# On the Influence of Task Size and Template Provision on Solution Similarity

Tobias Haan, Michael Striewe[1]

**Abstract:** In most cases of programming education, there is not a single correct answer to a given task. Instead, the same problem can be solved by two or more pieces of program code that look very different. At the same time, two or more pieces of program code that look very similar may actually solve very different problems. It is thus not easy to foresee which degree of similarity one can expect for all or at least the correct submissions to a given programming task. Since several applications may benefit from some kind of prediction of the similarity, this paper presents first, preliminary results from research on that topic. In particular, it presents results from an empirical study on the influence of exercise size and template provision. Results indicate that both factors are not suitable as simple predictors and that other factors have to be taken into account as well. Nevertheless, the results help to generate hypothesis for more detailed subsequent studies.

**Keywords:** Programming Education; Program Code Similarity; Solution Space

## 1 Introduction

Computer programming is known to be a domain in which there is typically not a single correct answer to a given programming problem. Depending on the type and complexity of the problem, there might be (1) different algorithmic strategies to solve it (e. g. recursive or iterative), (2) different possible implementations for the same strategy (e. g. a for-loop or a while-loop), (3) different syntactical representations of the same implementation (e. g. `i++` or `i=i+1`), and (4) virtually unlimited options for naming identifiers (e. g. variable names). Consequently, two pieces of program code can look very different but both solve a given problem. At the same time, two pieces of program code can look very similar, but produce entirely different results due to a subtle but important deviation.

While this reduces the value of a direct comparison between two pieces of program code, programming education knows nevertheless many useful applications for measuring program code similarity on the large scale. Similarity of program code can be used to cluster the “solution space” and consequently provide similar feedback to similar solutions [Gr12]. There are also approaches for automated grading support that suggest what manually crafted feedback can be applied to which submission based on similarity [He17]. The gains in efficiency of such methods directly correlated to the similarity of submissions. Approaches

[1] Universität Duisburg-Essen, paluno - The Ruhr Institute for Software Technology, Gerlingstraße 16, 45127 Essen, Deutschland, michael.striewe@paluno.uni-due.de

doi:10.18420/abp2021-7

that require the preparation of several sample solutions for different solution strategies (like e. g. [Ol19]) may also benefit from a reliable prediction of the expected similarity or diversity of submission. An analysis of the solution space can also be used to determine the relative difficulty of a task or the degree of freedom it allows [SG13]. Another quite common application is the search for plagiarism [PPM00]. Finally, structural similarity of program code can also be used for grading [Na07, TRB04]. In any of these cases, the result of a single comparison needs careful interpretation as already mentioned above. However, also general properties of the programming task may have a remarkable influence on the results. A solution template that is provided with the task may obviously cause some minimum of similarity between all solutions, regardless of their correctness. Detailed specifications within the task description like asking for a recursive solution or asking to implement a particular class inheritance structure also limit the solution space at least for correct solutions. However, the size of such effects seems not to be entirely predictable and thus it is hard to foresee what degree of similarity or diversity among solutions can be expected for a particular task. The possibility for better predictions in that area can be considered beneficial particularly in the context of technology-enhanced assessment. Knowledge about the size of the solution space can e. g. help to determine the quality of tests and feedback rules. At the same time, clustering of similar submissions can take into account if some minimal similarity is enforced due to the use of large templates.

The goal if this paper is to systematically look for possible correlations between the size of solutions, the size of provided code templates and the kind of programming task in a sample set of exercises from a programming lecture. To do so, it applies three different measures for program similarity to hundreds of solutions for six programming exercises from an introductory lecture on object-oriented programming in Java. The goal is to generate first useful insights that help to generate hypotheses for more detailed subsequent studies.

## 2 Measurement of Program Code Similarity

There are several ways to measure the similarity of program code. One class of techniques is based on graph comparison that is applied to the syntax tree or control flow graph of a program. Another class is based on lexical comparison that is applied directly to the program code. Finally, there is also the class of metric based comparisons, in which abstract representations of programs in terms of metric values are compared. For the sake of brevity, only the three approaches and tools used throughout this study will be explained in more detail, followed by an overview on other approaches.

**Clone Doctor** (CloneDR)[2] is a commercial tool for detecting “code clones”, which are pieces of code within a larger program that are identical or very similar. Code clones are usually considered problematic with respect to software maintainability. Software quality assurance thus tries to detect such clones. The approach used by CloneDR is based on the

[2] http://www.semdesigns.com/Products/Clone/

structure of abstract syntax trees [Ba98] and thus does not care about lexical properties like variable names or code formatting. Syntax trees are cut into sub-trees and hash values are computed for these trees. Sub-trees with identical hash values are considered identical, while other sub-trees are ranked by similarity. For the purpose of this paper, we use the number of code lines that are identified as clones by CloneDR and relate it to the total size of a program. The result is a similarity value in percent.

**Deckard** is another tool that searches for code clones based on syntax trees. Different to the previous approach, it introduces the notion of "characteristic vectors" to represent sub-trees [Ji07]. Based on these vectors and the ability to merge several vectors it is able to ignore additional, intermediate tree nodes like blocks or parentheses. Besides that, it also uses clustering and hashing of vectors for similar sub-trees. It appears to be both faster and more accurate than CloneDR [Ji07].

**LAV**[3] is a general-purpose tool for static program verification. Among other techniques, it uses control flow analysis and constructs a control flow graph for that purpose. That graph can also be used to analyse the similarity of two pieces of program code using "neighbour matching" to determine the degree of similarity [Vu13]. Different to both previous approaches, using the control flow graph not only ignores lexical differences but also other syntactical differences in the implementation that have no or only very limited influence on the control flow.

There are several other approaches that have not been used in this study. TBCCD [Yu19] and Holmes [Me20] are also tree-based clone detectors but use additional information from a lexical analysis of program code or the analysis of program dependency graphs, respectively. Both approaches are quite new and seem to perform better than existing approaches, but there is only little documentation available and they could thus not be applied easily on our set of test data. Deltacon [KVF13] is an algorithm for measuring the similarity of general graphs. It could thus also be applied to syntax trees, but it is specialised on comparing graphs with known node correspondence and thus focuses on the detection of differences in the connectivity. Hence, it is not suitable for the purpose of our study.

## 3 Empirical Study

The overall plan of the study is to use existing submissions for programming exercises, compute similarity values for these and relate these to other characteristics of the exercises and submissions, i. e. to the code (template) size and the correctness of the respective submission. No sophisticated statistical methods will be used, as it is the intention to look for obvious correlations (or their absence) and to generate hypothesis for future research.

[3] urlhttp://argo.matf.bg.ac.rs/?content=lav

### 3.1 Data and Method

The empirical study is based on submissions to programming exercises from winter term 2019/20. We used six exercises from an introductory lecture on Java programming. Submissions were collected via an e-assessment system with automated feedback and students were free to submit as often as they wanted. Consequently, we collected incomplete and incorrect submissions, but also submissions that received full credit. Table 1 gives an overview on the size of each exercise (measured in lines of code for template and average submission) and the number of submissions. The templates followed the usual style of writing Java code with not more than one statement per line and the vast majority of submissions are written the same way. Hence, lines of code is indeed a usable approximation for the size of the code.

| | Avg. total submission size (LOC) | Template size (LOC) | Share of template on avg. submission | Number of total submissions | Number of correct submissions |
|---|---|---|---|---|---|
| Exercise 1 | 45 | 15 | 0.33 | 1352 | 452 |
| Exercise 2 | 228 | 145 | 0.64 | 4058 | 1182 |
| Exercise 3 | 158 | 46 | 0.29 | 3078 | 795 |
| Exercise 4 | 529 | 365 | 0.69 | 4790 | 652 |
| Exercise 5 | 69 | 34 | 0.49 | 5422 | 1897 |
| Exercise 6 | 211 | 116 | 0.55 | 3432 | 1005 |

Tab. 1: Overview on the six sample exercises used in the empirical study.

Exercise 1 is concerned with variables and the `if`-statement. The code template provides method signatures and students are asked to implement simple calculations within these methods. Exercise 2 focuses on constructors, arrays and loops. Students receive two Java files as code template with predefined method signatures and are asked to perform some operations (like searching, sorting or summarizing) on 1- and 2-dimensional arrays. Exercise 3 asks the students to implement a list data structure and perform operations on it. Students received a single file as code template and an additional file with code that must not be altered. Exercise 4 is similar to the previous one, but deals with a binary search tree as data structure. Exercise 5 is the one with the most files to be edited since it is concerned with inheritance structures. Students receive six files and must make changes and additions to four of them to realize the correct inheritance of classes and methods. Exercise 6 deals with enumerations, interfaces and maps. Again, student receive six files, but only need to alter and extend three of them. The size of the code templates given in Table 1 only refers to files that need to be touched by students and ignores files that must not be altered.

For each exercise, we created a clone detection report with CloneDR and performed a pair-wise comparison of all submissions with Deckard and LAV. We normalized all results

on a scale from 0 (no similarity) to 1 (full similarity). For Deckard and LAV, we used the average and median similarity values for a first analysis. Admittedly, both measures are weak as they can produce misleading results if the distribution of similarity values has more than one peak. However, both measure are very close to each other in most cases and a more detailed inspection of the distribution of similarity values revealed no cases in which there were two or more clear peaks. Nevertheless, the following results are only a first step towards a more detailed analysis. Some indicators for the need for a more detailed search for peaks or clusters are discussed below.

### 3.2 Results

The results from all comparisons are summarized in table 2. As a general observation, CloneDR produces clearly higher similarity values than the other tools, while LAV produces slightly higher values than Deckard in all cases except for exercise 2.

There is no obvious trend on the difference between correct submissions and all submissions. CloneDR produces lower similarity values for correct submission for all exercises except for exercise 5. Deckard and LAV produces lower values in the same three cases and higher values in the other three cases. There is no obvious correlation that these are exercises with a remarkable high or low share of correct submissions. In exercises 5 and 6 the difference between the values for all and correct submissions is larger than 0.05 in all cases (except for exercise 6 in CloneDR), while it is below 0.02 in almost all other cases.

Moreover, there is no obvious correlation between the similarity of submissions and the size or share of the code template. Exercise 1 and 3 have a low share of the code template (0.33 and 0.29) and also the lowest similarity values in all tools. However, exercise 5 has a moderate share of the code template (0.49) but high similarity values in all tools, including the highest for correct submissions in CloneDR and LAV. Exercises 1, 3 and 5 are the ones with the smallest code templates, but they span the entire range of similarity values. In turn, exercise 4 has the highest share of the code template (0.69), but remarkable lower similarity values in Deckard and LAV than exercise 2, in which the code template has a quite similar share (0.64).

There is also no obvious correlation between the exercise size in terms of code produced by students (which is the difference between the average submission size and the size of the code template) and the similarity values. Exercises 1 and 5 have a similar amount of code produced by students (30 and 35 lines), but very different similarity values. At the same time, Exercise 4 and 5 have similarity values that are close to each other, but require an entirely different student contribution (164 vs. 35 lines).

| | CloneDR | | Deckard | | LAV | |
|---|---|---|---|---|---|---|
| | all | correct | all | correct | all | correct |
| Exercise 1 | 0.480 | 0.448 | 0.224 | 0.221 | 0.238 | 0.235 |
| | | | 0.196 | 0.195 | 0.209 | 0.207 |
| Exercise 2 | 0.744 | 0.743 | 0.447 | 0.467 | 0.442 | 0.462 |
| | | | 0.447 | 0.468 | 0.442 | 0.463 |
| Exercise 3 | 0.713 | 0.695 | 0.196 | 0.195 | 0.207 | 0.207 |
| | | | 0.156 | 0.174 | 0.166 | 0.186 |
| Exercise 4 | 0.830 | 0.817 | 0.386 | 0.366 | 0.408 | 0.389 |
| | | | 0.370 | 0.360 | 0.394 | 0.383 |
| Exercise 5 | 0.759 | 0.842 | 0.398 | 0.454 | 0.422 | 0.481 |
| | | | 0.394 | 0.454 | 0.421 | 0.486 |
| Exercise 6 | 0.799 | 0.779 | 0.367 | 0.422 | 0.388 | 0.448 |
| | | | 0.337 | 0.420 | 0.359 | 0.449 |

Tab. 2: Similarity values for all and only correct submissions. For CloneDR, the overall share of clones is listed as provided by the tool report. For Deckard and LAV, average (first row per exercise) and median (second row per exercise) values from the pair-wise comparison of submissions are reported.

### 3.3 Discussion

The missing obvious correlations in the results indicate that neither exercise size in terms of code created by students nor template size in terms of lines of code provided to students are good predictors for the size of the solution space. This is not surprising on the first glance, since there are other factors like the task descriptions (that may prescribe or disallow certain patterns) and the knowledge level of students that may also increase or limit the solution space. Nevertheless, it is interesting to see that exercise 1 is among the exercises with the lowest similarity values although it is small, has a clearly defined scope and is used at the beginning of the lecture where students are expected to have a low level of knowledge. Different to that, exercise 2 is used slightly later in the lecture and theoretically gives much more possibilities to implement algorithms on arrays, but is nevertheless among the exercises with the highest similarity value.

The relative small differences between the similarity values for all and correct solutions in exercises 1 to 4 and the missing trend towards an increased or decreased similarity for correct submissions seems to hint towards a lack of discrimination between the two categories. In fact, there may be two competing factors, one for each category: On the one hand, correct submissions can be expected to be similar if student stick closely to solution patterns and strategies they have seen in the lecture or other sources. Deviations from these patterns may cause errors and thus increase the diversity of all submissions, but not the

correct ones. On the other hand, incorrect submissions may be intentionally incomplete if students have only worked on some part of the exercise so far. Consequently, they have not yet touched and filled parts of the provided code template. That may cause a higher similarity in all submissions that will vanish once they work towards a more complete submission. For exercises 5 and 6 a larger difference with higher similarity values for correct submissions could be observed. This may indicate that there are only few correct possible solutions on these exercises. For further studies, we may need to create more categories, ignore submissions that are incomplete attempts, or apply advanced statistical methods to find clusters that can be explained by other means.

With regard to tools, the results did not match our expectations we had based on the underlying approaches. Although CloneDR and Deckard are somewhat similar in their approach, they produce quite different results. At the same time, Deckard and LAV produce quite similar results although their approaches are different. Since the relative high absolute values from CloneDR do not match our subjective impression of submission similarity, we will most likely focus on the other tools in subsequent studies.

## 4 Conclusions

We investigated the influence of exercise size and template provision on submission similarity. Results indicate that neither exercise size nor template size seem to be simple predictor for the size of the solution space.

Nevertheless, the study provides useful insight that helps to generate hypotheses for subsequent studies. The first is, that a more fine-grained discrimination of submissions (e. g. creating groups of submission also for partially correct submissions or partially unchanged templates) helps to describe better how the similarity between submissions evolves over time. This may be particularly useful in the context of automated grading, where scores are easily available for each submission and where grouping by scores is more or less meaningful depending on the actual similarity of submissions. A second hypothesis is, that exercise size and template size have to be studied in conjunction with other factors like task instructions and previous knowledge of students to come up with a predictor for the size of the solution space. This is important for the further development of tool support, because some factors are much easier to measure and quantify than others. Finally, there is also room for the hypothesis that existing approaches for computing the similarity of program code are not entirely suitable for the context of (small) programming exercises. Further studies on the agreement between tools as well as on the agreement with a subjective human understanding of code similarity may help to develop approaches that are specifically tailored for the use in conjunction with programming exercises.

## Bibliography

[Ba98] Baxter, Ira D.; Yahin, Andrew; Moura, Leonardo; Sant'Anna, Marcelo; Bier, Lorraine: Clone detection using abstract syntax trees. In: Proceedings of the International Conference on Software Maintenance. IEEE, pp. 368–377, 1998.

[Gr12] Gross, Sebastian; Mokbel, Bassam; Hammer, Barbara; Pinkwart, Niels: Feedback Provision Strategies in Intelligent Tutoring Systems Based on Clustered Solution Spaces. In (Desel, Jörg; Haake, Joerg M.; Spannagel, Christian, eds): DeLFI 2012: Die 10. e-Learning Fachtagung Informatik. Hagen, Germany, pp. 27–38, 2012.

[He17] Head, Andrew; Glassman, Elena; Soares, Gustavo; Suzuki, Ryo; Figueredo, Lucas; D'Antoni, Loris; Hartmann, Björn: Writing Reusable Code Feedback at Scale with Mixed-Initiative Program Synthesis. In: Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale. pp. 89–98, 2017.

[Ji07] Jiang, Lingxiao; Misherghi, Ghassan; Su, Zhendong; Glondu, Stephane: DECKARD: Scalable and Accurate Tree-Based Detection of Code Clones. In: 29th International Conference on Software Engineering (ICSE'07). pp. 96–105, 2007.

[KVF13] Koutra, Danai; Vogelstein, Joshua T.; Faloutsos, Christos: Deltacon: A principled massive-graph similarity function. In: Proceedings of the 2013 SIAM International Conference on Data Mining. SIAM, pp. 162–170, 2013.

[Me20] Mehrotra, Nikita; Agarwal, Navdha; Gupta, Piyush; Anand, Saket; Lo, David; Purandare, Rahul: Modeling Functional Similarity in Source Code with Graph-Based Siamese Networks. CoRR, abs/2011.11228, 2020.

[Na07] Naude, Kevin Alexander: , Assessing Program Code through Static Structural Similarity. Master's Thesis, Faculty of Science, Nelson Mandela Metropolitan University, 2007.

[Ol19] Olbricht, Christoph: Trace-Vergleich zur Feedback-Erzeugung im automatisierten E-Assessment-System JACK. In: Proceedings of the Workshop Automatische Bewertung von Programmieraufgaben (ABP 2019). pp. 11–18, 2019.

[PPM00] Prechelt, Lutz; Philippsen, Michael; Malpohl, Guido: JPlag: Finding plagiarisms among a set of programs. publikation, Universität Karlsruhe, Fakultät für Informatik, Germany, January 2000.

[SG13] Striewe, Michael; Goedicke, Michael: Analyse von Programmieraufgaben durch Softwareproduktmetriken. In: SEUH. pp. 59–68, 2013.

[TRB04] Truong, Nghi; Roe, Paul; Bancroft, Peter: Static Analysis of Students' Java Programs. In (Lister, Raymond; Young, Alison L., eds): Sixth Australasian Computing Education Conference (ACE2004). Dunedin, New Zealand, pp. 317–325, 2004.

[Vu13] Vujošević-Janičić, Milena; Nikolić, Mladen; Tošić, Dušan; Kuncak, Viktor: Software verification and graph similarity for automated evaluation of students' assignments. Information and Software Technology, 55(6):1004–1016, 2013.

[Yu19] Yu, Hao; Lam, Wing; Chen, Long; Li, Ge; Xie, Tao; Wang, Qianxiang: Neural Detection of Semantic Code Clones Via Tree-Based Convolution. In: IEEE/ACM 27th International Conference on Program Comprehension (ICPC). pp. 70–80, 2019.