Human Priors in Hierarchical Program Induction

Mark K Ho* (mark_ho@berkeley.edu), **Sophia Sanborn* (sanborn@berkeley.edu)**,
Fred Callaway* (fredcallaway@berkeley.edu), **David Bourgin (ddbourgin@berkeley.edu)**,
Tom Griffiths (tom_griffiths@berkeley.edu)
Department of Psychology, UC Berkeley
Tolman Hall
Berkeley, CA 94720, USA

Abstract

People impose structure onto other agents' sequential problem-solving behavior. That is, they interpret actions in terms of a *likely program* that the observed agent was executing to solve a problem. But what prior expectations do people have about these programs? For example, in both cognitive science and computer science, *shortest description length* has been proposed as a general principle for inducing a program. However, there may be other criteria that bias how people reconstruct others' solutions: That they are symmetric, balanced, or organize child and parent processes in particular ways. Here, we report preliminary experiments and models that investigate peoples' priors on others' problem-solving programs. We first present a novel experimental paradigm in which participants were given examples of how a problem was solved and needed to reconstruct the program that generated the solution. Then, we discuss the application of our model of human program priors to these data. We find that shortest description length inadequately explains how people reconstruct others' problem solving programs.

Keywords: hierarchy; reinforcement learning; problem solving

Introduction

To solve complex problems, agents can organize their behaviors hierarchically.

Here, we explore the question of how people interpret *another's* problem-solving behavior from the perspective of Bayesian inference over programs. This view naturally raises the question of what *priors* people have over others' programs. For example, in algorithmic information theory, a minimum description length prior based on the Kolmogorov complexity of possible programs has been suggested as being optimal (Ming & Vitányi, 1990). This idea has also been proposed as an organizing principle of human cognition (Chater & Vitányi, 2003). However, there may be other criteria that guide how people interpret or "parse" a sequence of actions into a program. People may expect programs to have certain structural properties beyond just description length, such as that it observes certain types of symmetry, that the depth of the execution tree is limited, or that subprocesses are only reused in certain contexts.

Figure 1(a) shows a path in a the Lightbot game (`http://lightbot.com`). In this game, the robot needs to turn on all the lights (blue tiles) by moving around the grid. Figure 1(b-d) show the path represented as a sequence of discrete actions, and two distinct programs that produce the sequence. The first (c) is the human-written program that generated the sequence, while (d) is a algorithmically generated program consistent with the trace (i.e. the sequence of actions). Although they are both valid parsings using the same number of subprocesses, they differ in a number of ways. First, the program length, in terms of unique calls to a subprocess or ground action, is greater for the human-generated than algorithmically-generated program (33 versus 26). Second, although both exhibit symmetry and reuse of subprocesses, upon inspection they differ qualitatively. For example, the algorithmically-generated program nests shorter subroutines and calls one large subroutine (traversing a set of stairs) twice, while the human-generated program has one subroutine (move to and traverse the second set of stairs) that is only executed once. Observations about human-generated programs such as these motivate our investigation into what types of priors people bring to bear when *interpreting* action sequences as programs.

*Contributed equally to this work.

Model

Terms and notation

Formally, we model a problem as an undiscounted Markov Decision Process (MDP) M, defined by the tuple $< S, A, T, R >$: a state-space S; an action set A; a transition function that maps states and actions to next-states $T : S \times A \rightarrow S$ (for now we assume transitions are deterministic); and reward function that maps states to real values, $R : S \rightarrow \mathbb{R}$. In the Lightbot domain, the reward for turning on all the lights is 1, and otherwise 0. Additionally, once the task is solved, the environment "terminates" and no more actions can be taken.

In the context of problem-solving, we can think of a program as a sequence of instructions that solves a problem (or not). Instructions can be primitive actions that can directly modify the state of the world (e.g. `jump` can move the Lightbot agent to a new height and location in the grid). They can also be *subprocess calls* that are themselves sequences of instructions. Formally, then, we define a program π as a set of processes, $\pi = \{\sigma_0, \sigma_1, ..., \sigma_N\}$, where each subprocess is a sequence of instructions, $\sigma_i = (c_{i,0}, c_{i,1}, ..., c_{i,m_i})$. Each instruction is either a ground action or a subprocess call, and we define a special *root* subprocess, $\sigma_{\text{Root}} \in \pi$, that cannot be an instruction. By convention, we always set $\sigma_{\text{Root}} = \sigma_0$. Thus, for each j-th instruction of sub-process i, $c_{i,j} \in A \cup \pi \setminus \sigma_{\text{Root}}$.

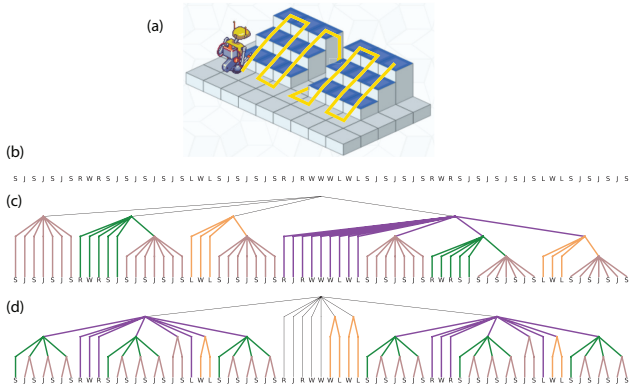

Figure 1: An action trace with multiple programs and equal subprocesses (four). (a) Lightbot domain and path in yellow. The goal of Lightbot is for the agent to turn on all the lights (blue squares) by moving around the grid. (b) Sequence of actions in the trace as a string (S = switch light on/off; J = jump forward; W = walk forward; L = turn left; R = turn right). (c) The execution tree of a human-produced program with description length of 33 that produces the sequence. Calls by a subprocess are represented by the same color. Subprocesses can call actions or other subprocesses. (d) The execution tree of an algorithmically generated program with description length 26.

A program π is a well-defined abstract object, but it only becomes meaningful once it is *executed*. A subprocess σ_i is executed by calling each instruction $c_{i,j}$ in order. If in state s, $c_{i,j}$ is a ground action a, then that action is taken and the environment transitions to the next state $s' = T(s,a)$. Otherwise, if $c_{i,j}$ is a subprocess, then that subprocess is executed. To execute a *program* π, we execute its σ_{Root} from an initial state s_0. When π is executed from s_0, it produces a unique *state-action trace*, $\zeta = (s_0, a_0, s_1, a_1, ..., s_T, a_T)$. It also yields a unique *execution tree*, an ordered tree where leaf-nodes are primitive actions in the trace ordered by time, non-leaf nodes are subprocesses, and edges indicate the parent subprocess for each instruction.

Priors for program induction

The problem of hierarchical program induction from observed behavior can be formalized as Bayesian inference over programs. An observer is given an action trace, ζ, and infers a posterior distribution over programs. This will be based on a deterministic likelihood $p(\zeta \mid \pi)$ that is zero or one depending on whether a program π produces the action trace ζ, and a prior over policies, $p(\pi)$:

$$p(\pi \mid \zeta) = \frac{p(\zeta \mid \pi)p(\pi)}{\sum_{\pi' \in \Pi} p(\zeta \mid \pi')p(\pi')}. \quad (1)$$

The denominator contains a normalizing term that is a summation over Π, the set of all possible programs. Note that because the likelihood, $p(\zeta \mid \pi)$, is deterministic, we can approximate this value by drawing samples from the set of programs

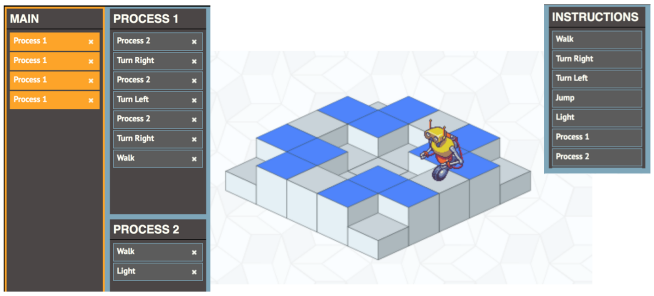

Figure 2: Basic interface for Lightbot paradigm. Participants can write simple programs for an agent whose goal is to walk around a 3D grid and turn on all the blue lights.

consistent with the trace rather than the set of all programs. However, even this constrained set is extremely large and difficult to sample from. In this work, we sample programs by partially enumerating the space of all consistent and *reasonable* programs, where a program is *reasonable* if all subroutines have at least two instructions and are called at least two times (this rules out degenerate subroutines that necessarily increase the total program length).

The goal of this work is to characterize the prior $p(\pi)$ that informs people's estimation of $p(\pi \mid \zeta)$. This prior could be represented in many ways, but here we assume that it takes the following form:

$$p(\pi; \theta) \propto \exp\{\theta^\top \phi(\pi)\}, \quad (2)$$

where $\phi(\pi)$ is a feature vector and θ is a weight vector. That is, the log prior probability of a program is a linear combination of features of that program—commonly known as a maximum entropy model (Berger, Pietra, & Pietra, 1996). Making the (strong) assumption that we have identified an appropriate ϕ, we can formulate the problem of characterizing priors over programs as the the problem of inferring θ given samples from $p(\pi \mid \zeta; \theta)$.

Program features

Under the standard Kolmogorov prior (Ming & Vitányi, 1990; Chater & Vitányi, 2003), the probability of a program is inversely proportional to the exponentiation of its length. However, people might be sensitive to additional features of a program's structure. We define three characterizations of program structure: (1) A program's *call matrix*, which captures which subprocesses are used by other subprocesses; (2) A program's *call-type matrix*, which captures the number of subprocess versus action calls made by each subprocess; and (3) the *execution depth counts*, which captures the number of times subprocesses or actions are called at different depths (starting from a depth of 0 at the root subprocess). Formally, a program's call matrix P is an $N \times N + |A|$ matrix with rows corresponding to the N subprocesses, $\sigma_0, \sigma_1, ...\sigma_N$, and the columns corresponding to the N subprocesses and $|A|$ primi-

tive actions available. $P_{i,j}$ is the number of times that subprocess i calls subprocess/action j (in a single execution). The call-type matrix T is an $N \times 2$ matrix where, for σ_i, $T_{i,0}$ is the number of subprocess calls in σ_i and $T_{i,1}$ is the number of action calls. Execution depth counts are represented by a matrix D, where entry $D_{i,j}$ is the number of times subroutine j is called at a depth i.

We can capture a number of program features with these representations. For instance, the Kolmogorov prior can be based off of the total length of the program, which is a property of the call matrix:

$$\phi_0(\pi) = \sum_{i=1}^{N} \sum_{j=1}^{N+A} P_{i,j}. \quad (3)$$

Additional features can be defined that capture aspects of program structure, such as the maximum depth of the execution tree, subprocess specialization, or relationships between child and parent processes. Table 1 summarizes the program features based on a program's structure matrices, P, T, and D, that are used in our model.

Table 1: Program Features used in MaxEnt Model. D is the execution depth counts, T is the call-type matrix, P is the call matrix, and H is entropy of a vector or set of counts ($H(\vec{x}) = -\sum_i \frac{x_i}{\bar{x}} \log \frac{x_i}{\bar{x}}$, where $\bar{x} = \sum_i x_i$).

Feature	Calculation
Tree depth	nrows(D)
Number of Subprocesses	N
Mean Subprocess Length	$\frac{1}{N}\sum_i \text{length}(\sigma_i)$
Std. Dev. of Subprocess Lengths	SD(length(σ_i))
Mean Call-type Entropy	$\sum_i H(T_{i,\cdot})$
Action-Call Entropy	$\frac{1}{N}\sum_i H(P_{i,N+1:\lvert A\rvert+N})$
Subprocess Call Entropy	$\frac{1}{N}\sum_i H(P_{i,0:N})$
Root Subprocess Length	length(σ_0)
Mean Children per Subprocess	$\frac{1}{N}\sum_{i,j} P_{i,j}$
Mean Parents per Subprocess	$\frac{1}{N+\lvert A\rvert}\sum_{i,j} P_{i,j}$
Mean Subprocess Entropy	$\frac{1}{N}\sum_i H(P_{i,0:\lvert A\rvert+N})$
Mean Action to Non-Action Ratio	$\frac{1}{N}\sum_i \frac{T_{i,1}}{T_{i,0}}$

Experiment

Our experiment had participants observe the problem-solving behavior of another agent and attempt to reconstruct the program that produced that behavior. In particular, we were interested in whether people's program induction was guided by factors other than a possible program's description length.

Paradigm

To assess what program features are more highly weighted in people's priors, we use an experimental paradigm based on the Lightbot game (`https://lightbot.com`) (Sanborn, Bourgin, Chang, & Griffiths, 2018) Figure 2. In the Lightbot game, participants write simple programs for an agent represented as a robot. The robot inhabits a 3-dimensional block world, and its goal is to turn on all the light tiles (blue) on the grid. There are 5 primitive actions (walk, jump, turn right, turn left, and toggle light). Participants can also store sequences of primitive actions as sub-processes. These sub-processes can be called by the main program, by one another, or even by themselves. These last two possibilities allow participants to write programs with simple recursion.

Method and Procedure

Mturk participants completed a tutorial involving three simple examples of the Lightbot task, including explanations of how to write and use sub-processes. They could construct programs by dragging, dropping, and deleting instructions into a program frame and could test their program at any time, which initiates an animation of the agent executing the program 2.

In the main portion of the task, participants were given one of two sets of six puzzles and solution trajectories. Participants were given a video showing an animation of the trace and could pause, rewind, or replay the trace as many times as they wanted as they constructed their own program. They were told that they would receive a bonus based on how closely the program they wrote matched the program that generated the trajectory. On each trial, once a participant wrote and ran a program that turned on all the lights, the experiment progressed to the next puzzle. They also had the option of skipping a level after six minutes without receiving a bonus.

After removing subjects who failed to complete the experiment or reported difficulties with the interface, we were left with a total of 77 subjects. We additionally constrain our analyses to only those solutions that exactly match the traces given to the participants. Due to a technical error, 47 participants were assigned to the traces from set 1 condition, and 30 to set 2.

Humans vs. randomly-sampled programs

We first compare the values of our defined features for programs written by participants and programs randomly sampled from the space of possible programs that are consistent with the given trace. We fit a mixed generalized linear model with the twelve defined features as main effects and program trace as a random effect. We find a significant difference between the human-induce programs and the randomly-sampled programs for all features ($p < .05$) except for action entropy ($\chi^2 = .12; p = .72$), and action call entropy ($\chi^2 = 3.39; p = .07$).

Figure 3 shows empirical values for randomly sampled programs and human-induced programs for six of the twelve features: total program length, tree depth, main program length, mean sub-process length, number of sub-processes, and call type entropy. Humans induce programs that are longer and have shallower trees than randomly-sampled consistent programs. Humans also write programs that have a shorter main program, use fewer, longer sub-processes, and are lower entropy in terms of call type. That is, humans show prefer-

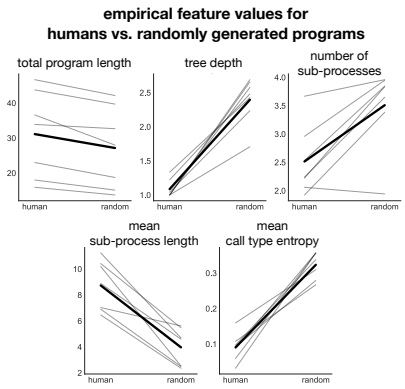

Figure 3: Empirical feature values compared to randomly generated program feature values. Bold line represents mean values while thin lines represent individual traces.

ence for offloading actions from the main program into subprocesses, inducing a compact main program that consists of repeated calls to a few sub-processes. This may align with the cognitive interpretation of the main program as a working memory space in which planning takes place and the subprocesses as stored representations—with the preference for a shorter main program according with the well-known storage constraints on working memory. It is important, however, to keep in mind that the "random" programs are constrained to be *reasonable* (defined on page 2), ruling out an infinite number of programs that have lengthy subprocesses that are never used. Thus, our findings do not conflict with previous work indicating that humans prefer to write shorter programs (Sanborn et al., 2018).

Modeling Results

We now turn to an evaluation of our full model against the Kolmogorov model that considers program length alone. We begin by fitting the model with all features to a subset of the human data by maximum likelihood estimation. The Kolmogorov model has fixed weights, -1 for length and 0 for all other features. We then evaluate each model on a held out test set of human programs. For each program in this set, we generate a randomly sampled program from the space of programs consistent with the corresponding trace. We then compute unnormalized log probabilities for the two sets (human and random) under each model. Because the scores assigned by the two models have different normalizing constants, we cannot compare them directly; however, we can still compare the ability of the models to distinguish between human and random programs using a non-parametric Wilcoxon signed-rank test. We use a one-sided test because a higher score corresponds to higher probability of human generation.

We repeated this procedure five times, training on four fifths of the data set and testing on the remaining fifth (a five-fold cross-validation). For all five folds, we find that the ranking produced by the full model produces a significant U statistic ($p < 0.05$). In contrast, the Kolmogorov model is significant in none of the folds ($p > 0.8$). Indeed, this is unsurprising given the empirical distribution of program length in human vs. random programs (Figure 3).

Discussion

In previous work, we have found that humans show a preference for solutions that are more compressible and yield shorter programs when generating solutions to Lightbot puzzles (Sanborn et al., 2018), in accordance with the normative Kolmogorov prior for minimizing program length. However, we find that humans do not simply optimize program length alone and are sensitive to other structural properties of programs. In this paper, we explicitly examine a number of such properties and propose a framework for modeling program induction as Bayesian inference over programs parameterized by such features. Empirically, we find that humans show a preference for programs that offload complexity to sub-processes, yielding shorter and simpler main programs, and show that our model provides a better fit to the human data than a model possessing a simple Kolmogorov prior alone.

In future work, we will conduct additional model comparisons to identify which of the program features we have proposed here are necessary to explain human choices. In addition, we plan to apply our model of program priors to the more unconstrained task of generating programs given a puzzle without a solution trace.

Acknowledgments

This work was supported by Air Force Office of Scientific Research grant number FA9550-18-1-0077.

References

Berger, A. L., Pietra, V. J. D., & Pietra, S. A. D. (1996). A maximum entropy approach to natural language processing. *Computational linguistics*, *22*(1), 39–71.

Chater, N., & Vitányi, P. (2003). Simplicity: A unifying principle in cognitive science? *Trends in cognitive sciences*, *7*(1), 19–22.

Ming, L., & Vitányi, P. M. (1990). Kolmogorov complexity and its applications. In *Algorithms and complexity* (pp. 187–254). Elsevier.

Sanborn, S., Bourgin, D., Chang, M., & Griffiths, T. (2018, July). Representational efficiency outweighs action efficiency in human program induction. In *Proceedings of the 40th annual cognitive science society.*