

Using Visualization to Detect Plagiarism in Computer Science Classes

Randy L. Ribler
Department of Computer Science
Lynchburg College
Lynchburg, Virginia 24501
ribler@lynchburg.edu

Marc Abrams
Department of Computer Science
Virginia Tech
Blacksburg, Virginia 24061
abrams@cs.vt.edu

Abstract

This paper introduces a number of general methods for visualizing commonality in sets of text files. Each visualization simultaneously compares one file in the set to all other files in the set. These visualizations, which can be computed in $O(n)$ time and space, are explained and then applied to the problem of detecting plagiarism in large computer science classes. A case study is presented and sample visualizations are provided. Finally, a new interactive tool that can be used to produce and manipulate these visualizations is presented.

1. Introduction

Computer science instructors usually require students to write their programs independently. Therefore, any nontrivial programming assignment should produce a number of unique solutions equal to the number of students submitting programs. Unfortunately, some students might not observe the rules regarding plagiarism, and copy all or part of another student's program and submit it as their own work. To reduce the chance of being detected, they sometimes change the names of variables or make other cosmetic changes, but often, in large classes where the perceived probability of detection is low, they make only the most minor alterations. Perhaps if better detection techniques were available this type of plagiarism could be effectively deterred.

This paper describes the production of visualizations that show how similar one file is to the other files in a *set* of files. When applied to plagiarism detection, they show how similar one student's program is to all the other programs submitted. These "one-to-many" file visualizations can be computed in $O(n)$ time. To see how every file in a set compares to every other file in a set requires n such visualizations, rather than the $O(n^2)$ operations required for pairwise comparison. While these methods are particularly well suited to plagiarism detection, they are not limited

to that application, and represent a general method for performing one-to-many comparisons among text files.

Visualization is especially appropriate for plagiarism detection because a high level of similarity between documents can be caused by factors other than cheating. So plagiarism detection generally requires two steps. The first step is to identify suspicious programs, and the second step is to examine sections that are similar.

2. The Categorical Patterngram (CP)

In plagiarism detection, we are attempting to identify *sequences* of characters that are present in more than one program. We define k as the minimum sequence length of interest. Every k -length character sequence defines an n -gram. N -grams are the sequences of n characters that exist in a text file. For example, the first n -grams of length three in a program that starts with the word "begin", are "beg," "egi," and "gin." We do not include spaces, carriage returns, tabs, or other non-printable characters in our n -grams.

The single file in a one-to-many comparison will be referred to as the *base file*. The entire set of files involved in the comparison will be referred to as the *ensemble* [2]. We define a *categorical patterngram* [9] as a visualization that displays, for each character position x in the base file, the number of ensemble files that contain the k -length n -gram that begins at that position. A point is plotted at coordinate (x, y) if the k -length n -gram beginning at character position x in the base file occurs at any location in y files of the ensemble.

2.1. A Toy Example

Suppose that we have three files that contain text as shown in Figures 1, 2, and 3. A patterngram that uses Toy File 0 as a base file is shown in Figure 4. The points in the graph have been labeled with the corresponding letters in the files. This is done only as an aid to understanding how

Toy0: This is a test.

Figure 1. Toy File 0

Toy1: Oh yes. This is a test too.

Figure 2. Toy File 1

Toy2: Toy2 has little in common with the other two.
This is common.

Figure 3. Toy File 2

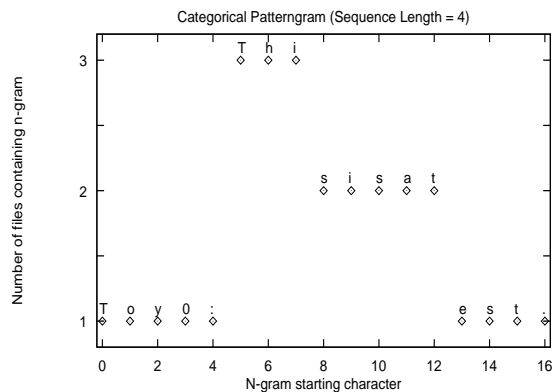


Figure 4. Patterngram of Toy File 0

the graph is generated. In real data the points are typically much too dense to allow this type of annotation.

The categorical patterngram for Toy File 0 shown in Figure 4 depicts the one-to-many comparison of Toy File 0 to Toy File 1 and Toy File 2. The first five characters in Toy File 0 produce five dots at y -coordinate one because the base file n -grams “Toy0,” “oy0:,” “y0:T,” “0:Thi,” and “:This” do not occur in the other two files. The sixth, seventh, and eighth characters (plotted at x -coordinates 5, 6, and 7, respectively) produce points at y -coordinate three because the n -grams that are initiated with those characters, “This,” “hisi,” and “isis,” occur in all three files. Characters eight through twelve each initiate n -grams that occur in one other file – in this case Toy File 1, so they are plotted at y -coordinate two. The remaining characters belong to sequences that do not occur in the other files and are therefore plotted at y -coordinate one.

In real applications the sequence lengths of interest are typically longer than four characters. In addition, the files are typically made up of thousands of characters, so it is not possible to label each point with its corresponding letter. However, these graphs are designed simply to show

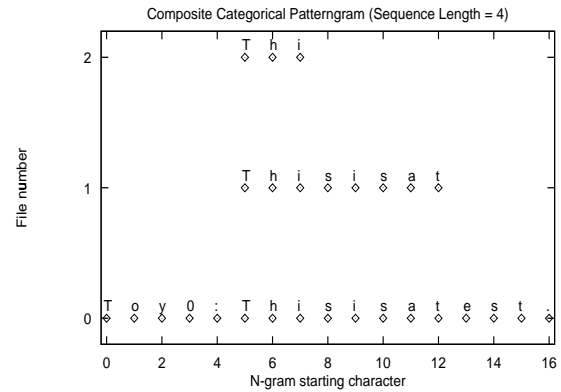


Figure 5. CCP of Toy File 0

whether or not similarities exist. Another technique will be described in section 5 that shows the n -grams that correspond to the similarities displayed in the patterngram.

The following notes highlight some important characteristics of the Categorical Patterngram (CP).

1. The n -grams that are unique to the base file are plotted at y -coordinate one.
2. N -grams in the base file that are common to many other files are plotted with a large y -coordinate. In plagiarism detection, these n -grams correspond to character sequences that naturally occur frequently without any collaboration between students.
3. The sequences that may be indications of plagiarism are indicated by points that have a relatively small y -coordinate greater than one. The value of the x -coordinate provides the viewer with an indication of where the n -gram occurs in the base file. The location of the n -gram in the other ensemble files is unavailable in this visualization.

Perhaps the most important information in the graph is provided by the presence or absence of clusters of horizontally contiguous points. For example, a contiguous section of points plotted at $y = 1$ indicates a section of code that is unique – a section where the majority of the points are plotted at $y = 1$ is also a good indication of independently developed code, particularly when the values that are not at $y = 1$ have relatively high y -coordinates. However, the absence of points at y -coordinate one for an extended section is a good indication of plagiarism. The dominant y -coordinate in such a section often indicates the number of collaborators involved.

3. The Composite Categorical Patterngram

The categorical patterngram defined in the previous section shows the degree of similarity that exists between one file and the rest of the ensemble of files. The visualization presented in this section, which we call the *composite categorical patterngram (CCP)*, shows which particular files are similar.

Each file is assigned a number in the range $[0, n - 1]$, where n is the number of files in the ensemble. A point is plotted at (x, y) if the k -length n -gram beginning at x in the base file occurs one or more times in file y .

Figure 5, shows the CCP for the Toy Files using Toy File 0 as the base file.

The following notes highlight some important characteristics of the Composite Categorical Patterngram (CCP).

1. Each of the files in the ensemble is assigned a unique integer value which is used as a y -coordinate in the graph. A point is plotted at the y -coordinate associated with the base file for every character in the base file. This usually appears as a horizontal line.
2. If a unique sequence exists in the base file, extending from character number i to character number j , then all the points plotted in the interval $[i, (j - k)]$ will be at the y -coordinate associated with the base file.
3. If a sequence in the base file interval $[i, (j - k)]$ matches a sequence anywhere in file m , then $(j - k - i + 1)$ points will be plotted at y -coordinate m in the interval $[i, (j - k)]$. This means that there will be points plotted at the y -coordinate associated with a matching file at the x -coordinates associated with the match in the base file.
4. If a sequence in the base file matches multiple files in the data set, then points will be plotted at the multiple y -coordinates associated with the matching files. These points will be plotted in the x interval corresponding to the position of the matching n -gram in the base file.

4. Creating the Patterngram

A memory-efficient algorithm that can be used to produce the patterngram and the composite patterngram uses a hash function to map the n -grams to integers, and then uses a hash table to record the number of occurrences and file id's of each k -length n -gram contained in the ensemble. Because a string table is used to hold all the files in memory, the memory requirement for the hash table is independent of k , and the total memory requirement is proportional to n , where n is the total number of characters in all the files.

Additionally, the use of recursive hashing by cyclic polynomials [4], provides a constant hashing time regardless of the size of k . In practice we have found the memory requirement for the algorithms to be approximately ten times larger than the memory required to store all the files in memory. Using this algorithm, an average Pentium-class machine can produce all the patterngrams for programs in an average computer science class in just a few seconds.

5. Pattern Text View

The *pattern text view* provides an additional many-to-one visualization for text files. It formats the text from the base file so that each character that begins a non-unique sequence is highlighted, either using an underlined font or using an alternative font color. The text view has a simple relationship to the patterngram. If the value of the y -coordinate corresponding to the k -length n -gram beginning at that character is one, then the character is not highlighted. If the corresponding y -coordinate is greater than one, then the character is highlighted. Thus, letters beginning unique n -grams are distinct from letters beginning non-unique n -grams.

6. Changing Variable Names

Sometimes students will change the variable names in a program in an attempt to mask plagiarized code. This technique will reduce the efficacy of the patterngrams, but will not eliminate it altogether. However, the effects of variable name changes can be defeated through the insertion of a simple preprocessing step known as *tokenization*. The tokenization converts all the alphanumeric strings in a program into single characters. The special characters, which include everything other than the alphanumeric characters, are left unaffected. This produces three effects. The first is that two files that contain the same programs with different variable names will produce identical patterngrams. The second effect is a reduction in the size of the data set. The third effect is that the ensemble becomes more homogeneous, so an increase in k will probably be justified. The results of our case study seem to indicate that this transformation has little adverse effect on the visualizations and has all the positive effects described.

7. A Case Study

A case study was performed to see if these visualization methods could be successfully applied in a classroom situation. The students had been provided with a small sample program and asked to expand it. The sample program contained approximately 30 lines of code. The average completed program was less than 150 lines. Students submitted

their programs via email, and each program received was assigned a sequential submission number.

7.1. Typical Graphs

Figure 6 shows the patterngram ($k = 10$) for submission number 23. The majority of the points are plotted at y -coordinate one, indicating that most of the ten-character sequences in this file are unique to this file. There are a number of points plotted at y -coordinate two, but these are relatively evenly distributed. Because n -grams that occur frequently are not of interest here, all points with ($y \geq 10$) are plotted at ($y = 10$). This provides a better visualization of the points in the interval 1 to 10, which is the critical range in this application. This graph is typical of a graph associated with a program that was written independently.

The CCP ($k = 10$) for submission number 23 is shown in Figure 7. The solid horizontal line formed by contiguous dots at $y = 23$ is present because all the n -grams in submission 23 must of course exist in submission 23. CCPs will always have a horizontal line of points at the y -coordinate corresponding to the file whose n -grams are used as the basis for the graph. Figure 7 also shows a large number of points plotted in the range $x = 0$ to $x = 500$. These points correspond to the characters in the email message header. The headers contain many of the same words and phrases, although the headers are not identical. The effects of the headers are visible in all the CCPs in this data set. There are other places, such as near $x = 700$, where a particular n -gram from submission 23 is found to be present in virtually all of the graphs. These matches are caused by common words or phrases which naturally occur in independently developed programs or by the sample program that the instructor provided. Color has been added to the displays to distinguish between the base file (blue), the infrequently matched n -grams (red), and frequently matched n -grams (green).

7.2. Submissions Showing Collaboration

Figures 8 and 9 show the CP and CCP for submission number 45. This program is nearly identical to submission number 44, which was submitted by a different author. The nearly solid lines in the CCP from near $x = 1500$ to near $x = 2500$ are a clear indication that the code sections of these two programs are the same.

While the plagiarism in Figures 8 and 9 is quite blatant, several more subtle instances of plagiarism, such as that shown in Figure 10 were also detected.

To further investigate the efficacy of this approach, we conducted a number of experiments in which we tried to copy a program from the original submissions and disguise its presence. We made copies and changed all the variable

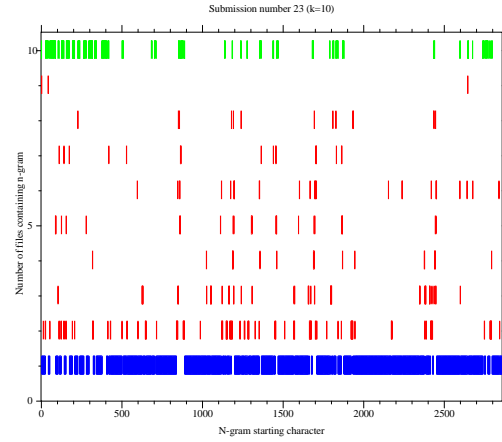


Figure 6. Patterngram of Submission 23

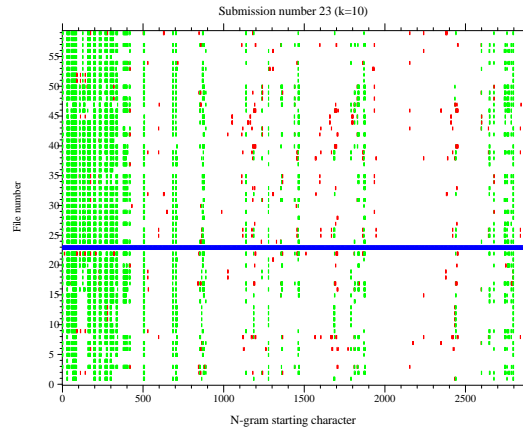


Figure 7. CCP of Submission 23

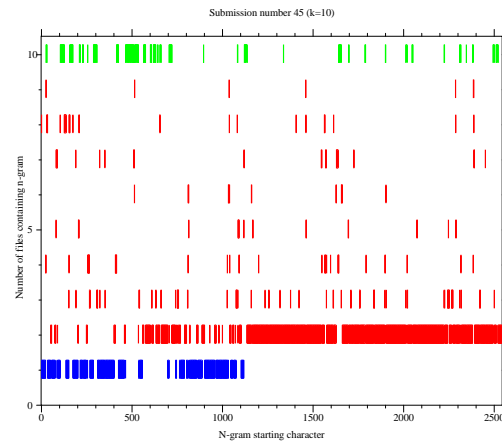


Figure 8. Patterngram of Submission 45

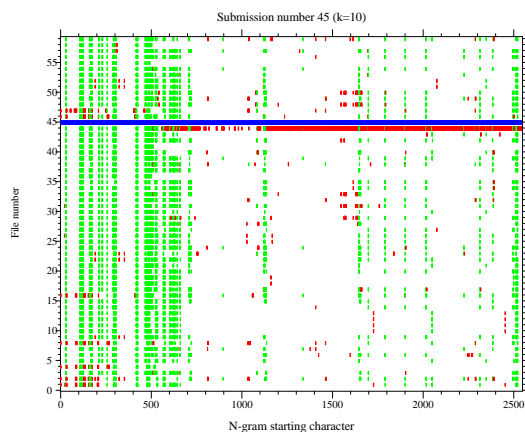


Figure 9. CCP of Submission 45

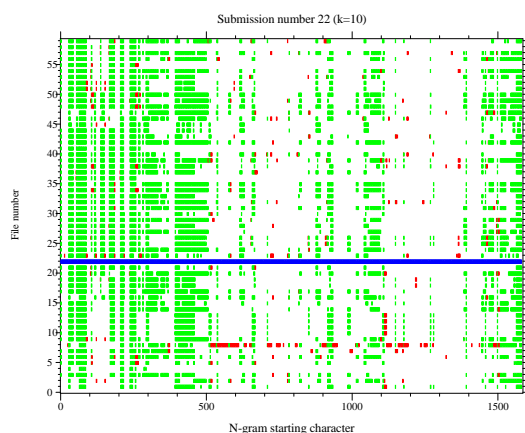


Figure 10. CCP of Submission 22

names, we made copies and reversed all the lines, we even copied as few as five lines from one program to another. All of these attempts were visible in our graphs.

8. Status and Future Directions

We have incorporated all the visualizations presented in this paper into CHITRA, a Unix-based tool originally designed to analyze computer and network performance data and, more recently, into an interactive tool called *Same*, which runs under Windows98 or WindowsNT. For more information, or to download this software, visit http://lasi-lynchburg.edu/ribler_r/public/same.

We are currently investigating other applications in which the patterngrams might be useful. These include software configuration management (visualizing how a file changes across versions), genetic algorithms (visualizing how a population of genes converge), and computer performance analysis (detecting and identifying cyclic behavior).

References

- [1] M. Abrams. CHITRA95 — *User Manual*. Computer Sci. Dept., Virginia Tech, Blacksburg, VA 24061-0106, May 1996. <URL: <http://www.cs.vt.edu/~chitra>>.
- [2] M. Abrams, N. Doraswamy, and A. Mathur. Chitra: Visual analysis of parallel and distributed programs in the time, event, and frequency domain. *IEEE Trans. on Parallel and Distributed Systems*, 3(6):672–685, Nov. 1992.
- [3] E. H. Chi, P. Barry, E. Shoop, J. V. Carlis, E. Retzel, and J. Riedl. Visualization of biological sequence similarity search results. In *Proceeding Visualization '95*, pages 44–51, Atlanta, GA, Oct. 1995.
- [4] J. D. Cohen. Recursive hashing functions for n-grams. *ACM Transactions on Information Systems*, 15(3):291–320, July 1997.
- [5] M. A. Hearst. Tilebars: Visualization of term distribution information in full text information access. In *Conference proceedings on Human factors in computing systems*, pages 59–66, 1995.
- [6] J. Mothe and T. Dkaki. Interactive multidimensional document visualization. In *Proc. SIGIR'98*, Melbourne, 1998. ACM.
- [7] L. T. Nowell, R. K. France, D. Hix, L. S. Heath, and E. A. Fox. Visualizing search results: Some alternative to query-document similarity. In *Proc. of SIGIR96*, Zurich, Aug. 1996.
- [8] R. Ribler, A. Mathur, and M. Abrams. Visualizing and modeling categorical time series data. In *Symposium on Visualizing Time-varying Data*, volume NASA Conference Publication 3321, Williamsburg, VA, Jan. 1996. ICASE and NASA/LaRC. <URL: <http://www.cs.vt.edu/~chitra/docs/95vtvdRMA/95vtvdRMA.html>>.
- [9] R. L. Ribler. *Visualizing Categorical Time Series Data with Applications to Computer and Communications Network Traces*. PhD thesis, Virginia Polytechnic Institute and State University, 1997.
- [10] M. Rorvig, M. M. Smith, and A. Uemura. The n-gram hypothesis applied to matched sets of visualized japanese-english technical documents. In *Proceeding of the 1999 Annual Meeting of the American Society for Information Science, Knowledge: Creation, Organization and Use*, Washington D.C., Oct. 1999.
- [11] W. Scullin, T. Kwan, and D. Reed. Real-time visualization of world wide web traffic. In *Symposium on Visualizing Time-varying Data*, volume NASA Conference Publication 3321, Williamsburg, VA, Jan. 1996. ICASE and NASA/LaRC.
- [12] I. M. Soboroff, C. K. Nicholas, J. M. Kulka, and D. S. Ebert. Visualizing document authorship using n-grams and latent semantic indexing. In *Proc. NPIV '97*, Las Vegas, Nevada, 1997.
- [13] D. Wu, J. Roberge, D. J. Cork, B. G. Nguyen, and T. Grace. Computer visualization of long genomic sequences. In *Proceeding Visualization '93*, pages 308–315, Atlanta, GA, 1993.