Sequential Monte Carlo Program Synthesis with Refinement Proposals

Anonymous Author(s)

1 Introduction

Given a set of input-output examples specifying the behavior of a function, the problem of *inductive synthesis* is to generate a (deterministic) program producing the observed outputs on the given inputs. For example, presented with input-output examples like $[4, 8, 3, 12, 2] \to [12]$ and $[2, 5, 3, 2] \to [5]$, a synthesizer might produce a program representing the rule *index into the input list using the first element of the list as an index.* The challenge in synthesis arises from searching over a combinatorially large space of programs, specified by a grammar – classic techniques often seek to either *prune* portions of this space that are provably incorrect or *guide* the search to prioritize more promising portions of the space.

Recent work [1] suggests synthesizing deterministic programs by instead searching over a space of *probabilistic* programs. The idea is that the marginal likelihood that a probabilistic program produces the observed input-output examples can be used as a measure of how close that probabilistic program is to the deterministic solution. A probabilistic program that accurately captures the high-level structure of a solution will assign a higher likelihood to the examples. This program can then be iteratively refined, gradually replacing probabilistic components with deterministic structure until a complete deterministic program is found. [1] presents preliminary evidence for the promise of this idea, walking through an example coarse-to-fine trajectory in detail, but doesn't instantiate the idea in a search algorithm.

In this abstract, we seek to develop and evaluate synthesis algorithms that draw on this insight along with recent advances in efficient evaluation of exact marginal likelihood [2]. We first explore whether MCMC over a space of probabilistic programs aids in the synthesis of deterministic programs, finding that while in the long run this outperforms baselines, the cost of likelihood evaluation is steep and fails to pay off when given a shorter search budget.

To close this gap, we develop a sequential Monte Carlo (SMC) algorithm based on the idea of refinements from [1], implement this approach in Julia, and find that it solves more problems faster than a range of MCMC-based approaches.

2 MCMC Approaches to Program Synthesis

We consider the problem of synthesizing programs from the deterministic core of the grammar given in Listing 2.1 (*excluding* the probabilistic primitives shown in ***purple***) to solve tasks from the list manipulation dataset of [8], which contains problems like the one in the intro. Each problem has 10 input-output examples, where where each input and output is a list of natural numbers.

Listing 2.1 All primitive operations in the grammar, with probabilistic ones shown in ***purple***

$$
\begin{array}{lcl}
\mathit{nat} & ::= & 0 \mid \cdots \mid 9 \mid x \mid \textbf{randnat(0.5)} \mid \\
 & & \mathsf{index}(\mathit{nat}, \mathit{list}) \mid \mathsf{car\_or}(\mathit{list}, \mathit{nat}) \mid \\
 & & \mathsf{length}(\mathit{list}) \mid \textbf{if } \mathit{bool} \textbf{ then } \mathit{nat}_1 \textbf{ else } \mathit{nat}_2 \\
\mathit{list} & ::= & \mathsf{nil} \mid x \mid \textbf{randlist(0.5, 0.5)} \mid \\
 & & \mathsf{cons}(\mathit{nat}, \mathit{list}) \mid \mathsf{cdr\_or}(\mathit{list}_1, \mathit{list}_2) \mid \\
 & & \mathsf{mapi}((\lambda x_x x_i . \mathit{nat}), \mathit{list}) \mid \\
 & & \mathsf{filteri}((\lambda x_x x_i . \mathit{bool}), \mathit{list}) \mid \\
 & & \mathsf{append\_one}(\mathit{list}, \mathit{nat}) \mid \mathsf{append}(\mathit{list}_1, \mathit{list}_2) \mid \\
 & & \textbf{if } \mathit{bool} \textbf{ then } \mathit{list}_1 \textbf{ else } \mathit{list}_2 \\
\mathit{bool} & ::= & \mathsf{true} \mid \mathsf{false} \mid x \mid \textbf{flip(0.5)} \mid \\
 & & \mathit{nat}_1 = \mathit{nat}_2 \mid \mathit{nat}_1 > \mathit{nat}_2 \mid \\
 & & \textbf{if } \mathit{bool}_1 \textbf{ then } \mathit{bool}_2 \textbf{ else } \mathit{bool}_3
\end{array}
$$

We can formulate this as a Bayesian inference problem. Starting from a PCFG prior over the deterministic core of the grammar in Listing 2.1, the goal is to synthesize programs that have a high likelihood of generating the 10 observed input-output examples for the task. When approaching this through MCMC, we use a Metropolis-Hastings kernel that uniformly chooses an existing subexpression to resample from the prior as is common in the MCMC-based synthesis literature [2, 4, 5, 9, 12]. Next, we need to pick how to evaluate the likelihood of our data under a program.

Sparse-MCMC: While the observed input-output examples should ultimately have a likelihood of one under a deterministic solution, a noise-free likelihood that simply marks all other incorrect programs as having some fixed likelihood l would offer very little guidance in the search. Such an approach would have to stumble across a solution through blind luck.

Edit-Dist-MCMC: To improve upon the sparsity of the l-or-one likelihood, a common approach in the literature of MCMC over deterministic programs is to use an *edit-distance* or *noise* likelihood [6, 10]. Here, the generative model explains the observed outputs as random corruptions of the input. We follow [5] in using an efficiently-computable Levenshtein distance based metric.

Prob-MCMC: A key hypothesis we are testing in this work is that searching over a space of *probabilistic* programs with a likelihood given by their probabilistic semantics may lead

LAFI 2026, January 2026, Rennes, France
2026.

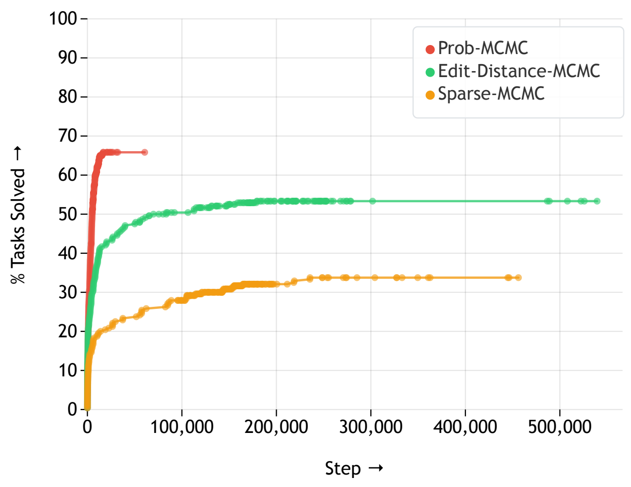

Figure 1. Percent Tasks solved vs MCMC Steps.

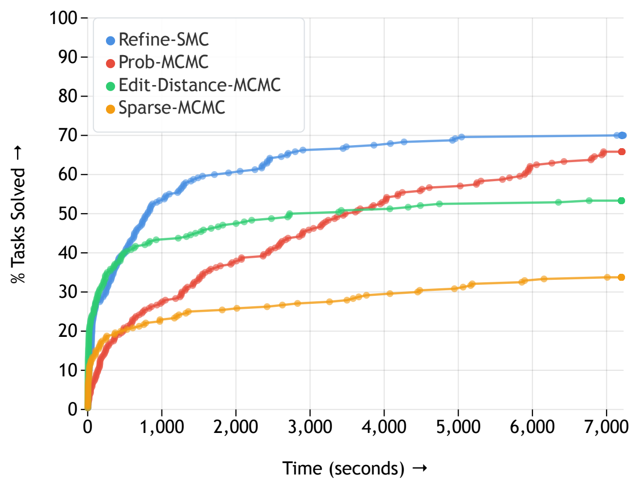

Figure 2. Percent Tasks solved vs Wall-Clock Time.

to better inferences than these alternatives. To explore this hypothesis we extend the core DSL of Listing 2.1 to include the probabilistic primitives shown in ***purple***. We use the Pluck language [2] for efficient exact marginal likelihood evaluation. There is no edit distance or noise likelihood on the outputs of these programs, though the inference over the space of probabilistic programs can be viewed as *inferring* the noise model during the search.

Each synthesis approach was run on the first 80 tasks of the dataset, and we average over the results of 3 attempts at solving each task, where each attempt at a task has a time limit of 2 hours. Figure 1 at first shows Prob-MCMC in a promising light – switching to the extended space of probabilistic programs results in dramatically higher solve rates than Sparse-MCMC or Edit-Dist-MCMC, as a function of the number of steps of MCMC. The latter algorithms run for up to hundreds of thousands of steps while almost all MCMC runs terminate before 10,000 steps and achieve a higher performance. However, Figure 2 presents a plot against wall-clock time, which shows a more nuanced story. While ultimately Prob-MCMC outperforms the Edit-Dist-MCMC and Sparse-MCMC, it does so at a steep computational cost, causing it to lag considerably behind Edit-Dist-MCMC in solve rate for the entire first hour of runtime.

This motivated us to develop an alternative sequential Monte Carlo approach to searching over the space of probabilistic programs, inspired by the idea that successful search paths follow a coarse-to-fine trajectory as outlined in [1].

3 Refinement Proposals for SMC

We consider a sequential Monte Carlo (SMC) algorithm, which we refer to as Refine-SMC, that at each timestep targets a posterior distribution over syntax trees of increasing maximum size. Since all of our tasks are of type $list \rightarrow list$, we begin with the Dirac delta distribution on the size-one program ($\lambda xs.$***randlist(0.5, 0.5)***), which is the program that samples a random list of natural numbers where both the length of the list and its values are geometrically distributed.

At each step of SMC, we use a proposal which chooses a probabilistic leaf (a ***purple*** grammar construct) and randomly samples a depth one expression from the grammar to replace it with (for example `cons(`***randnat(0.5)***`, xs)`). We call these *refinement* proposals because they take a coarse hypothesis and refine it by one step, introducing some new deterministic structure that may increase the likelihood of generating the observed outputs.

The weight update of a particle is computed following [3, 7] by

$$w = \frac{p_t(x_t)}{p_{t-1}(x_{t-1})} \cdot \frac{L(x_t \rightarrow x_{t-1})}{K(x_{t-1} \rightarrow x_t)}$$

where L is a simple reverse kernel that selects uniformly at random the possible coarsenings of the expression. The algorithm can be seen as a form of coarse-to-fine SMC [11].

The results for the SMC approach are presented in Figure 2. Refine-SMC outperforms all other approaches, only performing worse than Edit-Dist-MCMC briefly, early in synthesis. Furthermore, an analysis of which tasks are solved by each method reveals that the set of tasks that Refine-SMC could solve was a strict superset of the tasks that Edit-Dist-MCMC can solve (evaluated through 5 trials on every task each with a one-hour timeout). These results suggest that refinement-based SMC is a promising approach to the synthesis of deterministic programs.

References

[1] Maddy Bowers, Alexander K. Lew, Vikash K. Mansinghka, Joshua B. Tenenbaum, and Armando Solar-Lezama. 2024. Toward Probabilistic Coarse-to-Fine Program Synthesis. Presented at the ACM SIGPLAN Workshop on Languages for Inference (LAFI 2024), co-located with POPL 2024, London, United Kingdom.

https://popl24.sigplan.org/details/lafi-2024-papers/18/Toward-Probabilistic-Coarse-to-Fine-Program-Synthesis

[2] Maddy Bowers, Alexander K. Lew, Joshua B. Tenenbaum, Armando Solar-Lezama, and Vikash K. Mansinghka. 2025. Stochastic Lazy Knowledge Compilation for Inference in Discrete Probabilistic Programs. *Proc. ACM Program. Lang.* 9, PLDI, Article 222 (June 2025), 25 pages. https://doi.org/10.1145/3729325

[3] Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. 2006. Sequential monte carlo samplers. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68, 3 (2006), 411–436.

[4] Noah D. Goodman, Joshua B. Tenenbaum, Jacob Feldman, and Thomas L. Griffiths. 2008. A Rational Analysis of Rule-Based Concept Learning. *Cogn. Sci.* 32, 1 (2008), 108–154. https://doi.org/10.1080/03640210701802071

[5] Qiantan Hong and Alex Aiken. 2024. Recursive Program Synthesis using Paramorphisms. *Proceedings of the ACM on Programming Languages* 8, PLDI (2024), 102–125.

[6] Qiantan Hong and Alex Aiken. 2024. Recursive Program Synthesis using Paramorphisms. *Proc. ACM Program. Lang.* 8, PLDI, Article 151 (June 2024), 24 pages. https://doi.org/10.1145/3656381

[7] Alexander K. Lew, George Matheos, Zhi-Xuan Tan, Matin Ghavamizadeh, Nishad Gothoskar, Stuart Russell, and Vikash K. Mansinghka. 2023. SMCP3: Sequential Monte Carlo with Probabilistic Program Proposals. In *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics (AISTATS) (PMLR, Vol. 206)*. 7061–7088.

[8] Joshua S Rule, Steven T Piantadosi, Andrew Cropper, Kevin Ellis, Maxwell Nye, and Joshua B Tenenbaum. 2024. Symbolic metaprogram search improves learning efficiency and explains rule learning in humans. *Nature Communications* 15, 1 (2024), 6847.

[9] Feras A Saad, Marco F Cusumano-Towner, Ulrich Schaechtle, Martin C Rinard, and Vikash K Mansinghka. 2019. Bayesian synthesis of probabilistic programs for automatic data modeling. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 1–32.

[10] Eric Schkufza, Rahul Sharma, and Alex Aiken. 2013. Stochastic superoptimization. *ACM SIGARCH Computer Architecture News* 41, 1 (2013), 305–316.

[11] Andreas Stuhlmüller, Robert X. D. Hawkins, N. Siddharth, and Noah D. Goodman. 2015. Coarse-to-Fine Sequential Monte Carlo for Probabilistic Programs. arXiv:1509.02962 [cs.AI] https://arxiv.org/abs/1509.02962

[12] Yuan Yang and Steven T Piantadosi. 2022. One model for the learning of language. *Proceedings of the National Academy of Sciences* 119, 5 (2022), e2021865119.