Final Report (SimFix++)

Yangruibo Ding (yd2447)
Joe Huang (jch2220)
Columbia University, New York, NY

SimFix++

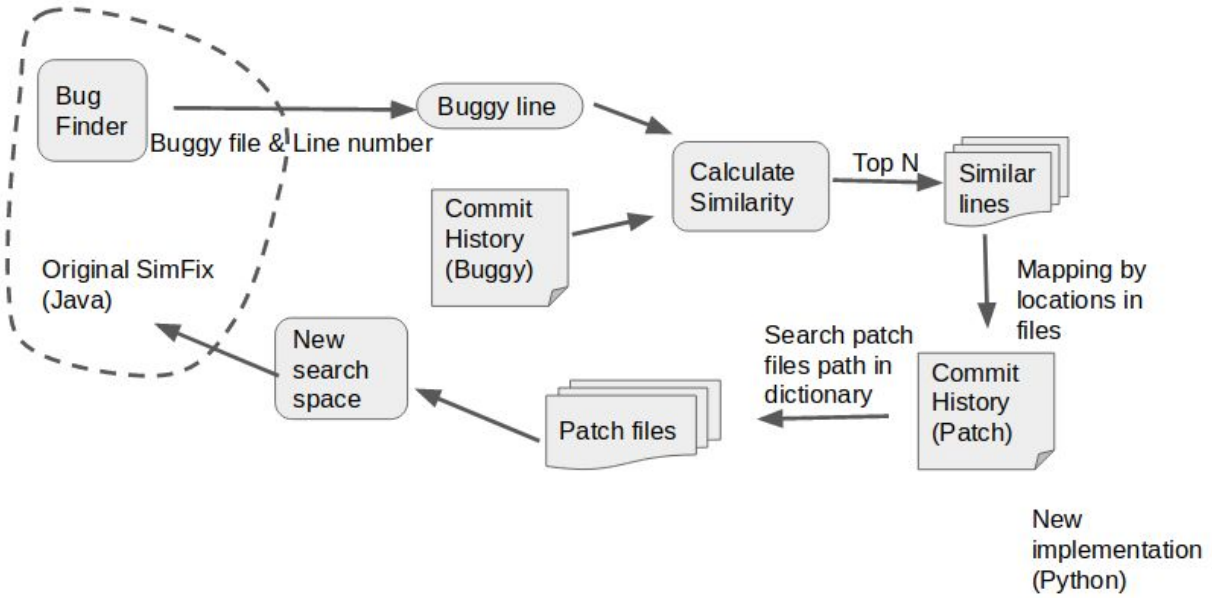


Figure 1. The workflow of the Simfx++

We give our tool a name of “SimFix++” because the tool is developed based on the SimFix[1] tool.

We designed an automatic bug fixing tool that takes a different approach to utilize the existing patch. Previously, SimFix applies those existing patches to reduce the search space of others produced from similar code. Consequently, common bug fixing modification behavior could be prioritized, and the unlikely fix could be prevented. However, only the fixed patch codes are used to determine if the modification is applicable and any relationship between the patch’s buggy code and current buggy code is not captured.

To address this issue, we take an alternative approach and use the existing patch differently. Instead of finding the modifications in patches, we will compare the current buggy code with the buggy code inside the commit history. The basic idea of our model is based on the assumption that “similar buggy code should be repaired in similar ways”. Specifically, the similar buggy code in the large size of commit history should give us at least some hints about how to repair the current buggy lines.

[1] "GitHub - xgdsmileboy/SimFix: Automatically fix programs by ...." https://github.com/xgdsmileboy/SimFix. Accessed 6 May. 2019.

Figure 1 shows the additional implementation of Simfix++. First, for every buggy lines of code, we calculate a similarity score with buggy codes from commit history. For the most N similar ones, we take the corresponding patch codes from commit history and use them as patch candidate for the current buggy code. The candidates are then used as the search space by the original Simfix.

Through this project, we hope to provide a novel way to automatically fix a program and investigate the performance of this technique. Moreover, we want to use this project as a starting point for further research and continue to improve the limitation we faced.

Deliverable

All of our codes and documentation are on a private GitHub repository[2] that is shared with Professor Kaiser and TA.

The dataset we use will upload to a separate private Github repository.

Evaluation

Dataset

We use 68 one-line bugs from Defects4J[3], a Java benchmark collection of bugs from open-source projects, to test the performance of a naive version of SimFix++: [*75 one-line bugs - 7 one-line bugs from Mockito]*. SimFix did not do the experiments on Mockito, so we don't have the material to perform patch validation.

Note: *Our tool should not be limited to one-line bug domain. Using one-line bug here is due to the time limit of this project.*

Performance

In general, SimFix++ can fix 38 bugs in Defects4j in total, which means **four** additional bugs can be fixed by our new implementation. More importantly, this project introduces a novel approach to a new commit-history-based model to build the patch search space, and the overall performance is improved by 12% approximately.

SimFix (original)

---

[2] "Robin-Y-Ding / Repositories · GitHub." https://github.com/Robin-Y-Ding/Simfixplusplus. Accessed 6 May. 2019.
[3] "GitHub - rjust/defects4j: A Database of Real Faults and an ...." https://github.com/rjust/defects4j. Accessed 6 May. 2019.

- Correct(34)

SimFix++

- Original Correct(34)
- Additional Correct(4)
    - chart(1) : 8
    - closure(2) : 92, 93
    - lang(1) : 26

RQ1: Why these four bugs can be fixed by our naive model?

Chart-8

The most similar commit history successfully reveals the bug-fix pattern, which means the result meets the intuition of our model once we assumed. However, be careful about the cheating from the dataset: Do we have an overlap between the test data and the commit history? Will they be the same piece of data?

After carefully analysis, we found that our dataset came from the "training" dataset and "valid" dataset of S. Chakraborty's openNMT github project[4], and the test data came from S. Chakraborty's "test" dataset (as shown below), then because of the standard method to split data in machine learning

Commit history: link

```
1992   { this ( time , RegularTimePeriod.DEFAULT_TIME_ZONE , Locale . getDefault () ) ; }

1992  /zf8/sc2nf/CCRecom_exp/Defects4jProjectPatches/patches/jfree_jfreechart/3045/parent/source_org_jfree_data_time_Week.java
```

The test file: link

```
8   { this ( time , RegularTimePeriod.DEFAULT_TIME_ZONE , Locale . getDefault () ) ; }

8  /zf8/sc2nf/CCRecom_exp/Defects4j/Chart/8/parent/source_org_jfree_data_time_Week.java
```

If the dataset is correct, which means the validate dataset should have no overlap with the test dataset, then our result should be good.

Closure-92,93

The similar commit history did not give us the correct bug-fix pattern, but it gives us correct patches directly.

SimFix++ can fix these two bugs because of the power of both our naive model and the SimFix itself. Specifically, our model brings a similar commit history as a complement of previous

[4] "saikat107 (Saikat Chakraborty) / Repositories · GitHub." https://github.com/saikat107/OpenNMT-py. Accessed 6 May. 2019.

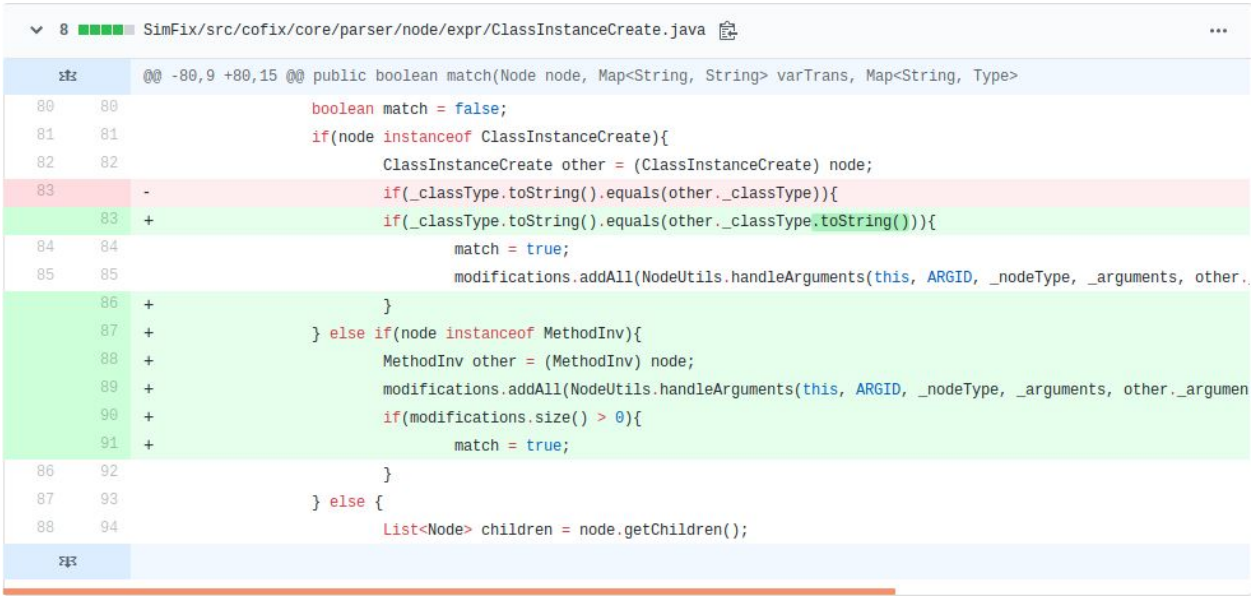
“similar code snippet search space”, which is just the current project. SimFix cannot fix this bug because the bug is in “Closure” project, but the correct patch we generate is from “Lang” project. Then we say our model brings a wider range of search space and more useful information.

Again, we did not find the bug-fix pattern as the intuition of our model once assumed, but combined with the power of SimFix, the model does fix these two bugs.

Lang-26

Instead of using a similar commit history, we also investigate the original Simfix code and found the reason behind its inability to fix certain bugs.

One kind of bugs that original wasn’t able to fix is the new instance initialization, which we thought should be relatively easy to be fixed. After hacking the source code, we found out that SimFix itself has bugs leading to this inability. In addition, we also see that using method to initialize the instance is also not considered as a fix. To resolve those issues, we added a new pattern matching that allows method invocation as a way to initialize an instance as well as fixing the bug inside SimFix. The changes lead to fixing Lang-26.



RQ2: Why other one-line bugs cannot be fixed by our naive model?

Problem:

1. Search space for buggy commit is not big enough: Our model cannot capture the correct bug fixing pattern. Our dataset contains approximately 24000 pieces of commit history, which is a relatively small number if we want to cover most of the bug fix patterns.

Specifically, similar code can be buggy out of several reasons and thus will be fixed in different ways, but the small size of our commit history may not cover the specific pattern which is able to repair the current bug. There are some examples to show this problem: Example1. Example2.

2. Our naive model currently ignores the "context information" of the buggy code, and instead, the model focuses mainly on the context information of "Patches of the similar buggy code". This ignorance has severely weakened the performance because we cannot fix the bug which has wrong variable names, wrong method invocation and other bugs that depend on the context information. There are some examples to show this problem: Example1, Example2

Team contribution

Both Robin and Joe:

1. Debugged the whole SimFix project to:
    a. Find the potential entrance for our new functionalities
    b. Try to find the reasons why SimFix cannot find some specific kinds of bug
2. Developing the idea about how to transform the logical model into plausible implementation and integrate the implementation into original SimFix.

Robin:

1. Implemented the new functionalities of SimFix++.
2. Designed and implemented an interface to link the original java code (from SimFix) with the newly implemented Python parts (SimFix++).
3. Designed and implemented the bugs-analyzing tool. The tool can automatically generate a file (example here) for each bug-repair commit, which contains:
    a. The original buggy line and corresponding patch line
    b. Top N (N can be explicitly indicated or 20 as default) similar buggy lines (with similar scores implicitly) and corresponding patch lines.
    c. The project and file names for the similar commit history.
4. Generated analysis files for 75 one-line bugs[5] in Defects4J database. These files are important for our general analysis of SimFix++'s performance.
5. Constructed a dataset of Defect4J's commit history (Derived from S. Chakraborty's openNMT github project[6]) with a dictionary mapping both the buggy lines and patch lines to their source files. In total, the dataset has 24597 buggy lines and 24597 patch lines.
6. Finished the evaluation of the SimFix++.
7. Found the bug of SimFix: Even feeding with the correct patches, SimFix still cannot fix the Lang-26 bug

---

[5] "GitHub - KTH/sequencer: Sequence-to-Sequence Learning for End-to ...." https://github.com/KTH/sequencer. Accessed 6 May. 2019.
[6] Dataset derived from this project: "saikat107 (Saikat Chakraborty) / Repositories · GitHub." https://github.com/saikat107/OpenNMT-py. Accessed 6 May. 2019.

Joe:

1. Debugged why SimFix couldn't fix Lang-26 bug and applied the appropriate fix, such that instance initialization could be considered as candidate fix.
2. Added an additional pattern in the Instance Initialization to include instances that are created by method invocation (example: Integer a = SomeClass.getInteger(123);).
3. Created the branch in SimFix to allow different techniques to be applied through user-input arguments
4. Investigated how to convert flatted AST tree to AST tree using the Gumtree[7] API and thus enabled the calculation of code difference using PyPI apted[8].

What did I learn?

Robin: This project is really a good start point for my future research. This is the first time that I experienced the process of coming up with a topic that I am interested in and finally making this idea a real implementation. During this semester, I learned 1. how to start an interesting research topic, 2. how to expand the topic by reading others' papers and thoughts, 3. how to effectively communicate with the team members 4. how to transform a logical model into a plausible implementation. I learned how to start research and keep going deeper inside until getting some results, which will make much sense in my future research experiences. Besides, I built a good background in Automatic Program Repair (APR) techniques and went deep inside a pretty good APR tool - SimFix. The background knowledge I gained this semester is a great starting point for my future research in APR-related fields.

Joe: At the start, I didn't know much about the implementation details of automatic program repair and the Simfix paper. Through this project, I had a chance to see how a functional APR connects the idea from its paper to the code level and fixes bugs in real-world projects. It was a lot of code hacking and debugging in order for us to integrate our implementation with SimFix and it was very interesting to understand how each level of the SimFix code works. However, because of the Simfix program structure, directly using Robin's idea was not as easy as we thought and we had to work around that by transforming our dataset and using additional preprocessing. Overall, I had learned a lot about the details of APR and the reasons behind each fixed/unfixed bugs. There are several suggestions that Professor Ray had discussed with us during the meeting and in comparison, it is easy to see why our current model is not working as well as we think, including context and search space limitation. There are still many problems faced by the APR and I hope that this project could be a good starting point for Robin's research.

---

[7] "GitHub - GumTreeDiff/gumtree" https://github.com/GumTreeDiff/gumtree. Accessed 6 May. 2019.
[8] "PyPI documentation" https://pypi.org/project/apted/. Accessed 6 May. 2019.