Test Task for Project
Logical coupling analysis in Git repositories

Roman Kovalev

May 2024

Introduction

The test task for this project involves developing a tool to calculate pairs of developers who contribute most frequently to the same files/modules in a GitHub repository. This report describes what has been implemented for the test task, but its main value lies in discussing what else can be done for this problem.

Problem Analysis

So, we have a GitHub repository. This means we have a list of commits in the repository with all the contained information: commit date, author, and modified files.

To begin, it makes sense to limit the number of commits we will analyze. This is necessary because the idea of "grouping developers by similarity"implies up-to-date data based on the set of active developers and the interests of current project participants. Additionally, it's worth mentioning that repositories of large projects include dozens or even hundreds of daily commits, which can provide us with a lot of irrelevant information. Since we are interested in current information, it is reasonable to limit ourselves to commits from the past few days.

Now that we have defined our input data, let's introduce a numerical estimation of the "similarity"between two developers based on the files/modules they have modified. This can be done in several ways, but we will choose the simplest: the number of files where changes were made by both the first and second developers.

We can then consider a complete weighted graph where vertices represent developers and edge weights represent the number of shared files for each pair.

Then, we can rephrase the original problem as follows: *Find the maximum weight matching in a complete weighted graph with no negative weights.*

Implemented Solution

In this task, I implemented a greedy algorithm that, at each iteration, selects a pair of developers who are most similar among all available developers.

The time complexity of this algorithm is $O(n^3)$.

Unfortunately, the greedy algorithm does not guarantee that the found matching will have the maximum weight, but we will obtain some pseudo-optimal division.

Exact Solution

Let's reduce our problem to the Assignment Problem.

Assignment Problem:
Given a bipartite weighted complete graph. We need to find a maximum weight matching.

Let the original graph be represented as: $G = \{V,\ E = V \times V,\ f : E \to \mathbb{Z}_+\}$ Let's duplicate each vertex - set V' - and consider graph H:

$$H = \left\{V \cup V',\ E',\ g : E' \to \mathbb{Z}_+\right\}$$

The weight of an edge in graph H between two vertices is zero if they both belong to the same set V or V', and is equal to the weight of the edge between their preimages in the original graph otherwise. The weight of an edge between copies of the same vertex is also zero.

Now we have a bipartite weighted complete graph $H_{n,n}$ vertices. If we find a solution to the Assignment Problem on this graph, we will straightforwardly restore a maximum weight matching on the original graph.

It is worth mentioning that the Assignment Problem is solvable in $O(n^4)$. The reduction of the original problem to the Assignment Problem works in $O(n^2)$, so the best division into pairs can be achieved in $O(n^4)$.

In the paper, it is shown that there are randomized solutions for the Assignment Problem with worse asymptotics but better practical performance. In our case, it would be reasonable to use one of these solutions.

Algorithm Demonstration

A public repository of the library pytorch was used as the test repository.

1. Run the file *Main.py*.
2. Specify the full path to the cloned repository (*/home/roman-not-hehe/Desktop/sandboxes/pytorch*).
3. Specify the number of days to analyze the repository.
4. Obtain information about the number of commits by authors who made changes to the repository during this time and the total number of commits.
5. Since the pairs obtained might likely be "leftover"pairs of developers, meaning their "similarity"is zero, we need to choose whether to display these pairs on the screen or not.
6. Next, the program displays pairs of developers with the number of files they both modified.
7. Finally, the program outputs the average number of shared files per pair, as well as their median value.

The first example

The second example