Line Tracker – Tracking Source Code Lines

Ahmed Shuaib (110184126), Elliot Adams (110142940)
Department of Computer Science
University of Windsor

December 10, 2025

Abstract - This project investigates the problem of tracking source code lines across different versions of a file. Using concepts from language independent diffing techniques presented in prior work, this project provides an approach using techniques such as preprocessing, similarity-based candidate selection, and unchanged line detection. A dataset of 23 file pairs with 556 total line mappings was used. The method identified most mappings and demonstrated pros and cons of hybrid strategies.

Keywords - line tracking; diff; file comparison; source code; simhash.

Introduction

In order to properly advance a software through development phases, tracking the evolution of source code is critical. It can be tough for developers to know what specific lines or sections of code has been altered. Traditional tools such as Unix diff can fail in cases of reordered lines or small text modifications. Using a language independent hybrid diff can improve accuracy with strategies like context similarity.

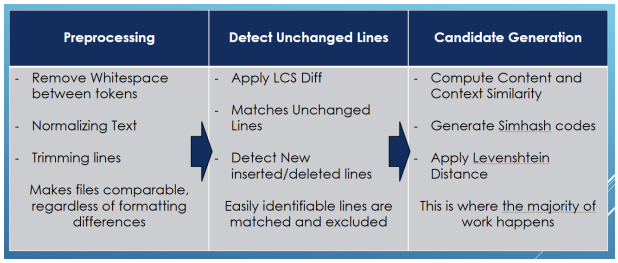
In this project, we attack the same problem. We decided to design a method modeled after the LHDiff workflow, making use of strategies such as preprocessing, unchanged line detection, simhash, and similarity scoring. In order to validate our algorithm, we used a dataset of pairs of files with subtle and large changes specifically designed to test the algorithm. Our results show that a hybrid technique such as this can correctly determine most line mappings properly. The technique successfully identified the majority of line changes.

Data Collection

To evaluate our approach, we used a dataset containing 25 pairs of source code files, each consisting of an original version and a modified version. The files were chosen regardless of language, with differed typical developer edits, and with diversity among file size and change complexity.

Technique Description and Evaluation

Whether it be tracking team changes in a python project, debugging java code, or even just visualizing differences between regular text files, tracking changes between versions of anything is important. This is why it is vital to use an approach that is reliable and language independent. We designed our method as such. The following section will introduce the initial instructions of our algorithm, explain the main process of matching lines, and delve into the complicated methods that we utilized to create our algorithm. Figure 1 displays a table with summarized information of the process.



Step 1: Preprocessing:

Each line is normalized by Removing excessive whitespace, Lowercasing for language independent comparison, Trimming lines

In order to compare files regardless of formatting, you must remove the formatting. It is necessary to create an environment where two files can be compared without the need for extra function of ignoring any formatting. In order to avoid any extra function, we entirely remove the formatting by deleting whitespace between tokens, normalizing text by making it all lowercase, and trimming lines by removing indents.

Step 2: Detecting Unchanged Lines:

For the next step, we reduce easy bulk line matches by: Applying LCS Diff, Matching lines that appear unchanged, Matching lines that swapped positions

For this step, we apply a LCS (Lowest common subsequence) Diff algorithm to match lines that are easy to match. Lines that are easy to match include unchanged lines, because the LCS Diff algorithm can easily detect them. Secondly, lines that are swapped are also matched by the LCS Diff algorithm because their content is still exactly the same. Using this method, the bulk of lines that can easily be matched are removed from the pool so the more complicated functions can run smoother.

Step 3: Candidate Generation Using Hybrid Similarity:

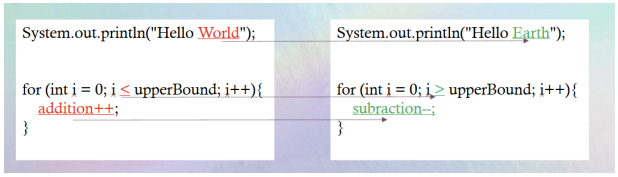
For remaining unmatched lines, we compute: Content Similarity, Context Similarity, Simhash values for each line, Levenshtein Distance

From whatever is left after running LCS Diff, we evaluate similarities using strategies such as content similarity, which uses Levenshtein distance to calculate how similar a line is to another edited version of the same line. We also test context similarity, which compares the lines around the line we are checking, this is useful for things like detecting line changes within function blocks. During this process, a Simhash value is applied to each line to keep track of it. With the addition of a Levenshtein distance "score", we can make thresholds where if a line is similar enough to another, we match it, and if it is too different, we treat is as an entirely different line. Using this criteria, a line is evaluated and the highest scoring candidate is selected as the match to it.

After testing, our results have proven the effectiveness of our methods to a greater than satisfactory degree. Out of 23 sets of files, our algorithm correctly matched 503/556 of the lines in 100 seconds. That is an average of 21.8/ 24.2 lines per pair of files at 4.34s each. This extends full confidence in our method, proving the validity of language independent diff algorithms.

Presentation of Line Mapping Information

For the visualization of the line mappings, we would create a GUI that allows users to see lines from the original file, with an indicator showing the movement/changes that the line underwent in the second file. Figure 2 displays an example.



Notice the underlined text in file 1 is red and the underlined text in file 2 is green. This allows the user to see what has changed, and also creates an indicator difference between the first file (red) and the second file (green). There is also an arrow indicator pointing from the line from file 1 to the line from file 2. If it were implemented, the line could also point up or down in case of line movements. The settings for a GUI such as this could be edited at the user's discretion.

Conclusion

Our project on language independent line tracking has proved valuabale. Using strategies from other existing diff software such as UNIX Diff and LHDiff such as Levenshtein distance and Simhash, we were able to create an algorithm that can match lines from one file to another no matter the code language. Our algorithm successfully matched 503/556 lines in 100 seconds while testing with 23 sets of files. Overall, this project was a great learning experience and developed our skills as programmers part of a group.

References

[1] Muhammad Asaduzzaman, Chanchal K. Roy, Kevin A. Schneider, Massimiliano Di Penta, "LHDiff: A Language-Independent Hybrid Approach for Tracking Source Code Lines."

[2] G. Canfora, L. Cerulo and M. Di Penta, "Tracking Your Changes: A Language-Independent Approach," in IEEE Software, vol. 26, 2009.

[3] Spacco and C. Williams, "Lightweight Techniques for Tracking Unique Program Statements," 2009 Ninth IEEE International Working Conference on SCAM, 2009.