Detecting copied submissions in computer science workshops

Dick Grune
Matty Huntjens

Vakgroep Informatica
Faculteit Wiskunde & Informatica
Vrije Universiteit
De Boelelaan 1081
1081 HV AMSTERDAM

November 1989

ABSTRACT

In most computer science workshops, the students’ submissions are graded by more than one supervisor. This allows a student to submit work that has already been submitted before by another student to another supervisor, with minimal chance of being detected.
This leads to an unfair situation in which students receive points for work they have not performed and from which they have learned nothing. This paper describes how the Section Computer Science of the Vrije Universiteit in Amsterdam attempts to prevent this situation.

Introduction

Computer science workshops with multiple supervisors pose a difficult problem: “How can we see that the submitted work is not a copy?” Copy detection is not trivial, as may be seen from the fact that the workshop Introductory Programming at our faculty involved 1 coordinator and 8 supervisors, for a total of 181 students, each submitting 10 programs. This paper describes a solution based on the automatic mutual comparison of all submissions.

The Similarity Tester

The wish to have a program for finding common sections in two program texts did not first originate in plagiarism detection. Rather, its basis lay in the experience that a large (semi-)completed software project often contains stretches of similar code. When writing large software objects −operating systems, compilers, editors− the programmer often needs code similar to code that has already been written elsewhere in the same project. A common technique is to use a text editor to make a copy of the original code and adjust it to its new purpose. Good software engineering practice would probably dictate turning the code into a generic module and instantiate it for its different purposes, but few languages offer this possibility. Also, the construction of a proper interface for a generic module is an intellectually intensive affair; so the cut-paste-adjust technique is still popular.

Once the project has settled down, the question arises if these copied pairs can be found back, to see if they can be combined profitably. Most copy pairs will no longer be identical and will have undergone more or less profound modifications; unrecognizable copies are not interesting in this context, however. These considerations prompted the first author to design and implement a program, the *software similarity tester*. This program reads the code to be examined, reduces the text to a string of 8-bits “essential tokens”, and then repeatedly finds the longest common non-overlapping substring. Finally it prints a report of its findings.

The criteria for the reduction to essential tokens are programmed in a replaceable module in the similarity tester. They usually involve reducing all identifiers and numbers to a single token, <idf>, all strings to a single <string>, all characters to <char>, removing all comment and lay-out, and replacing each keyword of the programming language by a separate token. The other program text symbols (commas, semicolons, etc.) are accepted unchanged as tokens. A typical 2000-character file will result in a skeleton of about a 100 tokens.

It is not at all clear a priori that this approach has any chance of success. It is easy to imagine that even slight editing will change the skeleton of a copied piece of code to such a degree that the matching procedure described above will no longer recognize it. It is equally easy to imagine that this procedure is fraught with false positives. Practice shows, however, that meaningful results can be obtained. The dangers sketched above do materialize, but can be kept in check by specifying a minimum length for the common substrings. A good value is 24 essential tokens; values below 18 lead to spurious similarities to be reported, for example between declaration lists; values over 30 cause some meaningful matches to be missed, for example short for-statements. With a proper definition of “essential token” and a suitable value of the minimum common substring length, the software similarity tester is found to be a useful tool for locating stretches of similar code; the large majority of the reported similarities is meaningful.

Testing for Copies

Having had these good experiences with the similarity tester, we started playing with the idea of using it to find copied work in students’ submissions. Here, however, the doubts raised above were felt even more strongly; after all, each submitted program tries to solve the same programming assignment. Would this not create such a large similarity between the submitted programs that the similarity tester would see copies everywhere? And then there is the devious student, who knows quite well that his program will be checked automatically and how is fully aware of how the similarity tester works. Would not this student be able to apply simple systematic modifications to the original code, so that the result would no longer be recognized as a copy?

What is “to look like”?

To find an answer to the first question we need to examine the notion of “similarity” more closely. We define the similarity $S_N(A,B)$ of two program texts A en B under a minimum string length N as follows. Suppose we want to construct a copy of A by using segments of B only, restricting ourselves to segments of N essential tokens or more. The more A and B are similar, the larger the part of A we can construct this way. $S_N(A,B)$ is then defined as the percentage of essential tokens in A that can be constructed this way.

Usually, $S_1(A,B)$ will be near 100%, since each token in A will occur almost certainly somewhere in B too. And for sufficient large values of N, $S_N(A,B)$ will be equal to 0%, unless A is an exact copy of B. When N increases from 0 to the length of A, S will decrease monotonously from (almost) 100% to (almost) 0%. The similarity percentage S will decrease more sharply with increasing N for non-related A and B than for related A and B. Experiments show that for N=24 non-related solutions to a programming assignment have an S roughly between 20% and 50%, whereas related solutions have an S of 75% to 80% at least. This allows us to make a clear distinction between the two.

We will now examine assignment *j2* from the workshop of the Introductory Programming course described above, for which 39 submissions had been made. Each of these was compared to each of the 38 others, yielding 1274 match percentage values; these match percentages were then grouped into ranges 1-10, 11-20, ..., 91-100. Figure 1 shows the bar diagram of the frequencies of the various match percentages. We can see that in this assignment most of the submissions are between 1 and 10% similar. The distribution then trails off to 41-50%, followed by a few outliers. Each of the outliers signals an unusual situation.

Figure 2 shows the percentages for a single submission, compared to all other submitted programs. The presence of a match with a high percentage (61-70%) indicates that perhaps a copy has been detected. Human intervention is then required to decide if indeed copying has taken place, which one is the original and if perhaps both derive from a third source.

Possible countermeasures

Assessing the vulnerability of this test to actions of “clever” students requires us to delve deeper into the process of handing in and evaluating the students’ submissions. We will again take the workshop Introductory Programming as an example. The workshop consists of eleven programming assignments of increasing difficulty, called *a* through *k*. The six assignments *a* through *f* are trivial and serve as rote exercises only; the four assignments *g* through *j* are the main course; and assignment *k* is optional. Finished programming results are sent by electronic mail to the supervisor, who tests the proper functioning of the program. If the program is allright, it is stored in a database specific to the assignment. If not, the supervisor talks to the student, explaining what is wrong. The student then corrects the work and the procedure is repeated.

The programs in the database are automatically compared to each other, using a comparison process of which the similarity tester is the algorithmic core. If the database contains n programs, this process results in a list of $n-1$ percentages; a sample list is shown in Figure 3. The supervisor can access these lists. Programs with a match percentage of 60% or higher are examined personally by the supervisor, to verify originality. If there is a strong suspicion of copying, this is reported to the workshop coordinator. Match percentages of 80% or higher are always reported automatically to the workshop coordinator by the comparison process.

If everything is in order, the program code is printed and graded by the supervisor for style, structure, conformity to guidelines, etc.

Students are informed at the start of the workshop that their submissions will be checked automatically for copies. The student who desires to submit copied code without being caught faces the following challenges:

- The essential tokens of the fraudulent code must differ regularly from those of the original or the similarity tester process will sound the alarm. This means that changing names systematically, adding or removing comment and/or layout, and code motion are ineffective. It is important to note that the similarity test finds the *set of longest common substrings*, which is order-insensitive, unlike the UNIX program *diff*, which find the *longest common subsequence*, which is order-sensitive.
- The submitted code is tested by the supervisor on a testbed specific to the assignment, and has to function properly. This means that the student cannot compose the submitted code from copied segments of programs of other students. Getting such an assembly to work properly would probably involve at least as much work as doing the exercise in the first place. The test also denies the student the possibility to hand in a totally unrelated text, for example a UNIX manual page. Such a text would show a match percentage close to 0% but would not compile, let alone run correctly.

  Occasionally, a student submits his program under the wrong assignment name, or a supervisor stores it in the wrong database. Suppose , a solution to assignment *j2* is submitted as a solution to assignment *j3*. In that case the match percentage is close to 0%, which is noted by the supervisor, who can then correct the situation. Extremely low match percentages are as suspicious as very high match percentages.
- If the code passes these two hurdles, it will still be graded by personal inspection by the supervisor. This means that is has to look good and the student cannot afford to add dummy statements at irregular intervals to throw off the similarity tester. Adding useless parameters to procedures would also attract the supervisors attention.

Experiences

Of course we cannot guarantee that there is no strategy to create a copied submission that will successfully circumvent the above obstacles, but we are not aware of any such strategy. It seems reasonable to suppose that such a strategy, should it exist, would require an amount of work similar to that of just doing the assignment. Changing for-statements to while statements, splitting larger procedures into smaller ones, choosing a somewhat different data structure, are techniques that come to mind. But the result of such manipulations would hardly be a copy any more, and breaking the system would be interesting only as a sport.

It turns out that copied submissions are the exception. The similarity tester detects between one and ten attempts to submit copied material per workshop, depending on size and organization of the workshop. Attempts to mislead the similarity tester have never been observed. For obvious reasons, we have no information about undiscovered, successful attempts.

All cases found above the 80% threshold turned out to be real copies upon inspection. In each case the correspondence between the printed programs was seen to be very strong, even stronger than the 80% would suggest. For example, identifiers matched textually, or comments were identical, two features that were not included in the essential tokens and thus could not contribute to the match percentage.

Originally we set the threshold at 60%, but this turned out to be too low. First, short programs (one to one-and-a-half pages) contain so much standard material that the reported match percentage is too high; and, second, even if the match percentage was reported correctly, it is difficult to tell if a short program is a copy if it differs for 40% from the purported original. No such doubts remain with a threshold of 80%.

All students confronted with suspicion of submitting copied work confirmed this suspicion, sometimes after some discussion. Often their defense was that since they had taken great pains to change the program, it was no longer a copy. These changes were, however, usually restricted to systematically replacing identifiers and changing the order of declarations, so basically the same program was submitted. We have seen above that the similarity tester is insensitive to such changes. The 10 to 15% of code that *was* different usually consisted of rewritten output statements. The algorithm itself was never changed.

The goal of our computer science workshops is to teach the students something rather than to test and evaluate them. Therefore it is no problem when students seek help from other students. It is immaterial if they learn from the classroom lectures, the books used in class, the workshop, the supervisor, or a fellow student; in all cases they learn something and that is the only thing that matters. Selection and evaluation takes places during written tests. This attitude does not, however, imply that doing the workshop is optional. There is a big difference between hearing from somebody how a problem can be tackled and then tackle it yourself, and obtaining the complete solution from somebody else and handing it in as original work. The difference lies in the learning value.

Conclusion

Our present impression, though unproven, is that the only way to circumvent the similarity tester is by writing original code. This means that anybody who submits somebody else's work as his own, even after modifications in comment, identifiers, layout and order of program units, is exposed as a cheater.

Figures

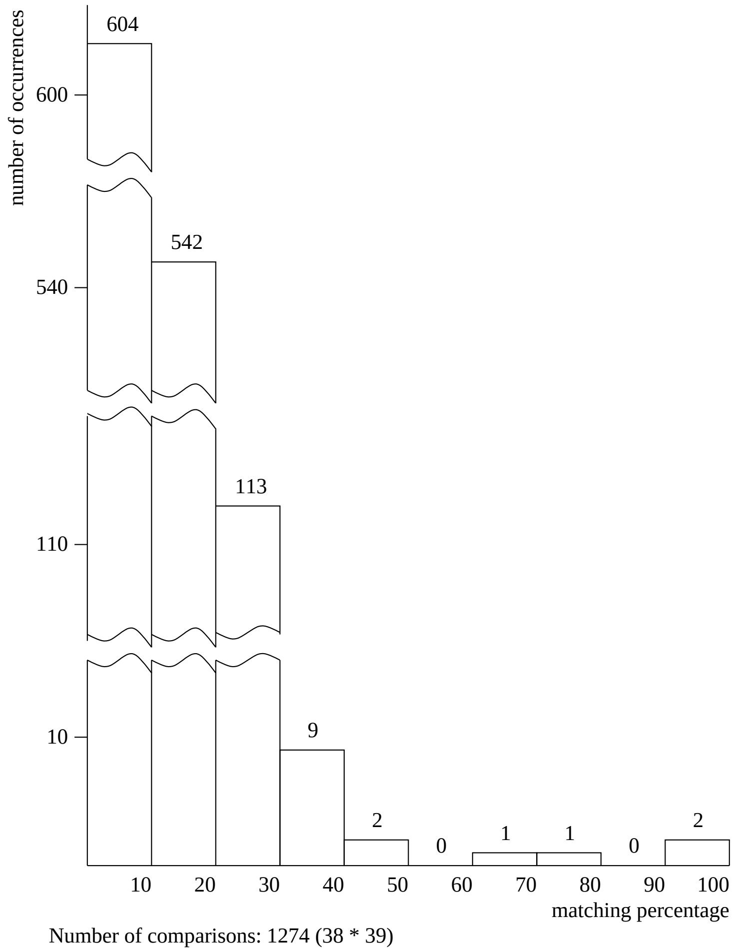

Figure 1.
Bar diagram of the match percentages of 39 submissions to assignment *j2*

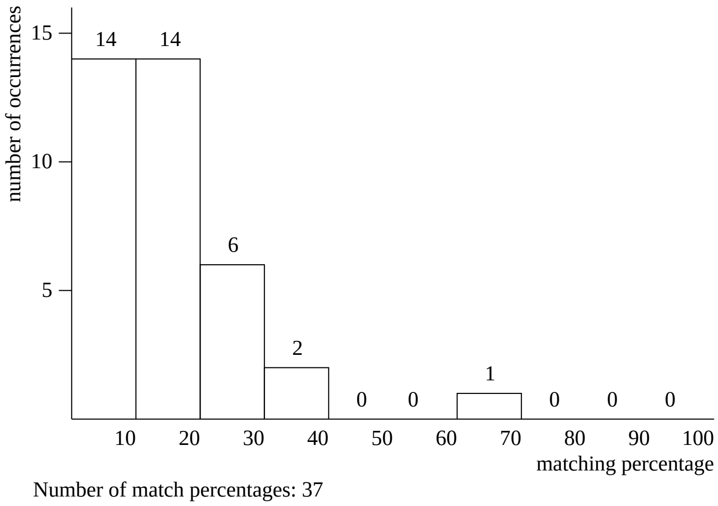

Figure 2.
Bar diagram of the match percentages of student xxxxxxx's submission for assignment *j2*

Student xxxxxxx

69%	yyyyyyy	Lines: 195	Tokens: 627
31%	???????	Lines: 96	Tokens: 282
31%	???????	Lines: 78	Tokens: 284
26%	???????	Lines: 66	Tokens: 243
25%	???????	Lines: 66	Tokens: 233
24%	???????	Lines: 66	Tokens: 226
23%	???????	Lines: 58	Tokens: 211
23%	???????	Lines: 52	Tokens: 214
21%	???????	Lines: 61	Tokens: 196
20%	???????	Lines: 49	Tokens: 188
20%	???????	Lines: 48	Tokens: 183
18%	???????	Lines: 53	Tokens: 171
18%	???????	Lines: 41	Tokens: 169
16%	???????	Lines: 39	Tokens: 152
14%	???????	Lines: 31	Tokens: 131
14%	???????	Lines: 37	Tokens: 127
13%	???????	Lines: 33	Tokens: 124
13%	???????	Lines: 32	Tokens: 125
12%	???????	Lines: 29	Tokens: 115
11%	???????	Lines: 22	Tokens: 103
11%	???????	Lines: 29	Tokens: 104
11%	???????	Lines: 21	Tokens: 101
11%	???????	Lines: 29	Tokens: 104
10%	???????	Lines: 25	Tokens: 94
10%	???????	Lines: 25	Tokens: 94
10%	???????	Lines: 25	Tokens: 95
10%	???????	Lines: 23	Tokens: 91
10%	???????	Lines: 25	Tokens: 91
9%	???????	Lines: 24	Tokens: 88
8%	???????	Lines: 11	Tokens: 73
8%	???????	Lines: 18	Tokens: 79
8%	???????	Lines: 23	Tokens: 78
8%	???????	Lines: 18	Tokens: 77
8%	???????	Lines: 25	Tokens: 80
3%	???????	Lines: 8	Tokens: 32
3%	???????	Lines: 7	Tokens: 33
3%	???????	Lines: 13	Tokens: 28

Figure 3.
List of match percentages between student xxxxxxx's submission of the program for assignment *j2* and the 38 other *j2* submissions.
One submission with a match percentage of 0% has been left out.
The names of the students have been replaced by question marks for privacy reasons.