University of Zurich[UZH]

Department of Informatics

University of Zürich
Department of Informatics
Binzmühlestr. 14
CH-8050 Zürich
Phone. +41 44 635 43 11
Fax +41 44 635 68 09
www.ifi.uzh.ch/dbtg

UZH, Dept. of Informatics, Binzmühlestr. 14, CH-8050 Zürich

Prof. Dr. Michael Böhlen
Professor
Phone +41 44 635 43 33
Fax +41 44 635 68 09
boehlen@ifi.uzh.ch

Zürich, 14. September 2026

BSc Thesis (18 points)
Topic: A Similar-File Index for the Software Heritage Graph

Background and Motivation

Software Heritage is an archive of the source code and commit history of public software projects [6]. It stores every version of every file once, under the hash of its bytes, and it records which files each commit contains. Our group has tools that answer, for a file, which commits contain it and which commits introduced it [7, 2]. They recognise a file only when its bytes are identical. Files change, though. They are edited from commit to commit, copied into other projects, patched. To follow where a piece of code comes from, one needs the files in the archive that are similar to a given file, not only the identical ones. This thesis builds an index that returns them, on a subset of the archive.

What counts as similar

Take a file, remove blank lines and the spaces at the start and end of every line, and cut it into pieces of three consecutive lines. Lines 1 to 3 form the first piece, lines 2 to 4 the second, and so on, so a file of 200 lines gives 198 pieces. The similarity of two files is the number of pieces they have in common, divided by the number of different pieces the two files have together. It lies between 0 and 1 and is called the Jaccard similarity. Suppose five lines of the 200 are edited. Each line sits in three pieces, so about fifteen pieces change. The two versions then share 183 pieces and have 213 different pieces together, and their similarity is 183 divided by 213, about 0.86. A copy in which half of the lines were rewritten lands around 0.3. We call two files similar when their similarity is at least 0.5. Git uses the same threshold when it decides whether a renamed file is still the same file [3].

How the index works

Comparing a file with every other file is far too slow for millions of files, so every file gets a short fingerprint instead. Take a hash function, apply it to every piece of the file, and keep

the smallest value. Do this with 128 different hash functions, and the file has a fingerprint of 128 numbers. This is MinHash [1, 4]. Its key property is easy to see. Two files have the same smallest value under a hash function exactly when the piece with the smallest value, among all pieces of both files, is one they have in common. The chance of that is the similarity. So the share of the 128 positions on which two fingerprints agree is an estimate of the similarity of the two files, and the files themselves need not be read.

Comparing a fingerprint with every other fingerprint would still be too slow, so the fingerprint is cut into blocks, for example 32 blocks of 4 numbers. All files whose 4 numbers in a block are equal go into the same bucket. Two files with similarity 0.5 share a bucket with probability 0.87, two files with similarity 0.3 with probability 0.23, and unrelated files almost never. A query for a file reads only the 32 buckets of that file, compares its fingerprint with the fingerprints found there, and keeps the files that agree on at least half of the positions. The index consists of two tables on disk. One maps a file, by its identifier in the archive, to its fingerprint, and the other maps a bucket to the files in it. A new file is fingerprinted once and appended to both tables, so the index grows by a fixed number of bytes per file and is never rebuilt.

How we check that it works

There are two checks. First, for a sample of 500 files, the similar files are computed exactly by comparing the file with every other file. That is slow but correct, and the index's answer is compared with it. Second, the commit history tells us which files are versions of each other. Suppose `query.py` has content A when it is added, and becomes B, C, and D after three edits. The Git log records exactly that. So for the query B the files that should be found are A, C, and D. A is one edit away from B, C is one edit away, and D is two. We call that the distance. When content A also appears at another path, it has been copied, and the versions that follow at the other path count as well. These versions and copies, read from the history, are the ground truth. For the 500 sample files we count how many of them the index finds. That share is the recall. It is reported by distance, because versions far apart in the history have drifted away from each other and are not similar in our sense anymore, so no index can find them. On Django, 92% of consecutive versions are similar, and ten versions apart 60% still are. The files the index returns although they are not similar are counted too, with the exact similarity.

Data and what is prepared

Stage one is Django, a Python web framework, with 51,000 commits and 90,000 versions of Python files. Stage two is the popular-500-python subset of the archive, 443 popular Python projects with 17.7 million files, which is on our server together with local Git clones and our commit tools. The Django files, the ground truth from the history, the exact answers for the 500 sample files, and an evaluation script are prepared in advance, so the student starts directly with the algorithm. The reading is Chapter 3 of [4], which explains fingerprints and buckets with worked examples, the original paper on MinHash [1], and [5], which shows that keeping only a few bits of every number in the fingerprint is enough. The student needs Python and Git.

Plan (15 weeks, five blocks)

Block	What is done	Delivered, and done when
1 Weeks 1–2 Understand	The student reads Chapter 3 and the MinHash paper, loads the prepared files and ground truth, and reproduces the evaluation of the prepared exact answers. Then the Django files are looked up in the archive by their hash, and our commit tools are run on a few of them.	A reproduced evaluation table, one page on MinHash in own words, and a script that prints the containing and introducing commits of a file. *Done when* the student can explain on one file why near versions are found and far ones are not.
2 Weeks 3–4 First index	Fingerprints and buckets are built with a ready-made Python library over all Django files, and the first 100 sample queries are answered. Then the number of hash functions, the bits kept per number, and the block size are varied.	The first recall table by distance, a table of settings with bytes per file, and the chosen settings. *Done when* the recall at distance 1 reaches 90% of what the exact comparison finds.
3 Weeks 5–7 Own index	The library is replaced by own code for fingerprints and buckets. The two tables go to disk, appending and querying are implemented, and buckets that grow huge, such as those of near-empty files or licence texts, are cut off at a fixed size. The full evaluation runs on all 500 queries.	A documented tool with build, append, and query, and the Django numbers: recall by distance, bytes per file, build time, and query time. *Done when* own code gives the same answers as the library and a query takes milliseconds.
4 Weeks 8–10 Whole subset	The Python files of all 443 projects are taken from the clones and looked up in the archive, and copies across projects are added to the ground truth. The index is built over all of them in parallel. Queries run on the full index, and the index is connected to our commit tools.	The index over the subset with its numbers, and for 100 files the commits that introduced a similar version. *Done when* the build runs end to end and the numbers stand next to Django's in one table.
5 Weeks 11–15 Writing	Week 11 is analysis, figures, and the outline. Weeks 12 and 13 produce the complete draft. Week 14 is the revision after the supervisor's comments, and week 15 the submission.	The thesis.

University of Zurich
Department of Informatics

Prof. Dr. Michael Böhlen
Professor

Supervisor: Mateo Dujić (`dujic@ifi.uzh.ch`)

Start date: October 1, 2026

End date: January 15, 2027

References

[1] Andrei Z. Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences (SEQUENCES)*, pages 21–29. IEEE, 1997.

[2] Mateo Dujic, Michael Böhlen, Sven Helmer, and Oded Lachish. Determining file provenance in directed acyclic graphs using partial sources, 2026. Under submission.

[3] Git project. Git reference manual: git-diff, option `-M` (rename detection). `https://git-scm.com/docs/git-diff`. Accessed September 2026.

[4] Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, 3rd edition, 2020. Chapter 3: Finding Similar Items.

[5] Ping Li and Arnd Christian König. b-bit minwise hashing. In *Proceedings of the 19th International Conference on World Wide Web (WWW)*, pages 671–680, 2010.

[6] Antoine Pietri, Diomidis Spinellis, and Stefano Zacchiroli. The Software Heritage graph dataset: Public software development under one roof. In *Proceedings of the 16th International Conference on Mining Software Repositories (MSR)*, pages 138–142, 2019.

[7] Guillaume Rousseau, Roberto Di Cosmo, and Stefano Zacchiroli. Software provenance tracking at the scale of public source code. *Empirical Software Engineering*, 25(4):2930–2959, 2020.