

LHDiff – A technique to track code changes

Albert Qyli, University of Windsor Computer Science
Salman Youssef, University of Windsor Computer Science
Omar Kayali, University of Windsor Computer Science

Badr Kayyali, University of Windsor Computer Science
Youssef Teffahi, University of Windsor Computer Science

Abstract

The goal of our project was to develop a technique called LHDiff that detects line correspondences between two source files. It considers structural, conceptual, and contextual similarities and differences to determine how lines evolve between versions. We designed our model by analysing existing tools such as BEST, Git, SCAM, and Diff, and incorporating their most effective concepts into our approach. Our evaluation across 25 pairs of files showed that LHDiff correctly identified approximately 92% of line mappings, demonstrating that it is effective for tracking source code evolution.

Keywords

Line mapping, cosine similarity, Sim hash, Levenshtein difference, LHDiff

Introduction

The purpose of LHDiff is to solve the problem of tracking how individual source lines evolve between two versions of a file. This is important because refactoring is a common practice in software engineering and observing how lines change over time can support debugging, maintenance, and code reviews. Combining three similarity signals in content similarity (using Levenshtein difference), structural similarity (using shape tokens), and context similarity (using cosine similarity) helped make our LHDiff technique effective and accurate. Our model maps which lines are identical as well as lines that are split and merged.

Data Collection

We took it amongst ourselves to each be responsible for 5 pairs of “old” and “new” files and present them in our GitHub to have it one common area. We created these source files ourselves, creating a range of simple to complex files, as well as each file containing different functionalities in each. We determined the line mappings by running our LHDiff model on

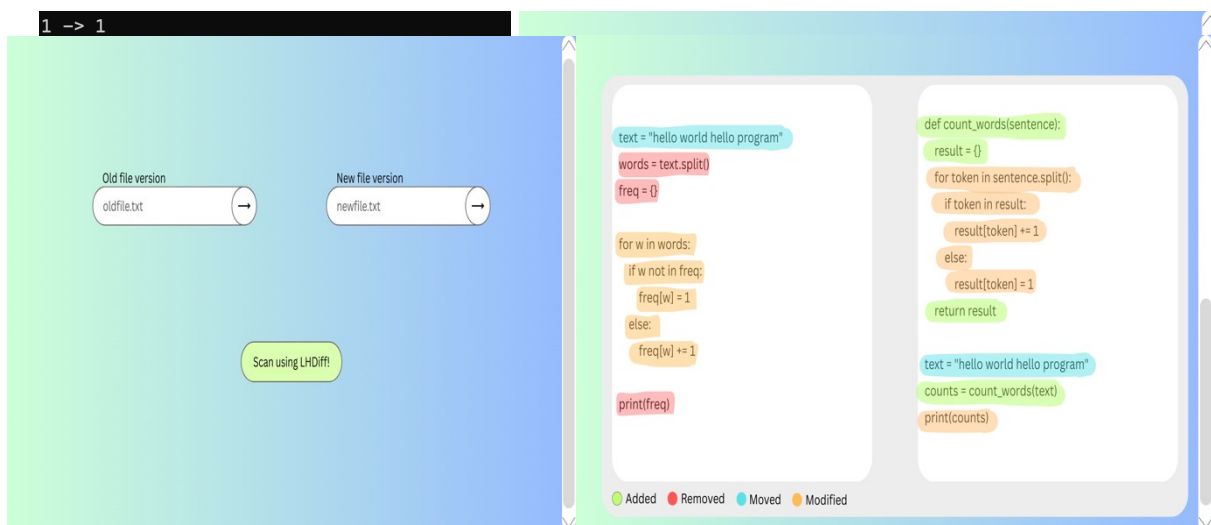
our local terminal and the output was shown as old -> new for matched pairs, and there was a section for unmatched deletions as well as unmatched additions.

Technique Description and Evaluation

Throughout the development of our model, we weighed several options on which is most effective and efficient for our model. Normalization which includes removing punctuation, ignoring blank lines, lowercasing everything, and collapsing whitespace was determined as an important first step. We determined exact matches using `find_unchanged`, and similar matches using similarity methods as previously discussed. To detect split lines, our model analyses the lines before and after the exact match of the new file. If the similarity between the old line and the neighbouring new line is greater than the split threshold, the new line is added to the mapping. To detect merged lines, our model works inversely of the line split method. It evaluates the lines before and after the exact match of the old file and if their similarity to the same new line exceeds the merge threshold, they are also mapped to that line. After several experimental trials, we found our model to be approximately 92% accurate. To calculate accuracy, we compared the mappings produced by LHDiff against manually verified ground truth and computed the ratio of correctly identified mappings over the total number of mappings.

Presentation of Line Mapping Information

If you were to run LHDiff on your local terminal, an example run would be “python3 LHDiff/LHDiff.py our-dataset/file17-old.txt our-dataset/file17-new.txt”. This would create an output as pictured below to the left with line mappings as well as unmatched deletions and additions. We also created a visualization of a GUI that prompts the user to enter the “old” and the “new” file in their respective fields as shown in the picture. If we were to have an application of LHDiff, it’d show the lines correspondence by a colour coded scheme. Green lines indicate they were added, red lines indicate they were moved, blue indicate they were moved, and yellow indicate they were modified.



Technique Research and Evaluation

Throughout this project, we took the liberty of learning each of the past techniques used to give ourselves a better understanding of everything.

Unix diff: Traditional Line Matching

Unix diff uses the Longest Common Subsequence algorithm. It steps through two files, compares each line, and highlights anything added, removed, or altered. Strengths consist of its speed and predictability; weaknesses consist of its inability to track moved code and illegible output during scattered edits.

W_BEST_LINE: Similarity-Based Matching

W_BEST_LINE searches for the closest matching line in the new file for each line in the old one. It scores similarity using normalized text and character comparisons and is helpful when lines get shifted around. Its strengths consist of spotting lightly edited or renamed lines, as well its overall stronger detection of modification than Unix Diff; however, it struggles with working fast on large files and still focuses on text rather than structure.

LDiff: Hybrid, Multi-Step Analysis

LDiff mixes ideas from Myers diff, range grouping, TF-IDF scoring, and Normalized Levenshtein Distance. It was designed for research on software evolution and targets a broad view of changes. It works well in detecting moved blocks and works across many languages but struggles with line splitting detection and slows down in larger ranges.

SDiff and Git

SDiff's behaviour suggests some structural awareness, possibly through tree-based methods. However, Git diff stays close to Unix diff in design. Small improvements come from Git's internal optimization tricks rather than fundamental algorithmic changes.

As we progressed in our research and understanding of these techniques, we saw that Unix diff was directed towards a shift in sequence matching, W_BEST_LINE was directed towards a shift in similarity scoring, and LDIFF in both structural and textual signals. All in

all, we found the research of these techniques highly beneficial in our overall understanding and development of our LHDiff model.