Centris

Centris is a tool for identifying open-source components. Specifically, Centris can precisely and scalably identify components even when they were reused with code/structure modifications. Principles and experimental results are discussed in our paper, which will be published in 43rd International Conference on Software Engineering (ICSE'21).

The prototype of Centris will be opened soon at IoTcube (https://iotcube.net).

How to use

Requirements

Software

- ***Linux***: Centris is designed to work on any of the operating systems. However, currently, this repository only focuses on the Linux environment. Centris can be operated on Windows if some minor environment settings (e.g., the path of ctags parser used in OSSCollector) are changed.
- ***Python 3***
- ***Universal-ctags***: for function parsing.
- ***Python3-tlsh***: for function hashing.

How to install Python3-tlsh:

```
sudo apt-get install pip3
sudo pip3 install py-tlsh
```

Hardware

- We recommend a minimum of 32 GB RAM to utilize a large amount of OSS datasets for component identification.

Running Centris

OSSCollector (src/osscollector/)

1. Collect git clone URLs (will be contained in the component DB) into a single file, as shown in the sample file.
2. Specify the directory paths in OSS_Collector.py (line 17 to 21), where cloned repositories and their functions will be stored. Also you should specify the path of the installed ctags here.
3. Execute OSS_Collector.py

```
python3 OSS_Collector.py
```

(several warnings may occur due to encoding issues.)

4. Check the outputs (description based on the default paths).
    - ***./osscollector/repo_src/***: Directory for storing source codes of collected repositories;
    - ***./osscollector/repo_date/***: Directory for storing release dates of every version of all collected repositories;
    - ***./osscollector/repo_functions/***: Directory for storing extracted functions from all collected repositories.

Preprocessor (src/preprocessor/)

- You can create a component DB through preprocessing in two ways: using the full preprocessor (Preprocessor_full.py, used in the paper) or the lite version of preprocessor (Preprocessor_lite.py). The only difference is whether including the common functions of the two software even similar functions (for full), or consider only the exact same functions (for lite). If you create the component DB with the lite preprocessor, the elapsed time is much shorter than that of the full preprocessor, but instead, the component identification accuracy is slightly decreased.

1. Specify proper directory paths in the Preprocessor_full.py or Preprocessor_lite.py (what you specified in the above OSSCollector step 2).

2. Execute the corresponding Python script (Preprocessor_full.py or Preprocessor_lite.py).

```
python3 Preprocessor_full.py
```

or

```
python3 Preprocessor_lite.py
```

3. Check the outputs (description based on the default paths).
    - ***./preprocessor/componentDB/***: Directory for the constructed component DB;
    - ***./preprocessor/verIDX/***: Directory for storing index information for each collected OSS;
    - ***./preprocessor/metaInfos/***: Directory for storing meta-information for each collected OSS;
    - ***./preprocessor/weights/***: Directory for storing weights for each function in all collected OSS (this is for predicting utilized version of an identified component);
    - ***./preprocessor/funcDate/***: Directory for storing birth date of each function in all collected OSS;

Detector (src/detector/)

1. Specify the path of component DB and the directory where the result will be saved in Detector.py.
2. Run the Detector.py with the root path of the target software as an argument, in which you want to identify components of the software.

```
python3 Detector.py /path/of/the/target/software
```

3. Check the component identification results (default output path: ./detector/res/)

Reproducing the results presented in the paper

For several reasons (including commercial usage of Centris and the massive size of the dataset), we release the component DB which can only be utilized to identify the list of OSS components (i.e., no version information).

Therefore, there are two ways to reproduce the results of the paper:

1. The case of executing all three modules from component DB construction to component identification;
2. The case of using the dataset we provided.

Case 1. Executing all three modules.

1. Collect 10,000+ Git repositories for creating the component DB. Note that we collected C/C++ repositories with more than 100 GitHub stars as of April 2020 in the paper. This process, depending on the number of repositories you collect, can take anywhere from days to weeks.
2. Create the component DB using the Preprocessor_full.py
3. Execute the Detector.py.
4. See the results.

Case 2. Using the provided dataset.

Dataset: download here (5 GB).

1. Extract the downloaded file (Centris_dataset.tar).
2. There are four sample target software (ArangoDB, Crown, Cocos2dx, and Splayer, which are utilized in the in-depth comparison in the paper).
    - [2a] To check the detection result for these four target software programs, set "testmode" in line #194 of "Detector_for_OSSList.py" file to 1, and adjust only the file paths in lines #198 and #199 in the "Detector_for_OSSList.py" file.
    - [2b] To check the detection result for other software programs, set "testmode" in line #194 of "Detector_for_OSSList.py" file to 0.
3. Execute the "Detector_for_OSSList.py".

[2a]

```
python3 Detector_for_OSSList.py
```

[2b]

```
python3 Detector_for_OSSList.py /path/of/the/target/software
```

4. See the results (default output path: ./res/.)

About

This repository is authored and maintained by Seunghoon Woo.

For reporting bugs, you can submit an issue to the GitHub repository or send me an email (seunghoonwoo@korea.ac.kr).