Combining Statistical Top-down Deductions and Bottom-up Enumerations for Programming by Example

Report for CSE 573 Artificial Intelligence

Yihong Zhang
Paul G. Allen School of Computer Science & Engineering
University of Washington, Seattle
yz489@cs.washington.edu

Acknowledgement & Received Feedback

I would like to thank Ruyi Ji for his insights on MaxFlash, which makes this course project possible. I also appreciate constructive feedback from the reviewers, but unfortunately I do not have the time to work on improving this project, so I'll leave it as it is. Below are the feedback from the reviewers:

Major comments (1) This is a solid and interesting choice for a course project; it is good you were able to integrate your existing research interests. (2) It is interesting that the results are mixed, and not a strict improvement over MaxFlash, as you seem to have expected. The analysis of why is interesting; I would encourage you to think about cases where MaxFlash will fail or perform poorly but MaxDuet would be expected to perform better, and focus future experiments on those alone (MaxDuet does not need to be globally better; if it improves on a certain subset of cases that could still be useful). (3) The dataset needs to be described; simply providing a citation is not sufficient for a reader to evaluate the experiments. Please provide more details in the main text.

Minor comments (1) Some of the notation and terminology is not familiar to me; while I understand the overall problem setup and can infer it, you should fully explain the operators and terms used when writing for an audience outside your area (e.g. the operators in the equation in Sec. 1; what is an "incomplete" function). (2) It would be useful to provide pseudocode for the algorithms, instead of describing various steps and how they are modified.

Abstract

Programming by Example (PBE) is the task of automatically synthesizing programs according to the user-provided input-output examples. Due to its flexibility, it is widely applied in various domains. For example, FlashFill (Polozov and Gulwani, 2015) is a PBE tool in Microsoft Excel that auto-fills spreadsheets by synthesizing string manipulation programs. In this project, I propose to combine two recent optimization techniques in PBE literature to accelerate PBE task solving, i.e., using structural probability to guide deductive program synthesis (Ji et al., 2020) and combining deductive search with enumerative search for solving PBE tasks (Lee, 2021). Specifically, I add the support for enumerative search to MaxFlash, a program synthesizer that uses probabilistic models to guide the top-down deduction. Besides directly solving the synthesis problem, the enumerated programs are also used by the witness functions during probabilistic deductions. Experiments on the SyGuS benchmarks shows mixed results. Finally, I hypothesize and discuss several

Preprint. Under review.

potential limitations of the "enumeration-guided deductions" approach proposed in Lee (2021).

1 Introduction

Program synthesis is the task that, given a set of grammar and a formal specification, generates programs according to the grammar that are consistent with the specification. One important subproblem of program synthesis is programming by example (PBE), where the specification is given as a set of input-output examples. More formally, given a context-free grammar G and a set of input-output examples $\vec{e}$, PBE asks whether there exists a program P such that

$$P \in L(G) \wedge \bigwedge_{(i,o)\in\vec{e}} [\![P]\!](i) = o,$$

where $L(G)$ is the language specified by G. PBE has become the focus of many researches due to its easy-to-obtain and easy-to-interpret specifications and the practical usability in applications like database queries (Wang et al., 2017), string processing (Polozov and Gulwani, 2015), data wrangling (Gulwani, 2016), and data visualization (Wang et al., 2019). It's also an important subroutine of counterexample-guided inductive synthesis (CEGIS), which solves a broad class of program synthesis efficiently.

There are recently a number of proposed optimizations to speedup PBE solving. Among them, two of the optimization techniques are particularly promising. The first technique, presented as a tool named MaxFlash, is to guide the top-down deductive search of programs using statistical models (Ji et al., 2020). One of the central contribution of this work is its novel way of representing states for efficient state reuse. The second technique, instantiated in Duet, proposes to synergistically combine top-down deductions and bottom-up enumerations in a bidirectional search of satisfying programs, which effectively mitigates the shortcomings found in these two search strategies (Lee, 2021). One attractive property of Duet is the algorithm's independence from the specific deductive search strategy chosen. Therefore, it's natural to extend the combined framework presented in (Lee, 2021) with more sophisticated deductive search strategy. By combining the two optimizations together, I hope to obtain a performance gain over tools where only one technique is used.

To investigate into this problem, I implemented the bottom-up enumeration strategies for MaxFlash and uses the enumerated programs to aid the deduction procedure, as proposed by Lee (2021). I cal this new version of MaxFlash with the proposed optimizations MaxDuet. In the experiment, I compare the performance of MaxDuet against MaxFlash, which shows mixed result. To better explain the result, a discussion is offered in Section 5.

2 Related Works

In this section, I will discuss related works and compare MaxDuet with them.

A large class of PBE algorithms is based on search. Search algorithms for PBE can be generally classified into three categories:

- Enumerative search: given a grammar, enumerative search enumerates programs derivable from the grammar according to a specific order. In this report only bottom-up enumeration is considered, where programs are enumerated by composing smaller programs together. It turns out enumerative search is most effective for small programs. However, it is hard for enumerative search to synthesize large programs due to the colossal search space.
- Deductive search: deductive search derives programs by rewriting the specification into smaller specifications. Many modern deductive program synthesizers rely on *witness functions*. Popularized by the PROSE program synthesis framework (Polozov and Gulwani, 2015), witness functions calculate the inverse semantics of an operator. For example, given a specification string s, the witness for operator `concat(str,l,r)` considers possible inputs to `concat` that produces s. Generally, there may be infinitely many possible inputs, so most witness functions are incomplete.

	Enumerative search	Deductive search	Stochastic Search
EUSolver	✓		
Euphony	✓		✓
MaxFlash		✓	✓
PROSE	✓	✓	
Duet	✓	✓	
MaxDuet	✓	✓	✓

Table 1: Comparison with related works

- Stochastic search: stochastic search learns a distribution of likely programs that may satisfy the specification and samples programs from this distribution. Approaches to stochastic search for program synthesis vary from Probabilistic Context-Free Grammar to Neural Networks, but they all exploit the probabilistic models to speed up the search.

Many program synthesizers try to reconcile two of the search strategies for best performance. For example, MaxFlash (Ji et al., 2020) combines deductive search with a probabilistic model in an iterative deepening style. In particular, it encodes the context of a (sub-)specification into the state representation in a creative way, which allows efficient state reusing. On the other hand, Duet (Lee, 2021) synergistically combines deductive search with bottom-up enumerations by using the pool of enumerated programs to guide the behavior of the witness functions. It enjoys the completeness of enumerative search, meaning that it will finds such a solution as long as it exists, while is also able to synthesize complex programs efficiently compared to pure enumerative search. However, to my knowledge, none of the previous work utilizes the power all of the three approaches. In comparison, MaxDuet aims to takes advantages of the three approaches. A comparison with other related works is made in Table 1.

3 Approach

In this section, I propose MaxDuet and describes several implementation details of MaxDuet.

I built MaxDuet on top of MaxFlash and implemented the proposed optimizations in Lee (2021). In particular, the optimization consists of two parts: implementing bottom-up enumeration functionalities and using the enumerated programs to guide the synthesis.

Implementing bottom-up enumerations in MaxFlash During each iteration, MaxFlash uses a lowerbound to restrict the search space by searching for programs with probabilities higher than the lowerbound. The lowerbound is relaxed during each iteration. If a program cannot be found given certain very small lowerbound, MaxFlash will return "not found". It is possible for MaxFlash to be unable to find satisfying programs because the witness functions are not complete. To complement this, I added another iteration loop on top of the iterative deepening. In particular, if a solution cannot be found, instead of returning "not found", we expand the enumeration pool by increasing the target program size by 1 and then restart the iterative deepening. Because the output of witness functions are dependent on the pool of enumerated programs, the witness functions are able to generate more possible sets of arguments, allowing a larger search space to be explored. The enumeration procedure accepts the program size as an argument and, for every n-ary operator *op*, enumerates all the possible programs $op(p_1, p_2, \ldots, p_n)$ where p_i is from the current enumeration pool and the size of p_i sums up to $n - 1$. If any enumerated program is found to satisfy the specification, the program will be directly returned. To accumulate an initial pool of enumerated programs, MaxDuet will enumerate all programs with size under 3 at the beginning by default.

Using enumerated programs to guide the synthesis One central contribution of Duet (Lee, 2021) is the synergy between enumerative search and deductive search. In particular, Duet uses the output of enumerated programs to guide the witness functions. To have a sense of it, consider the operator $\mathsf{substr}(str, l, r)$. Given the specification $i \mapsto o$, Duet will search for all the enumerated programs P in the enumeration pool whose output contains o. For every such P, Duet produces a potential set of arguments where the first argument to substr is $[\![P]\!](i)$ and the second and third arguments are the left and right positions of o in $[\![P]\!](i)$, where $[\![P]\!](i)$ is the result of calling P on i. This

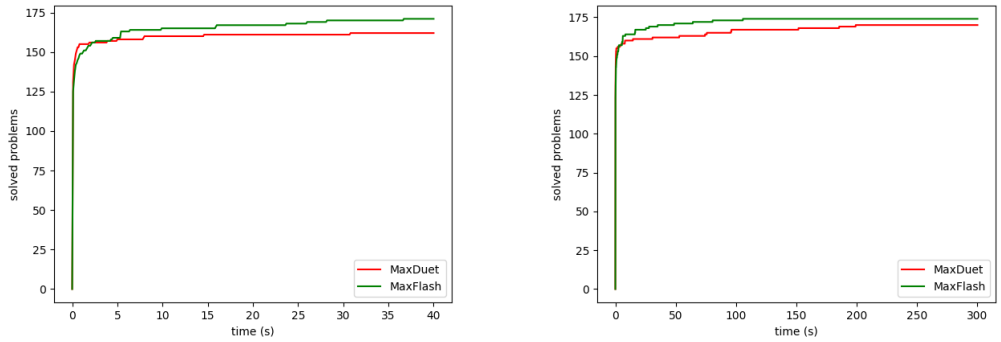

Figure 1: Comparison between MaxDuet and MaxFlash. Left: Time vs. solved instances for the first 40 seconds. Right: Time vs. solved instances for the first 300 seconds.

approach allows the witness functions to find the most relevant derivations that are likely to be solved. Moreover, it makes witness functions more flexible because the set of derivations can be continuously grown once it is found the current set of derivations cannot produce a solution. In comparison, most of the witness functions used by MaxFlash use a more limited despite similar strategy, where it only considers constants and parameters as the hints to possible arguments. Therefore, a lot of MaxFlash's witness functions can be seen as a special case of Duet's, where the pool of enumerated programs contains only size-1 programs. In MaxDuet, I implemented the witness functions proposed in Lee (2021, Figure 5).

Due to limited time, I only implement the proposed witness functions in MaxDuet for the string domain, although Duet also considers witness functions for bitvectors and circuits. The total implementation on top of MaxFlash is about 500 lines.

4 Experiment

I run experiments to compare the relative performance between MaxFlash and MaxDuet on my laptop using the SyGuS (Alur et al., 2019) dataset. The experiments consist of two parts, with one having a hard cutoff when programs run for 40 seconds and one 300 seconds. I run MaxDuet on both parts for the benchmarks. Due to limited time, I only run MaxFlash with a hard cutoff of 40 seconds. I use the data from the MaxFlash paper's experiment for the setting where the cutoff is 300s. This may cause inaccuracy due to difference across machines. However, this does not affect the result qualitatively because it turns out the machine running MaxFlash's original experiment is slightly slower than my laptop and even under this situation, MaxDuet still performs mostly worse than MaxFlash according to the numbers.

Figure 1 shows the result. The overall performance of MaxDuet is worse than MaxFlash: MaxDuet finds less programs in 40 seconds (162 vs 171) and in 300 seconds (170 vs. 174). Some hypothesis that explains the hypothesis is given in Section 5. Yet, MaxDuet does find solutions to 3 problems that MaxFlash cannot find, mostly because of the more flexible witness functions. Moreover, because of enumerative search's efficiency in solving problems that has a small solution, MaxDuet solves more problems in the first 0.5 seconds.

5 Discussions

Although it requires an extensive examination and experiments to explain the relative slowdown of MaxDuet compared to MaxFlash, there are several hypothesis, which are also the directions for future work.

1. Unlikely derivations: it turns out most problems solved by MaxDuet are also solved by MaxFlash. Therefore, it is doubtful whether enumeration-guided witness functions are necessary. For example, I'm having a hard time thinking about a case where the first

argument to $\mathsf{substr}(str, l, r)$ is not a parameter. Therefore, despite incomplete, carefully designed heuristics are good enough to solve most of the program synthesis problems with a superior speed.

2. Time spent during enumerations and derivation generations: enumerative search is time-consuming and takes a very long time when iteration increases due to its exponential nature. Worse, to produce the sets of possible arguments, the witness functions usually need to enumerate all of the programs in the enumeration pool, which is asymptotically slower than MaxFlash's witness functions.

Finally, although Duet claims to solve 81 out of 82 benchmarks in the SyGuS dataset, it tends to generate non-generalizable solutions for PBE tasks. For example, in several benchmarks examined, the generated program contains dozens of operators and many if-then-else operators, sometimes even more than the number of input-output examples. Such programs are unlikely to generalize, because every specification with n input-output examples can be answered by a program with $n - 1$ if-then-else operators trivially. I manually disable the if-then-else operator in two said cases. In one case, Duet failed to synthesize a satisfying program. In another case, Duet synthesize another program, which still failed to generalize. Note that Duet is designed to target a larger class of program synthesis programs than PBE, (i.e., Syntax-Guided Synthesis), which may explain their relative weakness in finding generalizable programs for PBE tasks. The authors of Duet may also be aware of the limited generalizablity, as they mark generalizablity as one advantage of EUSolver to Duet.

In summary, I proposed and implements MaxDuet based on MaxFlash. To my knowledge, it is the first tool that utilizes deductive search, enumerative search, and stochastic search for PBE solving. Each of the component works synergically with each other to find the satisfying solutions. However, experiment shows that MaxDuet performs slightly worse than MaxFlash. I discuss some possible explanations for the performance difference.

References

Rajeev Alur, Dana Fisman, Saswat Padhi, Rishabh Singh, and Abhishek Udupa. 2019. SyGuS-Comp 2018: Results and Analysis. arXiv:1904.07146 [cs.PL]

Sumit Gulwani. 2016. Programming by Examples - and its applications in Data Wrangling. In *Dependable Software Systems Engineering*, Javier Esparza, Orna Grumberg, and Salomon Sickert (Eds.). NATO Science for Peace and Security Series - D: Information and Communication Security, Vol. 45. IOS Press, 137–158. https://doi.org/10.3233/978-1-61499-627-9-137

Ruyi Ji, Yican Sun, Yingfei Xiong, and Zhenjiang Hu. 2020. Guiding Dynamic Programing via Structural Probability for Accelerating Programming by Example. *Proc. ACM Program. Lang.* 4, OOPSLA, Article 224 (Nov. 2020), 29 pages. https://doi.org/10.1145/3428292

Woosuk Lee. 2021. Combining the Top-down Propagation and Bottom-up Enumeration for Inductive Program Synthesis. *Proc. ACM Program. Lang.* 5, POPL, Article 54 (Jan. 2021), 28 pages. https://doi.org/10.1145/3434335

Oleksandr Polozov and Sumit Gulwani. 2015. FlashMeta: A Framework for Inductive Program Synthesis. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications* (Pittsburgh, PA, USA) *(OOPSLA 2015)*. Association for Computing Machinery, New York, NY, USA, 107–126. https://doi.org/10.1145/2814270.2814310

Chenglong Wang, Alvin Cheung, and Rastislav Bodik. 2017. Synthesizing Highly Expressive SQL Queries from Input-Output Examples. *SIGPLAN Not.* 52, 6 (June 2017), 452–466. https://doi.org/10.1145/3140587.3062365

Chenglong Wang, Yu Feng, Rastislav Bodik, Alvin Cheung, and Isil Dillig. 2019. Visualization by Example. *Proc. ACM Program. Lang.* 4, POPL, Article 49 (Dec. 2019), 28 pages. https://doi.org/10.1145/3371117