Efficiently and Precisely Searching for Code Changes with DiffSearch

Luca Di Grazia
University of Stuttgart, Germany
luca.di-grazia@iste.uni-stuttgart.de

ABSTRACT

Version histories of code contain useful information and these data are public, thanks to open source software. However, searching through large repository histories can be complex, because there is no specific tool to search for code changes. This paper presents DiffSearch, the first efficient and scalable search engine for code changes. Given a list of repositories and a query, DiffSearch can retrieve specific code changes in a few seconds. We design a language-agnostic approach that we test on three popular programming languages: Java, JavaScript, and Python, and we design a query language that is an extension of the supported languages. We evaluate DiffSearch in three steps. First, we measure a recall of 81.8%, 89.6%, and 90,4% for Java, Python, and JavaScript, respectively, and an average response time lower than five seconds. Second, we demonstrate its scalability with a large dataset of one million code changes. Last, we perform a case study to show one of its possible applications, where DiffSearch gathers a dataset of 74,903 Java bug fixes.

ACM Reference Format:
Luca Di Grazia. 2022. Efficiently and Precisely Searching for Code Changes with DiffSearch. In *44th International Conference on Software Engineering Companion (ICSE '22 Companion), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3510454.3522678

1 INTRODUCTION

Code version histories store a large amount of useful information about software, such as bug fixes, refactoring sessions, and optimizations. Developers and researchers can use this immense amount of data to improve their work and provide value to the community. For example, we can think about a large company that three years ago made a successful change of APIs in their code base. Now, the new manager decides for a new API transition and the software engineers would like to find the commits from the previous transition to make a similarly excellent job. Moreover, researchers can use specific code changes to make relevant studies on the software engineering field. For example, Wang et al. [17] build a test smell detector for Python. As a result, they need a collection of test smell code changes from other languages. Due to a lack of a search tool for code changes, they realize a manual analysis with a small dataset.

Unfortunately, as we mentioned above, an efficient tool that is aimed to search for code search is not available. A way to search for code changes is to use regular expressions (REGEX) on the commit diff output. However, REGEXs are complex to type for specific patterns and they do not scale well on a large dataset. Another method is the GitHub search feature[1] provided in their web interface. While, this feature searches for different version histories of the projects, it does not consider the code changes.

This paper presents DiffSearch, the first efficient and precise search engine for code changes. DiffSearch takes as input a query that describes a specific change pattern and in a few seconds, it retrieves a list of matching changes from a dataset of version histories. We want to underline three main contributions of this paper. First, we implement a language-agnostic approach that we successfully tested with Java, JavaScript, and Python. Second, we design a query language that is an extension of the supported programming languages, adding special keywords to retrieve specific patterns. Third, the tool scales well on large datasets, retrieving the results in less than five seconds for a dataset of one million changes.

We perform a large evaluation in three steps to check the quality of the results and the performance provided by DiffSearch. First, we measure a recall of 81.8%, 89.6% and 90.4% for Java, Python, and JavaScript, respectively. We also run a sensitivity analysis to show that it is possible to further increase the recall with a slight increase of the query time. Second, we analyze the scalability of the tool with different dataset sizes, concluding that on a large dataset of one million code changes, DiffSearch returns the results in an average of 2.3 seconds. Third, we perform a case study that shows that DiffSearch is able to retrieve 74,903 Java bug fixes. A web interface of DiffSearch is available online[2].

2 METHODOLOGY

2.1 Problem Statement

DiffSearch is a search engine for code search. For this reason, we first define what is a code change and its granularity.

Definition 1 (Code change). A code change consists of two snippets of code, one for the old code and one for the new code. These code snippets are consecutive lines of code extracted from a single hunk computed with a diff between a commit and its parent commit.

Second, we define the input of DiffSearch that consists of a query.
Definition 2 (Query). A query consists of two snippets of code that represent a specific code change pattern, written using the query language designed for DiffSearch.

For example, a query to find a Java API transition from Org.JSON[3] to Google GSON[4] for reading JSON files is:

```
n=new JSONObject(LT<0>);  → n=new JsonParser().parse(LT<0>);
```


ICSE '22 Companion, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9223-5/22/05…$15.00
https://doi.org/10.1145/3510454.3522678

[1] https://github.com/search
[2] http://129.69.217.114/diffsearch
[3] https://github.com/stleary/JSON-java
[4] https://github.com/google/gson

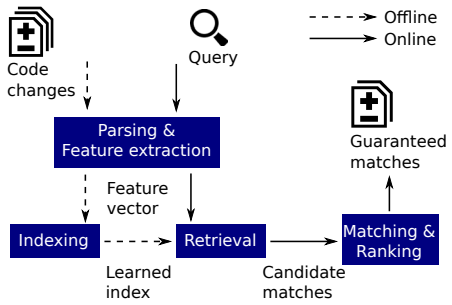

Figure 1: Overview of DiffSearch.

Given these two definitions, the problem is the following:
Definition 3 (Search for code changes). Given a set of code changes as a dataset and a query, find the subset of code changes that match the query. A matching occurs if the code change has the same tree structure or if the query is a subtree of a larger code change.

DiffSearch guarantees that every result of a search precisely matches the query. Based on the query above, suppose we have:

- Code change 1 (Matching):
  `n=new JSONObject("21");` ⟶ `n=new JsonParser().parse("21");`
- Code change 2 (Non-matching, because it is not an API change):
  `n=new JSONObject("22");` ⟶ `n=new JSONObject("23");`

2.2 Approach

The DiffSearch pipeline consists of two main stages as illustrated in Figure 1. First, there is an offline stage where the tool parses code changes,such as Code change 1 (CC1) and Code change 2 (CC2), from GitHub repositories and converts them into feature vectors. Second, DiffSearch has an online phase where it receives a query and retrieves a list of matching code changes.

More in detail, the "Parsing & Feature extraction" component builds the parse tree of the code changes using ANTLR4[5] and then it computes a set of features based on the nodes of the parse trees. For our example, the features encode, e.g., that a Google GSON call replaces a Org.JSON API call and that the literal LT<0> does not change. To enable quick retrieval with large datasets, the "Indexing" component indexes the feature vectors using FAISS [4].

When the index is ready, DiffSearch can start processing the queries in real time. The query goes through the two components described in the previous paragraph and the "Retrieval" component uses the query feature vector to perform a fast nearest neighbor search using FAISS to select k candidate code changes. Our default k is 5,000. However, this component cannot guarantee that all the candidates really match the query, i.e., both CC1 and CC2 are probably retrieved from this component due to their similarity. For this reason, the pipeline has another component called "Matching & Ranking". In this phase, DiffSearch compares all parse tree nodes between the query and the k candidates to guarantee that there are no false positives on the output. This phase is computationally expensive and it thus analyzes only k code changes. For our example, this component eliminates CC2, because it does not contain an API transition, and it returns CC1 as a search result to the user. At the end, DiffSearch shows the list of matching code changes in a ranked list based on the score computed in the "Retrieval" component. The query can also be a subset of a matching code change.

[5]https://github.com/antlr/antlr4

3 EVALUATION

Our evaluation focuses on three main experiments described below.

Recall. As we discussed above, the "Matching & Ranking" component guarantees that there are no false positives in the final results. However, the "Retrieval" component cannot guarantee that there are no false negatives when it retrieves k candidates. For this reason we measure the recall of DiffSearch in three steps. First, we extract three datasets of one million code changes from the top 100 repositories based on GitHub stars for Java, JavaScript and Python, respectively.[6] Second, we select 80 queries from the code changes extracted and we run DiffSearch with the "Retrieval" component. As a result, we process the 80 queries using the "Matching & Ranking" component on the full datasets. This slow experiment gives the 'expected' results for each query, because the "Matching & Ranking" component guarantees the precision. Last, we run DiffSearch with all the components and we compare the new results with the 'expected' ones. On average, across 80 queries and the parameter k=5,000, DiffSearch has a recall of 81.8%, 89.6%, and 90,4% for Java, Python, and JavaScript, respectively, and an average response time lower than five seconds. Increasing k, the recall reaches 89.6% for Java, 93.7% for Python and 95.6% for JavaScript when k=20,000, providing an acceptable answer time lower than 20 seconds.

Scalability. From the previous datasets, we extract 10,000 to 1,000,000 changes for the three languages supported to create ten smaller datasets. Using the same 80 queries, DiffSearch has an average answer time between 0.5 and 2 seconds without significant differences among the different datasets and the three languages.

Case study. In this case study we use DiffSearch in a real case scenario to build a dataset of Java bug fixes. We use twelve bug patterns defined by prior work [5]. DiffSearch gathers a dataset of 74,903 changes among these patterns. Computing the intersection with the results retrieved by SStuBs, DiffSearch finds 79.2% of their changes, a result consistent with the Java recall computed above.

4 RELATED WORK

Code search engines are popular among developers to find code based on method signatures [14], existing code examples [6], or natural language queries [3]. Our approach relates to works on searching for code that retrieve code snippets that match keywords [2, 3], test cases [14], or partial code snippets [8], but DiffSearch works with code changes. Moreover, there are works on mining repositories leverage version histories to extract repetitive changes [11, 13], predict changes [16], predict bugs [7], or to learn API usages [12]. These approaches usually work with all the code changes in a dataset, they support limited patterns [9], or they do not scale on large datasets [10]. DiffSearch can be useful to build large datasets to train learning approaches, e.g., fixes of particular bug patterns, how to apply them for automated program repair [1, 15].

5 CONCLUSION

We present DiffSearch,[7] a search engine for code change that can retrieve code changes, matching a given query in a few seconds. Moreover, we design this tool for developers and researchers that want to analyze large version histories of code in an efficient way.

[6]The offline stage can take up to four hours to build an index of one million changes.

[7]This work was supported by the European Research Council (ERC, grant agreement 851895), and by the German Research Foundation (ConcSys and Perf4JS projects).

REFERENCES

[1] Johannes Bader, Andrew Scott, Michael Pradel, and Satish Chandra. 2019. Getafix: Learning to Fix Bugs Automatically. In *OOPSLA*. 159:1–159:27.

[2] Sushil Krishna Bajracharya, Trung Chi Ngo, Erik Linstead, Yimeng Dou, Paul Rigor, Pierre Baldi, and Cristina Videira Lopes. 2006. Sourcerer: a search engine for open source code supporting structure-based search. In *Companion to the 21th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2006, October 22-26, 2006, Portland, Oregon, USA*, Peri L. Tarr and William R. Cook (Eds.). ACM, 681–682. https://doi.org/10.1145/1176617.1176671

[3] Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. 2018. Deep code search. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman (Eds.). ACM, 933–944. https://doi.org/10.1145/3180155.3180167

[4] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* (2019).

[5] Rafael-Michael Karampatsis and Charles A. Sutton. 2019. How Often Do Single-Statement Bugs Occur? The ManySStuBs4J Dataset. *CoRR* abs/1905.13334 (2019). arXiv:1905.13334 http://arxiv.org/abs/1905.13334

[6] Kisub Kim, Dongsun Kim, Tegawendé F Bissyandé, Eunjong Choi, Li Li, Jacques Klein, and Yves Le Traon. 2018. FaCoY: a code-to-code search engine. In *Proceedings of the 40th International Conference on Software Engineering*. 946–957.

[7] V. Benjamin Livshits and Thomas Zimmermann. 2005. DynaMine: Finding common error patterns by mining software revision histories. In *European Software Engineering Conference and Symposium on Foundations of Software Engineering (ESEC/FSE)*. ACM, 296–305.

[8] Sifei Luan, Di Yang, Celeste Barnaby, Koushik Sen, and Satish Chandra. 2019. Aroma: Code recommendation via structural code search. *Proceedings of the ACM on Programming Languages* 3, OOPSLA (2019), 152.

[9] Fernanda Madeiral, Thomas Durieux, Victor Sobreira, and Marcelo Maia. 2018. Towards an automated approach for bug fix pattern detection. *arXiv preprint arXiv:1807.11286* (2018).

[10] Matias Martinez and Martin Monperrus. 2019. Coming: A tool for mining change pattern instances from git commits. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 79–82.

[11] Stas Negara, Mihai Codoban, Danny Dig, and Ralph E Johnson. 2014. Mining fine-grained code changes to detect unknown change patterns. In *Proceedings of the 36th International Conference on Software Engineering*. 803–813.

[12] Anh Tuan Nguyen, Michael Hilton, Mihai Codoban, Hoan Anh Nguyen, Lily Mast, Eli Rademacher, Tien N Nguyen, and Danny Dig. 2016. API code recommendation using statistical learning from fine-grained changes. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 511–522.

[13] Hoan Anh Nguyen, Tien N. Nguyen, Danny Dig, Son Nguyen, Hieu Tran, and Michael Hilton. 2019. Graph-based mining of in-the-wild, fine-grained, semantic code change patterns. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*. 819–830. https://doi.org/10.1109/ICSE.2019.00089

[14] Steven P. Reiss. 2009. Semantics-based code search. In *31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Proceedings*. IEEE, 243–253. https://doi.org/10.1109/ICSE.2009.5070525

[15] Reudismam Rolim, Gustavo Soares, Rohit Gheyi, and Loris D'Antoni. 2018. Learning Quick Fixes from Code Repositories. *CoRR* abs/1803.03806 (2018). arXiv:1803.03806 http://arxiv.org/abs/1803.03806

[16] Michele Tufano, Jevgenija Pantiuchina, Cody Watson, Gabriele Bavota, and Denys Poshyvanyk. 2019. On learning meaningful code changes via neural machine translation. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*. 25–36. https://dl.acm.org/citation.cfm?id=3339509

[17] Tongjie Wang, Yaroslav Golubev, Oleg Smirnov, Jiawei Li, Timofey Bryksin, and Iftekhar Ahmed. 2021. PyNose: A Test Smell Detector For Python. *arXiv preprint arXiv:2108.04639* (2021).