Metareasoning in Symbolic Domains

Kevin Ellis (ellisk@mit.edu) **and Owen Lewis** (olewis@mit.edu)
MIT Department of Brain and Cognitive Science, 43 Vassar Street
Cambridge, MA 02139 USA

1 Introduction

Many AI problems, such as planning, grammar learning, program induction, and theory discovery, require searching in symbolic domains. Most models perform this search by evaluating a sequence of candidate solutions, generated in order by some heuristic. Human reasoning, though, is not limited to sequential trial and error. In particular, humans are able to reason *about* what the solution to a particular problem should look like, before comparing candidates against the data. In a program synthesis task, for instance, a human might first determine that the task at hand should be solved by a tail-recursive algorithm, before filling in the algorithm's details.

Reasoning in this way about solution structure confers at least two computational advantages. First, a given structure subsumes a potentially large collection of primitive solutions, and exploiting the constraints present in the structure's definition makes it possible to evaluate the collection in substantially less time than it would take to evaluate each in turn. For example, a programmer might quickly conclude that a given algorithm cannot be implemented without recursion, without having to consider all possible non-recursive solutions. Second, it is often possible to estimate ahead of time the cost of evaluating different structures, making it possible to prioritize those that can be treated cheaply. In planning a route through an unfamiliar city, for example, one might first consider possibilities which use the subway exclusively, excluding for the moment ones that involve bus trips as well: if a successfully subway-only solution can be found, one then avoids the (potentially) exponentially more difficult bus-and-subway search problem.

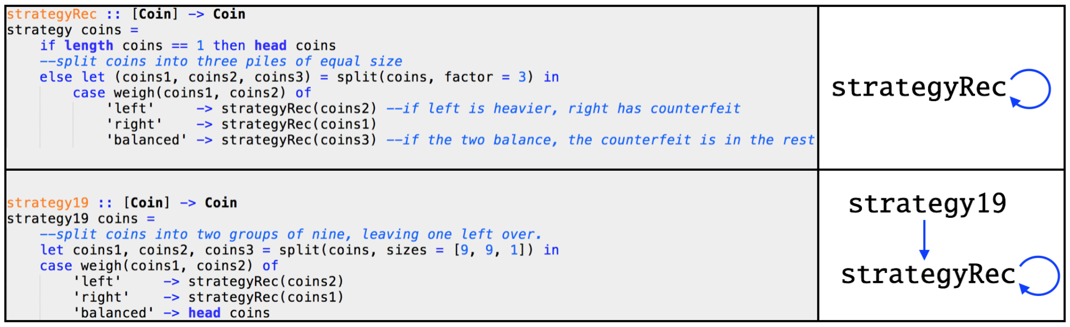
Here, we consider a family of toy problems [1], in which an agent is given a balance scale, and is required to find a lighter counterfeit coin in a collection of genuine coins using at most some prescribed number of weighings. We develop a language for expressing solution structure that places restrictions on a set of *programs*, and use recent program synthesis techniques to search for a solution, encoded as a program, subject to hypothesized constraints on the program structure.

2 Model

Our objective is to find a model, or theory, that satisfies some specification or minimizes some loss. At a high level the approach is to propose *metatheories*, which are schemas, or templates, that can be filled in to give a concrete theory. Our model has three components: (1) a language for metatheories; (2) a solver, which searches for a theory, given a metatheory, and (3) a policy, which selects metatheories to hand off to the solver.

2.1 A language for metatheories

Our goal here is to define a space of metatheories: templates that can be filled in to give a concrete testable theory. Our theories take the form of programs, so we want metatheories that restrict the structure of programs. We restrict the structure of a program by specifying



```
strategyRec :: [Coin] -> Coin
strategy coins =
    if length coins == 1 then head coins
    --split coins into three piles of equal size
    else let (coins1, coins2, coins3) = split(coins, factor = 3) in
        case weigh(coins1, coins2) of
            'left'     -> strategyRec(coins2) --if left is heavier, right has counterfeit
            'right'    -> strategyRec(coins1)
            'balanced' -> strategyRec(coins3) --if the two balance, the counterfeit is in the rest
```

```
strategy19 :: [Coin] -> Coin
strategy19 coins =
    --split coins into two groups of nine, leaving one left over.
    let coins1, coins2, coins3 = split(coins, sizes = [9, 9, 1]) in
    case weigh(coins1, coins2) of
        'left'     -> strategyRec(coins2)
        'right'    -> strategyRec(coins1)
        'balanced' -> head coins
```

Figure 2: Programs that solve coin weighing problems (left) and their call graphs, which are metatheories (right). The bottom program is one possible solution to the 19 coins in three weighings problem, and the top one solves 9 coins in two weighings, or 27 coins in three weighings.

the *call graph*, which is a directed graph whose vertices correspond to subroutines, and where there is an edge from a to b if a calls b. So a metatheory specifies all of the subroutines in a theory, as well as what subroutines can use what other subroutines.

Drawing inspiration from [2], we generate our metatheories from a grammar over directed graphs, shown in Fig. 1.

Our model automatically translates a metatheory into an input for the solver, using a template for subroutine bodies that specifies domain-specific program components. Fig. 2 gives example programs and metatheories.

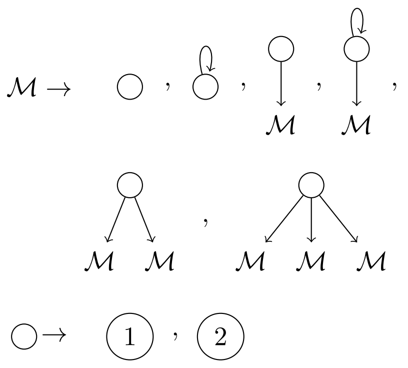


Figure 1: A grammar over metatheories. Metatheories are directed graphs generated by the $\mathcal{M}$ production; commas separate production rules. Each vertex (subroutine) has an arity (we arbitrarily consider only 1 or 2).

2.2 Solver

We use a general-purpose program synthesis tool, called Sketch [3], as our solver. The Sketch tool takes as input a partial program (referred to as a "sketch"), along with a specification. A sketch leaves some constants unspecified, leaving "holes", and Sketch can solve for the values of the holes that make the specification hold using a SAT solver. We automatically compile each metatheory into a sketch, and introduce holes that when assigned values will concretize the sketch into a specific theory. Sketch excels at discovering intricate low-level structure, but is less adept at making high-level decisions, such as how many subroutines a program should have, or what their types should be. Thus we left those decisions to our metareasoning process.

2.3 Metatheory selection policies

Given some metatheories, we need a policy that picks a promising candidate. One approach, inspired by Levin search and OOPS [4], is to try the next metatheory maximizing $P(\text{success})/\mathbb{E}(\text{search time})$ - heuristically, maximize successes per second. In this work, we did not attempt to model the probability of success, and so treat all metatheories as equally likely to succeed. So we're after the metatheory minimizing $\mathbb{E}(\text{search time})$ - the next metatheory which we can most quickly accept or reject.

We considered four metatheory selection policies. The first three minimize different proxies for search time, namely: (1) call graph size, (2) number of holes, and (3) variables in SAT problem solved by Sketch. The fourth policy uses linear regression to learn to predict search time based on problem features (the three above, plus the number of clauses in the SAT encoding.) See Fig. 3.

3 Problem domain

The family of problems we consider involves finding a counterfeit coin among a collection of genuine coins, given a balance scale and the knowledge the counterfeit coin is lighter than the genuine ones. The classic version of the problem has nine coins and two weighings, and may be solved as follows: first split the nine coins into three groups of three, and weigh the first two groups against each other. If one group is lighter, then it must contain the counterfeit coin; if they balance, the counterfeit is in the third group. Regardless, after the first weighing it remains to find the counterfeit among three genuine ones. This can be done by weighing two of them against each other: if one is lighter, it is the counterfeit, and if they balance, the held out third coin is.

We consider solutions of the following form: a subroutine takes in a collection of coins, divides them into subcollections, either by forming n equally-sized groups or by forming groups of specified sizes. It then weighs two subcollections against each other, and calls another subroutine depending on the outcome of the weighing. Fig. 2 shows two sample programs and their call graphs.

Note that the sample solution above is recursive: at each step it splits a collection into three equally-sized subcollections, and recurses into the one containing the counterfeit. This gives a hint of why our model will help: creating an arbitrary program requires choosing a large number of splits and subroutine calls, but once it is known the solution is recursive (as will be hypothesized by our metatheory search), only a few choices remain.

3.1 Results

We selected six problems within the coin weighing domain, some of whose solutions had simple recursive metatheories. We solved these problems with each policy and compared against a baseline sketch, one without any metatheory constraints. Fig. 1 gives the times to find a solution for each problem and policy.

We see that metareasoning speeds up search when the solution has some special structure (such as the recursive solution for 9 coins), and can allow the solver to discover solutions to problems otherwise outside its scope (eg, 27 coins/3 weighs). However, there is no free lunch: trying simpler metatheories first, as in 5 coins/2 weighs, hurts when the solution doesn't fit these simpler forms. But, in first proposing metatheories that it can reject quickly, it avoids catastrophic slowdowns, similar to [4] or other variants of iterative deepening. We see that the variables and learned policies do best, and note that it is exactly these that have access to the underlying constraints given to the solver. In general, metareasoners might do well to have access to features of the underlying representation sent to a backend reasoner.

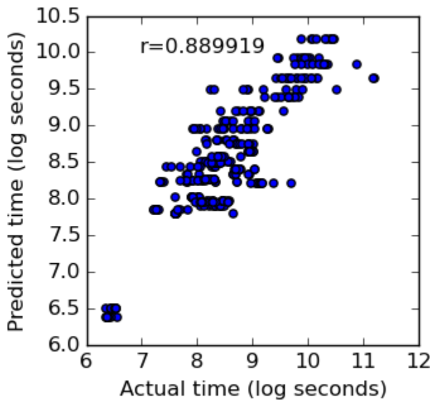


Figure 3: Predictions of our linear model for solver runtime. We regressed from problem features (# coins) and metatheory features (# holes, # clauses, # variables in the resulting sketch)

4 Related Work

The work described in [5, 6] consider theory learning problems in which inference involves an outer loop Metropolis-Hastings (MH) search over rules, and an inner loop Gibbs sampling

	baseline	graph	degrees of freedom	variables	learned
2 coins 1 weighs	0.60 sec	0.60 sec	0.60 sec	0.60 sec	0.60 sec
3 coins 1 weighs	0.65 sec	0.65 sec	0.65 sec	0.65 sec	0.65 sec
4 coins 2 weighs	4.73 sec	2.39 sec	2.39 sec	1.68 sec	2.39 sec
5 coins 2 weighs	7.57 sec	18.69 sec	18.69 sec	8.25 sec	18.69 sec
9 coins 2 weighs	29.16 sec	6.01 sec	6.01 sec	3.81 sec	3.81 sec
27 coins 3 weighs	≥ 3 hrs	253.55 sec	253.55 sec	–	–

Table 1: Solution time under different policies. Baseline: no metareasoning. Graph: try in order of call graph size. Degrees of freedom: try in order of # holes. Variables: try in order of # variables in SAT formula. Learned: use learned runtime model (see Fig. 3). Solver fails on most 27 coins/3 weighs metatheories, so we omit results for these cases.

assignment of latent predicates, roughly analogous to our metatheory search and solver. However, [5, 6] deploy Gibbs sampling on all MH proposals, whereas we use a policy to select only the most promising metatheories.

Like us, [7] models a search over algorithms, and like our learning-based policy, they use a feature-based problem representation to estimate algorithm runtime. Our model differs, in that it considers a potentially infinite collection of solutions, generated on-the-fly.Also like us, [2] considers an infinite set of over hypotheses generated by a grammars over graphs. Their focus was defining this expressive class of models, while our focus here is on metareasoning strategies for efficient inference for similarly structured spaces.

5 Open Questions and Future Work

While our metatheory language captures some important structural aspects of candidate solutions, it leaves others out. Among the first observations a human would make about the coin weighing puzzle, for instance, is that each weighing should place equal numbers of coins on the two sides of the scale. A human would be similarly quick to notice that if a weighing shows one batch of coins to be lighter than another, then the counterfeit coin must be among the light ones, and that future weighings in a recursive strategy should focus on this group. Building in these observations to our model would yield substantial speed ups, but at present there is no facility in our metalanguage for expressing such facts, and no mechanism by which the model could discover or prove them on its own.

Future work will apply this method to other domains, such as grammar induction. Like more general programs, a grammar's large-scale structure can be characterized by a call graph, in which nonterminals take the place of subprocedures, meaning that our approach could be applied more or less out of the box. Fig. 4 shows the call graph for an example grammar. Another cognitively interesting domain is theory learning. Many theories are naturally expressed as logic programs, which have strong formal similarities to grammars, and which, like grammars, can be assigned a call graph.

$$\begin{array}{rcl} S & \rightarrow & S\,a \\ & | & X \\ X & \rightarrow & S\,b \\ & | & c \end{array}$$

$$\begin{tikzcd} S \arrow[loop above] \arrow[d, bend left] \\ X \arrow[u, bend left] \end{tikzcd}$$

Figure 4: A grammar and its metatheory

In grammar and program induction problems, there are infinitely many solutions that match the training data, and a prior is required to choose among them. Thus, solving a grammar or program induction requires optimizing a posterior distribution over solutions rather then simply finding a single acceptable solution as in the coin weighing puzzle. This consideration gives an additional criterion for selecting metatheories: a good metatheory should tend to lead to primitive solutions with high prior probability.

Acknowledgments: This material is based upon work supported by the Center for Brains, Minds and Machines (CBMM), funded by NSF STC award CCF-1231216.

References

[1] Richard K Guy and Richard J Nowakowski. Coin-weighing problems. *American Mathematical Monthly*, pages 164–167, 1995.

[2] Charles Kemp and Joshua B Tenenbaum. The discovery of structural form. *Proceedings of the National Academy of Sciences*, 105(31):10687–10692, 2008.

[3] Armando Solar-Lezama. *Program synthesis by sketching*. ProQuest, 2008.

[4] Jürgen Schmidhuber. Optimal ordered problem solver. *Machine Learning*, 54(3):211–254, 2004.

[5] Tomer D Ullman, Noah D Goodman, and Joshua B Tenenbaum. Theory learning as stochastic search in the language of thought. *Cognitive Development*, 27(4):455–480, 2012.

[6] Yarden Katz, Noah D Goodman, Kristian Kersting, Charles Kemp, and Joshua B Tenenbaum. Modeling semantic cognition as logical dimensionality reduction. In *Proceedings of thirtieth annual meeting of the cognitive science society*, 2008.

[7] Falk Lieder, Dillon Plunkett, Jessica B Hamrick, Stuart J Russell, Nicholas Hay, and Thomas Griffiths. Algorithm selection by rational metareasoning as a model of human strategy selection. In *Advances in Neural Information Processing Systems*, pages 2870–2878, 2014.