Evaluation of students programming skills on a computer programming course with a hierarchical clustering algorithm

Davi Bernardo Silva*† and Carlos N. Silla Jr.†
†Pontifícia Universidade Católica do Paraná (PUCPR) – Curitiba, PR, Brazil 80215–901
*Instituto Federal de Santa Catarina (IFSC) – Jaraguá do Sul, SC, Brazil 89254–430
Email: {davibernardos, carlos.sillajr}@gmail.com

Abstract—**This Research Full Paper presents a computational hierarchical clustering approach to identify groups of college students from a Computer Science 1 (CS1) course that need extra help with the programming content. We first defined a set of features that characterize the student's programming skills in a CS1 course. Next, we applied a hierarchical clustering algorithm to bring together the students with similar skills by analyzing the source code they developed for different tasks. Finally, we evaluated the quality of the model and analyzed the different clusters. We processed a total of 630 source code tasks, for which our results indicate the formation of three clusters. We have found that one of the clusters has a large number of students with possible difficulties in programming. The other two clusters, although having different coding behaviors, reached high levels of knowledge about the contents taught. We conclude that features extracted from the students source codes can be used to group students into clusters that indicate their performance trends throughout the course.**

Index Terms—**CS1, Computer science education, Clustering**

I. Introduction

The popularization of access to technology, as well as the several employment opportunities, made higher education in the area of computing a preference for a considerable part of future university students [1]. However, for many students, the excitement of the new course is often interrupted by the obstacles present in the first programming courses. The difficulty in developing the expected logical reasoning in the curricular unit can lead to failing or dropping out of the course [2]–[5]. Regardless, teachers are increasingly concerned with making learning effective and enjoyable. The intervention of the instructor in a more timely manner is vital in the performance of the students [6]–[8].

Throughout a Computer Science 1 (CS1) course, students acquire different levels of knowledge. Some students readily understand the elementary concepts and can receive more advanced content while others still need assistance in key topics [9]. Both groups need attention. In the first case, the teacher should direct the learning to continue with progress and prevent the student from becoming disinterested in the course. In the second case, which is the most critical, it will require an even more considerable effort so that the student does not get lost in learning an giving up the course. Another aggravating factor for the teacher in controlling students' knowledge levels is that there are several ways to solve the same programming task and each one with its characteristics [10].

With the support of data mining techniques, the programming tasks performed by the students can be automatically grouped according to the similarity of the source code lines. From then on, the teacher can identify the knowledge levels of each group and provide targeted assistance. Thus, in this paper, we group students according to their programming tasks, such that each group is composed of members with similar or related programming skills. Our goal is to identify in which content each student group needs more assistance. Predicting final student grades is not in our interests, but we want to support the teacher's pedagogical decisions to assist each group of students better. Our main contributions are two-fold: *(i)* we present a set of features that, when applied in source code tasks, can characterize the student's programming skills in a CS1 course; and *(ii)* we evaluate the performance of this set of features with a hierarchical clustering algorithm to group students according to their programming skills.

The remainder of this paper is organized as follows. Section II covers background, and outlines related work. Section III describes the setup of our study. Section IV presents an analysis of the results. Section V further discusses our results. Finally, Section VI presents the conclusion and outlines future work.

II. RELATED WORK

Several researchers are interested in understanding the relationships and challenges found in teaching and learning to programming, as identified in the following systematic reviews [11]–[13]. In 2015, Ihantola et al. [11] identified the use of data mining techniques in teaching programming. In 2018, Luxton-Reilly et al. [12] conducted an exhaustive review of the literature on introductory programming. In 2019, Silva et al. [13] carried out a study to identify pedagogical strategies and support tools.

Among the primary studies found, there is special attention to predicting student performance to use this information for learning interventions. These predictions are usually driven by high failure and dropout rates in CS1 courses [7], [14]–[18]. Some studies try to predict the students' final grade [19] and those that try to form programming pairs as an alternative to accelerate learning [20]. Overall, the common goal among all these studies is to identify students who need assistance in the course. In contrast, their differences are usually related to the information collected and the techniques used to produce the results. It is well known that building a good machine learning model depends fundamentally on selecting good features [21]. Within software engineering research, some studies attempt to define metrics that characterize software development. An example is Bassi et al. [22] that presented a set of software quality metrics to measure the contribution of development team members. The metrics are related to the complexity, inheritance, coupling, and size of source code.

Real-time evaluation of source code tasks applying test cases was performed in [7], [18]. Fonseca et al. [7] introduced *CodeInsights*, an online tool that uses a *plug-in* installed in the programming environment to receive coding information (PHP, Java, or Python) and provides performance notifications of the students to teachers. Source code tasks are compiled and tested using the automated *black-box* testing software engineering procedure [23]. Information was collected, such as: an unusual number of lines of code, compilation errors, attempts per assignment, assignments not attempted, and unfinished assignments are collected. Performance is calculated based on the percentage of tasks completed correctly over a while, and the student is given a label related to their pace (slow, intermediate, or fast). Benotti et al. [18] presented *Mumuki*, an open-source online editor, in a process that, for each task, the system explains the theory and a programming example that involves the concepts needed to solve the task. The teacher is responsible for manually defining the test cases and can also define auxiliary functions. The tool includes an interactive console on which solutions and reusable functions can be performed, and automatic feedback is provided according to the tests. The system supports 17 languages, including Python, JavaScript, and C.

Other studies used supervised machine learning to classify students according to their source code tasks. Ahadi et al. [14] used the *TestMyCode* [24] tool to record student coding data from a CS1 course in Java. A set of features such as the average grade of students in other courses, the maximum percentage of automated tests achieved (*Correctness*), the number of steps used to solve the programming tasks (*Steps*), and finally, a categorical variable that indicates students' major. In total, ten features were used. These features were used to train and validate tree types of supervised machine learning classifiers, namely Bayesian models, learning rules, and decision trees [25]. Students were labeled according to their performance on: an exam question, course acting, and a combination of the two factors; the best results were obtained with the decision tree. Finally, they conducted an experiment on a new course to test the previous model. A replication of this study [14] using two new databases was performed by Castro-Wunsch et al. [15]. In this new context, Python was taught in a CS1 course, and the evaluation of a neural network model built with *TensorFlow* using a hidden layer [26] was added. Although several other features were explored, such as time spent, finishing time, pass test, fail test, and error count, only *Steps* and *Correctness* were used. Students were labeled between passing (grade above 50%) and failing (grade below 50%). Data from one of the courses were used to train the model and predict the students' final grade in the first weeks of the second course. The results were compatible with [14], but the neural networks presented higher precision.

The linear regression technique was used to predict student grades [17] and course performance [16]. Munson and Zitovsky [17] introduced *CodeWork*, a web development environment that collects source code interaction events in a CS1 course taught in Java. The features used by the model were: amount and frequency of work, compilation error resolution, source code size, time of day the student coded, student improvement based on their scores, number and duration of breaks, and the use of the copy and paste function. Ozturk et al. [16] introduced an online editor that allows students to compile and submit their C language source code assignments. The system captures keystroke-level data, but the metrics used for classifying students are: the amount of time spent in the solution of a task and the ratio of tasks solved. The ratio is calculated by dividing the sum of tasks solved by the maximum of tasks solved by a student of the same course. These two characteristics are used to generate classifiers. The performance obtained at the beginning of the course correlates with the final performance of the course.

Besides, some studies use unsupervised machine learning techniques to group students [19], [20]. Aottiwerch and Urachart [20] have developed a web platform for suggesting programming pairs from a questionnaire, integrated into the system, which is completed by the students. This questionnaire covers three themes: programming skills that consist of a basic programming test; learning behavior that involves issues of attitude, motivation, planning, knowledge-seeking and self-assessment [27], and finally, behavioral interoperability involving communication and teamwork skills [28]. The information collected was used as features for a partition clustering algorithm [29] that resulted in 4 clusters: *(i)* good and *(ii)* bad students in all respects, students with *(iii)* high, and *(iv)* low performance in programming skills. Ahadi et al. [19] used students' average weekly performance on programming tasks, which was calculated using automated unit tests to estimate students who pass, fail, and drop out. It was a six weeks CS1 course that was taught in Java language. They also identified that student performance declines over the course.

III. STUDY SETTING

The clustering task has been used in many areas to join related elements [29]. Usually, the data set is not labeled, so the clustering technique is classified as an unsupervised machine learning approach. In this way, the elements are clustered according to the similarity of the information available [30]. Knowledge of the application domain is imperative to the success of the model. The results are highly sensitive to the input parameters, such as the measure of similarity, the chosen clustering algorithm, and the employed feature set. In this section, we will explore the process of constructing our model and address these topics.

A. Research question

In this paper, we present an approach to clustering students from a CS1 course according to their respective programming skills that were extracted from previously submitted source code assignments. Particularly, we are interested in answering the following research questions:

RQ$_1$: Does the hierarchical clustering approach allow groups of college students from a CS1 course with significantly related skills to be brought together?

With the answer to **RQ**$_1$, we aim to describe the application of a clustering algorithm to a set of source code tasks and analyze their results. For this, we use an agglomerative hierarchical clustering algorithm present in the machine learning library *Scikit-learn* [31] available for Python programming language. We applied the *Kruskal-Wallis* statistical test to verify if there is statistical significance in our grouping and the *Mann-Whitney* statistical test to find the difference between the clusters [32].

RQ$_2$: Given the hierarchical clustering, is it possible to identify college students with potential difficulties in the contents taught in a CS1 course?

With the answer to **RQ**$_2$, we want to see if the groups represent related forms of reasoning. We performed a descriptive statistical analysis of the task submission information and the language resources used, as well as the correlation between the nine features categories of the clustering. To validate the findings, we triangulated the scores from two tests conducted throughout the course.

B. Data sample

Our data for the study come from a CS1 course organized at the Federal Institute of Southern Brazil in the year 2017. The classes were taught by one of the authors of this paper and usually happened in a computer lab. The C programming language was chosen to implement the source code tasks. The course lasted 16 weeks, and the content covered is presented in Table I. Each topic consists of a set of source code tasks that range from 4 to 10 problems. The number of students who submitted at least one of the tasks, as well as the total amount of source codes processed per topic, is also shown in Table I. It should be noted that a student can submit at least one assignment to one topic and does not submit any assignment to another. Altogether, 630 source codes, developed by 34 college students, were processed and analyzed. Also, students took two tests: the first test covered contents 1 to 4 and in the second test contents 5 and 6.

C. Data preprocessing

The processing scheme followed by our model is presented in Fig. 1. Given the source code tasks that were submitted by students in text format (e.g., *file.c*), we performed natural language preprocessing to process our data sample. The first step was to join, in a single file, the source code of all tasks submitted by each student. Next, we removed a stopword list, that is, all source code comments, string contents, and variable names are removed. Thus, we intended to minimize noise and preserve the anonymity of the students. Then each word is considered as a token. Finally, we use the language keywords that are presented in Table II to perform a word count and form a feature vector (bag-of-words) that will be used as input to the clustering algorithm. A total of 46 keywords of the language were employed [33]. To decrease the dimensionality of our model, we have combined these characteristics into nine categories.

TABLE I: Content taught and task distribution.

ID	Content	# Tasks	# Students	# Codes
1	functions (by value)	6	28	193
2	functions (by reference)	4	17	71
3	text manipulation	9	14	117
4	struct	5	16	69
5	recursion	4	19	73
6	dynamic allocation	6	19	107

TABLE II: Set of extracted features and their categories.

Category	Description
input/output	*getchar, getche, getch, gets, fgets, putchar, puts*
conditional	*if, else, switch, case, default*
diversion	*continue, break, return, exit*
loop	*for, while, do/while*
function	*main, functions, argc, argv*
string	*strcpy, strcat, strcmp, stricmp, strlen, strupr, strlwr, strrev, tolower, toupper, isalpha*
file	*fopen, fclose, feof, remove, fflush*
data structures	*sizeof, typedef, struct, malloc, free*
library	*include, system*

D. Clustering method

The output of the clustering method consists of a set of clusters that share common characteristics. Each cluster has an indefinite number of elements that are assembled according to their similarity.

There are two main ways of grouping data, which are the partitional and hierarchical methods. Partitional methods divide the data instances into a multi-exclusive, cluster number that is defined in advance. An example is the *K-means* algorithm [29]. This type of algorithm does not work well with outliers in the data, in addition to presenting undesirable characteristics when the groups are of different sizes, densities, or have non-globular shapes.

The hierarchical clustering corresponds to a set of nested groups organized as a hierarchical tree; the grouping can occur following the divisive or agglomerative strategy [34]. In

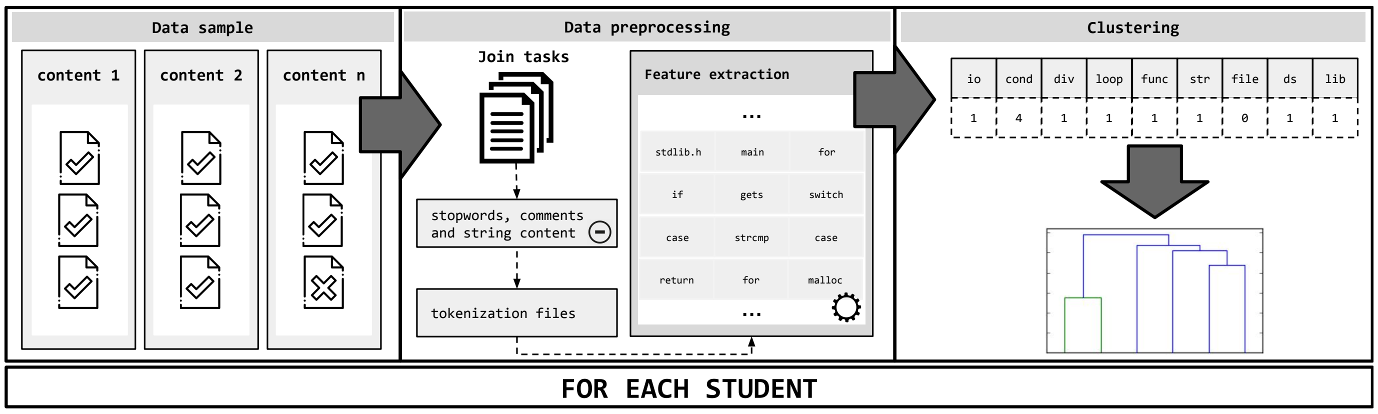


Fig. 1: Steps for student clustering.

the divisive, the algorithm starts with a cluster that includes all the elements; this cluster is divided into smaller clusters until each cluster contains an element. In the agglomerative, the algorithm starts with as many clusters as the number of elements; then, the closest clusters are joined until there is a single cluster. Hierarchical methods provide a tree view of the results called a dendrogram diagram; it shows the order and distances between the groups formed during the grouping [35]. It is possible to obtain different groupings depending on the level at which the dendrogram is cut.

As our database is small and the processing time is irrelevant, we opted to use the agglomerative hierarchical clustering algorithm. We made this choice due to the limitations of the partitional methods and the ease of visualization provided by the dendrogram. Its pseudocode is presented by Algorithm 1.

Algorithm 1: HIERARCHICAL AGGLOMERATIVE CLUSTERING

```
Input: matrix of similarity S $\in \mathbb{R}^{nxm}$
Output: Hierarchical clustering S
begin
    while all objects are not in the same group do
        update the distances between the clusters;
        find the nearest clusters;
        join in a new cluster;
    end
end
```

E. Distance measures

We tested two metrics to calculate the distance between elements used to define the cluster structure, such as the *Cityblock distance* and the *Euclidean distance* [36]. Each metric was applied from the data matrix, in which each row represents an instance. Then, the second distance matrix is calculated; each element of the matrix corresponds to a quantitative measure of the proximity between two pairs of objects. Equation 1 expresses the *Cityblock distance* (or *Manhattan distance*) *x* between the elements *u* and *v* within a multidimensional space.

$$x_{cityblock}(\overrightarrow{u}, \overrightarrow{v}) = \sum_{i=1}^{n} |u_i - v_i| \tag{1}$$

Equation 2 expresses the *Euclidean geometric distance x* between the elements *u* and *v* within a multidimensional space.

$$x_{euclidean}(\overrightarrow{u}, \overrightarrow{v}) = \left(\sum_{i=1}^{n} (u_i - v_i)^2 \right)^{1/2} \tag{2}$$

The linkage method defines how the distance between the clusters will be measured [34]. The most common methods are: *Single-Linkage*, which checks the minimum distance; *Complete-Linkage*, which checks the maximum distance; and the *Average-Linkage*, which checks the average of the distances. We calculated the similarity between the clusters with each one using the two distance metrics presented. We then evaluated the model to see which configuration performed best.

F. Evaluation of the model

We interpreted a dendrogram to find the appropriate number of clusters in the dataset. To verify this division's consistency, we apply the *Elbow method*, which is used to examine the percentage variation between clusters [37]. The goal is to add new clusters to the model to the point where the information gain is no longer significant. This point is identified on the graph as an "elbow".

We applied an internal validation index to check the quality of our clusters. We used the *Cophenetic Coefficient Correlation* [38] that correlates linearly to matrices of similarity and *Cophenetic's*. Equation 3 defines the *Cophenetic Coefficient cc*, where *x* is the distance, and *t* is the height of the node where the points met.

$$cc = \frac{\sum_{u<v} \left(x\left(u,v\right) - \bar{x}\right)\left(t\left(u,v\right) - \bar{t}\right)}{\left(\left[\sum_{u<v} \left(x\left(u,v\right) - \bar{x}\right)^2\right]\left[\sum_{u<v} \left(t\left(u,v\right) - \bar{t}\right)^2\right]\right)^{1/2}} \tag{3}$$

For a valid dendrogram, the resulting matrix must have a high correlation with the similarity matrix. The coefficient value can range from 0.00 to 1.00, the closer to 1.00, the higher the internal quality of the cluster.

IV. Analysis of Results

A. *Hierarchical clustering of students (RQ_1)*

In our clustering, we tested different metrics of distances and connection methods, as shown in Table III. We used an internal correlation index to check, which was the best configuration. So, the *Euclidean distance*, combined with the *Complete-Linkage method*, is the highest *cc* index (0.79).

TABLE III: Internal validation between clusters.

Metric	Method	*cc*
Euclidean distance	*Single-Linkage*	0.74
Euclidean distance	*Complete-Linkage*	0.79
Euclidean distance	*Average-Linkage*	0.78
Cityblock distance	*Single-Linkage*	0.72
Cityblock distance	*Complete-Linkage*	0.75
Cityblock distance	*Average-Linkage*	0.76

After executing the clustering algorithm using our dataset, we obtained the hierarchically organized separation of students, as shown in the dendrogram of Fig. 2. Each instance presented on the *x-axis* represents a student, and the horizontal lines of the *y-axis* indicate the *Euclidian distance* between instances. The most similar programming instances are joined at the beginning of the *y-axis*, for instance, students 21 and 26 who obtained the shortest distance from the cluster (10.1). The higher the value of the *y-axis*, the greater the calculated dissimilarity. The intersections between the vertical and horizontal lines represent the formation of clusters.

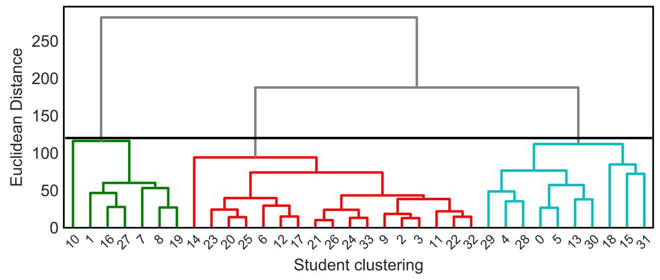


Fig. 2: Hierarchical clustering dendrogram of students.

In the diagram, we visually observe the most significant similarities of *Euclidian distance* at height 120 and therefore plot a cut line at this point. To confirm the most appropriate number of clusters, we use the *Elbow method*; the graph in Fig. 3 shows the obtained result. The *x-axis* represents the number of clusters, and the *y-axis* is represented by the *Euclidian distance*. Note that the acceleration of distance growth is more significant with three clusters.

According to the data, the results from *Kruskal-Wallis* *H*-test indicated that one or more of the three groups are significantly different ($H_{(2)} = 27.607$, $p < 0.05$). Therefore, we compared the samples with *Mann-Whitney U*-tests to find the differences between groups; in all comparisons, we obtained

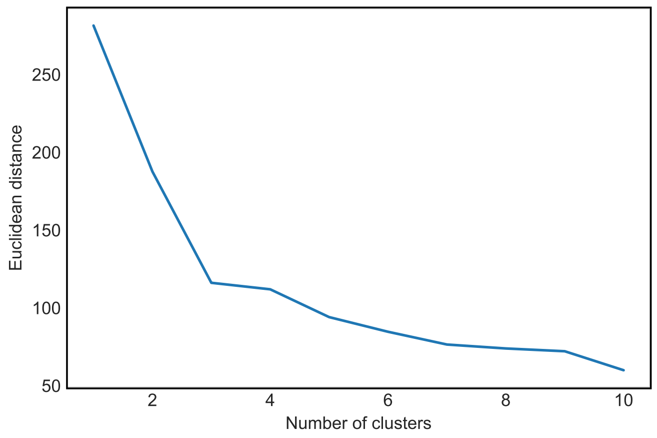


Fig. 3: Number of clusters using the *Elbow method*.

$p < 0.0167$, indicating that the samples were significantly different.

Details of the clusters obtained from the cut line are presented in Table IV. For this, the dendrogram reading was performed from left to right. The leftmost cluster was called Group A (GA) and brought together seven student instances (e.g., 10, 1, 16, 27, 7, 8 e 19); the central group, Group B (GB), joined 17 students; and rightmost group, Group C (GC), grouped ten students. Groups GA and GB were responsible for the submission of 76.3% of source code analyzed. Although GB had the largest number of students (50%), it was the group that submitted the fewest source codes.

B. *Struggling student clusters (RQ_2)*

To verify the distribution of the extracted features that were more present in each cluster, we plotted the graph presented in Fig. 4. The *x-axis* represents the groups, and the *y-axis* represents the sum of all categories for each student group. GA students are found to be concentrated in a small range of values, indicating that the data is more similar. They made the most use of language resources, which is between 509 and 518 occurrences. One of the students was considered an outlier, as he obtained a much larger amount of collected characteristics than the others (695 occurrences); it was responsible for sending 3.7% of the total submitted tasks. However, the lowest resource utilization was practiced by GB; their values range from 28 to 226, with 50% of occurrences below the median corresponding to 90. The GC group obtained the highest variance, with a standard deviation of 87.53; their minimum and maximum values were 251 and 521 occurrences.

In a more detailed analysis, on the distribution of resources by categories, we can observe a behavior similar to the previous one. In Fig. 5, the *x-axis* represents the groups, and the *y-axis* represents the sum of all occurrences of features for each student group.

The average percentage of submissions and their respective standard deviations, separated by content and by group, are also presented in Table IV. GA had the lowest rates for *text manipulation* and *recursion* content (85.7%). The lowest index

TABLE IV: Submission rate by cluster.

Group	# Instances	# Submission	Content ID					
			1	2	3	4	5	6
GA	7 (20.6%)	245 (38.7%)	100.0% ± 0.0%	100.0% ± 0.0%	85.7% ± 37.8%	88.6% ± 15.7%	100.0% ± 0.0%	85.7% ± 37.8%
GB	17 (50.0%)	148 (23.5%)	54.6% ± 42.3%	16.2% ± 36.4%	4.6% ± 18.9%	16.5% ± 31.0%	20.6% ± 38.8%	32.4% ± 45.4%
GC	10 (29.4%)	237 (37.6%)	97.1% ± 6.0%	70.0% ± 48.3%	58.9% ± 41.3%	44.0% ± 47.9%	77.5% ± 41.6%	63.3% ± 45.0%

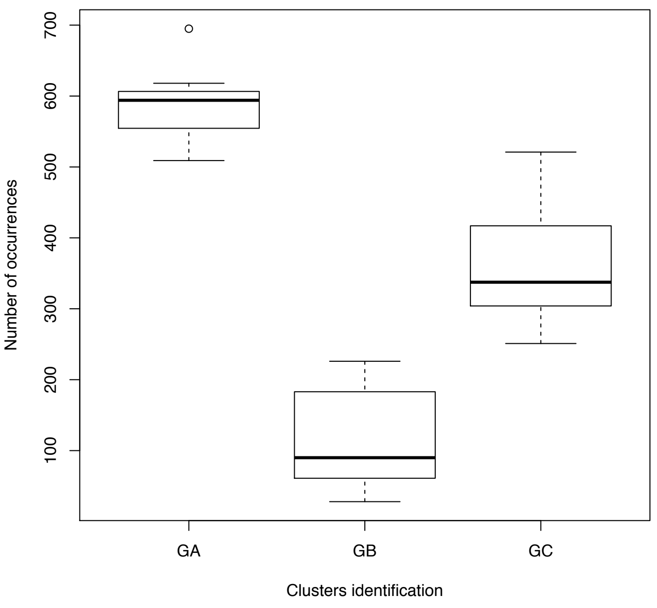


Fig. 4: Relationship between clusters and features.

TABLE V: Correlation rô of Spearman between features.

	Group	input/output	conditional	diversion	loop	function	string	file	data structures
conditional	GA	0.70							
	GB	0.42							
	GC	0.79							
diversion	GA	0.50	0.22						
	GB	0.58	0.00						
	GC	0.77	0.37						
loop	GA	0.73	0.29	1.00					
	GB	0.31	0.00	0.00					
	GC	0.09	0.10	0.34					
function	GA	0.37	0.94	0.33	0.43				
	GB	0.85	0.00	0.00	0.00				
	GC	0.67	0.05	0.01	0.18				
string	GA	0.70	0.82	0.53	0.64	0.56			
	GB	0.08	0.84	0.22	0.04	0.63			
	GC	0.17	0.65	0.58	0.94	0.71			
file	GA	0.95	0.47	0.14	0.64	0.80	0.04		
	GB	0.23	0.72	0.73	0.44	0.72	0.08		
	GC	0.70	0.02	0.90	0.10	0.22	0.74		
data structures	GA	0.37	0.82	0.85	0.31	0.20	0.88	0.72	
	GB	0.20	0.73	0.18	0.04	0.58	0.00	0.08	
	GC	0.08	0.20	0.69	0.01	0.21	0.49	0.02	
library	GA	0.94	0.34	0.15	0.38	0.70	0.09	0.00	0.64
	GB	0.15	0.00	0.00	0.00	0.02	0.01	0.22	0.02
	GC	0.75	0.59	0.05	0.79	0.72	0.30	0.66	0.55

of GB submissions was also the *text manipulation* content (4.6%), and GC was the *struct* content (44.0%).

To better understand relationships, we check the correlation between characteristics in each grouping. Table V presents the result of the *Spearman* coefficient. A large correlation value is 0.50 or higher; an average correlation has a value between 0.30 and 0.50; finally, 0.10 or less indicates a small correlation [39]. We highlight, in gray, relationships with a value of 0.80 or higher. The GA group presented the highest correlations: looping statements are fully correlated with the diversion statements; conditional statements are strongly correlated with functions (0.94); input/output commands are also highly correlated with the use of file manipulation commands (0.95) and the use of libraries (0.94). In GB, the combination of input/output commands and functions (0.85) is recurrent, in addition to conditional statements and string manipulation commands (0.84). In GC, the highest correlations were concentrated in string manipulation commands with looping statements (0.94) and file manipulation commands and diversion statements (0.90).

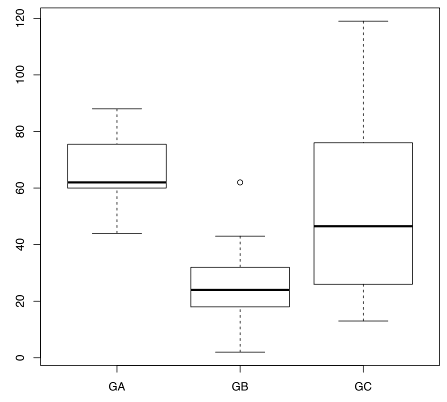
The tendency of the results obtained previously is confirmed by the students' performance in the tests performed during the course. In Fig. 6, we present the performance of the three student groups in the two tests (Fig. 6a and Fig. 6b) and then average the two tests (Fig. 6c). The *x-axis* represents the groups, and the *y-axis* represents the sum of all grades for each student group. The grades range from 0 to 10.

V. DISCUSSION

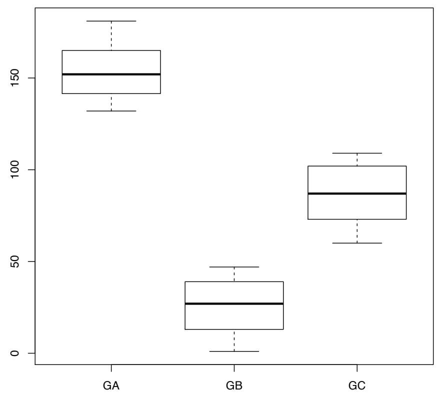
Our work is closely related to the studies using machine learning, especially those using the clustering technique. However, our focus is different because while most of the existing approaches try to predict students' performance in a given course, we are concerned with detecting the relationship between instances to determine which students have the most similar programming skills. In comparison with the works that analyze the students' source code, it should be noted that most of them develop specific online tools for this purpose. While these tools are interesting, they do not allow the teacher and the students to choose an integrated development environment (IDE) that suits them. Our approach has the advantage of being flexible, as it works directly on the source code.

Our results confirm the different levels of knowledge about the contents present among students. Therefore, our computational approach allows the teacher to adapt teaching strategies to make learning more engaging and focused on the specific difficulties of each group of students.

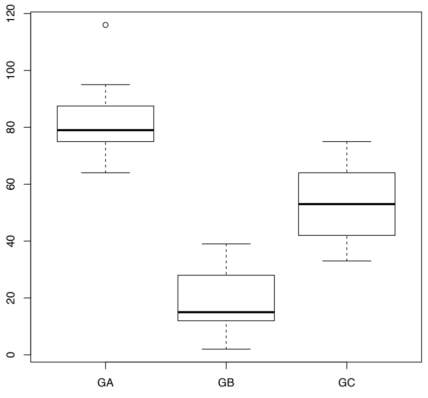
Among the possibilities for teacher intervention, extra class materials could be recommended, and tutoring classes could be scheduled according to each group's main needs. Also, pedagogical strategies that implement teamwork could be used to explore the balance between students' programming skills.



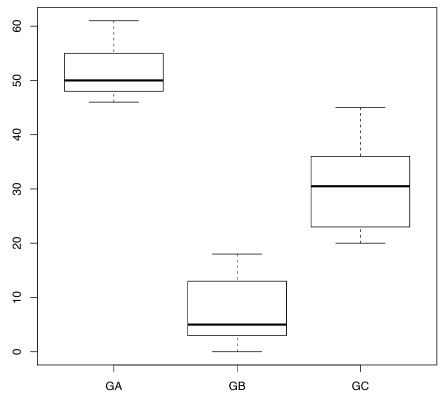
(a) Input/Output statement



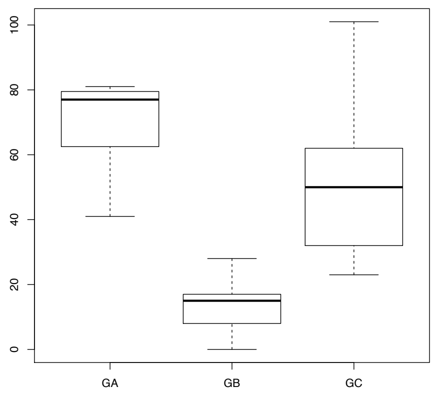
(b) Conditional statement



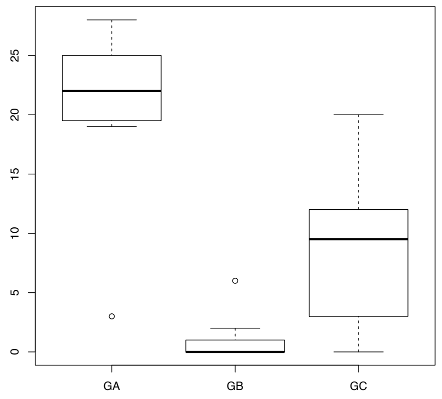
(c) Diversion statement



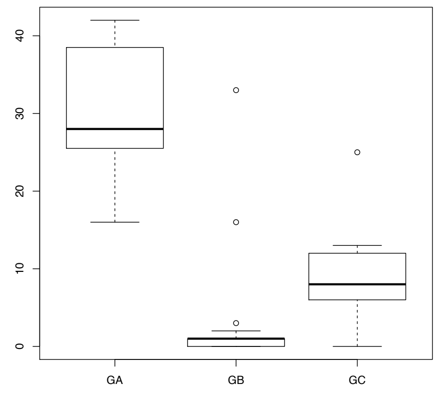
(d) Loop statement



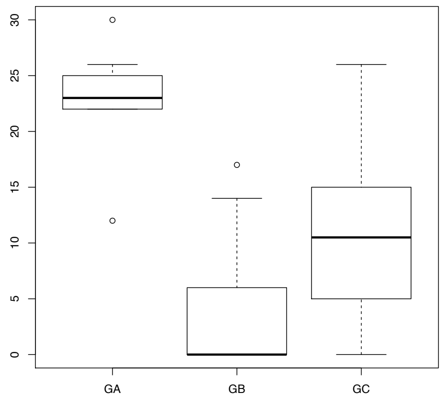
(e) Function



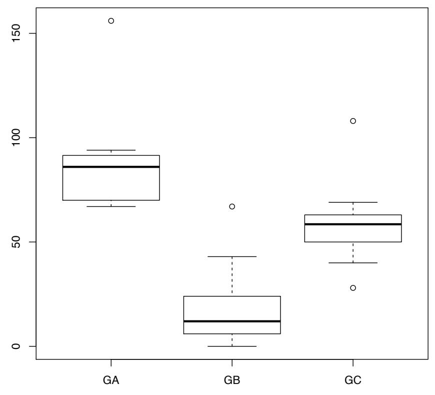
(f) String



(g) File



(h) Data structures



(i) Library

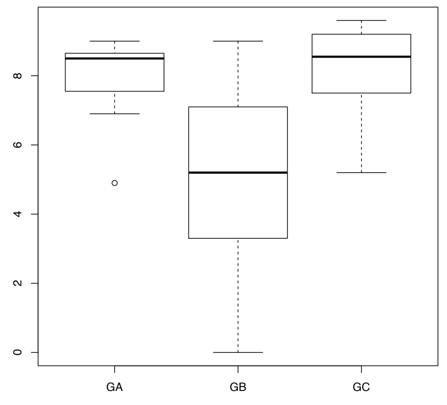
Fig. 5: Relationship between clusters and features.

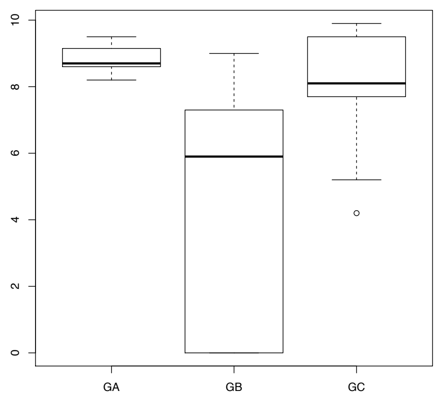
Another promising application would be to monitor students' evolution throughout the content taught; the teacher would have the opportunity to visualize, in real-time, the migration of students between groups and to identify the pedagogical strategies that are working for a specific class.

A. *Threats to validity*

The possible threats to the validity of this study are the following.

a) Sample size: our sample represents a specific context. We acknowledge that our data set comes from a single institution and that we do not test a large number of students; however, a much larger number of data could compromise a more accurate analysis of the results. New experiments, replicating our methodology, can be performed to increase the breadth of the results.


(a) Test 1


(b) Test 2

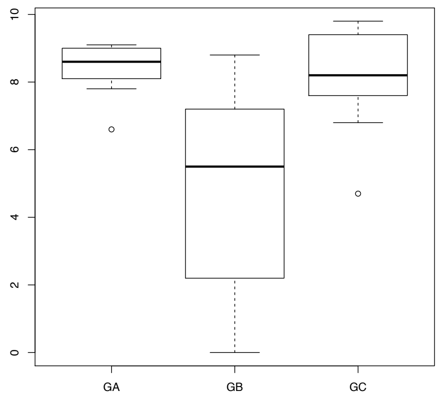

(c) Average

Fig. 6: Relationship between clusters and tests.

b) Employed features: we understand that the selection of features is determinant for a good clustering, so we review other works and select features with the focus of representing the programming skills of the students. We preprocessed the source code tasks and used keywords from the programming language itself that were subdivided into categories. We do not use language operators (arithmetic, relational and logical) and some data types (e.g., *int, double, char,* and so on), as they are frequent and tend to unbalance the model. Perhaps finding a way to put weight on features would be important for even more accurate results.

c) The number of clusters: the definition of the number of clusters can influence the detailing of the results and change the conclusions. We define a cut line at the height of 120 because the dendrogram obtained suggests that this is an adequate separation. Also, we use the *Elbow method* to confirm this trend. An even smaller granularity of clusters can be considered to find even more specific intervention niches; we chose not to do this to enable our analysis and provide an easy-to-manage view for the teacher.

d) Other factors: other factors specialize in our study, such as the programming language used in teaching, the pedagogical methodologies, the contents addressed, and the proposed source code tasks. However, these issues are decisions that need to be made to define the scope of the study. The replications of our work in other contexts are important for a higher generalization of results and a universal validation of the model.

VI. CONCLUDING REMARKS

We used hierarchical clustering to bring together students who have similar programming skills. We used a database of 630 C Language source code assignments written by a class of 34 undergraduate students. We collected a total of 46 keywords that were distributed in nine categories. After clustering, we defined a cut-off line with a *Euclidean distance* at the height of 120 and obtained three clusters (GA, GB, and GC).

We were able to quantify and analyze the language resources used by each cluster, as well as identify the weaker group of students. Statistically, the separation of our groups was significant and had a 95% chance of repeating itself in new experiments. The GB cluster had the largest number of instances (17); however, it was also the cluster that delivered the least tasks (23.5%). Consequently, it resulted in the lowest quantitative features collected and was the group that was most likely to have difficulty in programming classes. Regardless, the GA and GC groups showed signs of a greater understanding of the programming content taught in the class. They demonstrated good performance with the marks obtained in the two tests and demonstrated commitment in the delivery of the exercise lists. In general, GC tends to use language resources more economically. While GA, it makes more exploratory and abundant use of commands.

As future work, it would be important to remodel the extracted features. A consideration of the keywords most relevant to the contents addressed throughout the semester could deliver groupings with richer evidence. Therefore, extending the study to other classes with different teaching environments and even programming language settings could provide more solid results. Besides, it would be good to know how many submissions are needed to form robust clusters so that early interventions can be deployed to help at-risk students.

ACKNOWLEDGMENT

We thank the support received from the following Brazilian institutions: the Coordination for the Improvement of Higher Education Personnel (CAPES), the Brazilian National Research Council (CNPq), and the Araucaria Foundation (FA).

REFERENCES

[1] T. Camp, W. R. Adrion, B. Bizot, S. Davidson, M. Hall, S. Hambrusch, E. Walker, and S. Zweben, "Generation cs: The growth of computer science," *ACM Inroads*, vol. 8, no. 2, pp. 44–50, May 2017.

[2] J. Bennedsen and M. E. Caspersen, "Failure rates in introductory programming," *SIGCSE Bull.*, vol. 39, no. 2, pp. 32–36, June 2007.

[3] P. Kinnunen and L. Malmi, “Why students drop out cs1 course?” in *Proceedings of the Second International Workshop on Computing Education Research*, ser. ICER ’06. New York, NY, USA: ACM, 2006, pp. 97–108.

[4] C. Watson and F. W. Li, “Failure rates in introductory programming revisited,” in *Proceedings of the 2014 Conference on Innovation &#38; Technology in Computer Science Education*, ser. ITiCSE ’14. New York, NY, USA: ACM, 2014, pp. 39–44.

[5] A. Vihavainen, J. Airaksinen, and C. Watson, “A systematic review of approaches for teaching introductory programming and their influence on success,” in *Proceedings of the Tenth Annual Conference on International Computing Education Research*, ser. ICER ’14. New York, NY, USA: ACM, 2014, pp. 19–26.

[6] C. Szabo and N. Falkner, “Silence, words, or grades: The effects of lecturer feedback in multi-revision assignments,” in *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE ’17. New York, NY, USA: ACM, 2017, pp. 293–298.

[7] N. Gil Fonseca, L. Macedo, and A. J. Mendes, “Supporting differentiated instruction in programming courses through permanent progress monitoring,” in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE ’18. New York, NY, USA: ACM, 2018, pp. 209–214.

[8] A. Petersen, M. Craig, J. Campbell, and A. Tafliovich, “Revisiting why students drop cs1,” in *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*, ser. Koli Calling ’16. New York, NY, USA: ACM, 2016, pp. 71–80.

[9] J. Carter, S. White, K. Fraser, S. Kurkovsky, C. McCreesh, and M. Wieck, “Iticse 2010 working group report motivating our top students,” in *Proceedings of the 2010 ITiCSE Working Group Reports*, ser. ITiCSE-WGR ’10. New York, NY, USA: ACM, 2010, pp. 29–47.

[10] M. Gaudencio, A. Dantas, and D. D. Guerrero, “Can computers compare student code solutions as well as teachers?” in *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE ’14. New York, NY, USA: ACM, 2014, pp. 21–26.

[11] P. Ihantola, A. Vihavainen, A. Ahadi, M. Butler, J. Börstler, S. H. Edwards, E. Isohanni, A. Korhonen, A. Petersen, K. Rivers, M. A. Rubio, J. Sheard, B. Skupas, J. Spacco, C. Szabo, and D. Toll, “Educational data mining and learning analytics in programming: Literature review and case studies,” in *Proceedings of the 2015 ITiCSE on Working Group Reports*, ser. ITICSE-WGR ’15. New York, NY, USA: ACM, 2015, pp. 41–63.

[12] A. Luxton-Reilly, Simon, I. Albluwi, B. A. Becker, M. Giannakos, A. N. Kumar, L. Ott, J. Paterson, M. J. Scott, J. Sheard, and C. Szabo, “A review of introductory programming research 2003–2017,” in *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE 2018. New York, NY, USA: ACM, July 2018.

[13] D. B. Silva, R. d. L. Aguiar, D. S. Deconto, and C. N. Silla, “Recent studies about teaching algorithms (cs1) and data structures (cs2) for computer science students,” in *Proceedings Frontiers in Education. 49th Annual Conference*, ser. FIE ’19. Cincinnati, OH, USA: IEEE, October 2019.

[14] A. Ahadi, R. Lister, H. Haapala, and A. Vihavainen, “Exploring machine learning methods to automatically identify students in need of assistance,” in *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*, ser. ICER ’15. New York, NY, USA: ACM, 2015, pp. 121–130.

[15] K. Castro-Wunsch, A. Ahadi, and A. Petersen, “Evaluating neural networks as a method for identifying students in need of assistance,” in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE ’17. New York, NY, USA: ACM, 2017, pp. 111–116.

[16] A. Ozturk, P. Bonfert-Taylor, and A. Fugenschuh, “Using data to improve programming instruction,” in *DeLFI 2018 - Die 16. E-Learning Fachtagung Informatik*, D. Kromker and U. Schroeder, Eds. Bonn: Gesellschaft fur Informatik e.V., 2018, pp. 23–32.

[17] J. P. Munson and J. P. Zitovsky, “Models for early identification of struggling novice programmers,” in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE ’18. New York, NY, USA: ACM, 2018, pp. 699–704.

[18] L. Benotti, F. Aloi, F. Bulgarelli, and M. J. Gomez, “The effect of a web-based coding tool with automatic feedback on students’ performance and perceptions,” in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE ’18. New York, NY, USA: ACM, 2018, pp. 2–7.

[19] A. Ahadi, R. Lister, S. Lal, J. Leinonen, and A. Hellas, “Performance and consistency in learning to program,” in *Proceedings of the Nineteenth Australasian Computing Education Conference*, ser. ACE ’17. New York, NY, USA: ACM, 2017, pp. 11–16.

[20] N. Aottiwerch and U. Kokaew, “The analysis of matching learners in pair programming using k-means,” in *2018 5th International Conference on Industrial Engineering and Applications (ICIEA)*. IEEE, April 2018, pp. 362–366.

[21] R. Garreta and G. Moncecchi, *Learning Scikit-learn: Machine Learning in Python*. Packt Publishing, 2013.

[22] P. R. de Bassi, G. M. P. Wanderley, P. H. Banali, and E. C. Paraiso, “Measuring developers’ contribution in source code using quality metrics,” in *2018 IEEE 22nd International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. IEEE, May 2018, pp. 39–44.

[23] S. Nidhra and J. Dondeti, “Black box and white box testing techniques - a literature review,” *International Journal of Embedded Systems and Applications (IJESA)*, vol. 2, pp. 29–50, June 2012.

[24] A. Vihavainen, T. Vikberg, M. Luukkainen, and M. Pärtel, “Scaffolding students’ learning using test my code,” in *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE ’13. New York, NY, USA: ACM, 2013, pp. 117–122.

[25] C. Romero and S. Ventura, “Educational data mining: A review of the state of the art,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 40, no. 6, pp. 601–618, November 2010.

[26] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, “Tensorflow: A system for large-scale machine learning,” in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, November 2016, pp. 265–283.

[27] M. K. Khalil, H. G. Hawkins, L. M. Crespo, and J. Buggy, “The relationship between learning and study strategies inventory (lassi) and academic performance in medical schools,” *Medical Science Educator*, vol. 27, no. 2, pp. 315–320, June 2017.

[28] F. J. Rodríguez, K. M. Price, and K. E. Boyer, “Exploring the pair programming process: Characteristics of effective collaboration,” in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE ’17. New York, NY, USA: ACM, 2017, pp. 507–512.

[29] A. K. Jain, “Data clustering: 50 years beyond k-means,” *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651 – 666, 2010, award winning papers from the 19th International Conference on Pattern Recognition (ICPR).

[30] X. Zhu, A. B. Goldberg, R. Brachman, and T. Dietterich, *Introduction to Semi-Supervised Learning*. Morgan and Claypool Publishers, 2009.

[31] “Machine learning in python.” [Online]. Available: https://scikit-learn.org

[32] G. W. Corder and D. I. Foreman, *Nonparametric Statistics: A Step-by-Step Approach*. John Wiley & Sons, 2014.

[33] H. Schildt, *C: The Complete Reference*. McGraw-Hill Education (India) Pvt Limited, 2000.

[34] D. Mullner, “Modern hierarchical, agglomerative clustering algorithms,” *CoRR*, vol. abs/1109.2378, 2011.

[35] Y. Koren and D. Harel, “A two-way visualization method for clustered data,” in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’03. New York, NY, USA: ACM, 2003, pp. 589–594.

[36] U. Rani and S. Sahu, “Comparison of clustering techniques for measuring similarity in articles,” in *2017 3rd International Conference on Computational Intelligence Communication Technology (CICT)*, 2017, pp. 1–7.

[37] Y. Liu, Z. Li, H. Xiong, X. Gao, and J. Wu, “Understanding of internal clustering validation measures,” in *2010 IEEE International Conference on Data Mining*. Sydney, NSW, Australia: IEEE, December 2010, pp. 911–916.

[38] G. Cardona, A. Mir, F. Rosselló, L. Rotger, and D. Sánchez, “Cophenetic metrics for phylogenetic trees, after sokal and rohlf,” *BMC Bioinformatics*, vol. 14, no. 1, p. 3, January 2013.

[39] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Erlbaum Associates, 1988.