Learning Graph-based Heuristics for Pointer Analysis without Application-Specific Features (Artifact Manual)

This artifact aims to reproduce the main results in the paper "Learning Graph-based Heuristics for Pointer Analysis without Application-Specific Features" submitted to OOPSLA 2020. Specifically, following this manual will reproduce Table 1 and 2 in the evaluation section.

1 Getting Started Guide

1.1 Setup Instruction

We packaged our artifact in a VirtualBox (version 6.0.14) image with all of the necessary libraries installed. Unpacking the image requires at least 50G of free storage, and running the image to reproduce the main results of Table 1 and 2 needs at least 32GB of free memory.

- ID: "graphick" / PASSWORD: "1234"
- The analyzers for Table 1 and 2 are located in the directories "∼/`Graphick/Ctx_Abstraction/artifact/`" and "∼/`Graphick/Heap_Abstraction/artifact/`", respectively. The former corresponds to Table 1 and the other corresponds to Table 2. In each directory, you can find:

  - `doop`: A folder which contains modified Doop framework to support all the analyses in Table 1 (e.g., GRAPHICK,SCALER, ZIPPER, DATA, *2objH*, *Insens*) or Table 2 (e.g., GRAPHICK, MAHJONG, *Alloc-Based*, *Type-Based*).

  - `run.py`: A Python script for reproducing Table 1 or 2. This script performs points-to analysis with various heuristics. (Section 2)

1.2 Verifying Installation (Basic Testing)

Move to the directory "∼/`Graphick/Ctx_Abstraction/artifact/`". Then, run the following command, which analyzes a program `luindex` with GRAPHICK:

```
$ ./run.py graphick luindex
```

If the artifact is successfully installed, you will see the following results:

```
Running graph_ci pointer analysis for luindex ...
...
Pointer analysis START
analysis time: 23.49s
Pointer analysis FINISH
...
loading graph heuristic ...
elapsed time: 14.46s
...
Running 2obj-Graphick pointer analysis for luindex ...
...
Pointer analysis START
analysis time: 34.94s
Pointer analysis FINISH
...
#may-fail casts                      297
#poly calls                          682
#call edges                          29,045
```

The above results illustrate that the pre-analysis(e.g., `graph_ci`) and extracting an abstraction from the pre-analysis results take 23.49 and 14.46 seconds, respectively. The sum of these two costs corresponds to the analysis cost in the parentheses in Table 1 and 2. The main analysis (e.g., `2obj-Graphick`) takes 34.94 seconds. The results also show the results for each client (#may-fail casts, #poly-call sites, and #call-graph-edges metric). Note that it may report the different analysis time from the paper due to the differences in the experiment environments. In our paper, all the experiments are conducted on a machine with 64GB of RAM.

2 Step-by-Step Instructions

Following the instructions below reproduces Table 1 and 2. The command for running pointer analysis is as follows:

```
$ ./run.py <analysis> <pgm>|-all
```

If your current directory is "∼/Graphick/Ctx_Abstraction/artifact/", `<analysis>` can be one of the following analyses:

```
graphick, scaler, zipper, data, 2objh, insens
```

If you are in "∼/Graphick/Heap_Abstraction/artifact/", `<analysis>` can be:

```
graphick, mahjong, alloc_based, type_based
```

2.1 Reproducing Graphick in Table 1 and 2

To reproduce the columns Graphick in Table 1 and 2, move to "∼/Graphick/Ctx_Abstraction /artifact/" for Table 1 or to "∼/Graphick/Heap_Abstraction /artifact/" for Table 2. Then, run the following command:

```
$ ./run.py graphick -all
```

It takes about 3 hours in total. You can run Graphick for a specific program with the following command:

```
$ ./run.py graphick <pgm>
```

If you are in "∼/Graphick/Ctx_Abstraction/artifact/", `<pgm>` can be one of the following programs:

```
luindex, lusearch, antlr, pmd, chart, eclipse, jedit,
briss, soot, jython, findbugs
```

If you are in the other one, `<pgm>` can be:

```
luindex, lusearch, antlr, pmd, fop, chart, bloat
xalan, JPC, checkstyle, findbugs
```

For example, if you want to analyze `antlr` with Graphick, type:

```
$ ./run.py graphick antlr
```

2.2 Reproducing Other Analyses

Replacing `graphick` in the above commands with other analyses reproduces the corresponding results in the tables. For example, to reproduce the column Scaler in Table 1 (make sure that your working directory is "∼/Graphick/Ctx_Abstraction/artifact/"), type:

```
$ ./run.py scaler -all
```

If you want to construct all results of Table 1 or Table 2 completely, type:

```
$ ./runall.sh
```

It will reproduce the complete results of each evaluation table in our paper. To save user's time, it skips the cases if the analyses are not scalable in Table 1 and 2. For example, it skips analyzing `lusearch` with Alloc-Based analysis. Additionally, it skips the analyses if the analyses are not scalable in the current experimental environment. For example, it does not analyze `checkstyle` with `mahjong` in the given artifact because we limit the free memory for the current artifact to 32GB. If we give more memory, it can reproduce the same results with our evaluation table. Further, we would like to warn that it would cost a lot to construct the complete tables. The above command would take more than 10 hours for each table.

2.3 Learning Graph-based Heuristic

The artifact also includes learning script which reproduces Graphick used in Table 2. To reproduce it, change your working directory to "∼`/LearnGraphick/script/`". Then, run the following command:

```
$ ./learn.py
```

This procedure takes about 3 days. It will produce 96 features as a graph-based heuristic written in Datalog.