Visit my page @
https://y1fanhe.github.io

ADAPTIVE KNOWLEDGE-DRIVEN PROGRAM SYNTHESIS

Using Knowledge in Automatic Computer Programming through an Adaptive Genetic Programming System

Yifan He (*he.yifan.xs@alumni.tsukuba.ac.jp*), Claus Aranha (*caranha@cs.tsukuba.ac.jp*), and Tetsuya Sakurai (*sakurai@cs.tsukuba.ac.jp*), University of Tsukuba

筑波大学 University of Tsukuba

I. Program Synthesis

Program synthesis (PS) is a task to compose a computer program automatically by a computer. The atomic blocks of a program are called instructions, such as `int_add`, `str_length`, etc. In detail, we are going to compose a program based on two rules.

- A program is a sequence of instructions
- A program satisfies a given set of I/O data

Usually, we consider PS as the optimization problem formulated as below.

$$p^* = \arg\min_{p \in P_{ins}} \sum_{i=1}^{m} \|p(I_i) - O_i\|$$

p: program p^*: optimal program
I_i: i-th input O_i: i-th output
P_{ins}: all feasible programs defined by instruction set ins

II. Genetic Programming

Genetic Programming (GP) is an algorithm that searches programs through an "evolutionary process". In GP, the program is represented as a list of instructions (Fig.1).

The algorithm works as follows.

1. Initialize a random population of programs
2. Select good programs based on their errors
3. Variate the selected programs by adding and deleting instructions
4. Go to Step 2

Fig.1: Lisp-like program as a list of instructions

x0	x1	add	1	div	x1	mod	sqrt

III. Motivation of Using Knowledge

Despite the limited performance of using conventional GPs, several patterns in the human programming motivate us using knowledge in PS.

- Human uses knowledge from their own experience
- Human uses knowledge from other programmers' experience (import libraries)
- Human considers the entire problem as a combination of several sub-problems
- Human cumulates knowledge as time flow.

IV. Applied Scenario

In the future, we run GP on a server and solve problems consecutively. Every time when the GP solves a problem, it extracts and stores some ***"knowledge"***. When there are new problems to solve, the GP will select and use the helpful "knowledge" that are saved before. After a long time, this GP should grow and become clever enough to solve some hard problems, just as a human who is learning to program continuously.

As a first step, within this study, we assume the method to extract "knowledge" from a solution. We discuss how to select the helpful "knowledge" to a specific problem from a set of "knowledges".

V. Proposed Method

Knowledge in this study refers to the ***sub-program*** which is represented as a sub-list of the entire list of instructions.

To use the knowledge (sub-programs), we propose ***replace mutation***. It randomly replaces same length of instructions in a program with a given knowledge (Fig.2).

Fig.2: replace mutation

x0	x1	add	1	div	x1	mod	sqrt

selected sub-prog. | x1 | sub |

⬇

x0	x1	add	x1	sub	x1	mod	sqrt

We further design an ***adaptive strategy*** to select helpful sub-programs from a set (which contains both helpful and unhelpful programs).

The strategy works as follows.

1. The sub-program set prepared before the search (extracted before) is marked as A
2. Prepare a set $S = \emptyset$ to store successful sub-programs
3. If S is empty, randomly select a sub-program from A
4. If S is not empty, randomly select a sub-program from S with 50% probability; otherwise randomly select a sub-program from A
5. Perform replace mutation based on the selected sub-program, add the sub-program to S if the error becomes less after the replace mutation

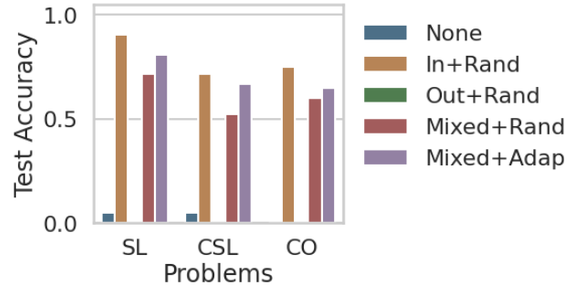
VI. Experiments & Results

We compared five settings on three problems in the General Program Synthesis Benchmark in Fig.3, namely small or large (SL), compare string lengths (CSL), and Count Odds (CO).

- **None**: no knowledge is used
- **In+Rand**: knowledges are extracted from the solution of the problem by hand, and are randomly selected
- **Out+Rand**: knowledges are randomly generated, and are randomly selected
- **Mixed+Rand**: knowledges are from the union set of In+Rand and Out+Rand, and are randomly selected
- **Mixed+Adapt**: knowledges are from the union set of In+Rand and Out+Rand, and are selected by adaptive strategy

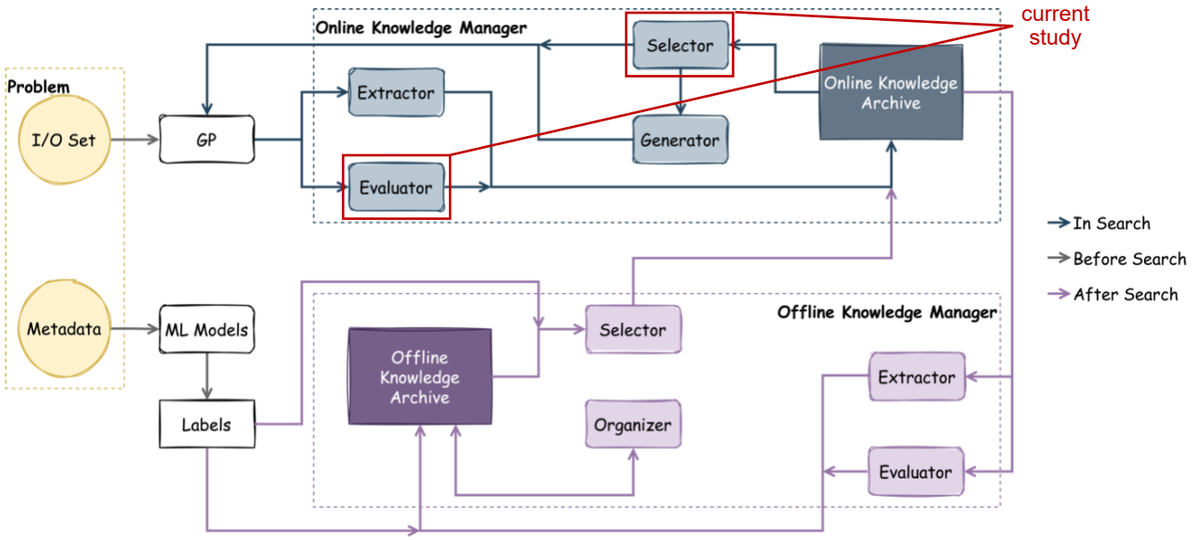
Implication: it is important to use adaptive strategy to select "useful" sub-programs

Fig.3: results on three benchmarks



VII. Design of the Entire Adaptive System

Fig.4: overview of the adaptive knowledge-driven program synthesis system



Y. He, C. Aranha, and T. Sakurai. Adaptive knowledge-driven program synthesis. Poster presentation at 2021 international collaborative workshop of RUB-UGA-UT (CollaboTICS)