Measuring creativity in computer programming: A code distance approach

By

Elijah Chou

Dr. Davide Fossati
Advisor

Computer Science

Dr. Davide Fossati
Advisor

Dr. Angela Porcarelli
Committee Member

Dr. Emily Wall
Committee Member

2023

Measuring creativity in computer programming: A code distance approach

By

Elijah Chou

Dr. Davide Fossati
Advisor

Computer Science

2023

Abstract

Measuring creativity in computer programming: A code distance approach
By Elijah Chou

We propose a novel approach to measure student creativity in computer programming. We collected a set of Java programming problems and their solutions submitted by multiple students. We parsed the students' code into abstract syntax trees, and calculated the distance among code submissions within problem groups using a tree edit distance algorithm. We estimated each student's creativity as the normalized average distance between their code and the other students' codes. Pearson correlation analysis revealed a negative correlation between students' coding performance (i.e., the degree of correctness of their code) and students' programming creativity in some circumstances. Further analysis comparing state (features of the problem set) and trait (features of the students) for this measure revealed a correlation with trait and no correlation with state. This suggests that our proposed measure is likely measuring specific traits that a student has, possibly originality, and not some coincidental feature of our problem set. We also examined the validity of our proposed measure by observing the frequency at which human graders agree with the measure in ranking the originality of pairs of code. Our proposed creativity measure achieved moderate agreement with the majority vote of human graders in ranking creativity. The Pearson correlation and state vs. trait analyses were repeated on student code written in Python, and similar findings were observed in the Python dataset as well.

Measuring creativity in computer programming: A code distance approach

By

Elijah Chou

Dr. Davide Fossati
Advisor

A thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements for the degree of
Bachelor of Science with Honors

Computer Science

2023

Acknowledgments

I would like to thank my thesis advisor, Dr. Davide Fossati, for introducing me to such an interesting project to work on and for guiding me throughout the entire research process. I would also like to thank Dr. Arnon Hershkovitz from Tel Aviv University for working with us and offering us his expertise in creativity for education research. It was my honor and pleasure to work with these professors in my final year of undergraduate studies at Emory University. I would like to thank Dr. Emily Wall and Dr. Angela Porcarelli for taking time out of their busy schedules to serve on my honors committee. Finally, I would like to thank all my family members and friends for supporting me throughout my undergraduate career. I was able to finish as strong as I did thanks to the encouragement and care they provided me throughout the toughest moments here at Emory University.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Ever since the term “computational thinking” (hereby abbreviated as CT) was coined by J. Wing in her short 2006 article published in the ACM, this concept of using analytical and algorithmic thinking for problem solving became widely popular in academia [45]. Well-known companies such as Google referred to ISTE’s CT Toolkit, which has been used to curate curriculums to help students develop CT, for foundational training in computer science [1, 42]. Even though the topic is popular, papers discussing CT in modern research are still refining the definition of CT and its relation to both the world at large and in the classroom [27]. There is also little quantitative research done that shows how effective these educational materials are at achieving any sort of outcome, whether it be increasing one’s CT or improving one’s academic grade.

Creativity is another topic that researchers have tried multiple decades to quantitatively measure. It is a quality that has also been studied in research under the context of problem solving and education [25, 13]. Creativity has also been shown to have a strong positive correlation with CT [35], which may suggest that developing one can help foster the other. However, there is no established metric of measuring creativity in the context of computer programming. Even in a recent study that attempted to establish a metric through automatic parsing of Scratch code, the results were limited by the lack of consensus among experts who were labeling the data from their experiments [24].

This study aims to create a measure that not only can objectively and quantitatively determine the creativity of an individual through their code, but also evaluate the measure and compare the relationship between the new creativity measure and student outcomes. With a more consistent, objective measurement of creativity, we could potentially use it to evaluate the effectiveness of CT curricula and improve CT in people both in and out of the classroom.

Chapter 2

Background

2.1 Computational Thinking

While Computational Thinking originated from the field of computer science, or more broadly from the Science, Technology, Engineering, and Math (STEM) field, CT has been gaining popularity among other fields such as the humanities and arts as seen in some published studies from 2016 and 2020 [19, 43]. The first to discuss the concept of CT was Seymour Papert in 1980, who predicted that computational problem solving would help children solve problems in domains outside of STEM [32]. This prediction accurately illuminates the importance of CT in K-12 education in modern day as described in Barr's 2011 study [6].

CT is an important skill that can help develop better proficiency with computer science problems and also increase one's potential for general problem-solving as shown in Ruan et. al's 2017 study [36]. It has also been shown to influence mathematics, literacy, and computational problem-solving in a another study in 2011 [6]. Due to its displayed importance, various organizations such as the World Economic Forum and the United Nations Educational, Scientific, and Cultural Organization (UNESCO) considered CT to be a new, vital literacy that should be developed in every citizen in 2015 [14, 39]. As a result, many educational researchers have been experimenting with new ways to bring CT into the classroom [15].

2.2 Creativity

Like computational thinking, creativity was stated in 2017 to be a vital skill for modern citizens [37] that people can train starting at a young age as shown by some studies published in 2004 and 2010 [7, 44]. It was shown to promote academic achievement and motivate students in engaging more with learning in the classroom as demonstrated in 2009 and 2013 [3, 10]. Thanks to this, many researchers have studied creativity extensively through different perspectives, such as in 2009 by Kaufman and Beghetto [20]. Creativity was studied as a process by Guilford in 1950 [16] and as a personal trait by Parsons in 1971 [33]. It has been assessed through the products of creativity themselves by Martindale in 1989 [30]. While creativity is widely considered an important trait, there is still a lot of debate about the definition of creativity and how it can be measured, as reviewed by Kilgour in 2006 [21].

Even so, the general consensus is that creativity is a multidimensional concept that is composed of four key characteristics. The first is **Fluency** – the ability to generate a large number of ideas and directions of thought for a particular problem. The second is **Flexibility** – the ability to think about as many uses and classifications as possible for a particular item or subject. The third is **Originality** – the ability to think of ideas that are not self-evident or banal or statistically ordinary, but rather those that are unusual and even refuted. The last is **Elaboration** – the ability to expand an existing idea and to develop and improve it by integrating existing schemes with new ideas [30].

Creativity is a wide-ranging topic that has been discussed and examined through different perspectives. One question surrounding the creativity debate asks whether it is a fixed trait or a trainable and enhanceable skill as discussed by Amabile and Pillemer in 2012 [2]. Plucker and Beghetto in 2004 asked if creativity is domain-general or domain-specific [34], or in other words whether creativity can be employed equally effectively from one context to another. There is still a lot of uncertainty regarding this aspect of creativity, but some recent research by Baer in 2010 has suggested that it is in fact both domain-general and domain-specific [4]. It is clear that there is still a plethora of facets that can be discovered about creativity.

Overall, these questions help drive us to explore how creativity is expressed through the learning process and attempt to create a measure that can accurately capture one's creativity. Particularly, we are trying to quantify students' creativity through the key characteristic of originality by analyzing their code (which can be considered as products of their creativity) for ideas that are not statistically ordinary.

2.3 Creativity and Computational Thinking

Creativity in recent years was shown to have a significant correlation with computational thinking and was acknowledged to have a positive impact on all fields of study as shown by Romeike in 2007 [35]. More specifically, researchers in 2015 and 2016 demonstrated that computational problem solving helped inspire creativity in producing art [26, 40] and that creativity can help facilitate the process of solving computational problems as shown in 2019 [22]. It was also demonstrated by Liu and Lu in 2002 that standardized creativity tests were able to predict the creativity of computer programming solutions [28]. The reason behind the strong correlation between creativity and CT was suggested to be that the two share a set of thinking tools, such as observation, imagination, visualization, abstraction, and creation as reasoned by Yadav and Cooper in 2017 [46].

Most of the studies examining CT and creativity did so through different perspectives. Some studies such as the Seo and Kim 2016 study simply examine the mutual impact of one over the other and studied creativity within the scope of CT [40]. Doleck et al. had a similar approach and examined the association between creativity in the context of CT and academic success [12]. Other studies assessed the expression of creativity in CT tasks such as programming activities [8]. In addition, some studies specifically explored the association between creativity and CT [17].

A key study that this work is inspired by is Hershkovitz et al.'s study, where they found positive associations between computational creativity and the fluency and flexibility dimensions of creative thinking [17]. They further build on their findings with their 2021 study, where they found that CT had significant negative correlations with the flexibility and originality dimensions of creativity thinking [18].

Considering the significant correlation found between CT and creativity, we reasoned that if we could accurately measure creativity, it would also serve as a good way to assess the effectiveness of CT development curricula. With a reliable way to assess CT and creativity, education can be further improved upon by identifying good and bad practices in the classroom for CT and outlining a clearer future direction for CT curricula as a whole.

2.4 Calculating Creativity in Programming

It has been a common approach to use machine learning and artificial intelligence to predict creativity scores in programming [29, 24, 23]. One study that is highly similar to the approach we are proposing is one conducted by Kovalkov et al. They attempted to predict the creativity scores of Scratch programs by training a machine learning model. However, like many of the other cited studies, their study did not have conclusive results due to a discrepancy among human experts when grading the same Scratch programs for creativity [24]. While we expect to have similar troubles when validating our creativity measure, it would be very interesting to see whether our measure consistently agrees with human graders.

2.5 Tree Edit Distance

In order to create a more objective, computational approach to assessing student creativity through students' computer programs, we need a medium through which we can calculate differences between two programs in the first place. Fortunately, computer programs are written with specific, structured syntax, which enables them to be represented in other forms such as abstract syntax trees. In fact, code is very frequently converted into abstract syntax trees because code compilers do so in order to translate the program from one language to another. We deemed it most effective to exploit this characteristic of program code to computationally assess the differences between different student programs without needing to build a code converter from scratch.

Data that can be modeled as trees are commonly used in various applications, including abstract syntax trees, the JSON data format, natural language syntax trees, and even RNA secondary structures [38]. Due to the nature of these applications, calculating tree similarity was a topic of high interest. The standard that was established for calculating this was the tree edit distance. Tree edit distance is defined as the minimum-cost sequence of node edit operations that transform one tree into another. Edit operations were defined as node deletions, node insertion, and label renaming [47]. The tree edit distance was shown to be useful for applications such as computing text similarities [41].

Zhang and Shasha were the first to propose a recursive solution to calculating tree edit distance in their 1989 study [47]. Their solution recursively decomposes trees into smaller subforests. The new subforests are created by either deleting the leftmost or the rightmost root node of a given subforest. Algorithms that implement Zhang and Shasha's recursive approach are referred to as Zhang decompositions [38]. The Zhang Shasha algorithm is outlined at algorithm 1. For more information regarding the determination of "key roots" of trees and the specifics about computing the "TreeDist" used in the Zhang Shasha algorithm, please refer to Zhang and Shasha's original paper [47].

Algorithm 1: Zhang Shasha Algorithm

```
Input  : Trees $T_1$ and $T_2$
Output: $TreeDist(i, j)$, where $1 \leq i \leq |T_1|$ and $1 \leq j \leq |T_2|$
1 Preprocessing: Calculate l( ), LRkeyroots1 and LRkeyroots2;
2 for $i' := 1$ to $|LRkeyroots(T_1)|$ do
3     for $j' := 1$ to $|LRkeyroots(T_2)|$ do
4         $i = LRkeyroots1[i']$;
5         $j = LRkeyroots2[j']$;
6         Compute $TreeDist(i, j)$;
7     end
8 end
```

In regards to the runtime, the original proposed Zhang and Shasha algorithm runs in $O(n^4)$ time and $O(n^2)$ space for trees with n nodes. Demaine et al. was able to reduce the complexity further to $O(n^3)$ time with the same amount of space [11]. However, recent results showed that a subcubic TED solution may be unlikely to exist [9].

Chapter 3

Approach

3.1 Proposed Method and Definitions

To develop a new, computational method of assessing student creativity through their program code, we calculated and quantified the differences among student code by converting them into abstract syntax trees and deriving the distances between different programs. These tree distances were then averaged out so that the measure can be used to compare the creativity in one student's program with that in another student's program, both within the same coding problem. Normalizing the measure allowed us to compare the creativity of student programs across different coding problems, and it enabled us to generalize the proposed measure from the program level to the student level so that we can assign students with a creativity score. With the student creativity score, we could directly compare students in terms of their creativity in programming.

There is no universal definition of what creativity is in computer programming, yet. Even so, we still needed a definition of creativity in programming that would help direct and focus our approach to creating a new measure. For the purpose of this study, we defined **creativity in programming** as the ***difference of one***

student's code from other students' code for the same coding problem. The more unique or distinct a student's code is from his/her peers' code, the more creative it is. This definition was determined with consideration of the definition of the originality dimension of creativity as discussed in Section 2.2. Our measure for each student program should reflect this definition of creativity in programming because it is an average of distance from the code to all other code from other students for the same coding problem. By computing this value for each program, we would be able to identify which programs are more unique or original and indirectly assess creativity. While there may be some limitations regarding this approach to defining and measuring programming creativity, we deemed this method as a good first step towards objectively and computationally assessing creativity in programming.

Regarding the specific technologies utilized in this study, we used the JavaParser library to automatically generate abstract syntax trees for student written Java programs from an introductory undergraduate computer science course, and then use the Zhang Shasha algorithm to calculate the tree edit distance between each Java program. In addition to a Java implementation, we developed a Python implementation using the built-in Python ast module to generate the abstract syntax trees for student written programs from the Python dataset. We also implemented the Zhang Shasha algorithm with the same logic in Python code to calculate the tree edit distance between each student Python program.

We then defined our **creativity measure** as ***the z-score of the average of all tree edit distances of a program compared to all other programs written to address the same coding problem***. We also defined **student creativity** as ***the average of all z-scores assigned to the student's submitted programs***. We determined the relationship of our proposed creativity measure and a student's programming performance by calculating the creativity measure for each program and find the Pearson correlation coefficient between our proposed student creativity and the students' quiz scores.

For the purposes of this study, we defined a **"correct program"** as one that received full points after meeting all delineated requirements put forth by the instructor of the course for each coding problem. On the other hand, we also defined an **"incorrect program"** as one that lost any amount of points for failing to meet any or all requirements put forth by the course instructor for the respective coding problem. For clarity, we defined **"submitted programs"** as the collection of computer programs that contain both correct and incorrect programs from either a student or all students, depending on the context used. To further understand the effect of including creativity measures from incorrect problems on the relationship between the creativity measure and student programming performance, we calculated the Pearson correlation coefficient twice: once including the derived creativity measures from both correct and incorrect programs and another including only creativity measures from correct programs.

3.2 Creativity Measure Standardization

Since averages of tree edit distance can vary from one cohort of programs to another, we calculated the z-scores of the average tree edit distances in order to later calculate an overall creativity score representing a student's creativity. Different coding problems may require a longer or shorter total code length to fulfill all requirements depending on the depth and complexity of each problem, and this difference in length may cause the distribution of tree edit distances to vary more or less in magnitude among problems. This issue invalidates any comparisons between two programs written for different coding problems. By utilizing the z-score, we can make all calculated distances and averages more uniform across different coding problems while still retaining the variance and distributions found within each coding problem cohort. This should also enable us to generalize the derived creativity measures from the program level to the student level by calculating averages of the normalized tree edit distances per student and setting that as the students' creativity measure.

3.3 State or Trait Analysis

One way we assessed the validity of our creativity measure was by utilizing a state or trait analysis. In a separate study that attempted to determine whether students choosing to "game the system" could be better explained by state explanations or trait explanations, Baker trained regression models that "attempt[ed] to predict each student/lesson gaming frequency using a function on either the student, or the lesson." For context, **state** explanations are ones that "suggest that some aspect of the student's current state or situation guide a student to engage in that behavior." On the other hand, **trait** explanations are ones that "suggest that specific traits that a student has – such as personality characteristics or preferred meta-cognitive strategies – guide a student to engage in that behavior." A regression model that predicted gaming frequency with students as nominal variables was defined as a proxy for all possible **trait** explanations, and one that predicted with lessons as nominal variables was similarly defined as a proxy for all possible **state** explanations [5].

We conducted a similar analysis with our data: we used students as nominal variables to develop a model representing the trait explanations, and coding problems as nominal variables to train a regression model representing the state explanations. We decided to conduct this analysis to determine whether our creativity measure is better explained by state or trait explanations. If the trait explanations model fits the data better than the state model, that would indicate that our measure assesses some trait of the student and suggest that it would be worthwhile to continue exploring our proposed measure to see if the trait it assesses is creativity. On the contrary, if the state explanations model fits the data better than the trait model, this would indicate that our measure assesses some aspect of the coding problems being solved and that our proposed method is far from measuring the programming creativity in students. While this validation analysis does not prove that our measure assesses creativity, it does show that our measure quantifies some trait of the students and that it would be valuable to continue studying the creativity measure. It would also help us eliminate the possibility that our measure is instead measuring some other aspect such as the various coding problems found in our data.

In the Baker study, the regression models were compared to each other by their R^2 values and Bayesian Information Criterion (BiC) values. The regression model with a higher R^2 indicates that it explains the creativity measure better than the other model. In addition, the model that returns a lower BiC value is typically considered as the one that fits the data better. We considered these two measures when comparing our models and decided whether our proposed creativity measure can be better understood through state or trait explanations.

To account for the difference in the number of predictor variables between the state and trait models, we also calculated an adjusted R^2 for both of the models. The equation to calculate this is defined as:

$$Adj.R^2 = 1 - (1 - R^2)\frac{N-1}{N-k-1}$$

. By adjusting the R^2 through penalization of models with a greater number of predictors, we should be able to compare the two models in a fairer manner.

3.4 Creativity Measure Validation

In order to examine whether our new creativity measure is representative of each program's creativity, we validated the measure by curating a collection of pairs of submitted programs (with each pair having programs that belong to the same coding problem group) and had a professor, an experienced teaching assistant, and an undergraduate student majoring in computer science decide which one of each pair is more creative than another. We first observed how often the graders agreed with each other by calculating the kappa coefficient of agreement to demonstrate the difficulty of human graders agreeing on creativity in programming. We then determined whether our tree edit distance based measure agreed with the human graders regarding which one of each pair was more creative than the other. We conducted this system vs. humans comparison by first creating one decision for every pair of programs by taking the majority vote among the human graders to represent the human judgement. We then compared this aggregated human decision with the system's judgement by calculating the kappa coefficient once more to see how often the majority vote and our proposed measure agreed with each other.

3.5 Creativity Measure Stability

Another issue to explore for our creativity measure is its stability across different coding languages. The collection of student-written code from the introductory computer science course is composed only of Java programs. To explore whether the creativity measure is consistent across different languages, we also used Python programs obtained from a different university's introductory computer science course to calculate average tree edit distances and determined whether they have a similar correlation to their respective quiz scores. Even though the coding questions in the Java dataset may not be exactly the same as the ones in the Python dataset, the two sets should be similar and comparable since they were both taken from introductory, undergraduate level computer science courses that discussed similar topics.

To convert Python code into abstract syntax trees, we used Python's built-in AST module to parse Python code. The Zhang Shasha algorithm was also implemented in Python to calculate the tree edit distance between different programs.

Chapter 4

Experiments

4.1 Dataset

4.1.1 Primary Dataset: Java

The primary data used in this study includes Java programs written by undergraduate students who were enrolled and completed Dr. Davide Fossati's "CS 170: Introduction to Computer Science I" course taught at Emory University. The dataset includes data from years 2016 to 2018, and it includes both Fall and Spring semesters. To ensure student anonymity, each computer program has a corresponding Emory student ID that was encrypted for de-identification. All metadata pertaining to each program was stored in their file names. The metadata includes year, semester, course number, quiz number, encrypted student ID, coding question, total points earned, and maximum points possible. More descriptive statistics about the data can be found in Table 4.1. There is a total of 19,284 unique student programs, each written by one of the 867 students in our dataset, after the preprocessing stage. Another table describing the data after excluding the programs that did not earn maximum possible points before calculating the z-scores can be found at Table 4.2. There is a total of 12,475 student submitted programs after the omission of incorrect programs, or programs that did

Measure	Earned Score	Maximum Possible Score	Tree Edit Distance	Z-score of Distance
Mean	7.98	10.00	78.13	0
SD	8.65	9.01	66.44	1
Min	0	5	0	-1.91
25%	5	5	43.20	-0.70
50%	6	10	63.69	-0.32
75%	10	10	90.77	0.39
Max	50	50	762.88	11.00

Table 4.1: Descriptive statistics of all student submitted programs (Java data)

Measure	Tree Edit Distance	Z-score of Distance
Mean	62.41	0
SD	48.06	1
Min	3.63	-1.35
25%	36.17	-0.66
50%	51.79	-0.34
75%	74.93	-0.29
Max	670.23	8.89

Table 4.2: Descriptive statistics of correct (full score) student submitted programs (Java data)

not earn all possible points. These 12,475 programs were ones that the grading professor or teaching assistant deemed to fulfill all program requirements for their respective coding question. Since this data was taken from an introductory course, the efficiency and time complexity of student code was not considered when grading. The descriptive statistics on the data after aggregating by student is discussed later in Section 4.4.

4.1.2 Java Sample Coding Question and Student Programs

An example of one of the problems that students needed to solve with Java code is the following:

"Write a Java method upperCaseDiagonals(String s) that takes a string s and returns a 2D array of strings, with each string containing a single character from s, as shown in the examples below, where the elements on the diagonal are uppercase and the elements not on the diagonal are lowercase."

```
Example:

upperCaseDiagonals("DAYS") returns

{ {"D", "a", "y", "S"},
  {"d", "A", "Y", "s"},
  {"d", "A", "Y", "s"},
  {"D", "a", "y", "S"} }
```

Here is how one student solved the problem, earning 10 points out of 10 possible points:

```java
public class UpperCaseDiagonals {
    // write your method here
    public static void main(String[] args) {
        //test your method here
        printArray(upperCaseDiagonals("DAYS"));
        printArray(upperCaseDiagonals("weeks"));
    }
    // prints a 2D array
    public static void printArray(String[][] x) {
        for (int i = 0; i < x.length; i++) {
            for (int j = 0; j < x[i].length; j++) {
                System.out.print(x[i][j] + " ");
            }
            System.out.println();
        }
        System.out.println();
    }
    public static String[][] upperCaseDiagonals(String s) {
        int n = s.length();
        String[][] result = new String[n][n];
        String sL = s.toLowerCase();
        String sU = s.toUpperCase();
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (i + j == n - 1 || i == j) {
                    result[i][j] = sU.substring(j, j+1);
                } else {
                     result[i][j] = sL.substring(j, j+1);
                }
            }
        }
        return result;
    }
}
```

An example of how another student tackled the same problem and earned full score is shown below:

```
public class UpperCaseDiagonals {
    // write your method here
public static String [][] cased(String s) {
int n = s.length();
String [][] result = new String[n][n];
    for(int row = 0; row < s.length(); row ++) {
        for(int col = 0; col < s.length(); col++) {
            if(row==col || (row+col) == n-1) {
                result[row][col] = ("" + s.charAt(col)).toUpperCase();
            }
            else{
                result[row][col] = ("" + s.charAt(col)).toLowerCase();
            }
        }
    }
    return result;
}
    public static void main(String[] args) {
        //test your method here
        printArray(cased("DAYS"));
        printArray(cased("weeks"));
    }
    // prints a 2D array
    public static void printArray(String[][] x) {
        for (int i = 0; i < x.length; i++) {
            for (int j = 0; j < x[i].length; j++) {
                System.out.print(x[i][j] + " ");
            }
            System.out.println();
        }
        System.out.println();
    }
}
```

These examples highlighted how student solutions to the same coding problem can differ from one another, but still earn full scores upon assessment. This also allowed us to examine the creativity measure for only full score solutions when evaluating the

Measure	Earned Score	Maximum Possible Score	Tree Edit Distance	Z-score of Distance
Mean	5.32	7.08	15.70	0
SD	2.96	2.22	14.11	1
Min	0	4	1.40	-1.43
25%	4	5	8.05	-0.58
50%	5	7	12.73	-0.28
75%	7	10	18.19	0.24
Max	13	13	158	7.42

Table 4.3: Descriptive statistics of all student submitted programs (Python data)

correlation between a student's average creativity measure of full score solutions and his/her programming performance.

4.1.3 Secondary Dataset: Python

To test our creativity measure's stability across different coding languages, we used Python programs written by undergraduate students who took Dr. Davide Fossati's introductory computer science course that he taught at Carnegie Mellon University's Qatar campus from 2013 to 2014. To ensure student anonymity, each computer program has a corresponding student ID that was encrypted for de-identification. All metadata pertaining to each program was stored in their file names. The meta data includes year, semester, quiz/exam number, encrypted student ID, coding question, total points earned, and maximum points possible. More descriptive statistics about the data, which includes a total of 1,724 student programs after the preprocessing stage, can be found in Table 4.3. Another table describing the data after excluding the programs that did not earn maximum possible points can be found at Table 4.4. There is a total of 931 student submitted programs after the omission of incorrect programs. The descriptive statistics on the data after aggregating by student is discussed later in Section 4.4.

Measure	Tree Edit Distance	Z-score of Distance
Mean	11.86	0
SD	12.12	1
Min	0.12	-1.56
25%	4.71	-0.59
50%	9.42	-0.27
75%	15.36	0.29
Max	102.55	6.20

Table 4.4: Descriptive statistics of correct (full score) student submitted programs (Python data)

4.1.4 Python Sample Coding Question and Student Programs

An example of one of the problems that students needed to solve with Python code is the following:

"Write a Python function tinyTweet(t) that takes a string t. If the length of t is longer than 10, the function returns the first 10 characters of t. If t is 7 characters or shorter, the function concatenates a smiley string ":-)" to the end of t and returns the result. Otherwise, the function returns t."

```
Examples:
  tinyTweet("good morning everyone") returns "good morni"
  tinyTweet("hello") returns "hello:-)"
```

Here is how one student solved the problem, earning 10 points out of 10 possible points:

```
def tinyTweet(t):
    if 0<=len(t)<=7:
        return t+":-)"
    elif len(t)>10:
        return t[:10]
    else:

        return t
```

An example of how another student tackled the same problem and earned full score is shown below:

```
def tinyTweet (t):

    if len(t)>10:

        result = t[:10]

    elif len(t) <=7:

        result = t + ":-)"

    else:

        result = t

    return result


print tinyTweet ("good morning everyone")

print tinyTweet("hello")
```

4.2 Preprocessing

Student programs were first exported from their original source as text files that included all student source codes. These were then converted into Java files in preparation for JavaParser processing into abstract syntax trees. Once converted, the Java programs were sorted into separate folders according to which coding problem the program was written to solve.

Within the collection of student code, some programs were written to correct, or debug, "wrong" code. For these questions, students were given a common piece of code that contained a few errors that students were asked to fix. Since all students would ultimately submit code that may not differ from one another that much, we deemed it necessary to exclude these programs from our analysis.

Another subset of code that was excluded from the analysis were questions that asked students to answer theoretical questions in addition to writing code. This is because students may have earned full points on the theoretical component of the question, but may have earned no points on the coding aspect. The creativity measure calculated from these programs would most likely not have the same correlation with the final assignment score compared to other code-only questions.

Due to the implementation of JavaParser, student code that had any issues with compiling or other syntactical issues were also excluded from the final analysis. We would not be able to use these programs especially if ASTs cannot be created for them.

In regards to the Python code, we built a Python implementation with the same logic and calculations used in the Java implementation of the creativity computation. As such, we did not convert Python code into Java code. However, the same pre-processing steps that were outlined above were done on the Python dataset as well, including the exclusion of programs that students debugged, programs that included theoretical components, and programs that could not be converted to abstract syntax trees.

4.3 Tree Edit Distance Calculations

To compute the tree edit distance between two ASTs, a Java program written by a former student assistant to Dr. Fossati was repurposed to automatically calculate distances and averages within each coding problem folder based on the Zhang-Shasha tree edit distance algorithm. For each computer program in every folder, a tree edit distance was calculated between the AST of the program and the AST of one other program in the folder. This was added to a total distance sum for each computer program, and once this was repeated for every other computer program in the folder, a final average is calculated for each computer program. Once these averages were computed, a comma-separated values (CSV) file was written for each problem folder, with each row containing meta data about a computer program and its calculated average distance. This process was completed for both Java and Python programs with their respective implementations.

As discussed previously, we needed to normalize the average distances between programs because different programming problems require varying lengths of code. Longer code tends to have a greater distance than shorter code, so directly comparing averages across different coding problems would not be accurate. The z-scores of the average tree edit distances were calculated for each computer program within their respective coding problem groups. The CSV files were imported into a Jupyter notebook using Python and the pandas library, and the z-scores were calculated using the statistical functions of the SciPy library. For the purpose of exploring the effect of including or excluding non-perfect computer programs from the creativity measure for a student, the above steps were repeated twice: once including all computer programs available in the data, and the second including only solutions that earned the maximum number of points possible.

4.4 Aggregate by Student

The z-scores were aggregated according to their corresponding student to find the average z-score per student. The sum of all points earned by each student on every computer program they submitted was also calculated, along with the sum of all possible points each student could have earned for each submitted program. With this information, a programming performance per student was derived by dividing the student's total earned points by the total maximum points the student could have

Measure	Avg. Distance Z-score	Total Earned Score	Total Possible Score	Programming Performance
Mean	0.08	177.38	222.38	73.00%
SD	0.63	93.44	88.58	24.84%
Min	-0.74	0	10	0%
25%	-0.33	108.5	174	62.42%
50%	-0.12	195	244	80.56%
75%	0.28	257	289	91.42%
Max	6.12	471	577	100%

Table 4.5: Descriptive statistics of all students (Java data)

Measure	Avg. Distance Z-score	Avg. Z-score of Full Score Programs	Programming Performance
Mean	0.03	0.06	76.89%
SD	0.58	0.78	19.58%
Min	-0.74	-0.95	7.97%
25%	-0.35	-0.38	66.34%
50%	-0.14	-0.17	82.24%
75%	0.20	0.16	91.97%
Max	6.12	7.43	100%

Table 4.6: Descriptive statistics of students with at least one full score submission (Java data)

earned. We found that there was a total of 867 unique students in the dataset. Among these students, there were 816 students who had at least one submitted program in which they scored the maximum possible points. Descriptive statistics for the data aggregated by student is outlined below in Table 4.5. Other descriptive statistics regarding only students that had at least one submitted program that earned full score can be found in Table 4.6. This subset of students was used for the second part of the Pearson correlation analysis.

The same aggregation was done for the Python data, and similar descriptive statistics regarding the included students can be found in Tables 4.7 and 4.7. We found that there is a total of 160 unique students in the dataset. Among these students, there are 150 students who had at least one submitted program in which they scored the maximum possible points.

Measure	Avg. Distance Z-score	Total Earned Score	Total Possible Score	Programming Performance
Mean	0.02	57.28	76.24	74.33%
SD	0.41	40.33	48.17	20.69%
Min	-0.74	0	4	0%
25%	-0.23	14	14	60.76%
50%	-0.26	64	91	76.79%
75%	0.14	84.25	105	90.48%
Max	2.14	182	250	100%

Table 4.7: Descriptive statistics of all students (Python data)

Measure	Avg. Distance Z-score	Avg. Z-score of Full Score Programs	Programming Performance
Mean	0	0.01	77.13%
SD	0.35	0.48	17.11%
Min	-0.59	-0.98	33.33%
25%	-0.24	-0.28	66.13%
50%	-0.03	-0.02	77.76%
75%	0.11	0.20	90.68%
Max	1.25	2.73	100%

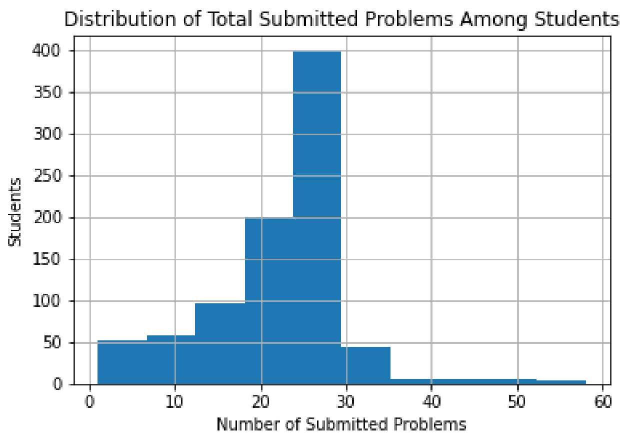
Table 4.8: Descriptive statistics of students with at least one full score submission (Python data)

4.5 Examining Data Distributions

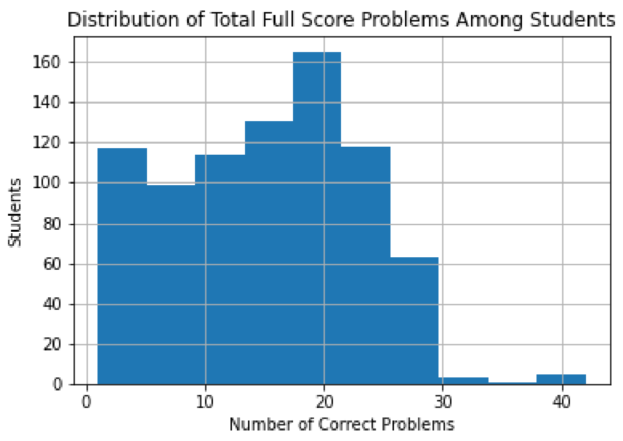
To ensure that the distributions of both submitted programs and only correct programs are similar, we graphed the distributions of the two sets of data in Figure 4.1a and Figure 4.1b, respectively. In addition, we also graphed the distribution of the proportion of correct programs over submitted programs by student in Figure 4.1c.

4.5.1 Measuring Clustering Tendency of Data

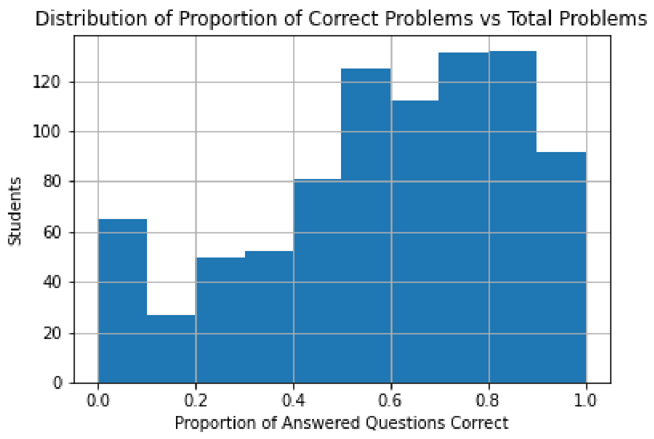
To further understand the distribution of the creativity measure in our data, we calculated the Hopkins statistic to quantify the clustering tendency of the calculated average z-scores. We calculated the Hopkins statistic twice: once with full dataset, and the other with the subset containing only correct programs. The results of the calculations can be found in Table 4.9. In both cases, the Hopkins statistic was found to be extremely close to 1 (0.997 and 0.995, respectively). This indicates that the



(a) Submitted programs



(b) Correct programs



(c) Proportion

Figure 4.1: Distributions of student submitted data after preprocessing

Inclusion Criteria	Hopkins Statistic
All Programs	0.997
Only Full Score Programs	0.995

Table 4.9: Hopkins statistics for measuring clustering tendency

data, with or without the inclusion of incorrect programs, is highly clustered.

4.6 Length vs. Uniqueness of Code

One concern regarding our proposed creativity measure is that it could simply assign higher creativity measures to programs with either more lines of code or greater overall length compared to other programs within the same coding problem group. One way that we explored this possible issue was to examine some programs to see whether programs with similar or the same total lines of code would earn noticeably different creativity measures. When we examined some programs within a few coding problems, we found that there were still programs with similar lengths that earned very different creativity measures. Two programs that exemplify this observation are included below. Note that both programs were similar in length (Program 1 has a total of 17 lines of code while Program 2 has a total of 16 lines), but they earned significantly different creativity measures. Program 1 had a creativity measure of -0.852 while Program 2 had a creativity measure of 1.764. Note that these are z-scores of distance averages, which indicate that these two programs were almost three standard deviations away from each other. Other than these two programs, we still found similar instances where programs with similar lengths of code had noticeably different creativity measures. From these observations, we inferred that our measure likely does not assign creativity measures solely based on length.

```
Program 1:
```

```
public class PickAndCount {

    // write your method here

    public static int pickAndCount(String s, int k){

        int result =0;

        for (int i=0; i<s.length();i++){

            if (s.charAt(k)==s.charAt(i)){

                result++;

            }

        }

        return result;

    }

    public static void main(String[] args) {

        // test your method here

        System.out.println(pickAndCount("fluffy",0));

        System.out.println(pickAndCount("fluffy",2));

    }

}
```

```
Program 2:


public class PickAndCount {

    public static void main(String[] args) {

        // test your method here

        System.out.println(pickAndCount("fluffy",0));

        System.out.println(pickAndCount("fluffy",2));

    }

    public static int pickAndCount(String s, int k) {
        int result = 0;

        for (int i = 0; i<s.length(); i++) {

            if(s.charAt(i) == s.charAt(k)) {

                result += 1;

                }

            }

            return result;

        }

}
```

4.7 State or Trait Analysis

As explained in an earlier section, the state vs. trait analysis is used to determine whether the creativity measure for each student can be better explained by state or trait explanations. State explanations, which were represented by the coding problems in this experiment, are ones that suggest that some aspect of the student's current situation guided a student to write the solution as he/she did. Trait explanations, which were represented by the students as nominal variables in our experiment, are ones that suggest that specific traits that a student has (which could be creativity) guided a student to write the solution as he/she did. If our results here showed that trait explanations are more likely to explain our creativity measure than state explanations, this would be a good indication that our measure does indeed quantitatively assess some trait of a student, and it may suggest that creativity or originality are some of the few possible traits it could potentially measure.

To create a regression model to serve as a proxy for the trait explanations of our creativity measure, we first consolidated all results of the tree edit distance calculations to one location. The total number of computer programs we had after the preprocessing steps and distance calculations was 19,284. From here, we generated two new datasets: one that treated students as nominal variables and another that treated coding problems as nominal variables. In both sets, the calculated creativity scores assigned to each computer program were included.

In the student dataset, we created a column for every unique student ID. If a program was associated with one student ID, the column corresponding to the student ID would be given the value of 1, and all other student ID columns would be set to 0. Essentially, we one-hot encoded for the student ID variable. The coding problem dataset was prepared in a similar fashion. This resulted with the student dataset having a total of 867 predictor columns for the 867 unique students and the coding problem dataset having a total of 101 predictor columns for the 101 unique coding problems.

Once the datasets were ready, we trained two multiple regression models using the linear regression model defined by the scikit-learn package in Python. The BiC values were calculated using the statsmodels Python package. We also trained two multiple regression models in R for a sanity check.

4.8 Validation of Student Creativity Measure

For this particular experiment, we wanted to demonstrate two particular notions. The first is that it is difficult for human graders to consistently agree on assessing creativity. This was shown in previous works when creativity experts had trouble agreeing with each other on grading the creativity of several Scratch programs [24]. We expected there to be discrepancy among the three human graders for this experiment as well. The second observation we wanted to demonstrate through this experiment is whether our proposed automatic computational method agreed with the human graders. If our results did reflect consistent or frequent agreement with human graders, we believe that this proposed creativity measure would be promising for further experimentation in the future.

To partially validate the student creativity measure, we decided to randomly select 30 separate pairs of programs, each pair from a different coding problem, and organize them onto a survey for graders to read and decide which of each pair was more creative. To isolate correctness of the program from biasing the human graders, only programs that scored full points were considered for random selection.

30 coding problems were randomly selected from the 101 problems we had in our data, and programs were separated into three equally sized bins based on their z-score of the distance measure. From those bins, one program was randomly selected from the lower group and another from the higher group to form the pair. Through random selection from high and low clusters, we would more likely select programs with more apparent differences to potentially aid and simplify the human graders' decisions. If a program with the maximum or minimum z-score of the coding problem group was selected, we would randomly re-sample the appropriate cluster again to avoid using an outlier in this experiment.

The chosen pairs of programs were copied onto a Google Forms survey to facilitate the ranking process and data collection. Each pair was separated into a separate section, and the coding question was listed prior to the two programs to brief the human graders of each problem's context so that they could make a better informed decision of which student program solved the problem more creatively. The instructions at the start of the survey informed the graders that the coding problems and programs were taken from Dr. Davide Fossati's CS 170 course from the years 2016-2018, and that only programs that earned full points were displayed.

4.9 Proposed Creativity Measure for Python Programs

Utilizing the built-in ast module in Python, we wrote code that could automatically parse Python code into ASTs and calculate tree edit distances with the Zhang-Shasha algorithm. This was done using the Java implementation as a basis, and therefore the logic and calculations used in the Python implementation should be the same as the ones found in the Java one. Student programs written in Python were processed into ASTs and the same tree edit distance calculations were done for each program as outlined above for the Java programs.

To determine if the creativity measure remains stable across coding languages, we ran the same Pearson correlation analyses to see if we would still observe a negative correlation between the creativity measure and the students' programming performance. We also replicated the state vs. trait analysis to see whether the results of the Java code analysis would be similar, if not the same, as the Python code analysis.

Chapter 5

Results

5.1 Student Creativity Measure vs. Programming Performance

To explore the relationship between our proposed creativity measure and a student's academic performance, we calculated the Pearson correlation coefficient between the student average z-score of tree edit distance and the average score earned by students for the submitted problems. In this analysis, all data that was included after the initial screening of excluding non-parseable and theoretical answers were used in the correlation calculation. This included a total of 867 students in this analysis. The final results are shown in Figure 5.1 and Table 5.1.

To better understand the effect of the inclusion of student submissions that did not receive full score on the average z-score of tree edit distance per student, we repeated the above analysis, but only using the average z-scores of computer programs that earned the maximum number of points possible. In this analysis, a total of 50 students were dropped due to them having no submissions that earned the maximum points possible. The second Pearson correlation was calculated using the creativity measures and programming performances of 817 students in total. The final results are shown

Inclusion Criteria	Pearson Coefficient	P-value
All Programs	-0.42	3.44×10^{-38}
Only Full Score Programs	-0.11	2.45×10^{-3}

Table 5.1: Pearson correlation of student average distance vs. programming performance (Java data)

in Figure 5.2 and Table 5.1.

From the figures and table, we observed that there was a statistically significant negative correlation between the students' average creativity measure and their average scores of submissions. The negative correlation and statistical significance persisted both with and without the inclusion of incorrect programs. This result could be explained by the time-limited environment of quizzes, where students that pursue more original approaches in one question could have less time overall to work on other questions. With less time, students could perform worse on quizzes overall compared to other students who submit more canonical solutions that may be simpler and/or may take less time to write.

5.1.1 Evaluating Pearson Correlation by Quadrants

In addition, due to the wide range observed in the number of programs submitted by each student after preprocessing the data, we deemed it necessary to recalculate the Pearson correlation coefficient for student data separated by groups, which were formed according to the number of submitted programs per student. We did this to try to find a more specific explanation for why there was a negative correlation between student programming performance and the student creativity measured through tree edit distances. Upon generating the summary statistics of the number of submitted programs by student, we found that the cutoff for the 25% group was 19 submitted programs and that the cutoff for the 75% group was 27 submitted programs. Lists of student IDs (one for students who had less than 19 submitted programs, one for

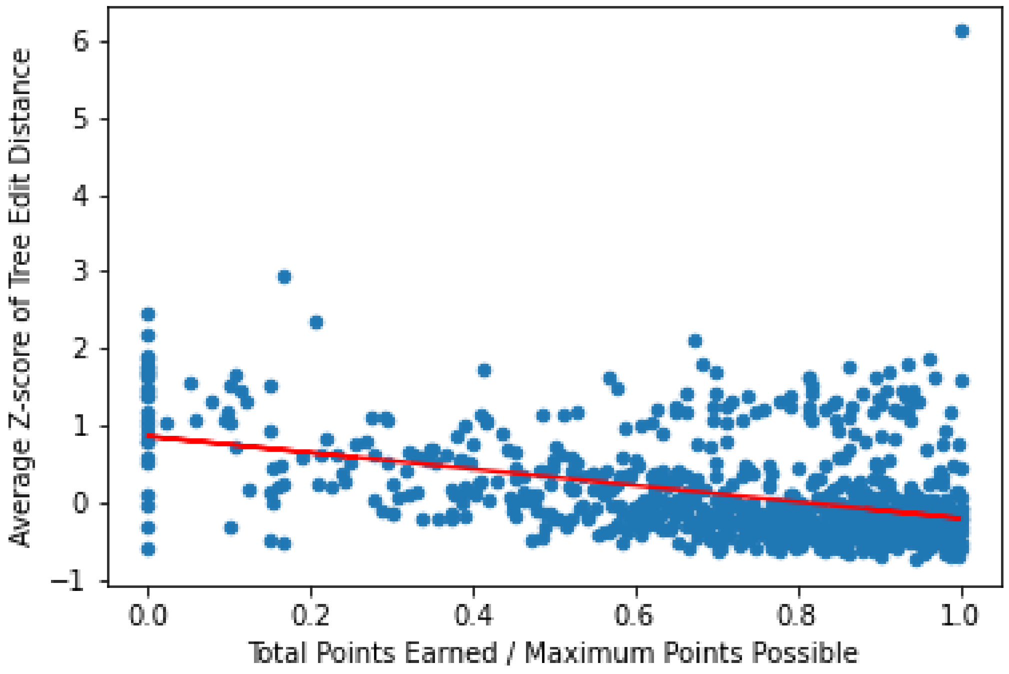


Figure 5.1: Student average tree edit distance (including all eligible programs) vs. programming performance (Java data)

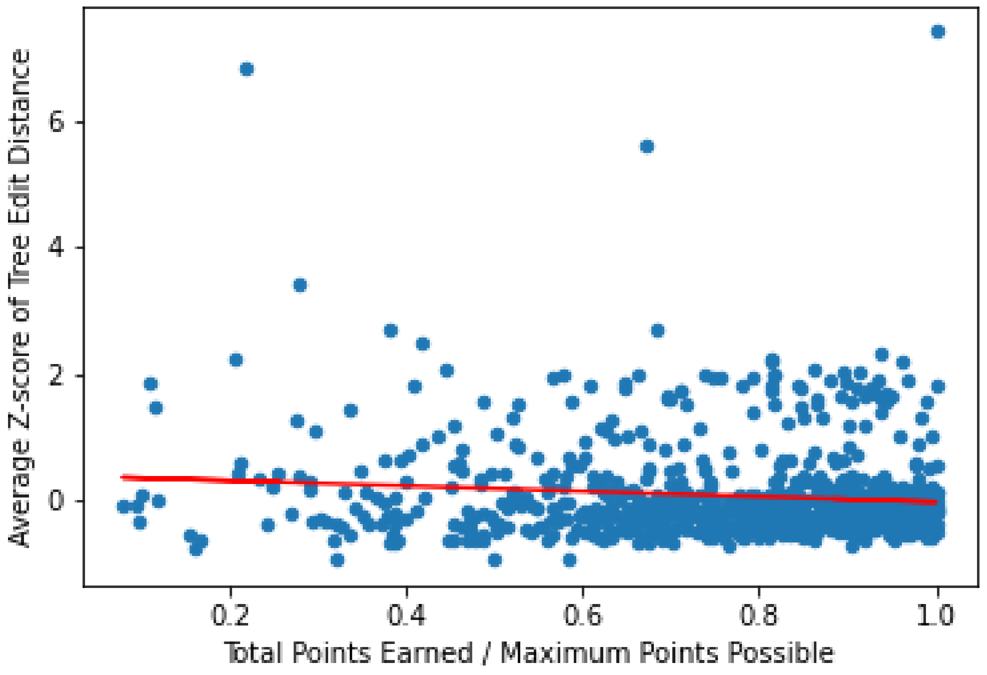


Figure 5.2: Student average tree edit distance (including only full score submissions) vs. programming performance (Java data)

Group	Pearson Coefficient	P-value
0-25%	0.001	0.99
25-75%	-0.05	0.27
75-100%	-0.11	0.06

Table 5.2: Pearson correlation of student average distance vs. programming performance by groups ordered by total number of submitted programs

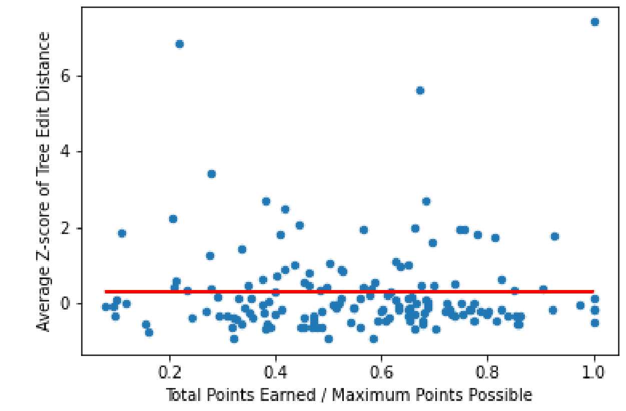
students who had more than 27 programs, and one for students who had between 19 to 27 programs) were created, and these lists were used to separate the data into different groups for analysis.

We derived and plotted the correlations of the 0-25% group, the 25-75% group, and the 75-100% group in Figures 5.3a, 5.3b, and 5.3c, respectively. The Pearson coefficients and their calculated p-values are included in the Table 5.2. With these results, we observed that the group of students who submitted more code overall had a statistically significant negative correlation between students' average creativity measure and their programming performance. This finding may not have a clear explanation, especially considering that this group of students includes students who had almost all of their submitted programs pass the preprocessing stage and students who took the course a second time and thereby having more submitted programs overall than usual.

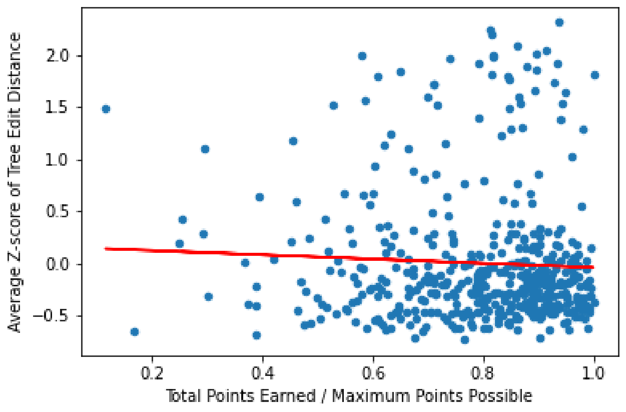
5.2 State vs. Trait Analysis

After training the multiple linear regression models, the R^2, adjusted R^2, and BiC values were derived using the same datasets that were used to train the models. The results of these calculations are summarized in Table 5.3.

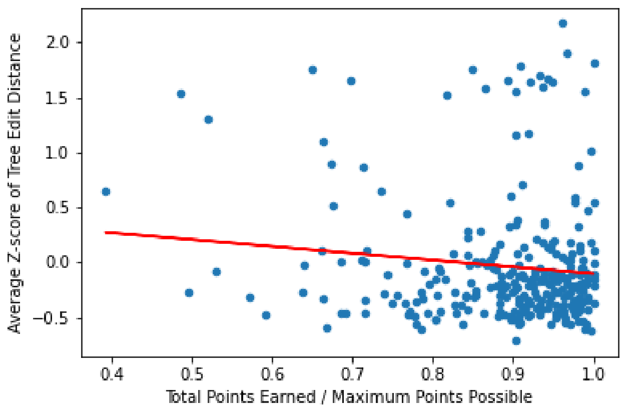
After a negative R^2 was observed, an additional sanity check was conducted on this analysis by completing the same steps in R. These results, which include an additional F-statistic and a corresponding p-value for each model, are organized in



(a) 0-25% Group



(b) 25-75% Group



(c) 75-100% Group

Figure 5.3: Pearson correlation of student average tree edit distance vs. programming performance by groups ordered by total number of submitted programs

Model	R^2	Adj. R^2	BiC
State (Coding Problem)	-2.05×10^{-6}	-5.27×10^{-3}	56606.72
Trait (Student)	0.29	0.26	55722.19

Table 5.3: State vs. trait analysis with Python scikit-learn and statsmodels (Java data)

Model	R^2	Adj. R^2	F-statistic	P-value	BiC
State (Coding Problem)	1.32×10^{-31}	-5.27×10^{-3}	2.51×10^{-29}	1	55732.1
Trait (Student)	0.29	0.26	8.78	2.2×10^{-16}	56616.6

Table 5.4: State vs. trait analysis with R (Java data)

Table 5.4.

In this experiment, we observed that the state explanations model failed to fit the data with an adjusted R^2 value of -5.27×10^{-3} and had a p-value of 1. On the other hand, we found that the trait explanations model fit the model better with an adjusted R^2 of 0.26 and a p-value of 2.2×10^{-16}. This indicates that the trait explanations explained our proposed creativity measure better than state explanations and suggests that the measure assessed a trait (or multiple traits) of the students. Since creativity or originality could be one of these traits, it would be interesting to further evaluate this creativity measure to examine whether it does indeed measure a student's creativity or another student trait.

5.3 Validation of Creativity Measure through Ranking

5.3.1 Ranking Agreement Among Human Graders

Once all three human graders (including a professor, an undergraduate teaching assistant, and an undergraduate computer science student) submitted their creativity rankings of each of the 30 pairs of programs given, we calculated the Fleiss' kappa inter-rater agreement coefficient to evaluate the agreement level among our human graders. We did this using the "irr" R package. Our findings of this analysis is arranged in Table 5.5.

From this analysis, we demonstrated that even among humans, it is difficult to determine which programs are more creative than others. This is similar to the issues that were discussed in Kovalkov et al.'s works where conclusive evidence of predicting creativity with machine learning models could not be obtained due to human expert disagreement [24].

5.3.2 Ranking Agreement Between Humans and System

In addition, we wanted to evaluate how well our creativity measure agreed with our human graders. Instead of simply adding the measure's ranking of the same 30 program pairs to the previous analysis, we deemed it more interesting to evaluate the agreement between our proposed measure and the majority vote of the three human graders. Specifically, we defined the majority vote's ranking of creativity between two programs as the program code that two or more human graders ranked as the one with higher creativity. Once we aggregated the rankings of all human graders to one choice for every pair, we then calculated Cohen's kappa inter-rater agreement coefficient to evaluate the level of agreement between our creativity measure and the human judgement as represented by the majority vote. The results of this analysis is also summarized in Table 5.5.

As discussed in McHugh's discussion of inter-rater reliability, we can interpret the 0.533 kappa statistic value as indicating moderate agreement between our proposed creativity measure and the majority vote of human graders [31]. This is already better than the inter-rater agreement found among the human graders themselves, and this suggests that our measure, while imperfect, has the potential to be a good representation of a group's opinion when ranking programs in programming creativity.

Experiment	Kappa Statistic	P-value
Human vs. Human vs. Human	-0.0714	0.498
Human (Majority Vote) vs. Creativity Measure	0.533	0.00341

Table 5.5: Kappa coefficients of inter-rater agreement from ranking creativity by human graders and creativity measure

5.4 Python Creativity Measure Analysis

5.4.1 Pearson Correlation in Python

Like the Java counterpart earlier, we calculated the Pearson correlation coefficient between the student average z-score of tree edit distance and the student programming performance for the Python data. In this analysis, all data that was included after the initial screening of excluding non-parseable and theoretical answers were used in the correlation calculation. This included a total of 160 students in this analysis. The final results are shown in Figure 5.4 and Table 5.6.

We repeated the analysis but only used the average z-scores of computer programs that earned the maximum number of points possible. In this analysis, a total of 10 students were dropped due to them having no submissions that earned the maximum points possible. The second Pearson correlation was calculated using the creativity measures and programming performances of 150 students in total. The final results are shown in Figure 5.5 and Table 5.6.

From the figures and table, we observed that there is a statistically significant negative correlation between the students' average creativity measure and their average scores of submissions. However, the negative correlation was no longer statistically significant after the exclusion of incorrect programs. This could be attributed to the significantly lower sample size of the Python data or to a possibility that the measure is not stable across coding languages.

Inclusion Criteria	Pearson Coefficient	P-value
All Programs	-0.43	1.28×10^{-8}
Only Full Score Programs	-0.04	0.66

Table 5.6: Pearson correlation of student average distance vs. programming performance (Python data)

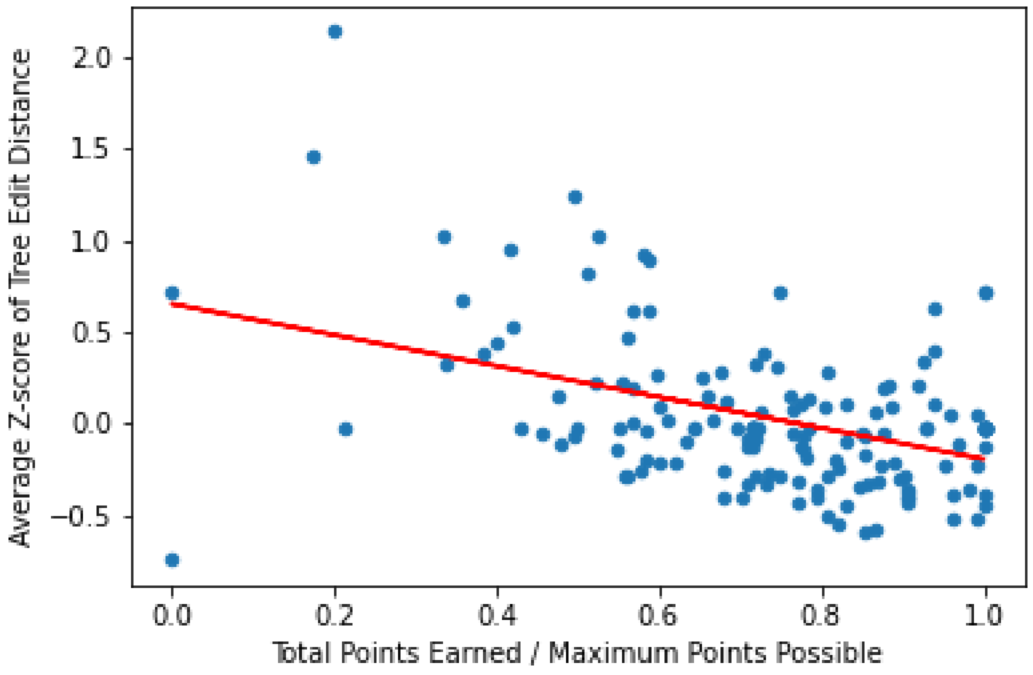


Figure 5.4: Student average tree edit distance (including all eligible programs) vs. programming performance (Python data)

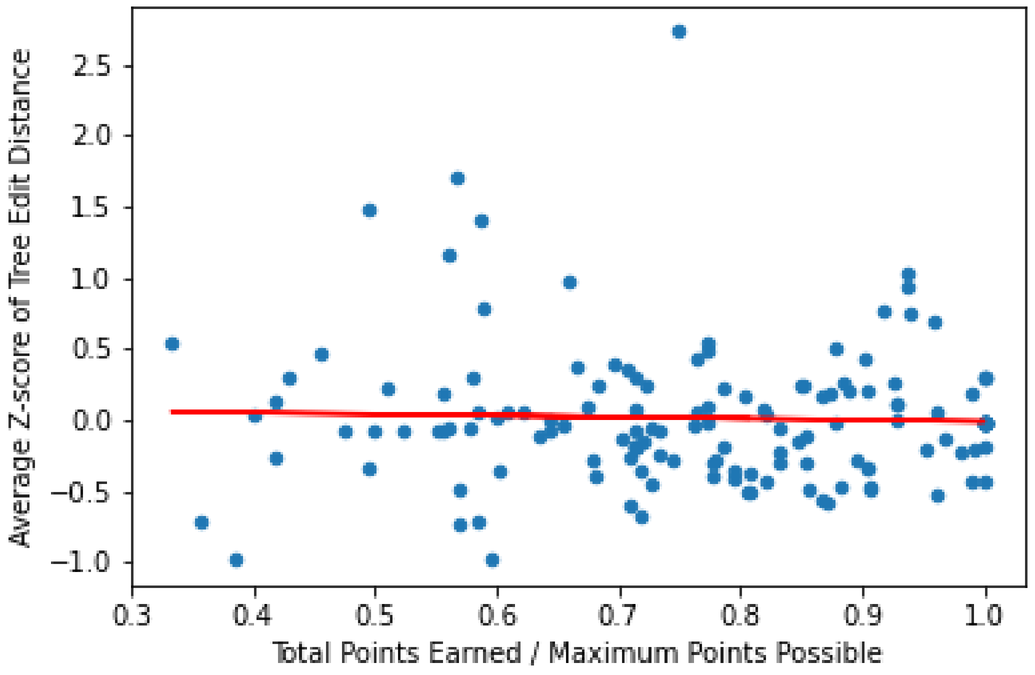


Figure 5.5: Student average tree edit distance (including only full score submissions) vs. programming performance (Python data)

Model	R^2	Adj. R^2	BiC
State (Coding Problem)	-6.23×10^{-4}	-0.02	5153.33
Trait (Student)	0.15	0.07	5799.72

Table 5.7: State vs. trait analysis with Python scikit-learn and statsmodels (Python data)

Model	R^2	Adj. R^2	F-statistic	P-value	BiC
State (Coding Problem)	5.30×10^{-31}	-0.02	2.56×10^{-29}	1	5160.79
Trait (Student)	0.15	0.07	1.76	8.92×10^{-8}	5807.17

Table 5.8: State vs. trait analysis with R (Python data)

5.4.2 State vs. Trait Analysis in Python

After training the multiple linear regression models on the Python data, the R^2, adjusted R^2, and BiC values were derived using the same datasets that were used to train the models. The results of these calculations are summarized in Table 5.3.

After a negative R^2 was observed again in the Python analysis for the Python data, an additional sanity check was conducted on this analysis by completing the same steps in R. These results, which include an additional F-statistic and a corresponding p-value for each model, are organized in Table 5.4.

In this experiment, we once again observed that the state explanations model failed to fit the data with an adjusted R^2 value of -0.02 and had a p-value of 1. On the other hand, we also found again that the trait explanations model fit the model better with an adjusted R^2 of 0.07 and a p-value of 8.92×10^{-8}. This indicated that the trait explanations explained our proposed creativity measure better than state explanations and suggested that the measure assessed a trait (or multiple traits) of the students. This result suggested that the proposed creativity measure is stable across coding languages and that the creativity measure is better explained by trait explanations than state explanations, which further bolstered the earlier indication that the proposed measure assessed a trait (or multiple traits), one of which could be creativity.

Chapter 6

Discussion

6.1 Negative Correlation between Average Tree Edit Distance and Programming Performance

Considering the statistically significant negative correlation between students' average tree edit distance and students' programming performance, this observation could be explained by the fact that these student programs were written under a time-limited environment through quizzes. With the extra time pressure, students who scored lower overall on quizzes might have received a lower distance average because they might have been less efficient in answering questions under those conditions. Students who performed better on quizzes were likely to be familiar with the concepts discussed and taught in class, and it is possible that they were able to implement correct solutions with less lines of code within the short amount of time allocated to them during the quizzes. The data could suggest that students who take a more creative or original approach in programming may perform worse overall in a class under time restrictions. However, we cannot confidently argue this with our results until further validation of our proposed creativity measure is conducted.

Upon further inspection of the Pearson correlation coefficients within groups based on total number of submitted programs, we observed that the only correlation that was near statistical significance was in the 75-100% groups. The students represented in this group were ones who either had a majority (if not all) of their programs pass the preprocessing stage, or took the CS 170 course twice after not performing well after their first semester. This negative correlation can be explained either by the explanation proposed in the previous paragraph or by the fact that students who retake the course would likely have a lower programming performance due to their worse first semester. Not enough analysis was done to determine which hypothesis was the most likely case, but further data exploration could reveal some interesting results.

6.2 Trait over State Explanations for Creativity Measure

Comparing both the R^2 and adjusted R^2 values in both the Python and R analyses, the model trained with the students as nominal variables was able to fit the creativity measure data better than the other model, with the student/trait model scoring an adjusted R^2 of 0.26 and the coding question/state model scoring an adjusted R^2 of -5.27×10^{-3} as determined by the R analysis. This suggests that the calculated creativity measure may be better understood through trait explanations rather than state explanations. This also may indicate that, in order to understand why students' programs get the creativity measure that is calculated for them, it could be more fruitful for future work to investigate trait explanations rather than state explanations.

In the Baker study, the other measure that was used to compare the two models was the BiC value [5]. Note that even though the state model had a lower BiC value of 55732.1 (versus the trait model's 56616.6 BiC value) as shown by the R analysis, we did not consider this as an indicator that the state model better fits the data because of it's F-statistic's p-value of 1. The extremely high p-value indicates that the model was not significant and that only the trait model somewhat fits the data.

6.3 Moderate Agreement Between Humans and Creativity Measure

When we calculated the kappa coefficients of inter-rater agreement for our creativity ranking experiment, we observed that, among the human graders, there was poor agreement when trying to rank one of two programs within a pair as more creative. This was evident as the Fleiss' kappa statistic for the three graders was a negative value, which is interpreted as poor agreement. This supports earlier observations of similar studies where human graders failed to reach unanimous decisions on grading creativity in Scratch programs [24]. Since there is much debate about creativity in general, there is also much difficulty for humans to agree on what is considered creative. Even so, we still conducted this experiment in hopes of validating our proposed creativity measure.

Instead of measuring inter-rater agreement of the creativity measure's ranking along with the other three human graders' rankings, we deemed it more productive and interesting to instead measure the agreement between the majority vote of the human graders and the creativity measure. After calculating the kappa statistic between these two, we found that there was a moderate agreement (indicated by a 0.533 kappa value) with a p-value of 0.00341. Even though the human graders could not agree among themselves on the creativity of programs, we observed here that the majority vote of a group of graders had a moderate agreement with the creativity measure's ranking of creativity in programs. While this does not necessarily indicate that our creativity measure does indeed assess programming creativity, this result demonstrated that our proposed measure has some promise in representing the majority vote of ranking creativity in a panel of graders. With a larger sample size and group of graders, we could potentially further validate our creativity measure.

6.4 Creativity Measure Stability

6.4.1 Negative Correlation in Python Data between Average Tree Edit Distance and Programming Performance

After repeating the Pearson correlation analysis on the Python data, we found that there was still a statistically significant negative correlation between students' average creativity measure and their programming performance. This finding provides some support for the hypothesis that our proposed creativity measure is indeed stable across different coding languages. This should be expected since the implementation of logic and calculations in the Java-based system is the same as the ones implemented in the Python-based system. While this does not provide any further insight regarding the reason behind the negative correlation, this result is still important since it suggests that this study could potentially be replicated in a different dataset that contains program code in a different coding language.

One surprising result from this experiment was that, unlike the Pearson correlation analysis conducted on the Java data, there was no statistical significance in the negative correlation when incorrect programs were excluded from the analysis. While this could indicate that our proposed creativity measure is not stable across coding langauges, we believe that this was most likely due to the significantly smaller sample size of full score submissions in the Python data vs. the Java data. The Python data had a total of 931 full score programs, while the Java data had a total of 12,475 full score programs.

6.4.2 Trait over State Explanations for Creativity Measure in Python

Comparing both the R^2 and adjusted R^2 values in both the Python and R analyses, the model trained with the students as nominal variables was able to fit the creativity measure data better than the other model, with the student/trait model scoring an adjusted R^2 of 0.07 and the coding question/state model scoring an adjusted R^2 of -0.02 as determined by the R analysis. This supports the inference made earlier for the Java data and suggests that the calculated creativity measure may be better understood through trait explanations rather than state explanations. This also indicates that the trait explanations for the creativity measure is stable across coding languages.

Since the state explanations model in the Python data analysis once again earned a p-value of 1, we opted to not consider the BiC value to compare the two multiple linear regression models. The state explanations model still does not fit the data, and even though the adjusted R^2 of 0.07 for the Python data is lower than the 0.25 adjusted R^2 of the Java data, the trait explanations model still fitted the data better than the state explanations model.

Chapter 7

Conclusion

Through the experiments conducted in this study, we demonstrated that our proposed creativity measure had a statistically significant negative correlation with a university student's programming performance under time constraints. We reasoned that this is reflective of the time-limited nature of quizzes because of how "non-canonical" or "creative" programs could take more time to complete and thereby reducing a student's overall time to work on other questions on the same quiz. While we had statistically significant results, we believe that further exploratory data analysis is required before any conclusions or interpretations can be supported by these results.

We also observed that the creativity measure was better explained through trait explanations rather than state explanations. In other words, our data showed that the variation in the creativity measure is better understood through the differences in students' characteristics or traits rather than the differences among the quiz coding problems that the students needed to solve. While this finding is derived from a very high-level view of state or trait explanations, we believe that this demonstrates that our proposed creativity measure could indeed measure student creativity since creativity is considered as a trait and is potentially the one trait (or one of many traits) that the creativity measure measured in a program.

Our results reinforced the notion that creativity, even in programming, is an intricate, complex concept that is not easily agreed upon between two or more humans. We demonstrated that there was poor agreement among three different human graders when they were tasked with ranking one program of two programs as more creative for 30 different pairs of program code. However, when we transformed the rankings of the three graders into one majority vote ranking and compared that to the creativity measure's ranking, we found that there was a statistically significant, moderate agreement between the two. Even though it is difficult for human graders to agree among themselves, we demonstrated that the creativity measure has the potential to represent the collective opinion of a group of graders through majority vote in ranking creativity in programming.

When replicating a few of our experiments on the Python data, we found similar results compared to the results from the Java data. We still observed a statistically significant negative correlation between students' average creativity measure and students' programming performance when considering all available data, and we still found that the trait explanations better explained our creativity measure compared to the state explanations. Even though there was a discrepancy of findings when analyzing only full score programs in the Pearson correlation analysis for the Python data, we believe that this is likely due to the low Python data sample size compared to the larger Java dataset.

Overall, our results indicate that the creativity measure, which utilizes abstract syntax trees and tree edit distances, is a good candidate for computationally assessing a student's creativity through their code. While further experimentation is needed to bolster the measure's validity of accurately assessing creativity, this study acts as a starting point for computational creativity research to start conducting extensive studies on the use of abstract syntax trees and tree edit distances as a standardized method of assessing student creativity in the classroom. Our results also suggest that this study can be replicated on datasets of different coding languages as well. If future work continues to show promise for our proposed measure, it could help instructors to better adapt their computational courses to each of their student's creativity level.

7.1 Limitations and Future Work

One of the caveats of our study was that our data was exclusively taken from a university course's quizzes. Not only should future works build upon this study's experiments, but replicating this study with data from either other courses of different education levels or simply just from different contexts (such as course homework) will be crucial in proving that this computational method is trustworthy and accurate universally.

To follow up from this study, further exploratory data analysis should be conducted to better understand the negative correlation between the proposed student creativity measure and student programming performance. Since some of the analysis done in this study aggregated the data by student, and students who took the course more than once were not excluded from the study, it was not clear why the students who had more submitted programs overall contributed the most to the negative correlation between the proposed creativity and programming performance.

Another limitation regarding our approach itself is whether our measure simply assigns higher values to programs that are longer and less efficient than others. While we did show that this was not always the case in Section 4.6, we did not completely rule out the possibility that our measure may be heavily influenced by length of code. While this study calculates creativity in programming only through code distance, future studies can attempt to penalize longer code to reduce the effect of length on the creativity measure.

Something that would have been interesting to explore is the clustering of the data based on the creativity measure and other variables such as quiz scores. By clustering the data, we would be able to find groups of programs that might lead to some interesting findings after examining them to find patterns of similarity among clusters or even consistent differences between different clusters. It may be worthwhile to cluster programs within coding problems and observe for any patterns, such as groups of students consistently being clustered together across different coding problems.

One principal experiment that should be done following this study is further validation of the measure. There should be a larger pool of human graders than we had for our study, and a more complex ranking of multiple programs should be conducted with the graders instead of our simple ranking between two programs at a time. Even though it has been a challenge to do so in the past due to disagreement among human graders themselves [24, 23], it will still be necessary to complete these studies in order for the proposed computational method to be more dependable as a measure.

Bibliography

[1] Exploring computational thinking. URL `https://edu.google.com/resources/programs/exploring-computational-thinking/`.

[2] Teresa M. Amabile and Julianna Pillemer. Perspectives on the social psychology of creativity. *The Journal of Creative Behavior*, 46(1):3–15, 2012. doi: https://doi.org/10.1002/jocb.001. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/jocb.001`.

[3] Kelly Anthony and Wendy Frazier. Teaching students to create undiscovered ideas. *Science Scope*, 01 2009.

[4] John Baer. *Is creativity domain specific?*, pages 321–341. The Cambridge handbook of creativity. Cambridge University Press, New York, NY, US, 2010. ISBN 978-0-521-73025-9 (Paperback); 978-0-521-51366-1 (Hardcover). doi: 10.1017/CBO9780511763205.021. URL `https://doi.org/10.1017/CBO9780511763205.021`.

[5] Ryan Baker. Is gaming the system state-or-trait? educational data mining through the multi-contextual application of a validated behavioral model. 01 2007.

[6] Valerie Barr and Chris Stephenson. Bringing computational thinking to k-12: what is involved and what is the role of the computer science education community? *ACM Inroads*, 2, 03 2011. doi: 10.1145/1929887.1929905.

[7] Ronald A. Beghetto. *Creativity in the Classroom*, page 447–464. Cambridge Handbooks in Psychology. Cambridge University Press, 2010. doi: 10.1017/CBO9780511763205.027.

[8] Vicki Bennett, Kyu Han Koh, and Alexander Repenning. Computing creativity: Divergence in computational thinking. pages 359–364, 03 2013. doi: 10.1145/2445196.2445302.

[9] Karl Bringmann, Paweł Gawrychowski, Shay Mozes, and Oren Weimann. Tree edit distance cannot be computed in strongly subcubic time (unless apsp can). *ACM Trans. Algorithms*, 16(4), jul 2020. ISSN 1549-6325. doi: 10.1145/3381878. URL https://doi.org/10.1145/3381878.

[10] Dan Davies, Divya Jindal-Snape, Chris Collier, Rebecca Digby, Penny Hay, and Alan Howe. Creative learning environments in education—a systematic literature review. *Thinking Skills and Creativity*, 8:80–91, 2013. ISSN 1871-1871. doi: https://doi.org/10.1016/j.tsc.2012.07.004. URL https://www.sciencedirect.com/science/article/pii/S187118711200051X.

[11] Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann. An optimal decomposition algorithm for tree edit distance. *ACM Trans. Algorithms*, 6(1), dec 2010. ISSN 1549-6325. doi: 10.1145/1644015.1644017. URL https://doi.org/10.1145/1644015.1644017.

[12] Tenzin Doleck, Paul Bazelais, David John Lemay, Anoop Saxena, and Ram B. Basnet. Algorithmic thinking, cooperativity, creativity, critical thinking, and problem solving: exploring the relationship between computational thinking skills and academic performance. *Journal of Computers in Education*, 4(4):355–369, December 2017. ISSN 2197-9995. doi: 10.1007/s40692-017-0090-9. URL https://doi.org/10.1007/s40692-017-0090-9.

[13] Loretta Donovan, Tim D. Green, and Candice Mason. Examining the 21st century classroom: Developing an innovation configuration map. *Journal of Educational Computing Research*, 50(2):161–178, 2014. doi: 10.2190/EC.50.2.a. URL `https://doi.org/10.2190/EC.50.2.a`.

[14] World Economic Forum and Boston Consulting Group. *New Vision for Education: Unlocking the Potential of Technology.* British Columbia Teachers' Federation, 2015. URL `https://books.google.com/books?id=h7ZorgEACAAJ`.

[15] Mariluz Guenaga, Iratxe Mentxaka, Pablo Garaizar, Andoni Eguíluz, Sergi Falip, and Isidro Navarro. Make world, a collaborative platform to develop computational thinking and steam. pages 50–59, 05 2017.

[16] J. P. Guilford. Creativity. *American Psychologist*, 5:444–454, 1950. doi: 10.1037/h0063487. URL `https://doi.org/10.1037/h0063487`.

[17] Arnon Hershkovitz, Raquel Sitman, Rotem Israel-Fishelson, Andoni Eguíluz, Pablo Garaizar, and Mariluz Guenaga. Creativity in the acquisition of computational thinking. *Interactive Learning Environments*, 27(5-6):628–644, 2019. doi: 10.1080/10494820.2019.1610451. URL `https://doi.org/10.1080/10494820.2019.1610451`.

[18] Rotem Israel-Fishelson, Arnon Hershkovitz, Andoni Eguíluz, Pablo Garaizar, and Mariluz Guenaga. The associations between computational thinking and creativity: The role of personal characteristics. *Journal of Educational Computing Research*, 58(8):1415–1447, 2021. doi: 10.1177/0735633120940954. URL `https://doi.org/10.1177/0735633120940954`.

[19] Filiz Kalelioglu, Yasemin Gulbahar, and Volkan Kukul. A framework for computational thinking based on a systematic research review. *Baltic Journal of Modern Computing*, 4:583–596, 05 2016.

[20] James C. Kaufman and Ronald A. Beghetto. Beyond big and little: The four c model of creativity. *Review of General Psychology*, 13(1):1–12, 2009. doi: 10.1037/a0013688. URL `https://doi.org/10.1037/a0013688`.

[21] Mark Kilgour. Improving the creative process: Analysis of the effects of divergent thinking techniques and domain specific knowledge on creativity. *International Journal of Business and Society*, 7, 01 2006.

[22] Siu-Cheung Kong. *Components and Methods of Evaluating Computational Thinking for Fostering Creative Problem-Solvers in Senior Primary School Education*, pages 119–141. Springer Singapore, Singapore, 2019. ISBN 978-981-13-6528-7. doi: 10.1007/978-981-13-6528-7_8. URL `https://doi.org/10.1007/978-981-13-6528-7_8`.

[23] Anastasia Kovalkov, Benjamin Paassen, Avi Segal, Kobi Gal, and Niels Pinkwart. Modeling creativity in visual programming: From theory to practice. In *Proceedings of the 14th International Conference on Educational Data Mining, EDM 2021, virtual, June 29 - July 2, 2021*. International Educational Data Mining Society, 2021.

[24] Anastasia Kovalkov, Benjamin Paaßen, Avi Segal, Niels Pinkwart, and Kobi Gal. Automatic creativity measurement in scratch programs across modalities. *IEEE Transactions on Learning Technologies*, 14(6):740–753, 2021. doi: 10.1109/TLT.2022.3144442.

[25] Elisa Kupers, Andreas Lehmann-Wermser, Gary McPherson, and Paul van Geert. Children's creativity: A theoretical framework and systematic review. *Review of Educational Research*, 89(1):93–124, 2019. doi: 10.3102/0034654318815707. URL `https://doi.org/10.3102/0034654318815707`.

[26] Kung Wong Lau and Pui Yuen Lee. The use of virtual reality for creating

unusual environmental stimulation to motivate students to explore creative ideas. *Interactive Learning Environments*, 23(1):3–18, 2015. doi: 10.1080/10494820.2012.745426. URL `https://doi.org/10.1080/10494820.2012.745426`.

[27] Yeping Li, Alan Schoenfeld, Andrea Disessa, Arthur Graesser, Lisa Benson, Lyn English, and Richard Duschl. Computational thinking is more about thinking than computing. *Journal for STEM Education Research*, 3, 05 2020. doi: 10.1007/s41979-020-00030-2.

[28] Ming-Chou Liu and Hsi-Feng Lu. A study on the creative problem-solving process in computer programming. 2002.

[29] Sven Manske and H. Ulrich Hoppe. Automated indicators to assess the creativity of solutions to programming exercises. In *2014 IEEE 14th International Conference on Advanced Learning Technologies*, pages 497–501, 2014. doi: 10.1109/ICALT.2014.147.

[30] Colin Martindale. *Personality, Situation, and Creativity*, pages 211–232. Springer US, Boston, MA, 1989. ISBN 978-1-4757-5356-1. doi: 10.1007/978-1-4757-5356-1_13. URL `https://doi.org/10.1007/978-1-4757-5356-1_13`.

[31] Mary L McHugh. Interrater reliability: the kappa statistic. *Biochem Med (Zagreb)*, 22(3):276–282, 2012.

[32] Seymour Papert. *Mindstorms: Children, Computers, and Powerful Ideas.* Basic Books, Inc., USA, 1980. ISBN 0465046274.

[33] M. J. Parsons. White and black and creativity. *British Journal of Educational Studies*, 19(1):5–16, 1971. ISSN 00071005, 14678527. URL `http://www.jstor.org/stable/3120652`.

[34] Jonathan Plucker and Ronald Beghetto. *Why Creativity Is Domain General, Why It Looks Domain Specific, and Why the Distinction Does Not Matter.* 01 2004. doi: 10.1037/10692-009.

[35] Ralf Romeike. Three drivers for creativity in computer science education. 2007.

[36] Lisa L. Ruan, Evan W. Patton, and Mike Tissenbaum. Evaluations of programming complexity in app inventor. page 2 – 5, 2017.

[37] Sameh Said Metwaly, Wim Van den Noortgate, and Eva Kyndt. Methodological issues in measuring creativity: A systematic literature review. *Creativity Theories – Research – Applications*, 4, 12 2017. doi: 10.1515/ctra-2017-0014.

[38] Stefan Schwarz, Mateusz Pawlik, and Nikolaus Augsten. A new perspective on the tree edit distance. In Christian Beecks, Felix Borutta, Peer Kröger, and Thomas Seidl, editors, *Similarity Search and Applications*, pages 156–170, Cham, 2017. Springer International Publishing. ISBN 978-3-319-68474-1.

[39] Cynthia Luna Scott. The futures of learning 2 : What kind of learning for the 21st century? 2015.

[40] Young-Ho Seo and Jong-Hoon Kim. Analyzing the effects of coding education through pair programming for the computational thinking and creativity of elementary school students. *Indian Journal of Science and Technology*, 9, 12 2016. doi: 10.17485/ijst/2016/v9i46/107837.

[41] Grigori Sidorov, Helena Gómez-Adorno, Ilia Markov, David Pinto, and Nahun Loya. Computing text similarity using tree edit distance. In *2015 Annual Conference of the North American Fuzzy Information Processing Society (NAFIPS) held jointly with 2015 5th World Conference on Soft Computing (WConSC)*, pages 1–4, 2015. doi: 10.1109/NAFIPS-WConSC.2015.7284129.

[42] Carolyn Sykora. Computational thinking for all, Apr 2021. URL https://www.iste.org/explore/computational-thinking/computational-thinking-all.

[43] Xiaodan Tang, Yue Yin, Qiao Lin, Roxana Hadad, and Xiaoming Zhai. Assessing computational thinking: A systematic review of empirical studies. *Computers and Education*, 148, April 2020. ISSN 0360-1315. doi: 10.1016/j.compedu.2019.103798.

[44] Lev Semenovich Vygotsky. Imagination and creativity in childhood. *Journal of Russian & East European Psychology*, 42(1):7–97, 2004. doi: 10.1080/10610405.2004.11059210. URL https://doi.org/10.1080/10610405.2004.11059210.

[45] Jeannette M. Wing. Computational thinking. *Commun. ACM*, 49(3):33–35, mar 2006. ISSN 0001-0782. doi: 10.1145/1118178.1118215. URL https://doi.org/10.1145/1118178.1118215.

[46] Aman Yadav and Steve Cooper. Fostering creativity through computing. *Commun. ACM*, 60(2):31–33, jan 2017. ISSN 0001-0782. doi: 10.1145/3029595. URL https://doi.org/10.1145/3029595.

[47] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18 (6):1245–1262, 1989. doi: 10.1137/0218082. URL https://doi.org/10.1137/0218082.