worlds.

Before describing executable actions, We have to introduce a variety of physics-related different-but-targeted problem classes that we carefully designed. Their specific semantics should be very clear when describing their solution rate in A.1. Here we want to talk about their semantics again from the perspective of user interface.

- ***Fishing in ToolUse***. Agent should get the goal object(s) (indicated by golden bounding box) from inaccessible area to the accessible target area(s) (indicated by red boundary). The goal magnet(s) must be stopped in target area(s) before CHECK. (in the first two lines of Fig. 2)
- ***Curling in ToolUse***. Agent should send object(s) from accessible area to the inaccessible target area(s) (indicated by red boundary) and make sure there is at least one object in each target area. Magnet(s) must be stopped in target area(s) before CHECK. (in the third to fourth lines of Fig. 2)
- ***FindTheDiff in Discovery***. Agent should search for objects with different physical properties(mass, magnetism, polarity, boundary) , distinguish them from other objects that are totally the same, and leave the result objects placed in the ground and other objects stored in the side bar. The set of objects only refers to the ones initially stored in the left-side bar. (in the fifth to sixth lines of Fig. 2)
- ***FindTheSame in Discovery***. Agent should search for objects with totally same physical properties(mass, magnetism, polarity, boundary), distinguish them from other objects that different from each other, and leave the result objects placed in the ground and other objects stored in the side bar. The set of objects only refers to the ones initially stored in the left-side bar. (in the seventh to eighth lines of Fig. 2)
- ***Building in Crafting***. Agent should build the given shape (indicated by red lines) by keeping at least one object in each closed area. Agent should make sure that the shape is in balance and will never be destroyed by magnetic forces before CHECK. (in the ninth to tenth lines of Fig. 2)
- ***Modifying in Crafting***. Agent should modify the given balanced building (indicated by gray lines) to the given shape (indicated by red lines) by keeping at least one object in each closed area. Agent should make sure that the shape is in balance and will never be destroyed by magnetic forces before CHECK. (in the tenth to eleventh lines of Fig. 2)

For submitting records, the player can not refresh the web page, or quit the game midway. The agent can perform the following actions in the environment:

- ***CHECK***. By clicking button "Check" you submit your solution or require another trial. There is the same action for the Gym environment on the Python version of ProbSol Worlds.
- ***QUIT***. By clicking button "Quit" you quit this trial and submit your solution. Gamer will return to the main interface. But we did not set this action option for the Gym environment on the Python version.
- ***GRAB, PLACE, STORE***. Left-click to grab, place, store an object. Same actions option in Python version too. The area with dark background is inaccessible for any operation.
- ***HOLD***. Right-click to HOLD an object for Gamer. Agent can only hold one object at one time.

- ***MANIPULATE***. Gamer can roll the wheel of the mouse to manipulate an object for changing the direction of boundary. Agent can only manipulate a held object.

The agent is subject to the following constraints in the environment:

- ***Objective Constraint***. Objective Constraint may contains Action Time, Action Number, Trial Time, Trial Number, and Solve Time. Action Time limits the number of actions in any single trial. Action Number limits the time before you taking any single action. Trial Time is the time limitation for any single trial. Trial Number limits the chances for re-tries in a problem-solving task. Solve Time is the time limitation for a problem-solving task It is noted that all the constraints are conjunctive, which means that once agent violate one of this constraints, this agent failed. We have adapted this constraint for two environments.
- ***Concept Knowledge Constraint***. Concept Knowledge Constraint requires gamer to answer question(s) about objects with specified properties. Gamer should click buttons in the top bar to choose objects that have properties specified by the questions according to color. We only implemented Concept Knowledge Constraint on the Web version of ProbSol Worlds. Constraint may contains magnets' Mass, Magnetism, Boundary. We set four kinds of magnets' mass – mall, medium, large, and extremely large. Concept Knowledge Constraint may want gamer to choose the smallest or biggest one with mass. We set three kinds of magnets' magnetism – small, medium, and large. (Of course, there are not only magnetism but also many factors that affect magnetism in ProbSol Worlds.)The knowledge constraint may be medium magnetism. We set four kinds of magnets' boundary – horizontal, diagonalA, vertical, diagonalB. It is noted that all the questions are conjunctive. Gamer need not to find all right answers, but once gamer find a wrong one or cannot find a right one, gamer fail.
- ***Theory Knowledge Constraint***. Theory Knowledge Constraint requires gamer to select a pair of objects with specified relationship. The knowledge constraint may be the same polarity pair or the same mass by the way. Knowledge Constraint(Both Concept and Theory Knowledge Constraint) only be checked when gamer successfully finishes the task. But wrong answer leads to failing results.

We can see that in Fishing's task0, the experimenter GRAB a dark blue magnet, and PLACE it in the operable area. Under the magnetic force and friction force, the two magnets moves closer to each other. And the dark blue magnet is obviously more massive. After the purple object stops completely, experimenter CHECK to completes the Fishing task. In Curling's task0, experimenter's first attempt failed very badly, and experimenter CHECK and started another trial. After successfully curling the purple magnet into the targeted area, the experimenter GRAB the yellow magnet, STORE back to the bar on the left side. Experimenter successfully complete the Curling task after CHECK. The experimenter HOLD the orange magnet in FindTheSame's task0 to check the direction of its boundary, and randomly MANIPULATE the red magnet a few times in task1 making it rotated. In the task0 of FindTheDiff, we can see that the three magnets can be attracted around the blue magnet, while the three magnets repel each other. Because the magnet homogeneity repulses and opposite sex mutual attraction, we can easily identify that the properties of these three magnets is exactly the same. In the task1 of Building and Modifying, we can more clearly understand the complexity of the multi-magnet environment, which makes it difficult to complete the task.

C API of ProbSol Worlds environment

In this section we describe the implementation details of the ProbSol Worlds for algorithmic evaluations.

We mainly want to explain in detail the observation space, action space, action mask, the feedback information (such as reward, done information, goal description, objective constraint) that can be obtained and so on of the Gym environment of ProbSol Worlds. apart from this, we would like to briefly introduce ProbSol_Dir_Env.

The initialization of the ProbSol Gym environment needs to specify the path of the corresponding configuration file (we provide enough configuration examples for learning). Its default size of 2-D environment

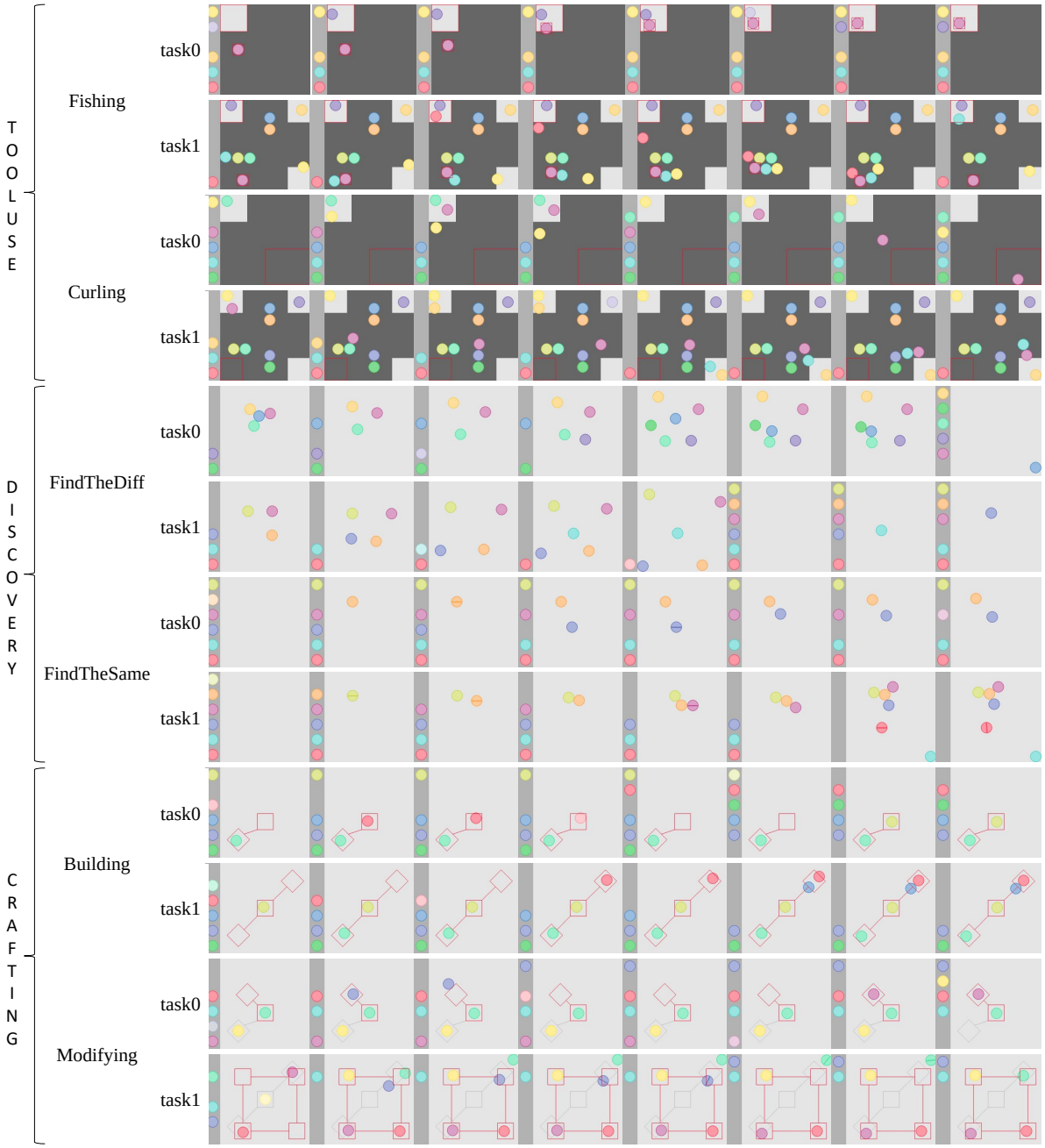

Figure 2: **Showcases of ProbSol Worlds.** We record and reproduce the actions that humans explore in the ProbSol environment to obtain our Showcases. We select the six environments of ToolUse (Fishing, Curling), Discovery (FindTheDiif, FindTheSame), and Crafting (Building, Modifying) running on the Web server. Each environment has 2 tasks. And we selected 8 screenshots of the frames during system operation for each task for showcases.

(excluding bar) is 480×480 pixels, the default surface kinetic friction coefficient $k = 20$, and the physics engine runs 60 times per millisecond defaultly.

Next, we will introduce the meaning of the interface of the API of Gym environment of ProbSol Worlds one by one.

- ***Observation Space***. The observation_space used in the Gym environment defines the characteristics of the observation space of the environment, Here we set as $spaces.Box(low = 0.0, high = 1.0, shape =$

$(max\_obj\_num * 7,), dtype = np.float32)$. The reset or step of the Gym environment can return an obs for the environment. It's a numpy row vector. Each group of seven cells describes the current state of a magnet.

Specifically, eath magnet's observations are: the state of magnet: stored, grabbed, deld, places (Bool value in Row $[0, 4)$), the position of the magnet(when placed, normalized x position in Row 4, normalized y position in Row 5), the boundary of the magnet(when held, normalized boundary in Row 6). But when agent win this game, obs is a unit row vector.

- ***Action Space***. action_space used in the Gym environment is used to define the characteristics of the action space of the environment, $Discrete(max\_obj\_num \times 3 + max\_grid\_num + 1600 \times 2 + 1 + 1 + 2 + 2 + 1 + 1)$. That is, we discretize all the actions and label them $[0, N_A)$in sequence.

  We define as follows:

  - The actions labeled $[0, max\_obj\_num)$ represent "free agent grab a stored object".
  - The actions labled $[max\_obj\_num, max\_obj\_num \times 2)$ represent "free agent grab a placed object".
  - The actions labled $[max_obj_num \times 2, max_obj_num \times 3)$represent "free agent grab a placed object".
  - The actions labled $[max\_obj\_num \times 3, max\_obj\_num \times 3 + max\_grid\_num)$ represent "grabbing agent store a grabbed object".
  - The actions labled $[max\_obj\_num \times 3 + max\_grid\_num, max\_obj\_num \times 3 + max\_grid\_num + 1600)$ represent "grabbing agent place a grab object in the discrete 2-D environment".
  - The actions labled $[max\_obj\_num \times 3 + max\_grid\_num + 1600, max\_obj\_num \times 3 + max\_grid\_num + 1600 \times 2)$ represent "grabbing agent hold a grabbed object".
  - The action labled $N_A - 6$ represents "holding agent place a held object".
  - The action labled $N_A - 5$ represents "holding agent grab a held object".
  - The actions labled $N_A - 4$ and $N_A - 3$ represent "holding agent manipulate a held object".
  - The action labled $N_A - 2$ represents None.
  - The action labled $N_A - 1$ represents CHECK.

- ***Action Mask***. Due to the multiple states of agent and object in this experimental environment, we may have a large number of actions that cannot be executed in some cases. In order to facilitate the design of agent and avoid a large number of invalid actions, we provide valid_action_mask interface, which provides a line vector with the length of N_A, where each value is a Boolean value. If it is 1, it means to perform the corresponding action; if it is zero, it cannot. Some studies have proved the working mechanism and experimental effect of invalid action masking. Experiments show that invalid action masking works well when the invalid action space is large , and the back-propagated gradient of invalid action is 0.

- ***Done Information***. It is just like traditional Gym environment, but we add more kinds of ends representation. Done Information will be returned when you execute a step.

- ***Goal Description***. We can get goal representation through call the interface of env.goal.We will get a normalized row vector with 55 columns. The first 7 columns represent the type of in a one-hot manner(six tasks above and WarmUp). Description of target rectangular areas is followed. We use the normalized coordinates of the the rectangular area's vertices to describe a target rectangular area (a total of 4×2 floating point numbers).

- ***Objective Constraint***. We can get constraint representation through call the interface of env.constraint. We will get a normalized row vector with 5 columns, which represent the normalized objective constraints.

- ***Now Constraint State***. This essentially represents the progress of the game with constraints. We can get this information by the interface of env.now_constraint_state.We highly recommend concatenating obs and now_constraint_state as input to the agent. Because when human plays the game in Web, we can clearly understand the progress of the game through the progress bar.