Exploring Probe-Style Grammar Updates in Budgeted Search Program Synthesis

Author: Chris Preda, C.D.M.Preda@student.tudelft.nl

Supervisor: Dr. Sebastijan Dumančić, TU Delft

1. Background

- **Program Synthesis** is the task of finding of a program based on certain specifications/constraints by the user. E.g. I/O examples, partial programs.
- **Budgeted Search:** Instead of tackling synthesis as one big search problem, attack it as multiple individual searches with a limited budget, where you alter your search in some way after each attempt.
- **Context Free Grammar (CFG):** Code building blocks whose combinations create the possible programs.
- **Probe** is a budgeted search method where we adjust the grammar probabilities using **Just-in-Time Learning** which uses bootstrapping to train a model using an update function and partial solutions. Searching through the tree is done using **Guided Bottom-up Search** which traverses the tree using the probabilities, calculating the cost in a size-based enumeration algorithm.

Algorithm 2 The PROBE algorithm

Input: CFG $\mathcal{G}$, set of input-output examples $\mathcal{E}$
Output: A solution P or $\perp$

```
procedure PROBE(G, E)
    G_p ← ⟨G, p_u⟩                                  ▷ Initialize PCFG to uniform
    LVL, B, E ← 0, ∅, ∅                             ▷ Initialize search state
    while not timeout do
        P, ⟨LVL, B, E, PSol⟩ ← GUIDED-SEARCH(G_p, E, ⟨LVL, B, E, ∅⟩)   ▷ Search with current PCFG G_p
        if P ≠ ⊥ then
            return P                                ▷ Solution found
        PSol ← SELECT(PSol, E)                      ▷ Select promising partial solutions
        if PSol ≠ ∅ then
            G_p ← UPDATE(G_p, PSol, E)              ▷ Update the PCFG G_p
            LVL, B, E ← 0, ∅, ∅                     ▷ Restart the search
    return ⊥
```

[1]

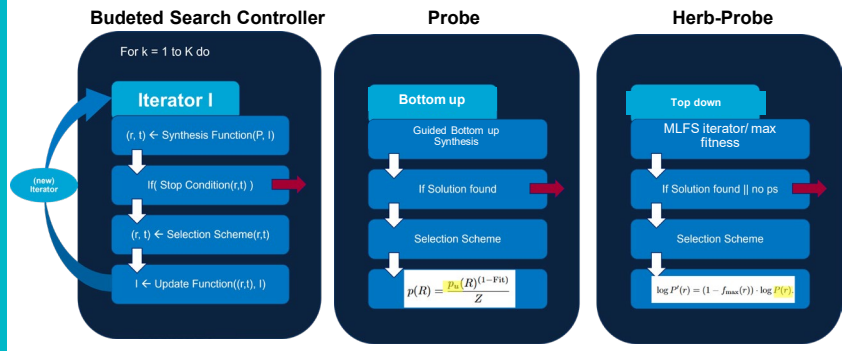
2. Research Question

How do we effectively leverage Probe-Style grammar updates, to improve program synthesis performance in Budgeted Search?

References

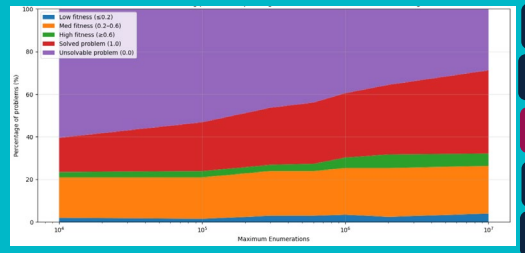
- [1] S. Barke, H. Peleg, and N. Polikarpova. Just-in-time learning for bottom-up enumerative synthesis. Proceedings of the ACM on Programming Languages, 4(OOPSLA): 1–29, Nov. 2020.
- S. Gulwani, O. Polozov, and R. Singh. Program synthesis.Foundations and Trends in Programming Languages, 4(1–2):1–119, 2017.
- T. Hinnerichs, R. G. Reid, J. de Jong, B. Swinkels, P. Wochner, N. Filat, T. Magurescu, I. Hanou, and S. Dumancic. Herb.jl: A unifying program synthesis library 2025.

3. Methods



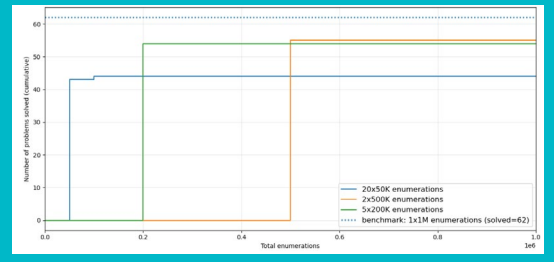
4. Evaluation

RQ1: Does increasing the total budget of a Herb-Probe run without any cycles lead to an increasing number of solved benchmarks?



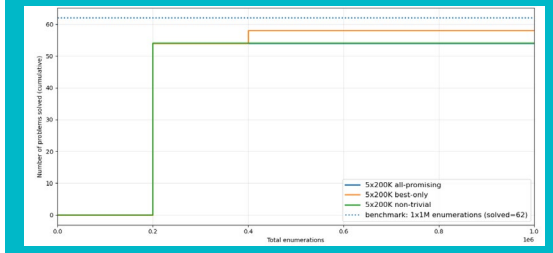
1x10K
....
1x1M
....
1x10M

RQ2: At a fixed total budget, do increased Herb-Probe cycles solve more benchmarks than a single search?

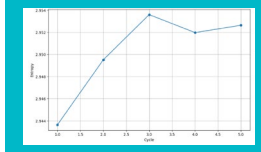
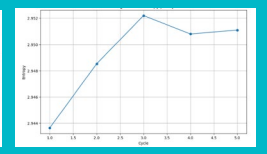
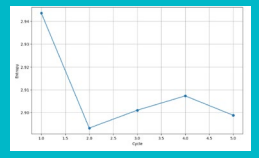


20x50K
5x200K
2x500K

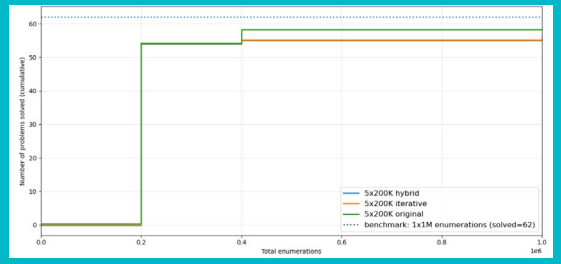
RQ3: Does the specific selector policy of Herb-Probe notably affect success?



all promising
non trivial > 0.2
best only

RQ4: Does Probe's original grammar update rule remain competitive for Herb-Probe?



Original (uniform p)
Iterative (previous p)
best only (normalised p)

RQ5: Does the shared framework allow comparing different approaches?

Approach	Solved Benchmarks
Herb-Probe (this work)	33 / 205
Structural Sketches	51 / 205
Promising Subprograms	31 / 205

EXTA: Benchmark vs optimal:

2x200K
best_only
original
1x400K

58 55

4. Conclusions & Future Work

- Strong selection pressure (best-only) is crucial
- Original Probe updater transfers best
- Currently restarts beyond cycle 2 rarely help
- Shared controller enables fair comparisons

- More drastic comparison
  - Try alternative iterators, updaters (decay), more permissive selectors
- Add standard benchmarks/hybridize