Journal of Interactive Learning Research (2010) **22**(3), 379-400

Automatic Plagiarism Detection with PAIRwise 2.0

KEVIN ALMEROTH AND ALLAN KNIGHT
UC-Santa Barbara
almeroth@cs.ucsb.edu
aknight@cs.ucsb.edu

As part of the research carried out at the University of California, Santa Barbara's Center for Information Technology and Society (CITS), the Paper Authentication and Integrity Research (PAIR) project was launched. We began by investigating how one recent technology affected student learning outcomes. One aspect of this research was to study the effect plagiarism detection had on student attitudes and behaviors in regards to plagiarism detection. As a means of executing this research, we developed PAIRwise, a plagiarism detection system for instructors, students, and researchers. In this paper we describe its design and explore its capabilities with regards to efficiency and effectiveness. Our findings show that PAIRwise can handle large document collections efficiently and find instances of verbatim plagiarism as well as similar passages with modifications to verb tense. Researchers have also approached us with proposals to use PAIRwise for purposes other than plagiarism detection. The original system, PAIRwise 1.0, however, was slow, hard to maintain and not easily used by researchers from disciplines outside of computer science. Therefore, this release of PAIRwise, version 2.0, has many capabilities and applications beyond plagiarism detection.

Keywords: Plagiarism, Natural Language Processing, Web 2.0, Document Collections

Technology has had, and continues to have, a major impact on contemporary and future classrooms in many unforeseen ways. Over the past twenty years, presentation slides and other media have become commonplace in the classroom. As innovative new technologies, especially in the fields of telecommunication and computer hardware/software, become more a part of daily life, these new technologies inevitably make their way into the classroom. Both instructors and students alike are adopting these technologies and using them in conventional and unconventional ways, as well as in appropriate and inappropriate ways.

Currently, the model used to introduce these technologies is to simply bring them into the classroom, use them, and as long as it does not "hurt" the students or interfere with the learning process, continue to use them. However, often instructors do not examine and quantify the impact of these technologies. For example, how does the ubiquitous availability of the Internet, and therefore access to vast amounts of information, affect students' choices about copying and properly citing the information they find? In its simplest form, this problem is really a matter of improper citation, a problem which proper education solves. Beyond simple errors on the part of students is the decision by some to purposely copy ideas from others. The focus of this paper is to explore ways in which instructors, students, and even researchers, can use technology to explore the similarities within collections of documents and, as one example, identify plagiarism.

Because of the ubiquitous access to information, researchers and instructors are interested in not only identifying cases of plagiarism, but also investigating the impact of plagiarism on pedagogy in general (Johnson, et al., 2004). Researchers have attempted to quantify how often students are deliberately dishonest in their schoolwork (McCabe & Drinan, 1999; McCabe & Trevino, 2001), and how often they intentionally plagiarize ideas from others (McCabe, Trevino, & Butterfield, 2001). Researchers have also been exploring how students respond to any attempts to identify instances of plagiarism. Researchers seek to determine if these attempts reduce how often students plagiarize, or if they simply try better techniques to mask their dishonesty. Beyond plagiarism detection, researchers are also interested in investigating how students use the information given to them as part of their course work.

The problem then, is how researchers, instructors, and even students look for and find similarities in documents. Any solution must be efficient and effective. The efficiency requirement is important because, if automatic identification of similarities takes significant amounts of time, then the automatic solution is undesirable and will not be used. If the time taken to

explore the documents is too great, the user experience suffers. The effectiveness of any solution is also of paramount importance. Any solution that is efficient, but not effective (*i.e.*, allows instances of plagiarism to go undetected) is not a useful solution. The work we have done in the PAIRwise project attempts to balance both efficiency and effectiveness.

Originally, our work on the PAIRwise project[1] sought to address these problems. After using PAIRwise 1.0 for some time and assessing its performance, we noted two main deficiencies. Most notable was the fact that PAIRwise 1.0 was difficult to set up and maintain. Furthermore, as researchers tried to use it for other applications, they found it inflexible. Adapting PAIRwise 1.0 for other uses was, at best, difficult, especially non-computer scientists. In order to make the use of PAIRwise easier, we decided to redesign the entire system and create PAIRwise 2.0. The goal was to create a system that was easier to install, use, maintain and adapt.

To ameliorate the deficiencies of PAIRwise 1.0, we completely redesigned it. PAIRwise 1.0 created static reports of the documents it compared. If there was a need to add new documents to the comparison set, PAIRwise 1.0 had to re-compare the entire set of documents. To solve this problem, we first made PAIRwise 2.0 more dynamic. This new feature not only makes PAIRwise 2.0 easier to use, but also makes it useful for other, more general types of research that go beyond plagiarism detection. Second, we simplified the installation and maintenance of PAIRwise 2.0. By doing so, more researchers and instructors can install and maintain it. Moreover, we created a desktop version of PAIRwise 2.0 that allows for a personal installation on many platforms, thus eliminating the need for an Internet-based client-server model. Finally, we employed updated techniques for plagiarism detection that our experiments show is not only faster, but also more effective for plagiarism detection, and more generally, for exploring the similarities between documents.

The evaluation of PAIRwise 2.0 has produced good results. We concentrated on two main goals for our analysis. First, we examined the computing speed of the various techniques used in PAIRwise 2.0. Second, we quantified how effective PAIRwise 2.0 was at identifying plagiarism. The results of the computational speed show that our newly developed techniques are faster than previous methods. Quantifying the effectiveness of PAIRwise 2.0 yielded a surprising result. We found that if we filtered some words from the set used for ranking documents according to their similarity, the results were not only obtained faster, but the technique was more effective at finding plagiarism. We believe that by filtering words that only occur once or twice in a document, PAIRwise 2.0 filters out documents that only

coincidentally share words.

The remainder of this paper is organized as follows. Section 2 discusses current research similar to ours. Section 3 discusses the features and techniques used by PAIRwise 2.0. Section 4 discusses the overall architecture of PAIRwise 2.0 and contrasts it with PAIRwise 1.0. Section 5 discusses the evaluation of PAIRwise 2.0. And finally, Section 6 discusses the conclusions of our research.

RELATED WORK

The related research we examined can be grouped into one of four categories. First is the large body of research that examines the prevalence of cheating in general, and plagiarism specifically. Second is plagiarism detectors designed specifically for computer programming code. Third is several successful business enterprises that provide plagiarism detection services. Finally, there are a few instances of researchers who have created basic plagiarism detection services on a smaller scale than the business enterprises.

Research about the extent of cheating and plagiarism among students has been conducted for years and covers all levels of education. Perhaps best known among these researchers is the work of Donald McCabe and his collaborators. Over the past two decades he has written at least 30 academic papers on the subject. His papers address a broad range of topics on academic integrity. These topics range from the effectiveness of honor codes (McCabe, Trevino & Butterfield; 1999; McCabe, Trevino & Butterfield; 2002; McCabe & Pavela, 2005) to quantifying the amount of cheating by students at both secondary and post-secondary schools (McCabe & Drinan, 1999; McCabe & Trevino, 2001; McCabe, Trevino & Butterfield, 2001). This body of work is quite extensive and offers keen insights into the problem. However, this research does not directly address the issue of automatic plagiarism detection, specifically with regards to its techniques or effectiveness.

Additional research by Johnson, *et al.* (Johnson, et al., 2004) looks at the effects of technology both in plagiarizing work and in detecting plagiarism. Copyfind was the basis for this research (Bloomfield, 2006). But Copyfind uses a very simple plagiarism detection technique only capable of detecting verbatim copying. Also, this software performs a binary comparison of the files, which is flexible enough to support all file formats, but also makes it easier for students to fool. Students who are determined to commit plagiarism and avoid detection can rely on several simple techniques. First,

they can use features that insert binary data into the text. Since the system is looking at binary data for verbatim matches, the insertion of binary data between words can lead to false negatives (*i.e.*, undetected instances of cheating). Second, it may be possible to fool the system by simply using different document file formats.

Two bodies of work specialize in detecting plagiarism of computer code. The Measure of Software Similarity (MOSS) project is widely used by instructors to check the honesty of programming students (Scheimer, Wilkerson & Aiken, 2003). It uses hashes to measure the similarity of computer code and supports a variety of languages. Users submit code to a web server. MOSS analyzes it and reports the similarities back to the user. CodeMatch is a private enterprise that provides software to find snippets of similar code in two programs (S.A.F.E, 2009). CodeMatch also supports a large number of programming languages and is useful for finding intellectual property in other code. Both of these bodies of work are meant for a very specific form of plagiarism—computer code—and do not support the more general problem of detecting plagiarism in written documents.

Several for-profit business enterprises provide plagiarism detection services. Best known amongst these enterprises are Turnitin.com (iParadigms LLC, 2009) and MyDropBox.com (MyDropBox.com, 2009). Turnitin.com is a well-known service that not only checks for plagiarism within a class, but also from sources on the Internet. Over the years several legal issues have arisen, mostly around student privacy and copyright issues. A recent court decision, however, seemingly resolves the copyright issues, barring any further appeals. MyDropBox.com offers a similar service. One major problem for both services, beyond the legal issues, is lack of research into effectiveness with respect to both detection and deterrence.

We previously developed a plagiarism detection system. The result of this work is the PAIRwise 1.0 Project (Knight, Almeroth & Bimber, 2007). Similar to Copyfind, PAIRwise 1.0 looked for verbatim matches in student documents and highlighted those matches. Unlike Copyfind, however, we only processed the text of the document, not the metadata. The result of this difference is that while PAIRwise 1.0 supported fewer file formats, it was harder to fool. There were, however, several problems with it, including difficulties with maintenance and the fact that it was still limited to verbatim matching.

PAIRWISE 2.0 OVERVIEW

The main application of PAIRwise 2.0 is plagiarism detection. It allows instructors, researchers, and even students to explore how similar a document is to other documents of a collection. The intent of the program is *not* to make a determination of plagiarism, but rather assist those interested in finding other texts with similar passages. Instructors still bear the responsibility of determining that plagiarism has actually occurred. Therefore, the goal of this project is to allow interested parties to organize documents into related collections (*i.e.*, courses and assignments) and then compare those documents to find similar passages. By highlighting these similarities, instructors can determine if these passages are plagiarized, improperly cited, or because of some other reason.

There are several challenges to creating such a document comparison system. First is actually finding similarities between documents. Processing text is computationally complex and may require large amounts of processing time. Any solution, therefore, must do so as quickly as possible. Most techniques to reduce the comparison time do so at some cost of effectiveness. Therefore, the biggest challenge is to choose techniques that find the proper balance between efficiency and effectiveness. The second major challenge is presenting document similarities to users in such a way that facilitates the process of identifying instances of plagiarism. This challenge is really a usability challenge, where the goal is to make the instances of plagiarism obvious and recordable. These listed goals do not represent the entire pool of possible challenges, but were our focus.

Both versions of the PAIRwise project address these challenges, though in different ways. PAIRwise 1.0 was our first foray in this area of research and provided many lessons for future projects. One of the purposes of this paper is to highlight these lessons and outline how PAIRwise 2.0 addresses the lessons learned from our previous work. We next describe some of the new features in PAIRwise 2.0 that are the result of these lessons.

DOCUMENT COLLECTION

The original version of PAIRwise did not collect documents. Instead, other tools were necessary. Since we had previously collected student documents by other means during the development of PAIRwise 1.0, we did not include this feature. However, as feedback was collected from PAIRwise 1.0 users, we realized three things. First, uploading large collections of documents for comparison was not easy. Users found it hard to upload archives (such as “tar” or “zip” files) for comparison. Second, organizing or

maintaining larger collections of documents was not possible. Finally and perhaps most importantly, by not collecting and storing collections of documents, dynamic analysis of collections was not possible. If users wanted to add new documents, they had to wait for PAIRwise 1.0 to perform all of the comparisons again. For useful plagiarism detection and other applications, this lack of functionality was not acceptable. PAIRwise 2.0 now supports document collection. We now allow users to better organize their collected documents and enable dynamic processing.

DYNAMIC REPORTING

PAIRwise 1.0 required that all comparisons of documents be done at one time. This requirement existed because PAIRwise 1.0 did not store documents or any intermediate form of the processed results. New documents added to a collection required re-processing of all documents. The problem with statically created reports is that they are not time efficient. If a collection of documents required four hours to compare all the documents, then adding just one new document requires another four hours to process. For PAIRwise 2.0, we deemed it important to support more dynamic functionality.

FULL-TEXT INDEXED SEARCHING

Another weakness of PAIRwise 1.0 was that it only identified instances of verbatim plagiarism. This weakness made it easy for students to create the false impression that their documents were unique simply by changing the tense or through the addition of function words. In order to make PAIRwise 2.0 more powerful and harder to cheat, it uses full-text indexed searching. Since a database stores all text from submitted documents, we can utilize its capabilities for full-text indexed searching to find similarities in documents even if they are not verbatim copies. Such databases use normalized text to create their search index. We exploit this feature by using the text of one document as a query string to search all other documents of a collection. The result is a ranking of all other documents based on their similarity to the original document. We show in this paper that this technique is both effective and efficient.

BROADER RESEARCH USES

An additional lesson learned from our initial research originates not from a deficiency of PAIRwise 1.0, but from an unforeseen need in the research community. Many researchers, especially those in the areas of education and sociology, have a need to find similarities in documents. Several researchers approached us with ideas for using PAIRwise 1.0 as a tool for document comparison. In several cases, PAIRwise 1.0 was sufficient for the work they suggested. But in many cases it was either not possible, or more work was needed to make the tool more practical and powerful. PAIRwise 2.0 addresses this need. Its design and implementation make it easier for future additions or changes to be made to better suit the needs of researchers.

PAIRWISE ARCHITECTURES

The main reason for creating PAIRwise 2.0 was to address some of the shortcomings of the original version and to better support other applications. At a high level, the main difference between the architecture of PAIRwise 1.0 and 2.0 is static versus dynamic reporting. PAIRwise 1.0 created reports once. PAIRwise 2.0, however, does it's reporting dynamically. Reports are generated at the time they are requested.

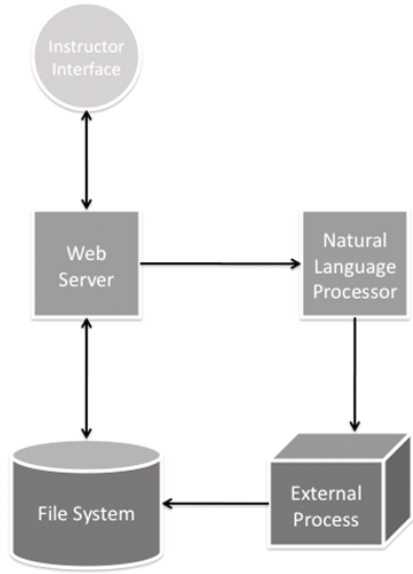

Figure 1. PAIRwise 1.0.

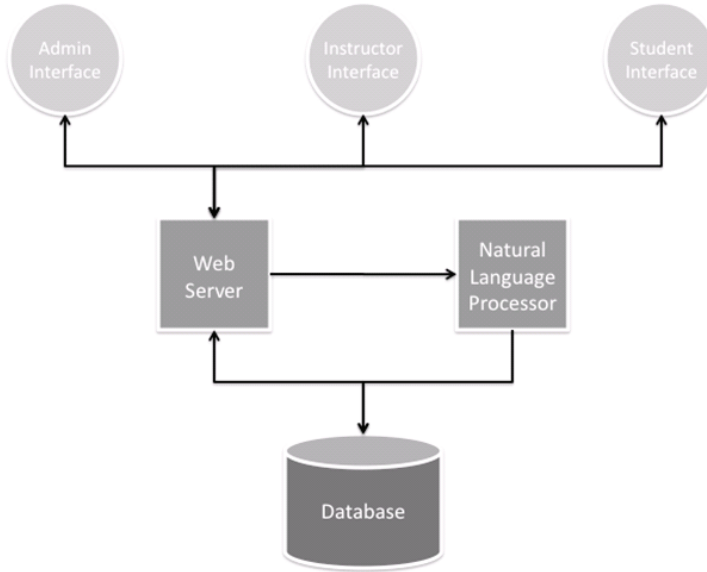

Figure 2. PAIRwise 2.0.

Figures 1 and 2 show the high-level system architectures of PAIRwise 1.0 and 2.0, respectively. The main differences in the figures are in the top and bottom. The middle layers are similar: a web server and a natural language processor. While the components are different with respect to implementation details, they accomplish similar tasks. At the top of the diagrams, both versions are composed of interfaces. PAIRwise 2.0, however adds two additional interfaces, one for administration and one for student submissions. At the bottom of the figures are the major differences between the two architectures. PAIRwise 2.0 uses a database, whereas PAIRwise 1.0 uses an external process and the machine's file system. The database is what allows PAIRwise 2.0 to be more dynamic. By saving the results of processing, PAIRwise 2.0 allows documents to be added or removed. PAIRwise 1.0 simply saved results to disk after the external process finished.

The following sections enumerate more details about each of these architectures and further expand on their differences.

PAIRWISE 2.0 ARCHITECTURE COMPONENTS

The architecture for PAIRwise 2.0 is divided into three distinct modules: the interfaces, the server, and the database. The interfaces allow three types of users to interact with PAIRwise 2.0. The server processes the submissions and generates reports for users. The database stores the results so that processing only occurs once.

PAIRwise 2.0 Interfaces

PAIRwise 2.0 adds two new interfaces. The original version only handled requests from instructors looking for instances of plagiarism. Collecting documents was not a part of its original functionality. PAIRwise 2.0, however, adds collection functionality to support dynamic document collections and support applications other than plagiarism detection. An administration interface is also new to PAIRwise 2.0.

The instructor interface allows for greater organization of documents and supports the concepts of courses and assignments. Instructors can create courses with multiple assignments. Furthermore, they can use PAIRwise 2.0 to collect assignments from students. Each assignment has a due date to limit students' ability to turn in assignments, or allow for late assignments by marking tardy submissions .

For students, the functionality is limited, but therefore, simple. They need only submit their assignments. The interface uses a course key to limit who can submit assignments. The instructor assigns a key to each course he or she creates. The key is shared with the students to allow them to submit their assignments.

PAIRwise 2.0 Server

The two main components that comprise the server module are the web server and the natural language processor. For the web server, PAIRwise 2.0 uses Apache with the Ruby on Rails framework (Gunderloy, 2009). In implementing the server, any framework will suffice, as long as it is as portable and lightweight. One attractive feature of Ruby on Rails is that it is independent of any particular web server. For server deployments, Apache is sufficient. Other servers are also well suited depending on the deployment. For installation on desktop computers as a stand-alone application, "lighthttpd" is a reasonable choice. If speed is more important, then Mongrel (Mongrel, 2009) may be a better choice. And since Ruby (Matsumoto,

2009) itself is platform neutral, the choice of Ruby on Rails fits with the goal of deployment on a desktop computer and on the web.

For natural language processing, PAIRwise 2.0 uses two important functions to normalize the text of documents before indexing them. First, it uses Ruby's built in string methods for global substitutions (mostly to remove punctuation when indexing) and string splitting (to detect word boundaries). Second, PAIRwise 2.0 uses the Porter Stemming method (Porter, 1980) to remove suffixes from words. The implementation of this method is, of course, written in Ruby. While Ruby is not currently the fastest run-time language, new virtual machines are increasing in speed and gaining parity with other language environments for web applications.

PAIRwise 2.0 Database

Figure 3 shows the design of the database schema for PAIRwise 2.0. The tables listed are not exhaustive, nor are the fields in each table, but the detail is sufficient as an example. There are a total of five tables in this diagram.

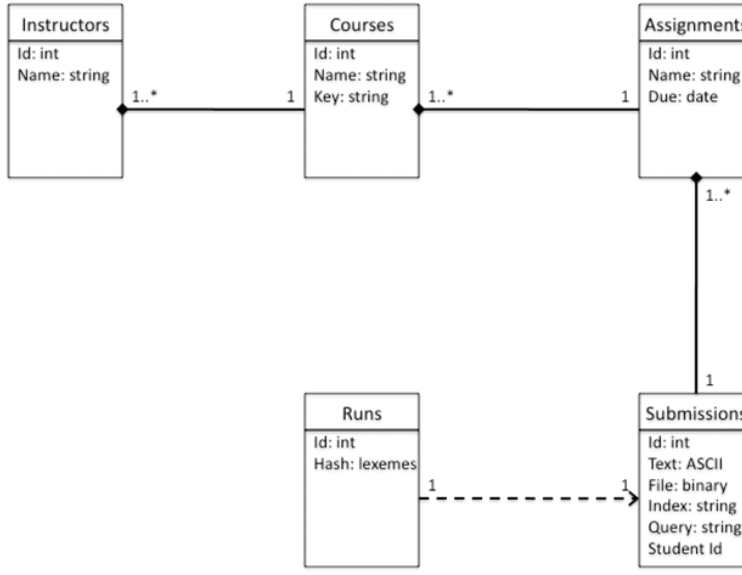

Figure 3. PAIRwise 2.0 Database Schema.

The first table is the Instructor Table. The diagram lists two fields for

the table: the Id and Name fields. Instructors create courses (and therefore own them) and assignments as well as view submissions while looking for instances of plagiarism. When instructors first access the PAIRwise 2.0 site, they submit their username and password and then arrive at a dashboard. The dashboard allows them to navigate to their courses and assignments and then access student submissions.

Next is the Course Table. Each course relates to exactly one instructor. Along with an instructor identifier, the table also has information such as the name of the course and a key. The key allows an instructor to restrict access to submissions for the course.

For any given course there can be any number of assignments. Students submit their documents with reference to a specific assignment. Besides the usual Name and Id fields, the Assignment Table also includes a due date. This date allows the instructor to set a deadline and/or identify when assignments are submitted late.

Perhaps the most important table is the Submissions Table. The role of this table is to store all the information associated with documents submitted by students. This information includes, but is not limited to, the name of the student, a student identifier and the document itself. The document, though, is stored in several formats to prevent repeated processing of the same information. One format is a plain ASCII text representation. This format facilitates displaying the document text for comparison to other documents. The text is also saved in the original text document format. Supported formats include Microsoft Word, Open Office OOXML, and HTML. Supporting other formats is also straightforward. Finally, the table stores the text in two additional formats to facilitate comparison with other documents. Because PAIRwise 2.0 uses the indexed text-searching feature of the database, there is a need for a normalized version of the text and a query string version. Since these formats do not change, they are saved as part of the database rather than recalculated for each comparison.

Finally, the Runs Table provides for efficient comparison of documents. This table stores a hash of each eight-consecutive-word run within a document along with its un-normalized text. In this way, PAIRwise 2.0 can quickly find common runs between two documents using simple hash functions. An MD5 hash is run on the normalized text to create these hash values. If two documents share many contiguous runs, they can be easily highlighted and brought to the attention of the instructor.

PAIRwise 1.0 Architecture Components

In contrast to PAIRwise 2.0, PAIRwise 1.0 was static and took several hours to finish processing larger sets of documents. Also, the barriers to installing PAIRwise 1.0 on a server system were quite high. For these reasons, installing it on a desktop computer was never a consideration. These problems and many others contributed to the instability of the system and made customization for other applications unnecessarily difficult. However, to better understand why PAIRwise 2.0 is such a leap in functionality, it is best to understand the design differences between the two architectures.

PAIRwise 1.0 Server

In terms of code, the original server was actually more complex than PAIRwise 2.0, but contained far fewer features. The main source of this complexity was implementation of many different processing techniques (*e.g.*, Internet searches). Additionally, since there was no underlying application framework, much of the design and code was dedicated to finding all the proper libraries and executables. Since PAIRwise 2.0 uses the Ruby on Rails framework, most of this functionality is already implemented, leading to a cleaner design with a greater feature set.

As the main concern of PAIRwise 1.0 was verbatim matching, the natural language processing was much simpler than for PAIRwise 2.0. It performed no normalization and only needed methods for splitting text using into words and into individual sentences. The text required no other processing.

PAIRwise 1.0 External Process

In PAIRwise 1.0, an external process actually performed the comparisons of documents. This external process was another source of unnecessary complexity in PAIRwise 1.0. The process had to continue to run separate from the web server (similar to how databases run as daemons in the latest version, but in PAIRwise 2.0, the database is a more logical place from which to run these daemons).

The role of the external process was to take the documents uploaded by the instructor and compare each document to one other, hence the name PAIRwise. Once completed, it created static web pages with the results

from the comparisons. The instructor later received an email informing them of the completion of the report and providing a link to the static reports. For large sets of documents, on the order of three or four hundred, processing took as long as several hours.

EVALUATION

The first goal of our evaluation was to quantify the difference in processing time for different minimum lexeme counts when ranking similarities between documents. When ranking the similarities, PAIRwise 2.0 performs a standard indexed text search. The search terms are all the unique, normalized words from a document. PAIRwise 2.0 can vary the number of words used in the search by using only those words that occur a minimum number of times. We refer to each word as a "lexeme", and the minimum count as the minimum lexeme count. For smaller minimum counts, the processing time is greater than for larger counts because lower minimum counts mean more search terms. For higher minimum word counts, the processing time is greatly decreased (one goal of this evaluation is to determine by how much), but may not identify instances of plagiarism. For example, for sufficiently large counts, the total number of lexemes used for searches could be as small as two or three. A simple way to escape detection is for two students to copy significant portions of text from each other, but agree on using different small sets of words more frequently in their respective essays. The key, therefore, to configuring PAIRwise 2.0 is to use a minimum lexeme count that balances processing time with the ability to find instances of plagiarism.

The second goal of this evaluation was to study how varying the minimum lexeme count affected PAIRwise 2.0's ability to find plagiarism or similarities in documents. Intuitively, there is a relationship between the minimum lexeme count and PAIRwise 2.0's ability to identify instances of plagiarism. Moreover, it is important to understand how the minimum lexeme count affects the ability to detect plagiarism even as the amount of copied text is varied. Ultimately, these experiments should be able to quantify what minimum lexeme count is effective for identifying possible instances of plagiarism for specific percentages of copying between documents. For example, UCSB requires that at least 10% of two documents share common text in order to pursue a case of plagiarism. Therefore, it would be important to know what minimum lexeme count still allows instructors using PAIRwise 2.0 to easily find instances of plagiarism when as little as 10% of the two documents are plagiarized.

METHODOLOGY

Our evaluation includes two different sets of tests. One of these tests measure processing time while the other measures the effectiveness of the system. The first test, therefore, is sensitive to the configuration of the system on which it was run. Therefore, a brief description of the configuration is necessary.

We performed all experiments on an Intel Core 2 Quad Q6600 running at 2.40 GHz with 2 GB of RAM and using the Fedora Core 8 Linux (Red Hat, 2009) distribution with kernel Version 2.6.23.1-42.fc8 (Linux Kernel Organization, 2009). The operating system ran in a typical X-windows configuration and daemons for SSH and other system functions.

PAIRwise 2.0 was written using the Ruby on Rails framework. We used Version 2.2.2 for these experiments. This version of Ruby on Rails uses Ruby Version 1.8.7. To normalize text, PAIRwise 2.0 used Version 0.5.4 of the Ruby Stemmer library. It also used Version 8.3 of the PostgreSQL Relational Database Management System (PostgreSQL Global Development Group, 2009). Since PAIRwise 2.0 requires support for fully indexed text searching, this version is the minimum version usable for PAIRwise 2.0. As a matter of convenience and in order to remove the performance effects of the web server, all test applications bypassed the web server and interacted directly with the Ruby on Rails environment. Doing so also avoided additional overhead added by the web browser.

EXPERIMENT PROCEDURES

The sets of experiments performed are independent of each other and therefore each type deserves some discussion. The first type of experiment measures performance. The second type of experiment, however, is quite different as it involves testing how well PAIRwise 2.0 performs when trying to find instances of plagiarism.

The goal of the first experiment was to quantify the relationship between the minimum lexeme count and the time taken to rank documents. Since PAIRwise 2.0 bases these ranks on indexed text searches, the expectation is that, as the number of documents increases, so too does the time required to calculate the ranks.

To quantify this relationship, we conducted the following experiment. We created document collections of various sizes. These sizes ranged from collections with two documents to collections with 500 documents. For

each collection, we computed rankings for each document with respect to all the others in the collection. For example, we ranked each document in a collection of ten documents against the other nine documents. Rather than observe overall times, we used an average time to hide the effects of document size and variations in system performance. In order to obtain the average time required, each experiment performed at least 200 calculations per document collection and recorded the results.

Finally, for the second set of experiments, we took a different approach. The goal of these experiments was to quantify the relationship between the minimum lexeme count and the ability of PAIRwise 2.0 to find instances of plagiarism. The main difference in these experiments was that we did not measure the speed of the algorithm, but rather the effectiveness of an algorithm optimized for performance. Therefore, the procedures for testing are necessarily different.

For this step in the evaluation, we performed five different experiments. During these experiments, we varied the minimum lexeme count (we used the minimums 0, 2, 3, 5, 10, and 25) and observed the rankings for a particular document. We also varied the size of the document collections. We added to each of these collections 7 more documents. Each of 6 documents contained progressively more verbatim text from the 7th document. These copies had 5%, 10%, 25%, 50%, 75%, and 100% of verbatim text from the control document. We then observed the ranking of each document in each of the collections. As no other document contained even 5% of verbatim text, each of these documents should, therefore, be at the top of the rankings for the control document. We recorded the rankings for each collection at all the varied minimum lexeme sizes. We also created an effectiveness score to quantify the ability to detect plagiarism.

Within any collection of documents that contains six documents with the amount of verbatim matching listed above, these documents should rank towards the top of all documents for the source document. Therefore, a perfectly effective ranking would put all six documents at the top of the rankings list. This situation is the basis for the scoring scheme used, with a perfect score being zero. Then, for each of the experimental documents, we assigned a weight and count of how far away it is from the perfect ranking position. For example, if the document with 100% verbatim copying is ranked tenth amongst a collection of documents, it is 9 positions out of place. To arrive at the final score, each copied document's distance from its optimal position is multiplied by its weight and the sum of these weights is the final score. We used the weights 6, 5, 4, 3, 2, and 1 for documents with 100%, 75%, 50%, 25%, 10%, and 5% verbatim matching, respectively. We

based these weights on the fact that the 100% matching document is less likely to be out of position than any of the documents with less copying. The same also holds for each other percentage of verbatim copying, as they are less likely to be out of place than the others with less copying.

TEST DATA

The data for all tests comes from two terms of student submissions for a Political Science class conducted in 2003 and 2004. This class was an undergraduate course and the subject of the student submissions dealt with the relationship between democracy and representation. Each document used the Microsoft Word file format. We further organized these student submissions into varying collection sizes as needed by the various experiments. The sizes varied from two documents up to 500. While this original set of data contained at least one example of plagiarism, we left out these examples to exactly control the amount of plagiarism for the third set of experiments.

RESULTS

Below are the results for the two experiments described above. Again, the first measures the speed of the algorithms used in PAIRwise 2.0, and the last quantifies the relationship between the minimum lexeme count and the ability to detect plagiarism.

Figure 4 shows the results when examining how the minimum lexeme count affects performance with regard to the size of the document collection. We conducted tests on collections of 2, 10, 40, 80, 200, and 500 documents. The x-axis graphs the minimum lexeme count. The test counts varied from 0 to 10. The y-axis graphs the average time taken for a series of rankings. We ran at least 200 samples for each test and recorded the average time. These times represent how long it will take for an instructor to wait, on average, to view the comparisons for any document submitted by a student. The times vary from tenths of a second to over 14 s.

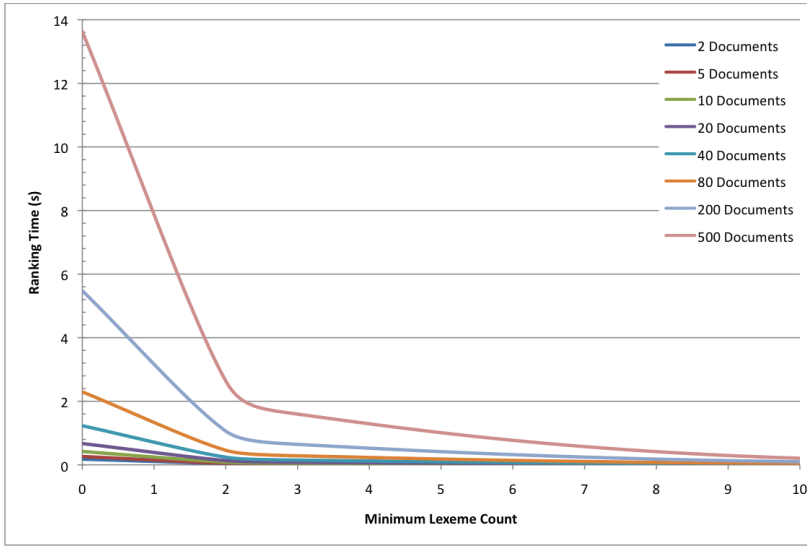

Figure 4. Ranking times for varying minimum lexeme counts.

Looking at the top line, for the collection of 500 documents, the average time taken for a search varies from 14 seconds to under 1 second. The time decreases as the minimum lexeme count increases because the larger the minimum count, the few the number of search terms used in any ranking computation. Each of the other document sizes exhibits similar behavior. Only lexeme counts of three or greater result in sub 2-second computation times. The implication is that three is perhaps the best value when considering usability and represents a good default configuration setting.

Varying the lexeme count, however, directly impacts the effectiveness of PAIRwise 2.0's ability to identify plagiarism. The following experiment attempts to quantify this affect. Figure 5 shows the results of quantifying the relationship between detection effectiveness and the minimum lexeme count. The x-axis shows the affects of increasing the size of the document collection. As the collection size increases, the effectiveness decreases (as shown by the increase of the effectiveness score graphed by the y-axis). The higher the effectiveness score, the lower the overall ability to find instances of plagiarism. In this graph, we varied the minimum lexeme count over the values of 0, 2, 3, and 5. The unexpected result is that the minimum lexeme count of 3 produces the lowest effectiveness score, and therefore, the best configuration for plagiarism detection.

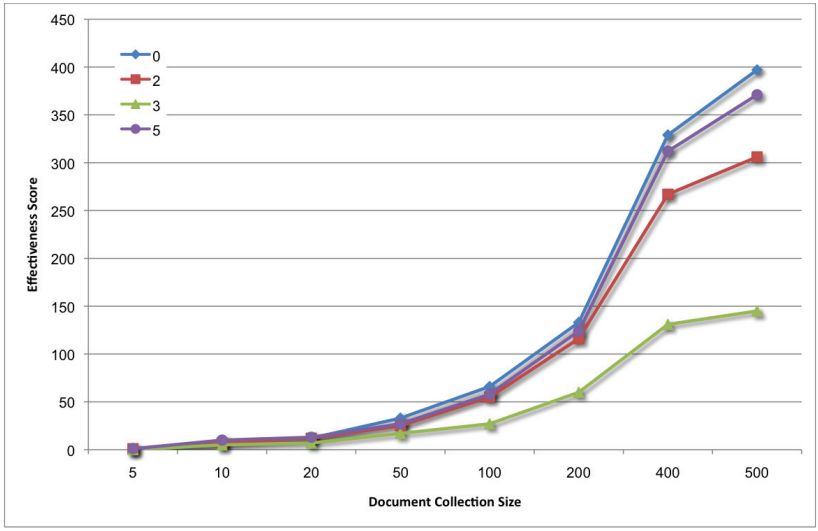

Figure 5. Effectiveness score vs. document collection size for varying minimum lexeme values.

From Figure 5, we observed that the worst configuration (minimum lexeme count=0) varied from a score of 1, nearly perfect, to almost 400 as the size of the collection increased. While these scores are high, the only documents that did not place in the top five were the documents with 10% and 5% verbatim copying. The best performing configuration had a minimum lexeme count of 3. In this configuration, the scores varied from a perfect score of 0 to a high of 150. These numbers indicate that the overall rankings of documents with significant plagiarism vary less than for any of the other tested configurations.

ANALYSIS

Several of the results from above have important implications that affect the scalability and effectiveness of PAIRwise 2.0. To meet all of our goals, PAIRwise 2.0 should scale well on small, medium, and large document collections as well as small and large documents.

The first set of experiments is also relevant to scalability and the ability to fine-tune PAIRwise 2.0 to meet users' needs. From these experiments, we see that reducing computation time is possible by controlling the number of search terms to use for ranking documents. Based on users' needs, they

can wait longer and get "better" results, or wait less and get less accurate results. For larger document collections, especially, this tunable parameter allows users to find the best setting for their needs. This fact is especially obvious when looking at setting the minimum lexeme count to 3. In the experiments we ran for a collection of 500 documents, we observed a reduction from 14 s to less than 2.5 s for the average time required to complete the ranking computations. And with this speed up, very little is lost with respect to PAIRwise 2.0's ability to find perceptible amounts of plagiarism.

The second set of experiments provided an unexpected result. We believed that removing any number of lexemes from the search term would result in less effective plagiarism detection. However, the experiments showed this hypothesis was not true. Actually, it seems, removing words that are not often repeated provides for more effective detection. This result, taken with the results from the second set of experiments, means that not only can PAIRwise 2.0 rank documents faster by increasing the minimum lexeme count, but can also be more effective. While the optimum minimum lexeme count may not always be 3, it does appear that some value greater than zero will work for large document collections and result in faster and more effective plagiarism detection.

CONCLUSIONS

In this paper we described the architecture of PAIRwise 2.0 and evaluated its effectiveness and efficiency. We found that it is both effective and efficient and learned that not only does it perform faster, but is more effective when less frequently used words are removed from searches for the purpose of ranking documents. We found that for documents of 500 to 2000 words, lexemes should occur at least 3 times within a document for the purposes of ranking documents. Furthermore, by limiting the number of words used in searches the speed up is on the order of 7 times for collections of 500 documents. We observed smaller speed-ups for smaller collections, but overall computing rankings took no longer than 2 seconds. Filtering out more than three words, however, led to less effective plagiarism detection. Finally, these findings are also relevant for other applications of PAIRwise 2.0. Since it remains effective and efficient, it remains a useful tool for exploring the similarities of documents in many disciplines including Education and Sociology as well as other domains not yet considered.

References

Bloomfield, L. (2006) Plagiarism Resource Site – Copyfind Software. http://plagiarism.phys.virginia.edu/software.html.

Gunderloy, M. (2009) Ruby on Rails 2.2 Release Notes. http://guides.rubyonrails.org/2_2_release_notes.html.

iParadigms (2009) Turnitin.com http://www.turnitin.com.

Johnson, D., Patton, R., Bimber, B. , Almeroth, K., & Michaels, G. (2004) "Technology and Plagiarism in the University: Brief Report of a Trial in Detecting Cheating", Association for the Advancement of Computing in Education (AACE) Journal, 12(3), 281-299.

Kneschke, J. (2009) lighthttpd fly light. http://www.lighttpd.net/.

Knight, A., Almeroth, K., & Bimber, B. (2008) Design, Implementation and Deployment of PAIRwise. Journal of Interactive Learning Research, 19(3), 489-508.

Kravets, D. (2009) Fair Use Bolstered by Student-Cheating Detection Service. http://blog.wired.com/27bstroke6/2009/04/fair-use-bolste.html.

Linux Kernel Organization (2009) Linux Kernel 2.6.23.1-42.fc8. http://www.kernel.org/pub/linux/kernel/v2.6/.

Matsumoto, Y. (2009) Ruby 1.8.7.http://www.ruby-lang.org/en/news/2009/04/18/ruby-1-8-7-p160-and-1-8-6-p368-released/.

McCabe, D., & Drinan, P. (1999, October 15). Toward a culture of academic integrity. Chronicle of Higher Education, p. B7.

McCabe, D, Trevino, K., & Butterfield, K. (1999) "Academic Integrity in Honor Code and Non-Honor Code Environments: A Qualitative Investigation," Journal of Higher Education 70(2), 211-234.

McCabe, D., & Trevino, L. (2001). Dishonesty in academic environments: The influence of peer reporting requirements. Journal of Higher Education, 72, 29-45.

McCabe, D., Trevino, L., & Butterfield, K. (2001). Cheating in academic institution: A decade of research. Ethics & Behavior, 11, 219-232.

McCabe, D, Trevino, K., & Butterfield, K. (2002) "Honor Codes and Other Contextual Influences on Academic Integrity," Research in Higher Education, 43(3), 357-378.

McCabe, D., & Pavela, G. (2005) "Honor Codes for a New Generation," Inside Higher Education http://www.insidehighered.com/views/new_honor_codes_for_a_new_generation.

McCullough, M. (2005) Using the Google search engine to detect word-for-word plagiarism in master's theses: A preliminary study. College Student Journal, 39(3), 435-441.

Mongrel (2009) Mongrel – Trac. http://mongrel.rubyforge.org/.

MyDropBox Suite (2009) Sciworth Inc. http://www.mydropbox.com/.

Oancea, A. (2008) Ruby-Stemmer. http://www.locknet.ro/projects/ann-ruby-stemmer.

Porter, M.F. (1980) An algorithm for suffix stripping. Program, 14(3), 130-137.
PostgreSQL Global Development Group (2009) PostgreSQL 8.3.7 Release Notes. http://www.postgresql.org/docs/8.3/static/release-8-3.html.
Red Hat (2009) Fedora Core 8 Documentation. http://docs.fedoraproject.org/desktop-user-guide/.
S.A.F.E. (2009) CodeMatch. http://www.safe-corp.biz/products_codematch.htm.
Schleimer, S., Wilkerson, D., & Aiken, A. (2003) Winnowing: local algorithms for document fingerprinting. International Conference on Management of Data. pp. 76-85.

Notes:

[1]The earlier work we call PAIRwise 1.0 and the later PAIRwise 2.0.