An analysis of students solutions on Algorithm and Data Structure programming exercises

Francisc-Sebastian Kalciov
West University of Timişoara
Department of Computer Science
Bachelor Study Program: Artificial Intelligence
Email: francisc.kalciov04@e-uvt.ro

Timişoara 2026

Contents

List of Figures

List of Tables

1 Introduction

In this paper, programming assignment solutions are analyzed in order to identify patterns of high similarity that may indicate potential plagiarism or similar approaches on how problems are solved.

The main objective is to explore how similarity scores are distributed across tasks and academic years, and to determine whether certain assignments or years show higher levels of similarity than others.

1.1 Problem

Programming assignments often lead to solutions that look very similar, especially when the problem requirements are strict or when standard algorithms are expected [SKS+20].

This makes it difficult to distinguish between normal similarity caused by common approaches and cases where solutions may be very similar.

1.2 Objectives

The main objective of this project is to explore similarity patterns in student programming solutions using data analysis and visualization techniques. This includes preprocessing and normalizing code, extracting simple metrics such as lines of code (LOC), and computing similarity scores between solutions using TF-IDF[1] vectorization and cosine similarity.

Another objective is to use visualizations to better understand how similarity behaves across different academic years and assignments. Static plots are used to summarize distributions and trends, while an interactive web-based visualization is designed to allow deeper exploration of highly similar solution groups. The final objective is to represent some of the results through a simple interactive dashboard implemented with D3.js and integrated into a React-based[2] web application deployed on Vercel[3].

2 Dataset Description

The dataset is from University of Hamburg, "Dataset of student solutions to algorithm and data structure programming assignments"[4] [PFSK+22], which consists of multiple CSV files containing task descriptions and solutions from students across three academic years (See Table: 1).

[1] TF-IDF (Term Frequency-Inverse Document Frequency): A statistical vectorization technique used in NLP to convert raw text into numerical format

[2] React: Open source JavaScript library used for building interactive UIs based on reusable components `https://react.dev/`.

[3] Vercel: A cloud platform that provides infrastructure for building, deploying and scaling web applications `https://vercel.com/`.

[4] `https://www.inf.uni-hamburg.de/en/inst/ab/lt/resources/data/ad-lrec`

All files that are part of the dataset are stored as CSV files and loaded from Google Drive into dataframes. During preprocessing, missing values (for example, missing code) are handled first, duplicate rows are removed, and code snippets are normalized to reduce formatting differences. This ensures that further analysis focuses more on logic.

Google Drive is used to store the dataset mainly because the analysis is completed on Google Colab[5] in Python, as it provides a runtime environment suited for visualizations, Machine Learning and a more structured view of the project [SS23].

Table 1: Dataset composition across three academic years.

Year	Task Descriptions File	Solutions File
Year 1	`year-1-task-descriptions.csv`	`year-1-solutions.csv`
Year 2	`year-2-task-descriptions.csv`	`year-2-solutions.csv`
Year 3	`year-3-task-descriptions.csv`	`year-3-solutions.csv`

2.1 Dataset structure

The following tables represent the structure of the dataset used for the analysis presented in this paper.

Table 2: Structure of the `year-1/2/3-solutions.csv`

Column Name	Description
`question_id`	Unique identifier for the programming task.
`student_id`	Anonymous identifier for the student.
`solution`	The source code submitted by the student.

Note: This file contains 541 rows and 3 columns.

[5]Google Colab: A free cloud-based development environment based on Project Jupyter `https://colab.research.google.com/`

Table 3: Structure of the `year-1/2/3-task-descriptions.csv` dataset.

Column Name	**Description**
`id`	Unique identifier for the task, matching `question_id` in solutions.
`title_ger`	German title of the task.
`title_eng`	English title of the task.
`task_description_tex_ger`	Task description in German, formatted for LaTeX.
`task_description_plain_ger`	Plain-text version of the German task description.
`task_description_plain_eng`	Plain-text version of the English task description.
`skeleton`	Code skeleton/template provided to students.
`tester`	Testing code or framework used to evaluate solutions.
`answer`	Reference solution or expected output code.

Note: This dataset contains for each year (1,2,3) 20, 10 and 12 rows with 9 columns.

In Table 4 is an example of a task description structured like in Table 3

Table 4: Example entry from `year-1-task-descriptions.csv`

Field	Value
`id`	`19_20-1-1-java`
`title_ger`	LCM – Das größte gemeinsame Vielfache (Java)
`title_eng`	LCM – Least Common Multiple (Java)
`task_description_plain_eng`	Similar to the already known "Greatest Common Divisor" (GCD), the Least Common Multiple (LCM) is also used in elementary mathematical applications, for example in the multiplication of heterogeneous fractions. This has applications ranging from the adaptation of cooking recipes to other portion quantities to the calculation of planetary orbits. The LCM is defined as follows: $lcm(n_1, n_2) = \frac{n_1 \cdot n_2}{gcd(n_1, n_2)}$. Implement a function that calculates the LCM of two given values. It may be based on the GCD implementation from the lecture. Note: The use of "import" is not allowed.
`skeleton`	See Listing 1
`answer`	See Listing 2

Note: This entry is not complete. Some fields are skipped.

Listing 1: Skeleton code for the example in Table 4

```
1  int lcm(int a, int b) {
2      return null;
3  }
```

Listing 2: Correct answer for the example in Table 4

```
1  int gcd(int a, int b) {
2      if (a < b) {
3          int tmp = a;
4          a = b;
5          b = tmp;
6      }
7      int r = a % b;
8      if (r == 0) {
9          return b;
10     }
11     else {
12         return gcd(r, b);
```

```
13      }
14  }
15
16  int lcm(int a, int b) {
17      return (a * b) / gcd(a, b);
18  }
```

3 Methodology

The project is implemented in Python using Pandas for data handling, Matplotlib and Seaborn for visualization, and scikit-learn for machine learning components. The preprocessing stage includes:

- Filter irrelevant columns.
- Handle missing values.
- Normalize code by removing unnecessary whitespaces, empty lines and inconsistent spacing.

After preprocessing, the number of lines of code (LOC) is computed for each solution, which is later used for further analysis.

All solutions from different years are then merged into a single dataframe. To compare solutions, TF-IDF vectorization is applied to the normalized code, using a custom tokenizer designed for code tokens.

Cosine similarity is computed between all solution pairs, and pairs above a high similarity threshold are flagged for further analysis.

In addition to similarity analysis, clustering is performed using K-Means, and anomaly detection is performed with Isolation Forest.

Finally, t-SNE is used to transform high-dimensional TF-IDF vectors into two dimensions for visualization.

4 Results and Visualizations

Several types of visualizations are used throughout the project.

First, box plots show how the number of lines varies across the three academic years (See Figure 1). This helps compare solution complexity between groups.

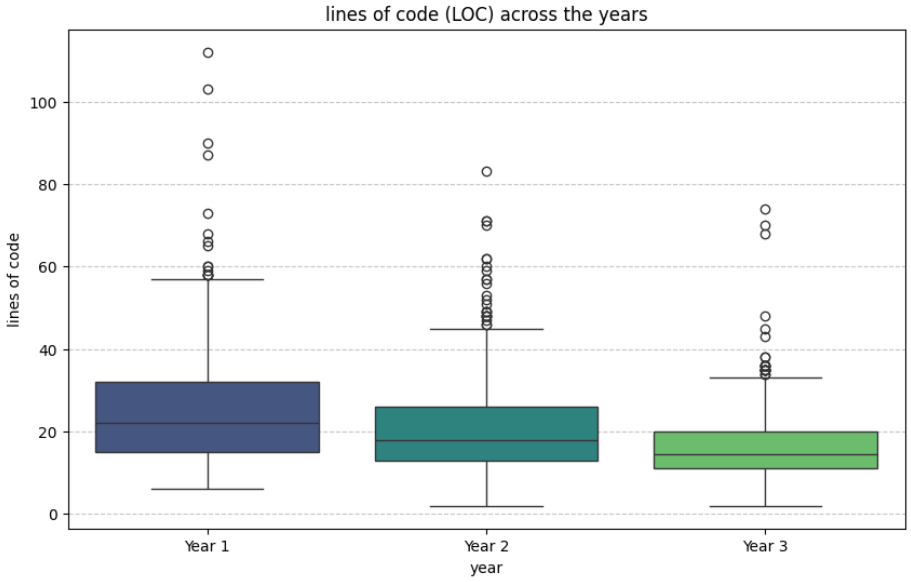

Figure 1: Lines of code (LOC) across the years

Similarity analysis produces distributions of highly similar solution pairs across years and specific questions.

Bar charts (See Figure 2, 3) are used to show which years and which exercises generate the most highly similar submissions

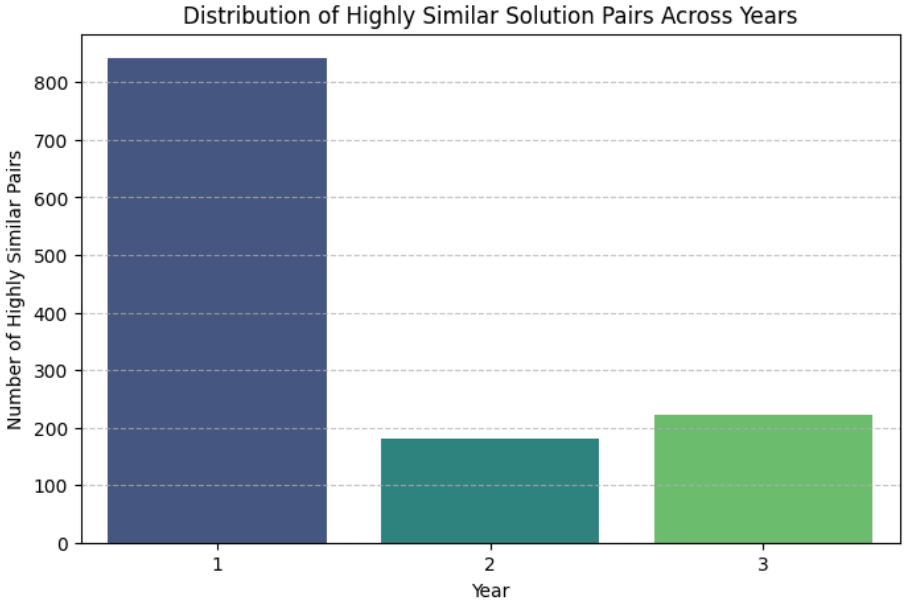

Figure 2: Distribution of highly similar solution pairs across years

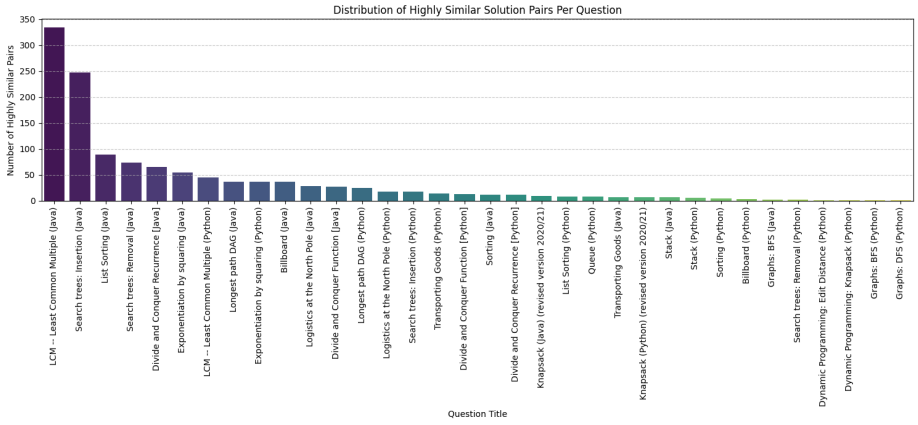

Figure 3: Distribution of highly similar solution pairs per question

Clustering results are visualized using a t-SNE scatter plot (See Figure 4), where each point represents a solution. Different colors indicate cluster grouping, and different markers highlight outliers detected by the Isolation Forest. This visualization makes it easier to see dense regions of similar solutions and

unusual cases.

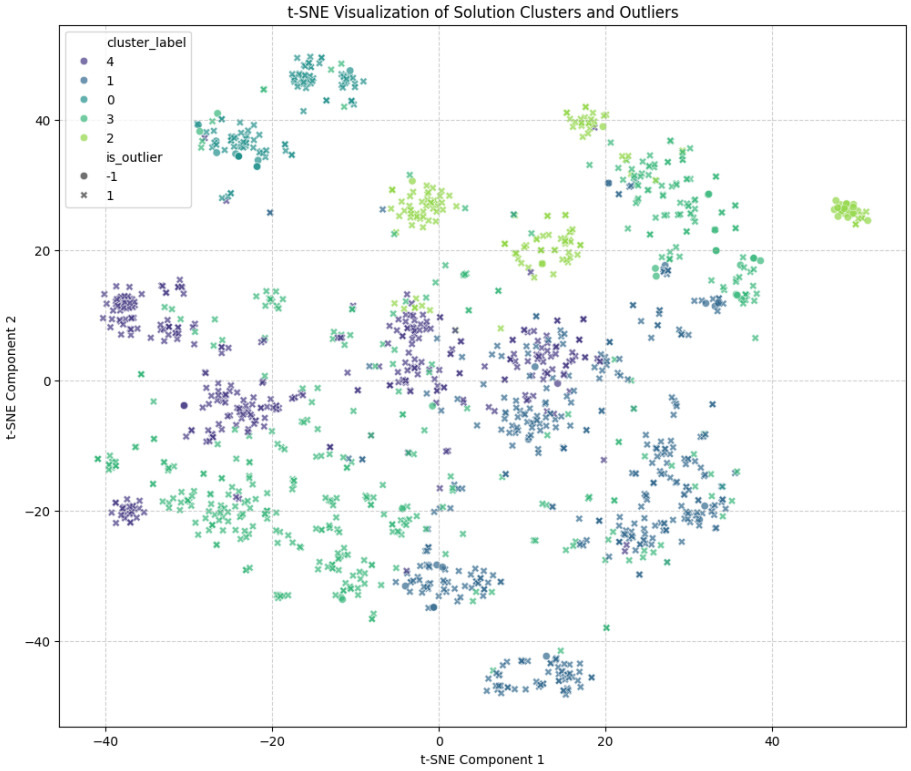

Figure 4: t-SNE Visualization of Solution Clusters and Outliers

The final outcome of the project is an interactive web-based application that visualizes similarity relationships between student programming results as a network graph (See Figure 5). The visualization is built using D3.js and integrated into a React application, which is deployed on Vercel and available on [Kal26b]. The data used by the web app is generated in Python, in Google Colab, and exported as two JSON files, one for nodes and one for links, using the preprocessing and similarity analysis pipeline described earlier.

The core visualization is a force-directed network graph that highlights groups of highly similar solutions.

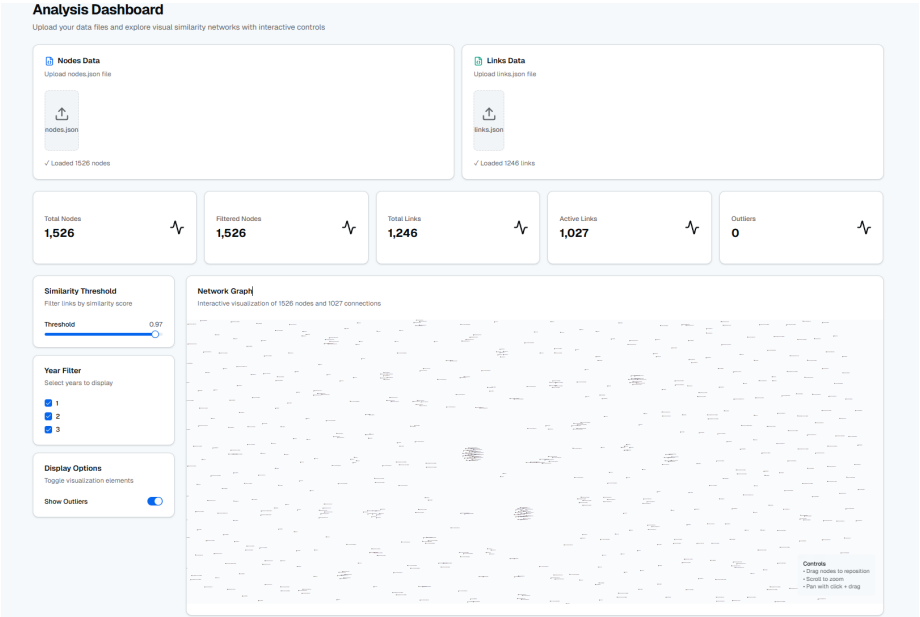

Figure 5: Web application overview

5 Discussion and Limitations

The analysis shows that some tasks consistently lead to higher similarity between student solutions, which may be caused by constrained problem statements or standard algorithmic approaches. Differences between academic years are also visible, suggesting changes in task design or student behaviour.

However, there are important limitations. Similarity scores alone cannot confirm plagiarism, since students may independently produce similar correct solutions. The similarity threshold used in the project is heuristic and chosen for exploration purposes. Additionally, the TF-IDF approach ignores deep semanting meaning and program structure.

The results should therefore be interpreted as indicators, not final judgements.

6 Conclusion and Future Work

This project demonstrates how data visualization and basic machine learning techniques can be used to analyze student programming solutions at scale. By combining preprocessing, similarity analysis, clustering and multiple visualization methods, meaningful patterns can be observed across tasks and years.

Future work could include an integration of more advanced code representations, such as abstract syntax trees or program dependency graphs, instead of relying only on token-based similarity.

From a visualization perspective, future work could include improving the web application to integrate additional interaction techniques, such as highlighting neighbors for a selected node or summarizing cluster-level statistics directly in the interface.

7 Appendix

The full source code for this project, including preprocessing, analysis and visualization steps, is available in the following repository: [Kal26a]

The web application is available on [Kal26b].

References

[Kal26a] Sebastian Kalciov. Data visualization and analysis github repository. `https://github.com/sebastiankalciov/data-visualization-and-analysis`, 2026.

[Kal26b] Sebastian Kalciov. Data visualization and analysis web app. `https://data-visualization-and-analysis.vercel.app/`, 2026.

[PFSK+22] Fynn Petersen-Frey, Marcus Soll, Louis Kobras, Melf Johannsen, Peter Kling, and Chris Biemann. Dataset of student solutions to algorithm and data structure programming assignments. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 956–962, Marseille, France, June 2022. European Language Resources Association.

[SKS+20] Simon, Oscar Karnalim, Judy Sheard, Ilir Dema, Amey Karkare, Juho Leinonen, Michael Liut, and Renée McCauley. Choosing code segments to exclude from code similarity detection. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '20, page 1–19, New York, NY, USA, 2020. Association for Computing Machinery.

[SS23] Dr. Shitalkumar R. Sukhdeve and Sandika S. Sukhdeve. *Google Colaboratory*, pages 11–34. Apress, Berkeley, CA, 2023.