Artifact for “GenTree: Using Decision Trees to Learn Interactions for Configurable Software”

KimHao Nguyen and ThanhVu Nguyen
University of Nebraska-Lincoln, USA
{kdnguyen,tnguyen}@cse.unl.edu

***Abstract*—This document describes the artifact package accompanying the ICSE’21 paper “GenTree: Using Decision Trees to Learn Interactions for Configurable Software” [1]. The artifact includes GenTree source code, pre-built binaries, benchmark program specifications, and scripts to replicate the data presented in the paper. Furthermore, GenTree is applicable to new programs written in supported languages (C, C++, Python, Perl, Ocaml), or can be extended to support new languages easily. GenTree implementation is highly modular and optimized, hence, it can also be used as a framework for developing and testing new interaction inference algorithms. We hope the artifact will be useful for researchers who are interested in interaction learning, especially iterative and data-driven approaches.**

I. Introduction

Testing, debugging, and analyzing highly configurable systems are challenging because of the exponentially large configuration spaces—in the worst case, every combination of option settings can lead to a distinct behavior [2]–[5]. The *configuration space explosion* presents real challenges to software developers because faults are often visible under only specific combinations of configuration options. Static analyses are difficult because of the path explosion problem and usage of external libraries that are difficult to model precisely.

An interaction for a location is defined as a logically weakest formula over configuration options such that any configuration satisfying that formula would cover that location. The GenTree [1] interaction learning algorithm is inspired by the iterative and dynamic approach of iGen [6] but can discover interactions under *arbitrary* boolean formula by leveraging the expressive power of decision trees. This document describes the reusable artifact package accompanying the paper, which includes source code, pre-built binaries, benchmark program specifications, and scripts to replicate the evaluation data presented in the paper.

II. Artifact Overview

A. Benchmark Parts

We divided the benchmark suite into two parts: a `fast` part that quickly generates results for reasonably sized programs and an `all` part that generates results for all programs. Both parts also run an exhaustive search to get the ground truth interactions if the configuration space is less than 10^7.

B. System Requirements

a) Hardware: The benchmarks presented in the paper were run on a workstation with a 64-core AMD Ryzen Threadripper 3990X @ 2.9 GHz CPU, 64 GB RAM, and at least 40GB of free disk space. Running the `all` benchmarks takes around 26 hours. However, the `fast` part could run on a normal laptop with around 8 GB RAM.

b) Software: We recommend evaluating the artifact on a Linux-based OS (tested on Ubunutu 20.04 and Debian 10.7) with Docker (tested with Docker 19.03.14 and 20.10.1). Before proceeding with the installation instructions, make sure you can successfully run the command `docker run hello-world` on the host machine.

```
# Step 1: Pull Docker image
docker pull unsatx/gentree_docker:icse21

# Step 2: Run container
docker run -it --rm --tmpfs /mnt/ramdisk \
   unsatx/gentree_docker:icse21 bash

# Step 3: Run GenTree (inside container)
cd ~/gentree/wd
./gt -J2 -cx -BF @ex_paper # example
./gt -J2 -cx -BF @ex_paper --full # example
    (ground truth)
./gt -J2 -cx -GF 2/id # coreutils id (C)
./gt -J2 -cx -YF 2/vsftpd # vsftpd (Otter)
```

Fig. 1: Commands to install GenTree

C. Installation

To install the GenTree artifact, follow the instructions in Figure 1. First, we pull the pre-built Docker image from Docker Hub. If the image is not available, we can import the permanently archived image at [7]. Then, we start the container and run GenTree tool. `@ex_paper` is the example C program listed in Figure 2 in [1]. `id` and `vsftpd` are benchmark programs listed in Table 1 in [1].

III. GenTree Output Format

For better interoperability, the GenTree output is designed to be both human and machine-readable. For each discovered interaction, GenTree outputs a block similar to Figure 2. *Blocks* are separated by the line “======”. In a block, there are three *components* separated by the line “–”. All lines started with the character ‘#’ are comments and should be ignored while parsing. In Figure 2, the comments tell us how many *hit* and *miss* configurations classified by the

```
======
# M/H: 51 / 137
# Last rebuild: iter 2 num_configs 30
L4,
-
(or (= u |0|) (= v |0|))
-
3 H 4 HM
======
```

Fig. 2: `GenTree`'s output for the interaction $\bar{u} \vee \bar{v}$ in the example program in Figure 2 in [1]

decision tree, which iteration the tree was last rebuilt, and how many configurations were used to build the tree. The first component in the block is a list of locations covered by the interaction (e.g., $L4$). The second component is an SMT-LIB 2.0 [8] formula generated by Z3 [9]. The last component is a serialized decision tree (pre-order traversal) for internal usage.

```
# Step 1: Check if GenTree is working.
# If got "Permission error", run "sudo chmod
   777 -R /mnt/ramdisk" and retry.
./gt -J2 -cx -GF 2/id

# Step 2: Clean up old results
./scripts/bm.sh --clean

# Step 3:
# - To run the fast part (~3m on i9-9880H)
./scripts/bm.sh --fast --bm
# - To run the all part (~26h on Ryzen 3990X)
./scripts/bm.sh --all --bm
```

Fig. 3: Commands to run the benchmarks (inside container)

IV. Evaluation

There are two parts to the evaluation process: generating interactions and analyzing them.

A. Generate Interactions

Follow the instructions in Figure 3 to generate the interactions and ground truths. The results are saved into files `res/<program>/a_<repeat>.txt`. Each program is run 11 times, so `<repeat>` is an integer from 0 to 10. The ground truth interactions, if available, are saved at `res/<program>/full.txt`. The output file format is described in §III.

B. Analyze Interactions

Follow the instructions in Figure 4 to analyze the generated interactions and obtain the results as presented in [1]. The analysis results are saved to `res/Analyze/<type>/<program>.csv`, where `<type>` is the analysis type (count number of each interactions type, the accuracy of inferred interactions compared to the ground truth, etc.). Each row in the `.csv` file presents the data of a single run, and the `MED` row at the end is the median value being reported in [1]. For a more detailed description of the analysis results, please refer to the `README` file in the Github source repository at [10] or the snapshot at [7].

```
# Step 1: Run analysis for Table II, III
# - fast part: ~5s on i9-9880H.
# - all  part: ~60s on Ryzen 3990X.
./scripts/bm.sh --all --analyze-all

# Step 2: Run analysis for Fig 9
# (optional, upto ~30m on Ryzen 3990X)
./scripts/bm.sh --all --analyze-progress
```

Fig. 4: Commands to analyze data (inside container) for Tables II, III and Figure 9 in [1]

The `GenTree` implementation and benchmark programs have some nondeterministic components, hence, the replicated results may not match exactly with the results in [1]. However, most of the time, they should be close or match exactly.

Acknowledgment

We thank the anonymous reviewers for helpful comments. This work was supported in part by awards CCF-1948536 from the National Science Foundation and W911NF-19-1-0054 from the Army Research Office. KimHao Nguyen is also supported by the UCARE Award from the University of Nebraska-Lincoln.

References

[1] K. Nguyen and T. Nguyen, "GenTree: Using decision trees to learn interactions for configurable software," in *International Conference on Software Engineering*. IEEE, 2021, p. to appear.

[2] P. Gazzillo, "Kmax: Finding all configurations of kbuild makefiles statically," in *Foundations of Software Engineering*, 2017, pp. 279–290.

[3] S. Zhou, J. Al-Kofahi, T. N. Nguyen, C. Kästner, and S. Nadi, "Extracting configuration knowledge from build files with symbolic analysis," in *International Workshop on Release Engineering*. IEEE, 2015, pp. 20–23.

[4] S. Apel, D. Batory, C. Kästner, and G. Saake, "Software product lines," in *Feature-Oriented Software Product Lines*. Springer, 2013, pp. 3–15.

[5] M. B. Cohen, P. B. Gibbons, W. B. Mugridge, and C. J. Colbourn, "Constructing test suites for interaction testing," in *International Conference on Software Engineering*. IEEE, 2003, pp. 38–48.

[6] T. Nguyen, U. Koc, J. Cheng, J. S. Foster, and A. A. Porter, "iGen: Dynamic interaction inference for configurable software," in *Foundations of Software Engineering*, 2016, pp. 655–665.

[7] K. Nguyen and T. Nguyen, "Artifact for GenTree: Using decision trees to learn interactions for configurable software," 2021. [Online]. Available: https://doi.org/10.5281/zenodo.4514778

[8] C. Barrett, A. Stump, and C. Tinelli, "The SMT-LIB Standard: Version 2.0," Department of Computer Science, The University of Iowa, Tech. Rep., 2010, available at `www.SMT-LIB.org`.

[9] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[10] K. Nguyen and T. Nguyen, "GenTree," 2021, accessed on 2021-02-01. [Online]. Available: https://github.com/unsat/gentree