Finding Lists of People on the Web

Latanya Sweeney
July 2003
CMU-CS-03-168

Also appears as Institute for Software Research International Technical Report
CMU-ISRI-03-104

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract

Among the vast amounts of personal information published on the World Wide Web ("Web") and indexed by search engines are lists of names of people. Examples include employees at companies, students enrolled in universities, officers in the military, law enforcement personnel, members of social organizations, and lists of acquaintances. Knowing who works where, attends what, or affiliates with whom provides strategic knowledge to competitors, marketers, and government surveillance efforts. However, finding online rosters of people does not lend itself to keyword lookup on search engines because the keywords tend to be common expressions such as "employees" or "students." A typical search often retrieves hundreds of Web pages requiring many hours of human inspection to locate a page containing a list of names. As a result, people may falsely believe online rosters provide more privacy than they do. This paper presents RosterFinder, a set of simple algorithms for locating Web pages that consist predominately of a list of names. The specific names are not known beforehand. RosterFinder works by identifying rosters from candidate Web pages based on the ratio of distinct known names to distinct words appearing in the page. Accurate classification by RosterFinder depends on the set of names used. Results are reported on real Web pages using: (1) dictionary lookup employing a limited set of known names; and, (2) dictionary lookup on utilizing an extensive set of known names. Privacy implications are discussed using the example of FERPA and online student rosters.

This research was supported in part by the School of Computer Science and the Laboratory for International Data Privacy at Carnegie Mellon University.

Keywords: Information retrieval, data linkage, data mining, privacy, policy

1. Introduction

In July 2002, a representative from admissions at Yale University complained to the United States Federal Bureau of Investigation that someone in admissions at rival Princeton University had illegally gained access to Yale’s admissions system in order to snoop on students who had applied to both schools [1]. Such information may have provided strategic knowledge to Princeton about its admissions decisions in comparison to Yale’s, but the price of acquiring the information illegally led to public embarrassment and potential legal action against Princeton [2].

The Yale online system reportedly allowed newly admitted students to access information using their last names, birth dates and Social Security numbers as passwords. Some of these students had also applied to Princeton, thereby providing Princeton officials access to the same personal information. By electronically pretending to be one of the applicants, Princeton officials were able to compare Princeton’s admission decisions to Yale’s. The incident was termed a “security breach” because password authentication was compromised.

Without minimizing the ethical issues this incident underscores, the very information Princeton sought illegally online can be provided legally over the World Wide Web (“Web”), without any security breach. By simply reviewing publicly available Web pages maintained at a school's Web domain, students in attendance can be identified by name. Such information is often provided on the Web from the school, through student organizations, or by the individual students themselves.

The problem is not limited to students in academia. Large amounts of data about people appear on the Web, and demand has emerged for finding person-specific information. For example, corporations often seek the identities of employees within competing organizations. Government efforts have spawned interest in locating lists of people for counter-terrorism surveillance. The ability to identify the names of a class of people (“rosters”) translates into strategic information for many uses.

A directory or list of names is a roster. Rosters are important because a roster is a classification of people belonging to the same category. Identifying a roster provides knowledge about the people listed. Rosters can be a formal or official register of names. Rosters can provide a written account or record that serves as a memorial or authentic evidence of a fact, an event, a condition, an act, or an occurrence involving people. Rosters can provide evidence that something takes place, a happening or an incident, or give a written account of people involved in a set of circumstances, a situation or a final outcome.

The irony of the Princeton example is that publicly available rosters of Yale undergraduates could have been used. Protecting rosters is not a matter solved exclusively by authentication or authorization, which are traditional areas of computer security; nor is the problem solved by encryption. Protecting the identities of groups of people is often not a matter of “breaking in” but requires coordinating and controlling that which is given freely. When public information is found and linked together, it can provide strategic knowledge that can compromise the privacy of an individual or the confidentiality of an organization. Data privacy is the study of computational solutions for solving privacy and confidentiality problems in shared data.

This paper is the first to formally present the retrieval problem of locating documents containing lists of people. Simple algorithms for finding online rosters are provided, related data privacy problems are discussed, and ideas for privacy solutions are proposed.

2. Manual Search For Lists

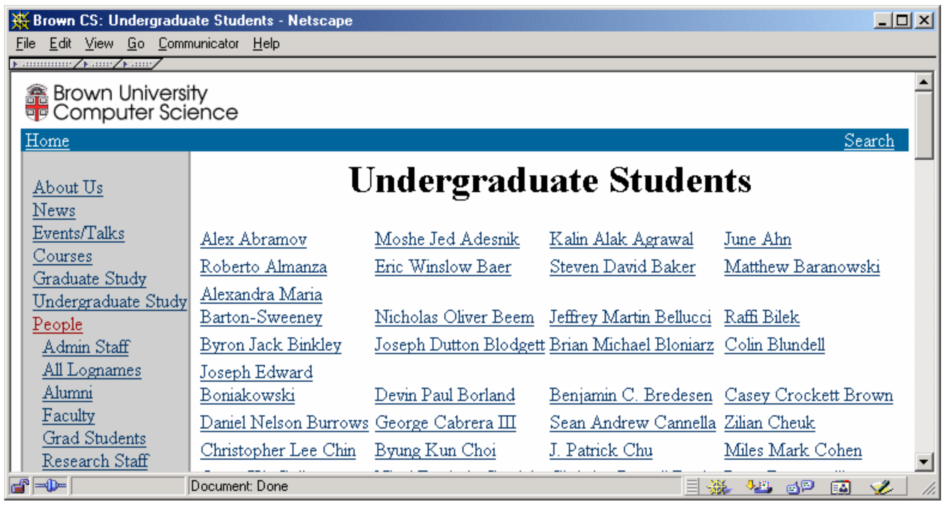
With so much personal information published on the Web and indexed by search engines, you might incorrectly believe you can easily find lists of people that share a given characteristic. Ironically, this is not the case. Finding rosters of people does not often lend itself to keyword lookup on search engines because the keywords are often widely used and as a result, many extraneous pages are retrieved. Another problem is the roster may not contain the searched keywords. Numerous search mechanisms do exist to locate a named person, but few options are available to locate a list of people whose names are not known beforehand. For example, to find a list of undergraduate computer science students at leading schools, searches on keywords {“people”, “undergraduate”} at each school website could be performed. Hundreds of Web pages containing these keywords are typically retrieved, requiring many hours of human inspection to locate a page having a list of names, if one is found all. Some schools have a policy prohibiting the publication of some kinds of rosters. Publicly available Web pages containing prohibited rosters may still exist at these school websites, but they are often difficult to locate by keyword searching. RosterFinder, presented herein, is a program that locates Web pages containing a list of names of a targeted group of people. The names are not originally known, but the list of names matches a specific criterion. RosterFinder works by automatically: (1) fetching candidate Web pages from search engines using website locations and keywords appearing in the page; and then, (2) identifying rosters from candidate Web pages based on the ratio of distinct known names to distinct words appearing in the page.

Explicit knowledge is provided to RosterFinder as a set of known names. Results are reported on real-world Web pages using: (1) dictionary look-up on a limited set of known names; and, (2) dictionary lookup on an extremely large set of known names.

3. Name Extraction Background

Methods for extracting proper names and their associated information from Web pages has been widely studied in the information extraction community [4, 5, 6]. Proper names of people reportedly represent about 10% of English newspaper articles [7] and systems have been constructed that accurately identify proper names in newspaper articles comparable to human performance (which is very good, 97% recall and precision) [3, 8]. Most approaches tackle name extraction by assuming names are basically unknown and not stored in dictionaries. These methods rely heavily on syntactic and semantic study of the appearance of names in text and related statistical language modeling. The result is typically a set of rules using trigger words and linguistic clues. Examples include taking advantage of “Mr.” and “Ms.” as well as “PhD” and “MD” and ways of recognizing “Abraham” and not “Abraham Lincoln” as the only proper name of a person in the sentence “a man named Abraham owned a Lincoln.”

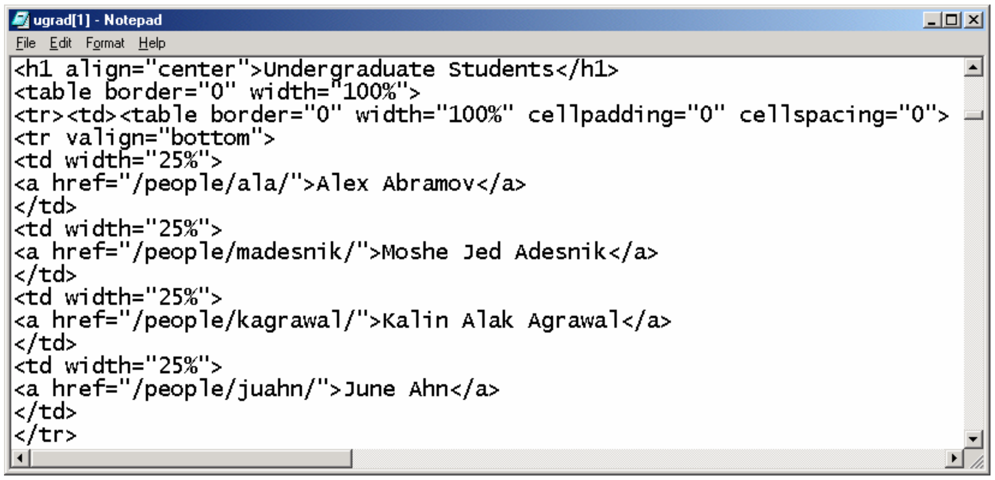
While prior work on name extraction has been heavily focused on newspaper articles, some attention has been given to non-journalistic texts. Using HTML tags, spelling cues (such as ‘@’ in an e-mail address), and layout cues, proper names have been reliably extracted from Web pages [3] and from email messages [9] to produce lists of names of people.



Undergraduate Students

Alex Abramov	Moshe Jed Adesnik	Kalin Alak Agrawal	June Ahn
Roberto Almanza	Eric Winslow Baer	Steven David Baker	Matthew Baranowski
Alexandra Maria Barton-Sweeney	Nicholas Oliver Beem	Jeffrey Martin Bellucci	Raffi Bilek
Byron Jack Binkley	Joseph Dutton Blodgett	Brian Michael Bloniarz	Colin Blundell
Joseph Edward Boniakowski	Devin Paul Borland	Benjamin C. Bredesen	Casey Crockett Brown
Daniel Nelson Burrows	George Cabrera III	Sean Andrew Cannella	Zilian Cheuk
Christopher Lee Chin	Byung Kun Choi	J. Patrick Chu	Miles Mark Cohen

Figure 1. Roster on a Web page listing undergraduate students at Brown University (formatted browser view). Source: http://www.cs.brown.edu/people/ugrad/



```
<h1 align="center">Undergraduate Students</h1>
<table border="0" width="100%">
<tr><td><table border="0" width="100%" cellpadding="0" cellspacing="0">
<tr valign="bottom">
<td width="25%">
<a href="/people/ala/">Alex Abramov</a>
</td>
<td width="25%">
<a href="/people/madesnik/">Moshe Jed Adesnik</a>
</td>
<td width="25%">
<a href="/people/kagrawal/">Kalin Alak Agrawal</a>
</td>
<td width="25%">
<a href="/people/juahn/">June Ahn</a>
</td>
</tr>
```

Figure 2. Roster on a Web page listing undergraduate students at Brown University (HTML source view). Source: http://www.cs.brown.edu/people/ugrad/

The task that is the subject of this paper, however, is not the extraction of names from text, but correctly classifying whether a fetched Web page consists primarily of a list of names.

Attempting to use name extraction methods to identify rosters does not produce reliable results. The biggest problem stems from the fact that lists of names are typically void of the linguistic context exploited by name extraction methods. In a roster, names are often just listed; See Figure 1. There is typically little or no linguistic text. Another approach is needed. The work presented in this paper takes advantage of explicit knowledge representation, i.e., the effective use of a stored dictionary of names. It is in sharp contrast to name extraction methods in appropriateness and in its simplicity.

4. Methods

This is the first work to formally introduce the problem of locating rosters. It is imperative therefore to precisely define terms from this vantage point, including common terms.

4.1 Definitions

The most primitive notion in this work is that of a "word." Some words are recognized as constituting the proper names of people. Identifying words in documents and determining which words are names are important concepts in this work. Definition 4.1 describes a "word." Sample words appear in Example 4.1.

Definition 4.1 (Word) A **word** is a string of two or more contiguous alphabetic characters.

Example 4.1 (Word) Shown in Figure 1 are 110 words. The initials "C." and "J." are not words. Appearing in Figure 2 are 52 words. These include HTML tags, such as "td" and "tr," as well as directory names, such as "madesnik" and "kagrawal."

The full name of a person includes a **given name** (culturally known as a first name in the US) and a **surname** (culturally known as a family or last name in the US). A "dictionary of names" is a distinct list of words that are known to appear in the proper names of people. A dictionary of names does not respect given or surname classifications. If a person bears the name "Martin Luther King," for example, then all three words may appear in a dictionary of names. Definition 4.2 describes a "dictionary of names" and a "name." Example 4.2 provides a sample.

Definition 4.2 (Dictionary of Names) A **dictionary of names**, **N**, is a finite set of words such that each word $w_i \in N$ is known to be part of the proper name of a person. A **name** is a word listed in a dictionary of names.

Example 4.2 (Dictionary of Names) {"Alex", "Abramov", "Moshe", "Jed", "Adesnik", "Kalin", "Alak", "Agrawal", "June", "Ahn"} is a dictionary of names containing names that appear in both Figure 1 and Figure 2.

Some ways in which proper names may be written is not fully supported. For example, the surname "O'Reilly" is recognized merely as "Reilly." The preceding "O" is not recognized as a word or part of a word. Similarly, "Malcom X" is recognized as "Malcom." Surnames that include a space character, such as "van Neumann," appear as two distinct names. Hyphenated surnames, such as "Hayes-Roth," also appear as multiple names.

Given a database of documents, this work is aimed at locating documents that contain rosters. The Web provides a very large, publicly available database of documents. Each document is fetched using a Web address. The combination of the Web address and its associated document is termed a "webpage." This is further described in Definition 4.3.

Definition 4.3 (Webpage) A **webpage** is a tuple (u, d) such that u is a string containing a full Uniform Resource Location (URL Web address) and d , which is referred to as the **document**, is the sequence of characters that comprise the content found at u.

Most webpage documents are written in the Hypertext Markup Language (HTML). Such documents have two appearances, depending on how they are viewed. The actual document has HTML tags embedded in its content. In an unformatted view of an HTML document, the tags are visible. But when an HTML document has its HTML tags removed, or is viewed through a browser, which interprets the formatting tags, then the textual content of the document appears unencumbered. Example 4.3 describes a sample webpage.

Example 4.3 (Webpage) Let d be the document located at URL u= "http://www.cs.brown.edu/people/ugrad/". Part of document d is displayed in the viewing areas of Figure 1 and Figure 2. (u,d) is a webpage. The browser view (or formatted display) of the first part of d appears in Figure 1. The source view (or unformatted display) of part of d appears in Figure 2. The formatting HTML codes are visible in Figure 2, but they are not visible in Figure 1.

In a webpage document, the order in which words appear is maintained and multiple occurrences of a word can occur. This is in contrast to a dictionary of names, in which names appear only once, and the order is not considered relevant.

This is the first work (and the first writing of this work) on locating documents that primarily consist of a roster. While this writing focuses heavily on documents retrieved over the Web, the work generalizes to other databases of documents with no loss of applicability. In such cases, references to the URL is omitted or replaced with references to an index of the documents.

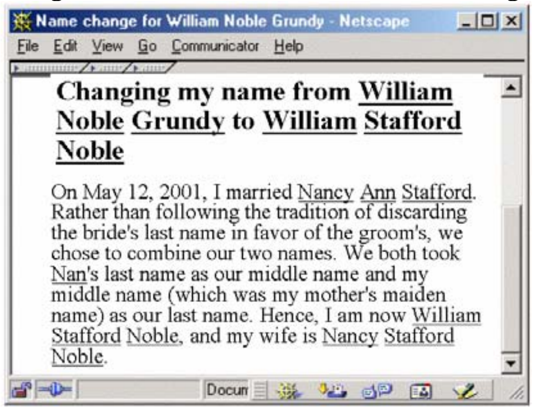
A "roster" provides a collection of names of related people arranged for reference or comparison, often in a vertical table or list format. Rosters appear as a list printed name by name. Definition 4.4 describes a roster; Example 4.4 provides a sample.

Definition 4.4 (Roster) Let **N** be a dictionary of names. A document d is a **roster** with respect to **N** if the content of d primarily consists of one or more itemized series of names, appearing name by name, where each name $n_i \in \mathbf{N}$. A webpage w=(u,d) is a roster if document d is a roster.

Example 4.4 (Roster) Figure 1 shows the first part of a roster of undergraduate computer science students at Brown University.

Several features in the definition of a roster require special comment. The word "primarily" and the phrase "name by name" in Definition 4.4 impose necessary conditions because not all documents in which names of people appear are rosters. For example, a document containing the title, authors, and abstract of a published paper is not a roster because the list of authors is not the determining element of the document's content. Figure 3 shows a document containing names, but the document is not a roster because the names are not catalogued.

While the names in a roster are itemized in a series, additional related text may appear between names. For example, a document containing a bibliographical listing of authors and titles is a roster. A document consisting of acknowledgements written as sentences containing lists of names is also a roster.


Name change for William Noble Grundy - Netscape
File Edit View Go Communicator Help
Docum

Changing my name from William Noble Grundy to William Stafford Noble

On May 12, 2001, I married Nancy Ann Stafford. Rather than following the tradition of discarding the bride's last name in favor of the groom's, we chose to combine our two names. We both took Nan's last name as our middle name and my middle name (which was my mother's maiden name) as our last name. Hence, I am now William Stafford Noble, and my wife is Nancy Stafford Noble.

Figure 3. Names (underlined) in a non-roster Web page. Source: http://www1.cs.columbia.edu/~noble/name-change.html.

If a document is a "roster," then the document's content may consist entirely of a single roster or may contain multiple rosters. The document's content may also include paragraphs of non-roster text. For example, a document containing a full academic paper with several sections, including bibliographic and acknowledgement sections as described above, is a roster.

The definition of a roster places no fixed constraints on the proximity of names to one another within the document; however, there is a relative measure of proximity imposed, because the list of names must be written on a name-by-name basis. No requirement is made that a surname and given name must both appear, and if they do both appear, the order is not specified.

The use of "roster" in this writing is limited to listing the proper names of people. However, these concepts generalize to "rosters" of other kinds of entities by simply specifying names appropriate to those entities in the dictionary of names used for analysis. Examples include names of places, journals, and companies.

A roster is defined with respect to a dictionary of names. When attempting to classify a document as a roster, the names comprising the roster are typically not known beforehand. Sometimes a superset of the names contained in the roster is used. Other times a dictionary containing an approximation of names is used, thereby introducing uncertainty.

The goal of this work is to find an automated means to correctly identify a document as a roster, even when the dictionary of names is approximated. This is the idea of a "roster detector" described in Definition 4.5. Example 4.5 provides a sample.

Definition 4.5 (Roster detector) Let **D** be a set of documents; $d_i \in \mathbf{D}$; **N** be a dictionary of names; $\Re$ be the set of real numbers between 0 and 1 inclusive; and, f:$\mathbf{D} \rightarrow \Re$ be a function such that $f(d_i)$ = r where $0 \leq$ r ≤ 1. The function f provides a certainty measure that d_i is a document containing one or more rosters with respect to **N**. A value of 0 means d_i has no roster according to f, whereas a value of 1 means f considers d_i to most certainly contain at least one roster. It is said that f is a **roster detector** and r is called the **rank** of the document. **A binary roster detector** returns either 0 or 1.

A roster detector, f_1, can be the basis for a binary roster detector, f_2, by introducing a threshold t, such that $0 \leq t \leq 1$ and $f_1(d_i) \geq t \Rightarrow f_2(d_i) = 1$ and $f_1(d_1) < t \Rightarrow f_2(d_i) = 0$.

Example 4.5 (Roster detector) Figure 4 shows *TwelveNames()*, which returns 1 if there are 12 or more distinct names in the dictionary of names that also appear in the document; otherwise, 0 is returned. *TwelveNames()* is a binary roster detector. The document in Figure 3 has 7 distinct names; *TwelveNames()* returns 0, correctly identifying its as a non-roster. The document in Figure 1 has 75 distinct names; *TwelveNames()* returns 1, correctly identifying it as a roster.

```
Algorithm: TwelveNames(d)
Input:    document d
Output:   1, if d is a roster and 0 otherwise.
Uses:     Dictionary of names N
Steps
    Let W be the set of distinct words in d                1
    if | N ∩ W| ≥ 12, return 1 else return 0               2
```

Figure 4. Roster detector *TwelveNames()* identifies a document as a roster if it contains at least 12 distinct names.

Not all roster detectors are good at detecting rosters. Some roster detectors may be better than others. To compare outcomes, terms used by the information retrieval community are useful. These are presented in the next subsection. Following that, a more robust roster detector, *RosterFinder()*, is presented. Results from experiments using *RosterFinder()* are then reported in terms of information retrieval measures. This paper ends with a discussion of privacy implications.

4.2 Information Retrieval of Rosters

The study of information retrieval systems is an active area of computer science research [15]. Information retrieval systems attempt to find relevant documents to respond to a stated request. A good system attempts to maximize the retrieval of relevant documents and minimize the retrieval of irrelevant documents. This paper describes information retrieval systems that detect relevant rosters; such systems are termed **information roster retrieval (IRR) systems** in this work. Operation of an IRR system may occur in one or two phases. In an initial phase, if present, the IRR system operates as a traditional information retrieval system. Based on a stated request, relevant roster and non-roster documents are selected from a very large collection of documents. The result is a database of pertinent documents that are candidate rosters. In a second phase, the IRR system uses a roster detector to determine which of the candidate documents are rosters. The final result is a list of relevant rosters, typically provided in a partial ordering based on the certainty the document is a roster. Alternatively, an IRR system can be described as an information retrieval system having a roster finding utility. A model of an IRR system is in Example 4.6.

Example 4.6 (Information roster retrieval system) Suppose an IRR system is to locate a roster of undergraduate computer science students enrolled at Brown University. The Web provides the collection of documents from which selections are made. A traditional search engine is used in the initial phase to provide a database of documents from webpages, where each document contains the words of one of the search strings in Figure 5 and has a URL that includes "cs.brown.edu". The top 10 webpages for each search string form a database of 124 distinct documents from which rosters are to be identified in the second phase. Results of the top 12 selections by a roster detector appear in Figure 6.

If there is no initial phase, the entire collection forms the database of documents from which rosters are identified. If the desired roster is not among the documents fetched in the first phase, or if it is not present in the collection at all, then the desired roster will not be found in the second phase. To avoid this kind of error being introduced in the first phase, performance measures in this work focus on the second phase. Unless otherwise noted, attention to an IRR system will be limited to the second phase, which involves the use of a roster detector to identify which documents in a database are rosters.

When searching for rosters among documents retrieved from a database, there are four possible outcomes. These are denoted in Figure 6. Evaluating information retrieval systems traditionally involves measures known as “precision” and “recall,” which are expressed using such outcomes [14]. These terms are adapted to IRR systems in Definitions 4.6 and 4.7, and samples are provided in Examples 4.7 and 4.8, respectively.

	Search string	Search string	
A	"academic advisor list"	"list undergraduate students"	O
B	"computer science undergraduate name"	"list undergraduates"	P
C	"computer science undergraduates"	"listing undergraduates"	Q
D	"degrees granted"	"people undergraduate"	R
E	"directory undergraduate"	"people undergraduate students"	S
F	"directory undergraduate students"	"personal home pages"	T
G	"index students name"	"student web page index"	U
H	"index students name Z"	"students home pages"	V
I	"index undergraduate student name"	"undergraduate student listing"	W
J	"index undergraduate students"	"undergraduate students"	X
K	"index undergraduates name"	"undergraduate students advisor"	Y
L	"last name"	"undergraduate students Z"	Z
M	"last name Z"	"undergraduates"	AA
N	"list computer science undergraduates"		

Figure 5. Set of search strings for identifying rosters of computer science undergraduates available on the Web.

Number of rosters retrieved (***rr***)
Number of non-rosters retrieved (***nr***)
Number of rosters missed (***rn***)
Number of non-rosters not retrieved (***nn***)

	Rosters	Non-rosters
Retrieved	**11 (*rr*)**	**1 (*nr*)**
Not retrieved	**33 (*rn*)**	**79 (*nn*)**

Figure 6. Retrieval outcomes of rosters and non-rosters in a database of 124 documents relating to undergraduate computer science students at Brown University.

Definition 4.6 (Recall) Let **D** be a set of documents and *f* be a roster detector. ***Recall*(*f*,D)** is the proportion of rosters retrieved by *f* from the set of total rosters in **D**. Based on the terms defined in Figure 5, ***Recall*(*f*,D)**= ***rr*/(*rr*+ *rn*)**.

Example 4.7 (Recall) Figure 6 shows retrieval results from roster detector, *f*, applied to a 124 document database, **D**, which has 44 rosters. When the roster detector was asked to retrieve its 12 most certain rosters, it returned 11 actual rosters. *Recall*(*f*,**D**)=11/(11+ 33)=25%. Retrieving 12 documents provided 25% of all possible rosters. The best possible result is 12/44 or 27%.

Definition 4.7 (Precision) Let **D** be a set of documents and *f* be a roster detector. ***Precision*(*f*,D)** is the proportion of rosters retrieved by *f* from the total number of documents retrieved. Based on the terms in Figure 5, ***Precision*(*f*,D)**= ***rr*/(*rr*+ *nr*)**.

Example 4.8 (Precision) Figure 6 shows retrieval results from a roster detector, *f*, which was asked to retrieve its 12 most certain rosters from a set of documents, **D**. Of the 12 retrieved, 11 were actual rosters. One was not. *Precision*(*f*,**D**)= 11/(11+ 1) = 92%. The best possible result is to retrieve 12 true rosters or 100%.

Consider a typical database having lots of rosters and non-rosters. In this, the general case, the following observations hold. A system having high recall and low precision requires a human to sift through the retrieved documents and reject the irrelevant information. A system having higher precision requires less sifting and therefore places less burden on the human. Results reported in Figure 6 achieve both a high level of recall (25% of 27% maximum) and high precision (92%). Another measure of burden on the human is the “false alarm rate,” described in Definition 4.8, with a sample provided in Example 4.9. In the general case, a high false alarm rate accompanies low precision.

Definition 4.8 (False alarm rate) Let **D** be a set of documents and *f* be a roster detector. The <u>**false alarm rate,**</u> ***FalseAlarm*(*f*,D)**, is the proportion of non-rosters retrieved by *f* from the total number of non-rosters in **D.** Based on the terms defined in Figure 5, ***FalseAlarm*(*f*,D)= *nr*/(*nr*+ *nn*)**.

Example 4.9 (False alarm rate) Figure 6 shows retrieval results from a roster detector, *f*, operating on a database having 80 non-rosters in a set of documents, **D**. When the roster detector was asked to retrieve its 12 most certain rosters, it returned one non-roster. *FalseAlarm*(*f*,**D**)= 1/(1+ 79) = 1%. The best possible result is to retrieve no non-rosters or 0%.

Recall, precision, and false alarm measures of a roster detector, *f*, can be influenced by the composition of the database, **D**. If **D** has very few rosters but lots of non-rosters, recall is a better measure of the performance of *f* than precision. As the number of documents retrieved by *f* increases in this setting, the false alarms increase. On the other hand, if **D** has lots of rosters but very few non-rosters, precision is a better measure of the performance of *f* than recall. As the number of documents retrieved by *f* increases in this setting, recall and precision increase.

Performance measures are useful in comparing roster detectors. The best roster detector provides the highest recall and precision and the lowest false alarm rate for a database. More generally, the best IRR system provides the highest recall and precision and the lowest false alarms for roster requests. Determining the best roster detector is the "roster retrieval problem," described in Definition 4.9 with a sample in Example 4.10.

Definition 4.9 (Roster retrieval problem) Let **D** be a set of documents; $d_i \in \mathbf{D}$; **N** be a dictionary of names; and, $\mathbf{F} = \{f_i\}$ be a set of binary roster detectors where each f_i:$\mathbf{D} \rightarrow \{0,1\}$. If for all $f_i \in \mathbf{F}$, there does not exist $f_j \in \mathbf{F}$ such that $i \neq j$, $recall(f_j,\mathbf{D}) \geq recall(f_i,\mathbf{D})$, $precision(f_j,\mathbf{D}) \geq precision(f_i,\mathbf{D})$, $falseAlarm(f_j,\mathbf{D}) \leq falseAlarm(f_i,\mathbf{D})$, and at least one of the following conditions is true: $recall(f_j,\mathbf{D}) \neq recall(f_i,\mathbf{D})$, $precision(f_j,\mathbf{D}) \neq precision(f_i,\mathbf{D})$, $falseAlarm(f_j,\mathbf{D}) \neq falseAlarm(f_i,\mathbf{D})$, then f_i is the best roster retriever with respect to **D** and **F**. The goal is to determine the most appropriate function *f*.

Example 4.10 (Roster retrieval problem) Let *f* be the roster detector providing the results on the set of documents **D** reported in Figure 5. *TwelveNames*, defined in Figure 4, provides the following results when retrieving 12 rosters from **D**: ***rr***=2, ***nr***=10, ***rn***=42, and ***nn***=70. Figure 7 provides comparative results of *f* and *TwelveNames*, demonstrating that of {*f*, *TwelveNames*}, *f* is the best roster retriever.

Care has been taken in this section to precisely define terms so others can build on this work, but this precision does not forecast complication. In the next section, a simple roster detector, *RosterFinder(),* is constructed to operate with modest computing resources. The real-world results provided in the section after next demonstrates *RosterFinder()* to be an effective solution to the roster retrieval problem. Its simplicity underscores privacy concerns, which are discussed at the end of this writing.

4.3 RosterFinder

"RosterFinder" identifies rosters from documents in a database based on the ratio of distinct known names to distinct words appearing in the document. A dictionary of names is used to explicitly recognize names. Figure 8 shows the roster detector algorithm known as *RosterFinder()*.

Given a document *d* and a dictionary of names **N**, *RosterFinder()* reports the percentage of distinct names to words appearing in *d*. This percentage is the rank of *d*.

	f	***TwelveNames***	**Best Possible**
Recall	25%	5%	27%
Precision	92%	17%	100%
FalseAlarm	1%	13%	0%

Figure 7. Comparison of the performance of two roster detectors, *f*, whose results are shown in Figure 6, and *TwelveNames*, which is defined in Figure 4, on the database of documents described in Example 4.6.

```
Algorithm: RosterFinder(d)
Input:    document d
Output:   rank of d, which is a value r such that $0 \le r \le 1$
Uses:     dictionary of names N
Steps
    let W = {$w_i$ | $w_i$ is a word in d}        1
    r = |N ∩W| / |W|                             2
    return r                                     3
```

Figure 8. *RosterFinder()* algorithm for determining the certainty a document is a roster.

Using a numeric constraint $0 \le$ *threshold* ≤ 1, *RosterFinder()* can be a binary roster detector by replacing line 3 in Figure 8 with the line shown in Figure 9. When the rank of the *d* is ≥ *threshold*, *d* is a roster (1 is returned) otherwise it is not (0 is returned).

```
    if r ≥ threshold return 1 else return 0      3
```

Figure 9. *RosterFinder()* algorithm modified to be a binary roster detector requires replacing line 3 in Figure 8.

RosterFinder() can be used within an information retrieval system to provide rosters relevant to a stated request. An example is provided in Figure 10.

```
Algorithm: RosterFinder IRR System
Input:    set of search strings Keys, set of websites Sites, and
        number of documents to retrieve per string k and overall m
Output:   a partially ordered matrix of m rows and 2 columns.
          Each row has a document d and its rank; [1][$r_1$] is
          most certain and [m][$r_m$] is least certain, i.e., $r_1 \ge r_m$
Uses:     dictionary of names N, collection of webpages Web
Steps
    let W = ∅                                            1
    for each s∈Keys and u∈Sites do:                      2
        W = SearchEngine(s + u, k) ∪ W                   3
    D = { $d_i$ | ($u_i$,$d_i$)∈ W }                     4
    for each $d_i$∈D do                                  5
        results[i][1] = $d_i$                            6
        results[i][2] = RosterFinder($d_i$)              7
    sort rows of results based on values in results[I][2]   8
    return results[1][] … [m][]                          9
```

Figure 10. A RosterFinder Information Roster Retrieval System using a search engine to fetch candidate documents and *RosterFinder()* to identify rosters from among them.

The IRR system in Figure 10 has two phases. The first phase appears in steps 1 through 3. A traditional search engine identifies webpages whose documents contain the words of one or more of the search strings and whose URLs contain one or more of the sites. The *k* "best" matching webpages are returned. Step 4 consolidates these documents into the database of candidate documents, **D**, which is used in the second phase. The rank of each document is computed in steps 5, 6 and 7 using *RosterFinder()*. The results are stored in a 2-dimensional matrix. Each row in the matrix contains a document in the first column and its rank in the second column. In step 8 the rows of the matrix are sorted in decreasing order based on rank. A document having the highest rank appears first and a document having the lowest rank appears last. The top *m* rows of the matrix are returned, thereby providing the *m* most relevant rosters.

Example 4.6 describes sample execution of the RosterFinder IRR system in Figure 10. *RosterFinder()* results are shown in Figure 6, and performance measures are reported in Figure 7. The roster detector denoted as *f* in those figures is *RosterFinder()*.

4.3.1 Correctness of RosterFinder

Most algorithms in computer science have a *right* answer, and are considered incorrect if they do not have a *right* answer. A heuristic guesses something close to the right answer. Heuristics are measured on "how close" they come to the right answer. *RosterFinder()* uses the simple heuristic that a roster has a higher ratio of distinct names to distinct words than does non-rosters.
Correctness is based on how close to the right answer proposed ranks come. Comparing recall, precision, and false alarm results from *RosterFinder()* to their best possible values describes its performance of *RosterFinder()*. The results reported for *RosterFinder()*, identified as *f*, in Example 4.6 are very good. Results from real-world experiments in the next section also show *RosterFinder()* to be very effective at identifying rosters.

The *RosterFinder()* heuristic has biased rankings. High-ranking rosters are those consisting almost entirely of a list of known names. Figure 1 provides an example of the kind of roster preferred by *RosterFinder()*. Care must be taken that the dictionary of names is likely to include names appearing in rosters. For example, if the dictionary of names consists only of the 7 names underlined in Figure 3, then the roster in Figure 1 will incorrectly receive a poor rank, having 0 names found.
Correct classifications of documents as rosters by *RosterFinder()* relies on: (1) a significant number of names in the roster appearing in the dictionary; and, (2) a much larger number of names appearing in the document than non-names.

Rosters incorrectly receiving poor rank by *RosterFinder()* typically have: (1) few numbers of roster names appearing in the dictionary of names; or, (2) too many non-name words appearing in the document. For example, a bibliography of technical writings is a roster that *RosterFinder()* tends to rank poorly when using a dictionary of proper names of people.

RosterFinder() can incorrectly award a high rank to a non-roster if: (1) the document text contains references to lots of different names; or, (2) the document text contains lots of words that are also names.
Non-rosters correctly receiving low rank from *RosterFinder()* may have lots of occurrences of a recurring name. For example, an article about the undergraduate program at Brown University may have many occurrences of "Brown," which is a common adjective and also a proper family name. The requirement in *RosterFinder()* to only consider distinct names and words helps it correctly classify these kinds of non-rosters.

4.3.2 Complexity of RosterFinder

Let **D** be a set of documents, $d \in \mathbf{D}$ be a document in **D**, and **N** be a dictionary of names. The complexity of *RosterFinder()* is linear in the number of words in a document and log in the number of names in the dictionary. This is computed as follows. The construction of *W* in line 1 of Figure 8 requires visiting each word in *d*, which is performed in time linear in the number of words in *d*. Determining which words in *W* are also in **N** (line 2 of Figure 8) requires looking up each word in *d*. This too executes in time linear in the number of words in *d*. **N** can be stored as a binary tree, making the search for a word in **N** execute in time log in the number of names stored in **N**. Overall, the complexity of *RosterFinder()* is $O(\ |d| \bullet \log|\mathbf{N}|\ )$. Even for dictionaries having a large number of names, *RosterFinder()* operates in real-time.

5. Experiments

The performance of *RosterFinder()* is examined in this section through a series of experiments. The overarching goal is to locate online rosters of computer science undergraduates at selected schools using the system described in Figure 10. The first phase of the system uses the Google search engine to construct databases of candidate documents and the second phase uses *RosterFinder()* to identify rosters from among the candidates.

Aspects of the system are tested. The first experiment lists the volume of webpages that relate to relevant search strings. Using a set of search strings, databases of relevant documents are constructed, one for each school. The second experiment reports on *RosterFinder()*'s classifications of rosters and non-rosters in each school's database. Recall, precision and false alarms are reported. The third experiment compares *RosterFinders()*'s performance using different sized dictionaries. The fourth experiment reports *RosterFinders()*'s ability to highly rank a targeted roster in each database. The target is the one (or no) seemingly complete roster of undergraduate computer science students appearing on the computer science department's official website at a school. This section ends with observations about other kinds of information gained from rosters found.

5.1 Materials

RosterFinder() requires a stored dictionary of names against which words are compared. Two dictionaries were used in these experiments. One dictionary, termed the **small dictionary**, was inherited from earlier work in which names were identified in the unrestricted text of clinical notes and physician letters in order to de-identify the text [13]. This dictionary contains 23,729 words, originating from given and surnames of patients and doctors. The other dictionary, termed the "big dictionary," resulted from extracting names of people from a very large death database and is further described below.

In the United States, a Social Security number is required for employment and necessary for many other purposes. The United States Social Security Administration maintains a death database, which contains a listing of everyone who had a Social Security number and who is deceased, based on official reporting to the Social Security Administration [10]. Social Security numbers were first issued in November 1936. As of March 2003, more than 70 million people were listed in the death database. A purchased copy of the death database was used as follows.

A full name was listed for each of the 70 million people in the death database. Given names and surnames were extracted to produce a list of distinct words, not respecting the original appearance of the word as a given or surname. The result was a list of 1,587,078 words, which formed the **big dictionary** of names used in experiments as noted.

A modest programming environment was used in these experiments to demonstrate the ease at which *RosterFinder()* can be deployed, thereby further underscoring privacy concerns. Components included: access to the World Wide Web, the Google Web API [11], which is publicly available, and Microsoft Access, a commonly available relational database program. All programs were written in Java. These programs used the JDBC-ODBC Bridge to store and retrieve information in Microsoft Access and used the Google API to perform automated web searches on the Google search engine. Processing was performed on an IBM Thinkpad A22p laptop with Pentium III processor.

5.2 Test Design

The subjects were 30 schools selected as follows. Each year *U.S. News* surveys computer science graduate programs and reports the top 70 ranked schools based on measures of quality and achievement [12]. Schools ranked in the top 21 were selected as subjects for these experiments. Another group of 6 schools were randomly chosen from the schools ranking between 26 and 70. Three additional schools were selected that were not in the top 70. Figure 11 contains an alphabetical list of the 30 subject schools.

5.3 Experiment: Search String Volume

The overall goal in these experiments is to locate rosters of computer science undergraduates at select schools. The 27 search strings listed in Figure 5 were used to search computer science department websites at the schools listed in Figure 11 in order to locate webpages that may contain rosters of computer science undergraduate students. This is consistent with the first phase of the RosterFinder IRR system. This experiment examines the search strings used and their role in constructing the database of candidate documents for a school.

Rebekah Siegel, who was a high school student at the time, provided the search strings. She was asked to locate rosters manually for MIT, Stanford and Princeton and write down each search string she tried and document the results. Figure 5 contains the list of search strings she used when attempting to locate seemingly complete, official rosters of students. Future references to search strings and their labels relate to those itemized in Figure 5, labeled, A through AA. Search string R is "people undergraduate," for example.

	School	Prec	M	R	Pos	Tot
1	Brigham Young University	67%	□	■	4	87
2	Brown University	92%	■	■	1	124
3	California Institute of Technology	67%	▨	■	1	112
4	Carnegie Mellon University	100%	■	■	10	159
5	Columbia University	100%	■	■	1	113
6	Cornell University	100%	□	■	1	161
7	Duke University	100%	■	■	3	130
8	Georgia Institute of Technology	100%	▨	▨	4	148
9	Massachusetts Institute of Technology	100%	▨	■	4	331
10	North Carolina State University	83%	■	■	2	47
11	Northeastern University	92%	■	■	2	133
12	Penn State University	75%	■	■	1	84
13	Princeton University	83%	■	■	1	119
14	Rice University	67%	□	□	---	101
15	Spelman College	58%	■	■	82	136
16	Stanford University	100%	■	■	1	29
17	University of Alabama	50%	■	■	6	59
18	University of California–Berkeley	92%	□	■	1	121
19	University of California–Los Angeles	92%	□	□	---	125
20	University of Illinois–Urbana-Champaign	83%	□	■	1	152
21	University of Maryland–College Park	67%	□	□	---	108
22	University of Michigan–Ann Arbor	92%	▨	▨	45	114
23	University of North Carolina–Chapel Hill	83%	■	■	1	119
24	University of Pennsylvania	75%	□	▨	1	125
25	University of Pittsburgh	75%	■	■	1	68
26	University of Texas–Austin	100%	▨	▨	1	152
27	University of Virginia	58%	■	■	3	116
28	University of Washington	50%	▨	▨	3	110
29	University of Wisconsin–Madison	100%	■	■	1	137
30	Virginia Tech	92%	■	■	3	127

Legend

■	complete
▨	partial
□	none

Figure 11. Search for undergraduate computer science rosters at 30 schools, reporting precision of rosters found in top 12 RosterFinder results, and whether a roster of undergraduates was found manually (M) or by RosterFinder(R), and if found by RosterFinder, its ranked position (Pos) in a database having a total number (Tot) of documents.

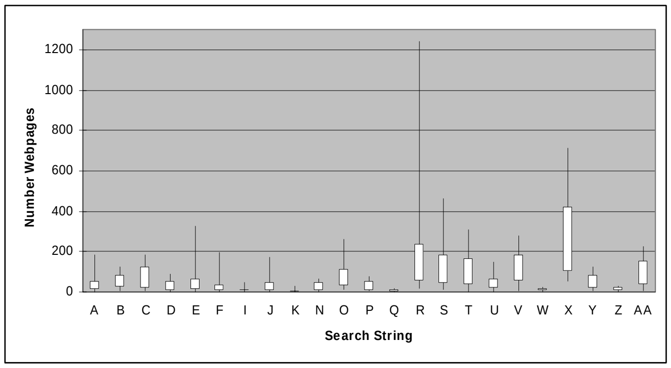


Figure 12. Statistics reporting results from web searches using search strings in Figure 5 on websites for schools in Figure 11. Line endpoints report the maximum and minimum number of webpages found at a school website for the search string, and the box outlines the middle two quartiles.

The search is limited to computer science website(s) at each school by restricting the URL of subject webpages to include one or more substring(s) custom to the school's computer science department website. Examples of computer science department website substrings used are: "cs.brown.edu," "cc.gatech.edu," "csc.ncsu.edu," "cis.upenn.edu," "ccs.neu.edu," and "cse.psu.edu," for Brown University, Georgia Institute of Technology, North Carolina State University, University of Pennsylvania, Northeastern University, and Pennsylvania State University, respectively. All future references to a school's website relates specifically to its computer science website(s) unless otherwise noted. Restricting the search to these websites provided two advantages: (1) it limited the number of webpages provided in response to a search; and, (2) it focused the search to the authoritative source of the rosters being sought.

The only exception was Spelman, which was the only college (no graduate students) included in the subject schools. The school wide website was used. The search strings were modified such that "undergraduate" was removed and as appropriate "computer science" was appended.

Figure 12 contains statistics that describe the number of webpages found by the Google search engine for the search strings. Results from 17 of the subject schools, randomly selected, are reported. The top point of each line is the maximum number of webpages retrieved for a school and the bottom point is the minimum number. R had its maximum, 1240, at Brown University's website, and its minimum, 17, at Brigham Young University' site. Searches K, Q, W, and Z provided very small numbers (maximum less than 30) at all schools. Half of all reported values appear within the box, so the size of the box depicts the spread of the distribution.

G and L are not pictured because they had extremely large maximums, 3380 and 16,600, respectively. Rosters were found in all search strings, but seemingly complete rosters of undergraduate computer science students were not found in E, F, I, J, K, P, Q, and U. Search strings H and M were not included in this experiment. Overall, the average number of webpages found for a search string at a school was 198, the minimum was 0, the maximum was 16,600, and the standard deviation was 965.

Documents from the top 10 matches from each search string were added to the database of the school from which the webpages were found, ignoring duplicates. These formed the databases of candidate documents from which rosters were subsequently detected. The total number of documents in each database for a school is reported in Figure 11 (Tot column). Overall, the average number of documents in a school's database was 122, the minimum was 29, the maximum was 331, and the standard deviation was 51.

5.4 Experiment: Roster Classification

RosterFinder() assigns each document in a school's database a rank. Documents are then sorted in decreasing order of rank. As the topmost documents are revealed in rank order, more and more non-rosters are expected to appear. *RosterFinder()* has done a perfect job if all rosters appear before non-rosters. Figure 13 shows recall and precision measures for *RosterFinder()* results from the first 3 subject schools. When the recall rate is low, the precision is near perfect (100%). As more documents are revealed, the recall rate decreases and the precision does also as increasingly more non-rosters appear.

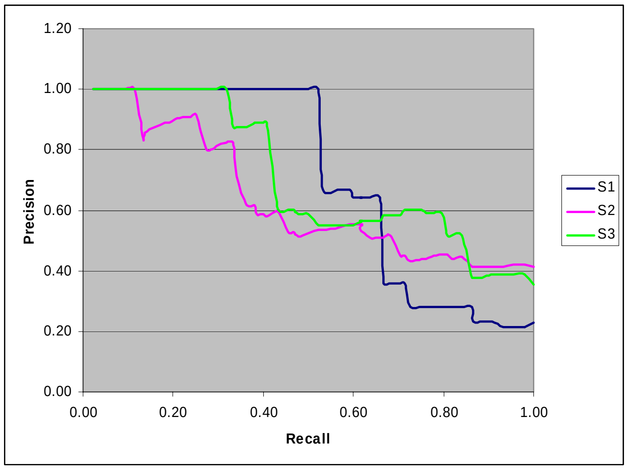


Figure 13. Recall and precision measures for 3 schools: Brigham Young (S1), Brown University (S2), and California Institute of Technology (S3).

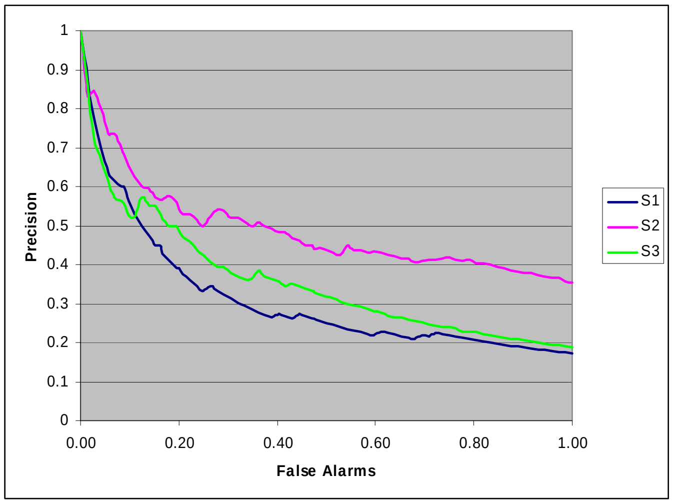


Figure 14. False alarm rate and precision measures for 3 schools: Brigham Young (S1), Brown University (S2), and California Institute of Technology (S3).

Ideal results resemble a step-down function where precision is 100% until all rosters in the database are retrieved and then, precision moves toward 0. Once all the documents have been retrieved, the final precision value is the proportion of rosters in the database. The steepness of the decay and the percentage of documents recalled before the decay begins is based on the number of rosters appearing in the database, not on the performance of *RosterFinder()*. The correctness of *RosterFinder()* is exemplified by the shape of the curve -- in particular, the high precision for the first documents retrieved.

Precision measures for all 30 subject schools are reported in Figure 11 when the 12 highest ranked documents are examined. Nine of the schools provided 12 rosters, thereby reporting 100% precision. The sizes of the databases cannot be ignored, however. Consider school S1, Brigham Young University. Its precision for the top 12 documents is only 67% in Figure 11, yet in Figure 13 its precision with low recall rates is perfect (100%). There are 89 documents in S1's database. When 12 of the 89 documents are provided, the recall rate is 53%. The curve in Figure 11 for S1 shows at 53% recall, precision drops significantly.

The results shown in Figure 13 are characteristic of the subject schools even though only results for 3 schools are shown. The results demonstrate *RosterFinder()* to be quite reasonable at detecting rosters. A human using the RosterFinder IRR system would have few non-rosters appearing in the top ranked results. Figure 14 shows how precision decays as the number of false alarms increase.

5.5 Experiment: Dictionary Size

All experimental results reported so far were based on *RosterFinder()* using the small dictionary. The previous experiment was repeated with each school's database using the big dictionary instead. Surprisingly, there was no substantive change in recall, precision or false alarms for any schools. The rank values assigned to a document were much higher overall because more words were being recognized as names. The ordering of documents changed, but not significantly.

5.6 Experiment: Targeted Roster Hunting

The overall goal of these experiments was not just to demonstrate the effectiveness of the RosterFinder IRR system at locating rosters, but to also demonstrate its ability to find a specific, targeted roster. In this case, "targeted students" are registered computer science undergraduates, by year or combination of years, and my include alumnae. A "targeted rosters" is a seemingly complete list of targeted students at each school. There may be one or no such targeted roster in a school's database. Graduate students are not included because there is much more information readily available on graduate students.

Marshall Warfield (a Masters student in English) was hired to navigate through each school's website by hand, following links related to undergraduate students and visually inspecting the contents of the webpages. He recorded whether a target was found, and if so, its URL was noted. All rosters reported by RosterFinder for a school website were also examined and hand tagged. Figure 11 provides a summary of comparative results, which is further discussed below.

Two kinds of rosters were recognized. A "complete target" is a targeted roster in which nearly all targeted students are included in the roster. A "partial target" is a targeted roster in which some targeted students appear, but there is no expectation that the roster is almost complete. For example, a list of students who gave their permission to be listed may comprise a partial target if few students appear to be listed.

Online phone directories provide a complete target if undergraduate students are included and the directory is made available from a computer science website. Most online directories were school wide, not computer science specific, and therefore not included in these experiments.

In all cases where a manual roster was found, RosterFinder located the roster. Figure 11 (column M) shows more than half (16 of 30) of the schools had complete targets found manually. RosterFinder found those same complete targets (see column R).

In 4 of the 6 schools where a partial target was found manually, RosterFinder located the same partial target, but, in the other 2 schools, the California Institute of Technology and the Massachusetts Institute of Technology (MIT), RosterFinder found complete targets. It also found complete targets in 4 schools where no manual target was found, and a partial target in a school where no manual target was found.

RosterFinder found the names of more than 15,000 targeted students from the subject schools. Student names are archived at http://privacy.cs.cmu.edu/dataprivacy/projects/icu/index.html.

5.7 Observations from Experiments

Here are some general observations. Several targeted rosters not only included the names of the students, but also their photographs. Rosters of students enrolled in particular courses were commonly found. Rosters of women and minority students were prevalent. A roster of Russian students studying in the United States, complete with contact information, was found; it exemplifies the kind of information found on foreign students.

Rosters can also provide inferences on the race and religion of students. Spelman College is a private, historically Black college for women located in Atlanta, Georgia. More than 97% of its students are black [16]. Approximately 98.6 percent of the students Brigham Young University are members of the Mormon Church [17].

6. Related Work

While this work is the first to examine the problem of locating lists of people, there has been prior work on data linkage and on trail re-identification where sensitive information about people is learned from fragments of seemingly innocent information left behind and given away freely.

Direct linkage allows people whose data are contained in a given de-identified dataset to be reliably re-identified by linking it to an explicitly identified dataset [18]. A de-identified dataset is void of any explicit identifiers, such as name or address. An explicitly identified dataset, containing name and address, is linked to the de-identified dataset using fields appearing in both datasets. For example, in the United States, {*ZIP code* (5-digit postal code), *gender*, *date of birth* (month, day, year)} uniquely identifies 87% of the population. A de-identified dataset containing these fields can be reliably re-identified.

Trails algorithms have been developed for learning a person's identity from the uniqueness of the trail of data fragments left behind [19]. For example, online consumers may visit websites and make purchases at some of the sites visited. At each site visited, the IP address is left behind in the website's log. At locations where purchases were made, the consumer's name and address are also left behind. When weblogs (IP addresses) and customer lists (names and addresses) are shared across organizations, trail algorithms can match people to their IP addresses (assuming static or persistent IP addresses).

In both data linkage and in trail re-identification, the task is to re-identify the person who is the subject of the data to the seemingly anonymous information given. In this work, however, a description of the kind of information (or roster) sought is provided, and not initial data. Then, explicitly identified information is directly uncovered.

7. Privacy Implications

This paper began by examining the Princeton-Yale incident, which underscores the confidentiality problems realized by competition. Information about the identities of Yale's students provided strategic advantage to Princeton by allowing Princeton to compare and possibly adapt its admissions policies. This paper ends by examining the nature of the changes enabled by today's technology and at the challenge to privacy that remains.

7.1 Technology's Erosion of Privacy

The experiments provided in this paper underscore the role of technology in exasperating privacy conflicts. Before 1996, resumes, school newspapers, class lists, and school phone directories had more privacy protection because of their physical boundaries. There were natural limits to their availability. Today, with so much of this information provided freely over the Web, physical boundaries are removed, and the data can be searched, sorted, and processed in ways not feasible when the information appeared only in print medium.

In the United States, the Family Educational Rights and Privacy Act (FERPA) addresses rights parents and students may have concerning school records. Section 5(a) and 5(b) of FERPA relate to this paper:

> (5) (A) For the purposes of this section the term "directory information" relating to a student includes the following: the student's name, address, telephone listing, date and place of birth, major field of study, participation in officially recognized activities and sports, weight and height of members of athletic teams, dates of attendance, degrees and awards received, and the most recent previous educational agency or institution attended by the student.
>
> (B) Any educational agency or institution making public directory information shall give public notice of the categories of information which it has designated as such information with respect to each student attending the institution ..."

In an earlier experiment and reported in Figure 11, no seemingly complete roster of undergraduate computer science students was found at MIT by manually navigating through the links at MIT's official site. However, RosterFinder did locate a seemingly complete roster of these students. This finding may be significant to privacy policy enforcement. MIT's stated policy, located at http://web.mit.edu/policies/11.3.html, states:

> "7.2 School, department and lab web pages - Faculty, staff and students must exercise caution in posting directory and other information to a web page that is accessible to MIT and/or to the public. Students have the right to withhold directory and other information from public distribution. Faculty and staff must receive permission from each student to post personal information and identification photographs to web pages."

The webpage identified by RosterFinder at MIT was used to match advisors to advisees. It contained a list of all enrolled undergraduates in computer science. This may be in violation of this policy. A list of advisor – advisee assignments was also found at the University of Illinois.

This example shows that even if a school policy restricts certain rosters from being publicly provided, enforcement can be illusive in a large organization. RosterFinder can be used for policy enforcement by insuring online lists at the organization's website are not rosters the organization has deemed strategically sensitive.

This example also demonstrates that it was not policies alone that historically provided privacy protection. It was privacy policies in the absence of today's technology that was the effective guard. So, how can technology allow the benefits technology currently allows society to enjoy, while also providing protections enjoyed in the absence of the technology? This lies at the heart of the data privacy challenge.

7.2 Data Privacy Challenge

Privacy conflicts emerge between all kinds of combinations of entities – people, governments, organizations, corporations and countries. Data privacy problems emerge because a lack of privacy protection for an entity often leads to situations in which strategic information about the entity is revealed to others. This can often cause the entity harm, distress or discomfort. Having the information about the entity can often empower others.

Privacy policy is one answer. Privacy policy is realized by personal actions, business practices, and government regulations. Privacy policy is limited to concise unambiguous word description and cost effective practices implemented with existing resources. Privacy policy is restricted to crude choices, such as providing the information or not. Consider FERPA as an example.

Data Privacy is the study of computational solutions for solving privacy and confidentiality problems in data. Computational solutions provide the opportunity to monitor, coordinate, and alter shared information in ways that provide scientific assurances about what can be inferred from the released information. Data are shared, but the version of the values shared or the way in which the data are shared has some assurances of protection. In comparison to computer security, data privacy controls inferences about the content of what is shared, not merely access to explicit values. In comparison to privacy policy, data privacy offers graduated solutions for sharing data, and is not limited to crude choices of either explicitly sharing values "as is" or not.

It is left as a challenge to computer scientists to develop and deploy data privacy technology to thwart RosterFinder. Some obvious first attempts might include limiting web access, storing rosters as image files rather than as text, or adding hidden words in the document making it difficult to search. Advancements in computer technology generated these privacy concerns. Advancements in privacy technology can solve them.

8. Acknowledgements

This work stems from the initial manual effort of Rebekah Siegel, who as a high school student demonstrated the kind of patience, perseverance and methodical dogma found in researchers many times her senior. James Morris, Janine Pischke and Walter Schearer in the School of Computer Science at Carnegie Mellon University (CMU) sponsored this project. The work was conducted in the Data Privacy Laboratory at CMU, and much gratitude goes to Marshall Warfield for manually detecting and classifying rosters, to Bradley Malin, Elaine Newton, Benjamin Vernot, Yiheng Li, Aaron White and Ralph Gross for numerous suggestions and comments, and to Sherice Livingston for administrative support.

References

[1] Ferdinand, P. and Barbaro, M. Yale Tells FBI of Rival's Breach of Web Site Princeton Suspends Admissions Official Over Snooping Into Student Files. *Washington Post*. July 26, 2002; p A02.

http://www.washingtonpost.com/ac2/wp-dyn?pagename=article&node=&contentId=A2983-2002Jul25&notFound=true

[2] Princeton Snoops on Yale. *Wired*. July 26, 2002. http://www.wired.com/news/culture/0,1284,54140,00.html

[3] Chen, H. and Bian, G. White page construction from web pages for finding people on the Internet. In *Computational Linguistics and Chinese Language Processing*, v.3 Feb 1998, pp. 75-100.

[4] *Proceedings of the Fourth Message Understanding Conference*, San Francisco, 1992. Morgan Kaufman Publishers.

[5] *Proceedings of the Fifth Message Understanding Conference*, San Francisco, 1993. Morgan Kaufman Publishers.

[6] *Proceedings of the Sixth Message Understanding Conference*, San Francisco, 1995. Morgan Kaufman Publishers.

[7] Coates-Stephens, S. *The Analysis and Acquisition of Proper Names for the Understanding of Free Text*, Kluwer Academic Publishers, Hingham, MA, 1993.

[8] Grisham, R. and Sundheim, B. Message Understanding: a brief history. In *Proceedings of the Sixth Message Understanding Conference,* San Francisco, 1996.

[9] Poibeau, T. and Kosseim, L. Proper Name Extraction from Non-Journalistic Texts. In *Computational Linguistics in the Netherlands Meeting*. W. Daelemans, K. Sima'an, J. Veenstra and J. Zavrel, 2001 (eds), Amsterdam/New York, 2001, p. 144-157.

[10] *Social Security Online History*, United States Social Security Administration. Available at http://www.ssa.gov/. Washington: 2003.

[11] *Google Web API for the Java Programming Environment.* Available at http://www.google.com/apis/. Mountain View: 2003.

[12] America's Best Graduate Schools: Science Programs, Computer Science. *U.S. News and World Reports*. 2003. At: http://www.usnews.com/usnews/edu/grad/rankings/rankindex.php

[13] Sweeney, L. Replacing Personally-Identifying Information in Medical Records, the Scrub System. In: *Proceedings of the American Medical Informatics Association*. Cimino, JJ, (ed.) Washington, DC: Hanley & Belfus, Inc., 1996, pp. 333-337.

[14] Baeza-Yates, R. and Ribeiro-Neto, B. *Modern Information Retrieval*, Addison Wesley, Boston, MA, 1999.

[15] *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Tampere, Finland, 2002. ACM.

[16] Spelman College Fact Book 2000-2001. Atlanta, GA 2001. http://www.spelman.edu/factbook/factbook0001/#enrollstats, Spelman College Fact Book 2000 – 2001

[17] Brigham Young University Home Fact File Students, http://www.byu.edu/about/factfile/stud-ff4.html#demo.

[18] Sweeney, L. *k*-anonymity: a model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10 (5), 2002; 557-570.

[19] Malin, B. and Sweeney, L. Compromising Online Anonymity with Trail Re-identification.. Carnegie Mellon University, School of Computer Science, Data Privacy Laboratory Technical Report, LIDAP-WP13. Pittsburgh: June 2003.