Interpreting News Through the Science of Networks

A Dissertation Presented
by
Andrew Mehler

to
The Graduate School
in Partial Fulfillment of the
Requirements
for the Degree of
Doctor of Philosophy
in
Computer Science
Stony Brook University

August 2008

Stony Brook University

The Graduate School

Andrew Mehler

We, the dissertation committee for the above candidate for the
Doctor of Philosophy degree, hereby recommend
acceptance of this dissertation.

Professor Steven Skiena, Advisor
Computer Science Department

Professor Amanda Stent, Chairperson of Defense
Computer Science Department

Professor Esther M. Arkin
Applied Mathematics and Statistics

Professor Matthew Lebo
Department of Political Science

This dissertation is accepted by the Graduate School.

Lawrence Martin
Dean of the Graduate School

Abstract of the Dissertation

Interpreting News Through the Science of Networks

by

Andrew Mehler

Doctor of Philosophy

in

Computer Science

Stony Brook University

2008

On-line news sources provide a large and comprehensive corpus of world events and news entities. The Lydia project (www.textmap.com) analyzes over a thousand on-line newspapers every day to discover news trends, sentiments, and geographic biases. The aim of the project is to deliver news analysis on a scale of content that would be impossible for a person to read, and to mine the data to learn information that a human would otherwise be unable to realize. We leverage network analysis techniques to understand this real world online news data. First, a geographic network based on cities is used to visualize our data, and to quantify which news entities have a geographic bias. We also consider the network formed by co-occurrences of people in news articles. We discover the network is scale-free, and show what we may learn from this network. We then show how we can clean the network, removing noise and *spurious* edges. Removing these spurious edges leaves the properties of the network essentially unaltered. Finally, we show how to discover communities in the network by using a small set of example members. Given some example members (20-400) we are able to discover communities of thousands of members. In addition, we describe the co-reference resolution technique, an important step in improving the reliability and robustness of th network

data. The algorithm for clustering co-references is shown, as well as a method of hashing names to quickly discover co-reference candidates.

Contents

[1] **This chapter is an extended version of [53].**

[2] This chapter is an extended version of [49].

[3] **This chapter is an extended version of [54].**

List of Tables

List of Figures

Acknowledgements

First off, I would like to thank my advisor, Steven Skiena. Without his help and understanding, this document would have been completely incomprehensible, and possibly non-existent.

I thank my friends and family. They have been supportive throughout, despite their continued bafflement as to why I can't fix their personal computers.

I give thanks to all the faculty members I have interacted with; I leave thinking them as the most notable quality of the university. I especially thank them for the Friday afternoon reading group (aka algorithm jam sessions).

I also thank all of my lab-mates. Dimitris 'D1' Papamichail, for teaching me about classical guitar, Yin Yang, and maybe a few computer science discussions in between. I Thank Levon Lloyd, and Dimitris 'D2' Kechagias for giving the project its start. Namrata Godbole, Prachi Kaulgud, Jae Hong Kil, Michael Papile, Manjunath Srinivasaiah, Izzet Zorlu, Alex Turner, and Alex Kim, who kept things going in the projects infancy. Sagar Pilania, and Shashank Naik for making heatmaps more interesting and colorful. The class of 2007 Masters students Sushma Devendrappa, Ashwin Subrahmanya, Anand Mallangada, Sandesh Devaraju, Jai Balani, and Lohit Vijayarenu. The 2008 students Gopalakrishnan Iyer, Sriram Kumaran, Gayathri Ravichandran, Jahangir Mohammed, and Paavan Shanbhag. And those who will carry on the torch of a project whose capabilities and relevance is constantly growing, I thank and wish luck to Mike Bautin, Wenbin Zhang, Yeongmi Jeon, Charles Ward, Anurag Ambekar, Akshay Patil, Swapna Reddy, Shrikant Shanbag, Yeongmi

Jeon.

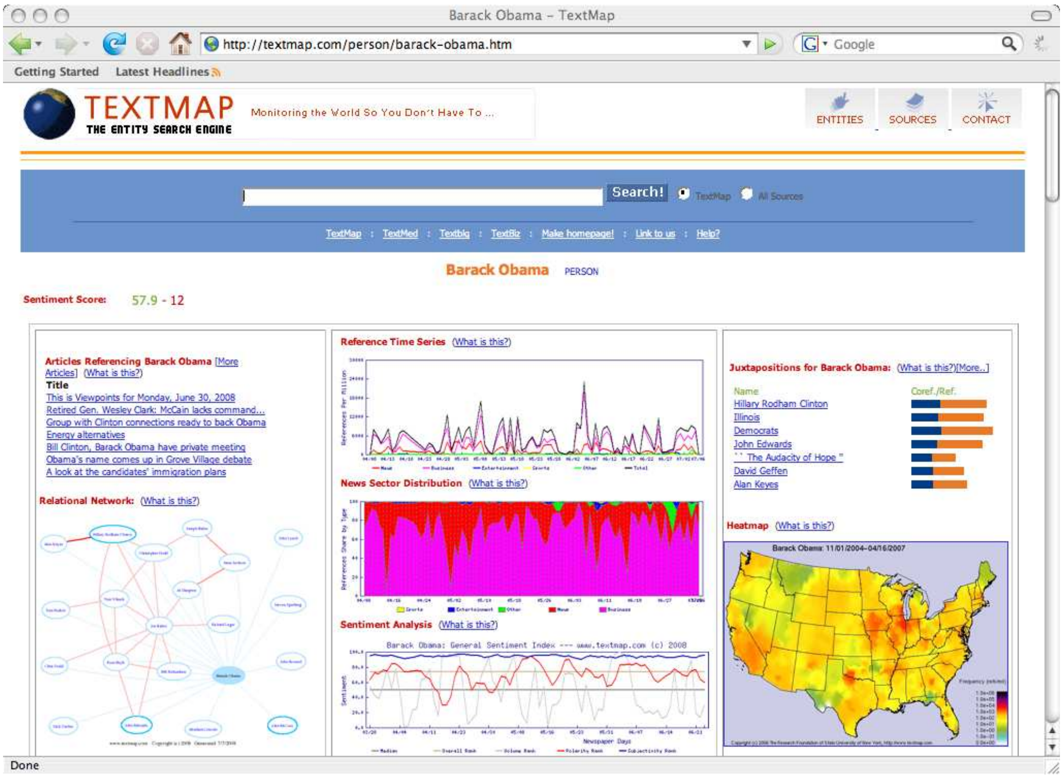
If I have neglected to thank anyone, I blame it not on lack of appreciation, but on my diminished capacity in focusing all my attention to the completion of this document.

The work in this thesis was completed with funding from The Research Foundation of State University of New York.

Chapter 1

Introduction

Online news sources provide a large and comprehensive corpus of world events and news entities. The Lydia project (www.textmap.com) analyzes over a thousand online newspapers every day to discover news trends, sentiments, and geographic biases. An example of the online front end of the project is shown in Figure 1. The aim of the project is to deliver news analysis on a scale of content that would be impossible for a person to read, and to mine the data to discover world facts that a human would be unable to realize. We leverage network analysis techniques to understand this real world online news data. First, a geographic network based on cities is used to visualize our data, and quantify which news entities have a geographic bias. We also consider the network formed by co-occurrences of references to people in news articles. We discover the network is scale-free, and show what we may learn from this network. We then show how we can clean the network, removing noise and *spurious* edges. Removing these spurious edges leaves the properties of the network essentially unaltered. Finally, we show how to discover communities in the network by using a small set of example members. Given some example members (20-400) we are able to discover communities of thousands of members. In addition, we describe our co-reference resolution technique. Resolving co-references is an important step in improving the reliability and robustness of



Barack Obama – TextMap

http://textmap.com/person/barack-obama.htm

TEXTMAP THE ENTITY SEARCH ENGINE Monitoring the World So You Don't Have To ...

Barack Obama PERSON

Sentiment Score: 57.9 - 12

Articles Referencing Barack Obama [More Articles] (What is this?)

Title

This is Viewpoints for Monday, June 30, 2008
Retired Gen. Wesley Clark: McCain lacks command...
Group with Clinton connections ready to back Obama
Energy alternatives
Bill Clinton, Barack Obama have private meeting
Obama's name comes up in Grove Village debate
A look at the candidates' immigration plans

Juxtapositions for Barack Obama: (What is this?)[More..]

Name
Hillary Rodham Clinton
Illinois
Democrats
John Edwards
" The Audacity of Hope "
David Geffen
Alan Keyes

Done

Getting Started
Latest Headlines
Google
ENTITIES
SOURCES
CONTACT
Search!
TextMap
All Sources
TextMap : TextMed : Textblg : TextBiz : Make homepage! : Link to us : Help?
Relational Network: (What is this?)
Reference Time Series (What is this?)
References Per Million
News
Business
Entertainment
Sports
Other
Total
News Sector Distribution (What is this?)
References Share by Type
Sentiment Analysis (What is this?)
Barack Obama: General Sentiment Index --- www.textmap.com (c) 2008
Sentiment
Newspaper Days
Coref./Ref.
Heatmap (What is this?)
Barack Obama: 11/01/2004–04/16/2007
Frequency (refs/mil)

Figure 1: Textmap (www.textmap.com) page of Barack Obama

the network data. An algorithm for clustering co-references is developed, as well as a method of hashing names to quickly discover co-reference candidates.

The architecture of the Lydia system has been described in detail in [48]. The main components of Lydia are

1. Spiders to download the news sources.

2. Named entity recognition, including co-reference resolution [49].

3. Various derivative analyses based on the named entity recognition.

The system can also be extended to other sources such as journal databases, financial reports, and blogs [47]. Some derivative analyses provided include question answering [42], spatial analysis and geographic bias (‘heatmaps’) [53], and sentiment analysis [29]. Searching our database is also discussed in [8].

The component of Lydia that we leverage most for further study is the *Named Entity Recognition* pipeline. This stage reads plain article text, and extracts the Named Entities (which we will call ‘actors’ or ‘entities’). These are the proper noun entities (people, places, organizations) that the articles discuss. For example, if the article mentions the string ‘Bill Clinton’, then the pipeline would extract that ‘Bill Clinton’ is a named entity, and its type is ‘PERSON’. When two named entities appear in the same article, we say there is a *co-occurrence*, or that there exists a *juxtaposition*. Clearly, much can be learned about an entity from other entities with who they are talked about (who is in their neighborhood). These neighbors in the articles can be represented as a network, and network science techniques can be applied.

1.1 My Publications

I have been working on problems related to news analysis since May 2003. My three publications on this work to date [49, 53, 54] are summarized below, and will also be covered in subsequent chapters.

- *Spatial Data Analysis.* In [53] the spatial analysis techniques used with Lydia are described. The main results of this are the ‘heatmap’ images that are seen on entity pages. Considering the maps as geographic networks, we interpolate ‘heat’ values from news data, and develop methods to quantify geographic bias. We describe the model for estimating geographic popularity, calculating how much bias a map shows, color schemes used, and the production process for rendering the maps.

- *Determining Co-References.* In [49] we describe our techniques for determining co-referential entities. This is the problem of determining that ‘George

Bush' and 'George W. Bush' actually are referring to the same person. We consider the vector of terms that co-occur with an entity, and base a similarity score on the cosine similarity of two term vectors. In terms of network science, co-reference candidates are vertices that are *structurally equivalent*. Since the space of terms is so large, we first reduce the dimensionality of the problem, by clustering the terms into 100 different groups.

- *Corrective Hashing.* Password corrective hashing is described in [54]. We show that a password scheme can retain nearly all of its security with increased usability by allowing a small number of errors in the password. While seemingly separate from the main themes of this thesis, corrective hashing is a needed method for reducing the complexity of finding co-references. It can find candidates for co-reference, as a way for correcting common spelling errors, and typos.

1.2 Network Science

More general knowledge of the world can be discovered from reading news articles besides just the topics of the articles. This includes identifying what communities there are, what the most important entities are, how the entities are inter-related, what is talked about positively and negatively, where geographically entities are talked about most, and the temporal interest in entities. Much of this can be learned by studying the networks that arise from Lydia's named entity extraction. Knowledge of the world can be optimized by simply studying the networks created by entities co-occurring in news articles. Among the networks we can consider are the entity co-occurrence network (for different definitions of co-occurrence), the entity - article participation network, or even the entity - author participation network. For example Figure 2 shows an example co-occurrence network focused on

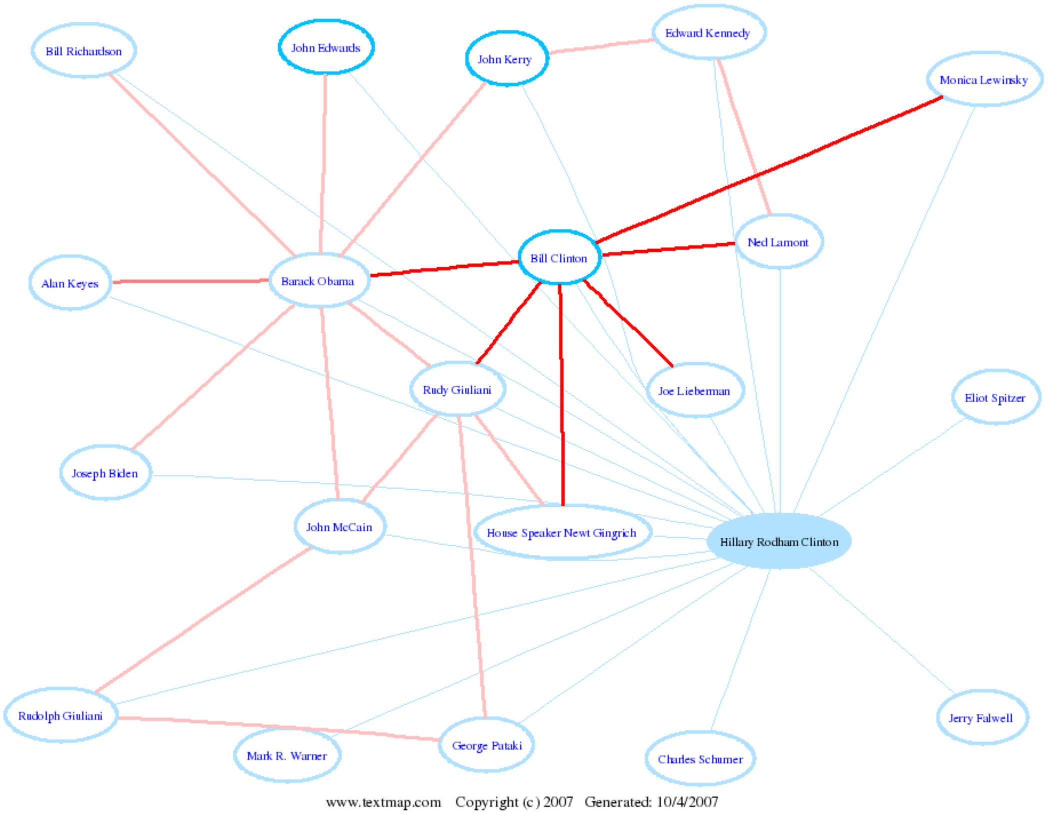


Figure 2: An example network, focused on Hillary Clinton.

Hillary Clinton. This network is taken from actual data accumulated up to 4 October 2007. For visualization, not all network edges are shown, only those deemed most important.

Network science techniques can be applied to these networks to discover new information about the world. Some of the problems that can be solved with social network analysis are: identifying spurious relationships, discovering communities (clusters), monitoring changes in communities/network over time, discovery of dominating and sibling entities, classification of entity type, co-reference and disambiguation, and predicting sentiment.

Network science studies how individual elements interact to exhibit phenomenon that cannot be described by individual interactions. The reductionist hypothesis, which states that a system can be understood by breaking it down into its smallest parts and understanding the interaction between individual particles, was rejected by Anderson in his 1972 paper [2]. Instead, network science tells us that complex interactions in a network lead to higher order phenomenon which are not predicted by individual interactions. This is why higher order disciplines such as chemistry, biology and economics exist [81].

One property displayed by social networks is the ‘Small World Phenomenon’, popularized by the Stanley Milgram experiment [55]. Real world networks tend to have small distances between vertices, and these small paths are easily found locally by greedy routing. That is, not only do short paths exist, but they can be easily found by the members of the network, using only information about a members neighbors [82]. Milgram showed this by having subjects in Nebraska route a letter to an unknown person in Massachusetts, where at each hop the letter could only be sent to someone the sender was on a first name basis with. Of the chains that were completed, the average length of the chain was only 6. This property of networks is somewhat conflicting with the property that real networks tend to have a high clustering coefficient (groups tend to form), since highly clustered networks would have vertices isolated from vertices not in their cluster [81]. Having both of these properties is what distinguishes small-world networks.

At the core of network theory is simple graph theory. Graphs are used to describe social networks, and many graph theoretic properties are commonly used to describe the properties of a social network. These include diameter, clustering coefficients, and centrality measures. Centrality measures tell us about the ‘importance’ of a node of a network. Observing how information flows through members of a network will show that nodes of high centrality have a large influence over the network. The spread of information through a population can be learned from studying the network properties. In some cases, as in viral marketing, we wish to target very few, but very important nodes that will lead to a cascade of an idea. But in the case of preventing the spread of a disease, we wish to discover what network connections should be altered to make a population resilient to an outbreak [81].

A major concept in network theory is that a network is dynamic. It changes and evolves over time. The network will cluster, and form groups, and these groups will change over time. The key observation is that although group formation and membership externally seems to be an independent decision of the entities, the structure of the network makes group movement predictable. Group formations are determined by network configuration to some extent, and not freely chosen by entities.

1.3 Properties of News Networks

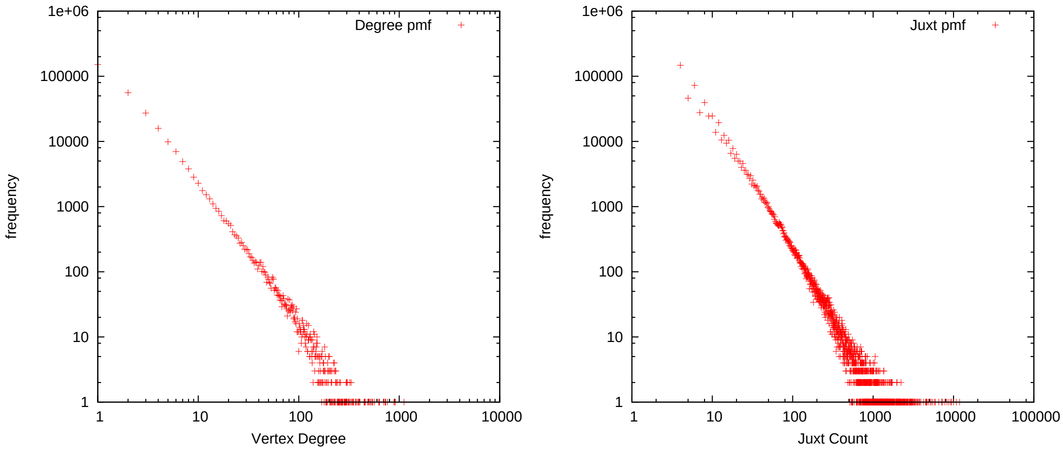


Figure 3: Degree (r) and Juxtaposition Count (l) Distribution (log-log) of News Entity Network.

1.3.1 Data Sets

Our analysis is done on three different sets of data. The principal area of interest is in online news data. We have 2 sets of data for this, one small set of very reliable, well known sources (i.e. The New York Times, The Washington Post, etc.); and a second large set of all sources we could acquire (including many smaller, local sources). The smaller set, called ‘goodnews’, is less noisy, and easier to experiment with given its size. However we feel the larger set, called ‘dailies’, is more likely to contain interesting signals.

Our goodnews news entity network consists of data extracted from 158 online news sources, starting from November 1 2004. There are 495,320 vertices (entities) in the network, and 2,249,568 edges. Restricted to people, there are 144,851 vertices and 265,779 edges.

Our dailies news entity network consists of data extracted from 917 online news sources, from November 1, 2004. There are 853,054 vertices (entities) in the network, and 4,753,134 edges. Restricted to people, there are 299,486 vertices and 594,884 edges.

For comparison, we also have a network obtained from pubmed abstracts, called ‘medline’. There are 29,282 vertices (entities) in the network, and 260,730 edges.

In addition, we analyze the networks on all extracted entities, and also just on the subset of people entities. The summaries of main net work properties are shown in Table 1.

Source	Categories	Vertices	Edges	Avg. Degree
goodnews	PERSON	144,851	256,779	3.545
goodnews	all	495,320	2,249,568	9.083
dailies	PERSON	299,486	594,884	3.973
dailies	all	853,054	4,753,134	11.144
medline	all	29,282	260,730	17.808

Table 1: Network Summaries.

Figure 3 shows a log-log plot of the degrees of the nodes of our network. The linear trend of the plot suggests a power-law distribution for node degrees, meaning our network is in the category of scale-free networks. Scale-free networks are a class of networks whose degree distributions obey a power law [12]; the probability of a node having degree k, $P(k)$ is

$$P(k) \propto \alpha^{-k}$$

where α is a constant depending on the network (typically between 2 and 3). This model suggests that entities display preferential attachment. That is, vertices will attach to popular vertices with higher probability; a rich get richer scenario. The model can be expanded with a *fitness* for each vertex to explain the popularity of new vertices [12, 7]. The highest degree nodes in our network are shown in table 2. We see three important political entities, and a popular Nascar driver. Also seen are some example entities having middle and low degrees.

High Degree		Mid Degree		Lowest Degree	
George W. Bush	1015	Alec Baldwin	125	Jeff Tweedy	15
Mark Martin	803	David Beckham	124	Vanna White	10
Barack Obama	736	Larry King	119	Seth MacFarlane	5
John McCain	663	Kurt Vonnegut	48	Martin Van Buren	3

Table 2: Example Degrees of Nodes in goodnews data.

1.3.2 Juxtapositions

The network we consider is a co-occurrence network. There is an edge between two entities if they both appear in the same sentence in an article. The weight associated with an edge is the number of times the two entities co-occur. Figure 3 shows the degree distribution of the co-occurrence counts of actors. This result coincides with our notion that the network is scale-free, that is the entities tend to connect to other popular entities.

The rest of this thesis is organized as follows: Chapter 2 discusses spatial analysis, and the generation of 'heatmaps'. Our co-reference resolution algorithm is described in Chapter 3. Chapter 4 discusses password-corrective hashing, a technique that is used in co-reference resolution. Chapter 5 shows how the network can be used to remove spurious edges. Discovering a community from seeds is in Chapter 6. Finally, Chapter 7 concludes with areas of future research.

Chapter 2

Spatial Analysis of News Sources [1]

2.1 Introduction

Lydia tracks the occurrences of hundreds of thousands different entities arising in news sources. An exciting consequence of this is that we can establish regional biases in the news, by analyzing the relative frequency that entities are mentioned in different news sources. We can report the results of our analysis through “heatmaps”, which are data maps reflecting interest in a given entity as a function of location.

Typical heatmaps of interest are presented in Figures 4-5. The heatmap for New York governor *George Pataki* is from October 2005, and focuses on his home state of New York; but also exhibits a secondary concentration in Iowa. This is explainable by Pataki’s consideration as a presidential candidate, and the significance of the Iowa Caucuses, the first test of the U.S. presidential primary season (such ambitions for Pataki were not realized). The heatmap for Phoenix Sun’s basketball star *Steve Nash* reflects home town fan interest in both his current and previous (Dallas Mavericks) teams. Heatmaps of geographical locations also show interesting biases. News interest in *Mexico* is significantly heavier around the U.S. / Mexico

[1] **This chapter is an extended version of [53].**

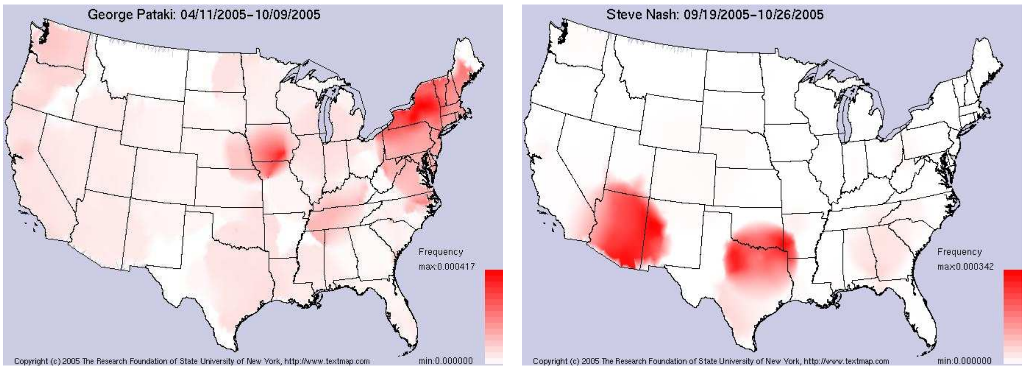

George Pataki: 04/11/2005-10/09/2005
Frequency
max:0.000417
min:0.000000
Copyright (c) 2005 The Research Foundation of State University of New York, http://www.textmap.com
Steve Nash: 09/19/2005-10/26/2005
Frequency
max:0.000342
min:0.000000
Copyright (c) 2005 The Research Foundation of State University of New York, http://www.textmap.com

Figure 4: Heatmaps for New York Governor *George Pataki* (l) and Dallas/Phoenix basketball star *Steve Nash* (r).

border, particularly in southern Texas. *Washington, DC* reflects national interest in its capital city, with stronger concentrations centered in the District of Columbia (reflecting local interest) and the State of Washington (reflecting natural language processing artifacts in resolving city references from state references). National figures such as President George Bush show little regional bias, while former international movie star Arnold Schwarzenegger is today primarily a state political figure.

It is the geographical bias among primary news sources which permits us to construct maps of relative interest in particular entities. An alternate way to study relative geographic interest is to compare the reference frequency of entities in a given news source. These biases are illustrated in Table 3, which present significantly overrepresented entities in each of three major American newspapers, as scored by the number of standard deviations above the mean frequency of reference over all sources. These over-represented entities include local political and business figures (e.g. *Brad Fitzpatrick* of LiveJournal and television star *Oprah Winfrey*, based in San Francisco and Chicago, respectively), local sports figures/teams (e.g. *Steve McMichael* and *Dwayne Wade*), and even local dialects (e.g. *Estados Unidos*

San Francisco Chronicle		Chicago Tribune		Miami Herald	
Entity	Score	Entity	Score	Entity	Score
Wuksachi Lodge	24.39	Steve McMichael	24.38	Estados Unidos	16.49
Brad Fitzpatrick	24.39	Chicago Tribune	23.20	Broward	14.65
Golden Gate Park	15.29	Richard J. Daley	15.89	Dwyane Wade	13.60
Bay Area	12.03	White Sox	13.86	Miami-Dade County	12.48
San Francisco, CA	10.20	Ozzie Guillen	10.42	Marlins	11.55
Giants	4.66	Oprah Winfrey	10.39	Adam Kidan	11.08

Table 3: Overrepresented Entities in Three Major U.S. Newspapers

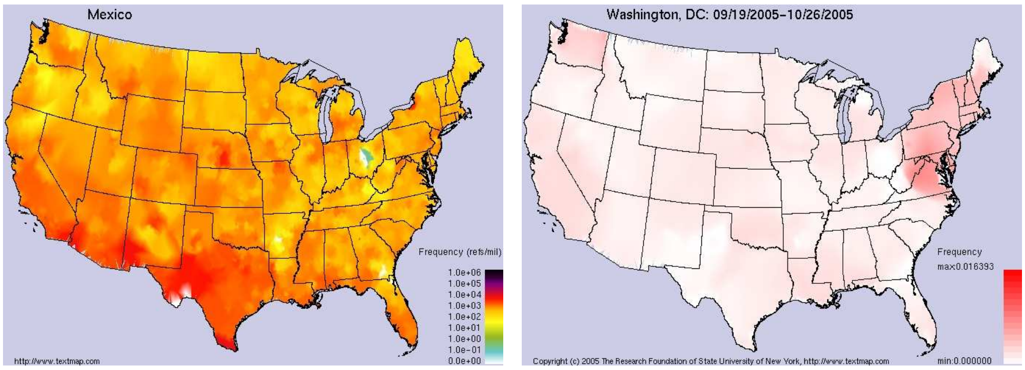


Figure 5: Heatmaps for two geographic locations, namely *Mexico* and *Washington, DC*.

in heavily Cuban Miami).

Also, we can also study over-representation by city, as in Table 4. We use the ‘heat’ measure as used in the heatmaps, searching for which terms are the most standard deviations above their mean.

Note that these source biases reflect interest in the primary location the given paper. However, the set of U.S. cities with spiderable online daily newspapers is surprisingly small, so sophisticated modeling and analysis are needed to interpolate this data throughout the United States. Our contributions are:

- *News Source and Coverage Analysis* – We discuss the basic mechanics of

San Francisco Chronicle		Chicago Tribune		Miami Herald	
Entity	Score	Entity	Score	Entity	Score
Gavin Newsom	10.84	Chicago, IL	8.57	Miami, FL	10.26
San Francisco, CA	10.56	Richard Daley	7.06	South Florida	9.53
Bay Area	8.44	Joan Humphrey Lefkow	5.20	Fort Lauderdale, FL	8.76
Pedro Feliz	5.36	Aon Corp.	4.69	Cuba	8.09
BALCO	5.29	Salvador Dali	4.54	Caracas	7.02
Kimberly Bell	5.02	Wrigley Field	4.42	Florida Marlins	6.91

Table 4: Most Overrepresented Entities in Three Important U.S. Cities

large-scale news acquisition and analysis, including spidering and duplicate document identification. We use visualization techniques to demonstrate how news sources are distributed around the country. We do *not* discuss the details of our entity extraction / NLP analysis, which has been previously presented in [49, 48, 42].

- *Source-Influence Modeling for Entity Analysis* – Interpolating entity distributions from roughly 500 different newspapers to reflect relative interest over the entire United States requires some sophisticated modeling. In this chapter, we present the details of our news source-influence model. This model is based on computing an appropriate sphere of influence for each newspaper, as a function of its circulation, location, and the population distribution of the United States. We also describe our model for allocating the relative contribution of all news sources influencing each location.

- *Visualization Techniques for Data Maps* – The engineering of our spatial news analysis system required a variety of decisions concerning visualization techniques, which may be of independent interest. These include the use of Delauney triangulations for surface interpolation and evaluations of assorted color scales.

- *Identifying Interesting Heatmaps* – Our system is capable of constructing heatmaps for thousands of different entities on a daily basis – far more than can be exhaustively viewed by any human observer. Many heatmaps are uninteresting in that they show no regional bias. Identifying the most interesting heatmaps for visual inspection requires the development of statistical methods for evaluating geographical bias.

  We propose a total of five different discrimination functions, each a variant of one of two general methods, namely variance analysis and connected-component histograms. We present the results of computational experiments that demonstrate that while all can successfully distinguish spatially-interesting entity maps from those of unbiased entities and random distributions, the best in practice appear to be the weighted variance and maximum gap discriminators.

2.1.1 Previous Work

Statistical geographic maps are studied in depth in [75]. The Moran coefficient is described as a measure of spatial autocorrelation. The Moran coefficient is given as

$$MC = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(X_i - \overline{X})(X_j - \overline{X}) / \sum_i \sum_j w_{ij}}{\sum_{i=1}^{n}(X_i - \overline{X})^2/n} \tag{1}$$

where w_{ij} is 1 for adjacent elements and 0 otherwise. When examining heatmaps, we do not suppose spatial autocorrelation to be of use; as we expect both geographically biased and unbiased maps to have high spatial autocorrelation, a result of the modelling and of the nature of news. A comprehensive overview of map layout, color, and all aspects of creation are also covered.

Miller and Han [56] give frameworks and algorithms for data mining tasks on spatial and geographical databases. They focus on representation, rule mining, clustering, outlier detection, and other data mining tasks. Work on smoothing has been done to help visualize data. Algorithms such as Head-Banging [57], [32] (a median based smoother), splines, surface modelling, wavelet transforms, are used to smooth noisy data. Our problem differs from these in that smoothing algorithms start with an exact data map, and hope to filter out noise. We do not start with an exact data map; we start with loose geographic information about heat, based on newspaper locations, and attempt to create the exact data. For instance, [57] examine mortality data maps. The data map begins with exact values of mortality rates, and their smoother makes this information easier to visualize. We however do not begin with any data. We must calculate the heat data for a city based on the newspapers from surrounding cities.

Lydia is the front-end analysis system we employ in this project to do entity extraction on our newspaper sources. An in-depth discussion on the architecture of the *Lydia* natural language processing (NLP) pipeline can be found in [48]. *Lydia* has been adapted to work on a variety of other text sources, including blogs [49], and served as the basis for a question answering system discussed in [42].

2.2 Text Acquisition and Analysis

The data for our analyses come from U.S. newspaper websites. In this section, we describe the mechanics of acquiring representative news text through spidering and duplicate article detection analysis before reporting an analysis of the coverage breadth of our news sampling.

2.2.1 Text Acquisition / Spidering

It is clearly infeasible for us to build custom spiders for each of the roughly 800 daily newspapers in the United States and approximately 300 daily English language newspapers overseas that Lydia uses. Instead, we developed a universal spider that downloads all the pages from a newspaper website, extracts all new articles, and normalizes them to remove source-specific formatting and artifacts.

Our spiders are built around the popular program *wget* [33] with the correct parameters; regulating the recursion depth (two levels suffices for most newspapers), user identification (via cookies), and wait time (for politeness, we never hit a website more than once per second). The news sources are divided by time zone, with many (at least 30) newspapers spidered in parallel across a given zone. Each download starts at 12:30AM local time. Each newspaper takes about 20-80 minutes to download, with a raw download size of 40-130MB. There are two reasons why we download more than 40 times the amount of data per newspaper day than we end up with: (1) *wget* downloads the entire directory structure of the website, including old articles and (2) each html file also contains a lot of other things including advertisements and navigational aids.

2.2.2 Duplicate Article Identification

An interesting issue we faced concerned identifying duplicate and near-duplicate news articles. Repeated instances of given news articles can skew the significance of our spatial trends analysis, so we need to eliminate duplicate articles before subsequent processing. Duplicate articles appear both as a result of syndication and because old articles are often left on a website and get repeatedly spidered.

Schleimer et al.[71] describe a clever solution to this problem in the context of plagiarism detection. By comparing hash codes on all overlapping windows of length w appearing in the documents, we can identify whenever two documents share a common sequence of length w, although at the cost of an index at least the size of the documents themselves. However, the index size can be substantially reduced by a factor of p with little loss of detection accuracy by only keeping the codes which are congruent to 0 mod p. This will result in a different number of codes for different documents, however. We discovered little loss of detection will happen if we select the *c smallest* codes congruent to 0 mod p for each article. The naive way to do this is to hash all $n - w$ windows for an n-character document in

$O(wn)$ time. However, The Karp-Rabin string matching algorithm [38] proposes an incremental hash code such that all codes can be computed in linear time.

Through experimentation, we discovered that taking the 10 smallest hashes of windows of size 150 characters that are congruent to 0 mod 100 gives (1) a good sub-sampling of the possible hashes in a document, (2) a reasonable probability that if two articles are near duplicates, then they will collide on at least two of these hashes and (3) a reasonable probability that if two articles are unique, then they will not collide on more than one of these hashes. Our experimental set of $3,583$ newspaper days resulted in a total of $253,523$ unique articles with $185,398$ exact duplicates and $8,874$ near duplicates according to our measure.

2.2.3 News Coverage Analysis

Every local newspaper has a readership centered around the city in which it is located. The size of the *sphere of influence* around a given paper is a function of (1) its readership (naturally measured by circulation or a proxy such as web hits) and (2) geographic population density. Details of our influence analysis will be presented in Section 2.3, but here we discuss its consequences on sampling density and potential for multi-source integration.

We have attempted to spider all of the roughly 800 daily U.S. newspapers we are aware of through authoritative web sources. Many newspaper websites are no longer active or are highly seasonal (particularly school newspapers which cease activity for summer and other break periods). Others (primarily small limited circulation papers) employ robot.txt files or even block IP addresses to prevent us from spidering them. The upshot is that we get occasional spidering data from roughly 600 online newspapers, however on any given day only about 500 sources are active.

We can use our model of influences to visualize the coverage of the sources we spider. Figure 6 presents a datamap where a city's heat is a function of the number of newspapers it is influenced by. We get significant coverage throughout the entire country, excepting isolated border locations around Maine, Minnesota, and Texas. The regions covered by more than ten sources are emphasized in the binary map of Figure 7. Many of these areas are surprisingly scattered around the country, reflecting intense competition among small papers in many local markets.

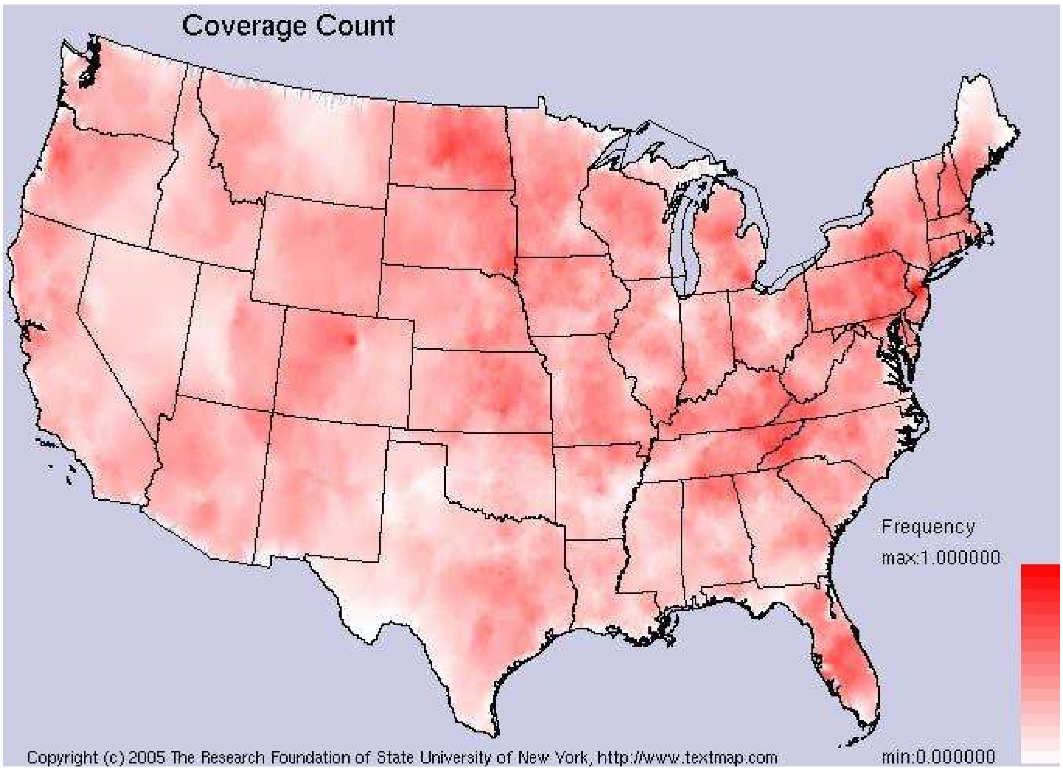

Coverage Count
Frequency
max:1.000000
min:0.000000
Copyright (c) 2005 The Research Foundation of State University of New York, http://www.textmap.com

Figure 6: The number of different news sources influencing each U.S. city.

Figure 8 measures the relative raw volume of text that influences residents of each location. The national media centers of California and New York are significantly overrepresented by this measure.

However, if we weight the word count by the influence (see Section 2.3 as in Figure 9 we see that the words of news any city gets is much more uniform, perhaps indicating a universal capacity for news volume.

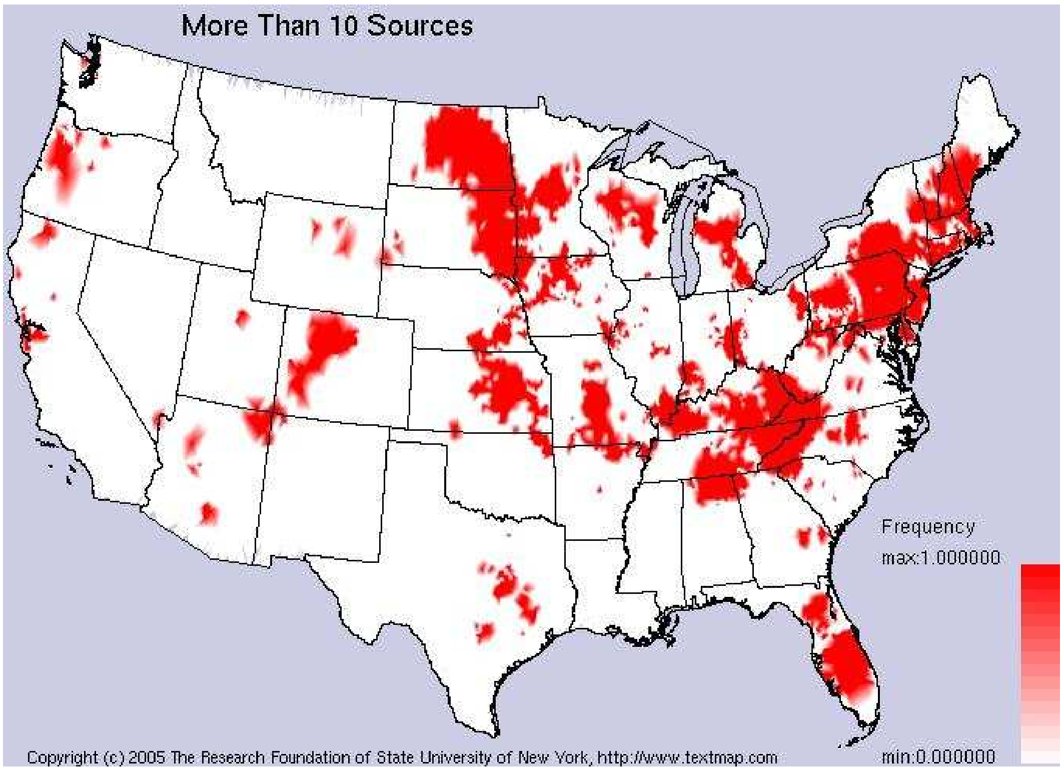

More Than 10 Sources
Frequency
max:1.000000
min:0.000000
Copyright (c) 2005 The Research Foundation of State University of New York, http://www.textmap.com

Figure 7: The number of cities influenced by more than ten sources.

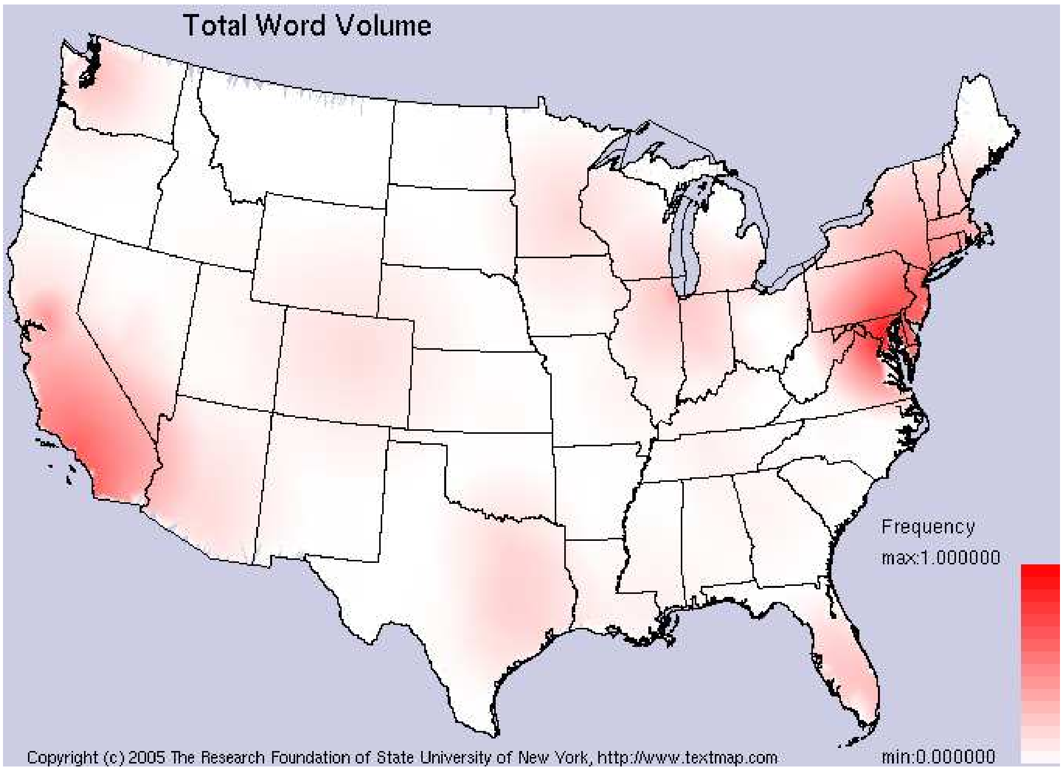

Total Word Volume
Frequency
max:1.000000
min:0.000000
Copyright (c) 2005 The Research Foundation of State University of New York, http://www.textmap.com

Figure 8: Media exposure by location as measured by volume of total words.

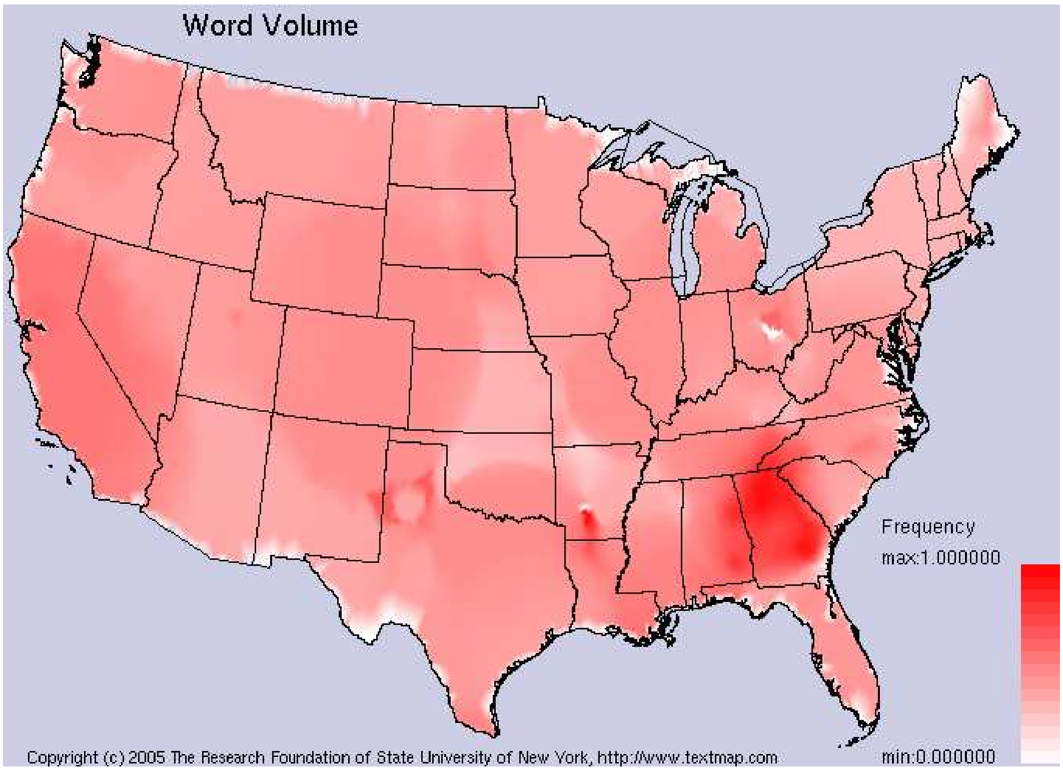

Word Volume
Frequency
max:1.000000
min:0.000000
Copyright (c) 2005 The Research Foundation of State University of New York, http://www.textmap.com

Figure 9: Average words per unit of influence.

2.3 Source-Influence Modeling

The *heat* any given entity e generates in a given location s is a function of the frequency of reference of e in each of the sources that have influence over s. The relative frequency of entity e in the source s_i is given by

$$\text{heat}(e,s) = \frac{\text{references}(e,s)}{\sum \text{references}(e_i,s)}. \tag{2}$$

Thus each heat value is from 0 to 1, giving the frequency of reference to the particular entity. A heat value of 0.05 implies that the entity is referenced once for every 20 entity references over the universe of all entities. The heat of an entity is a relative measure; even areas of high news volume may boast more absolute references to a given entity, but the popularity (heat) is determined by its *frequency* of reference.

The principle concern of heatmap construction is determining the ‘heat’ values, which is the estimate of the frequency of reference. The problem is trivial if we have just a single source (we just use the frequency reference of that source), but difficulties emerge with combining multiple sources. The sources should be combined in such a way that the more influential source has a greater say in determining the heat value. For instance, references in ‘The New York Times’ should have a greater effect than references in ‘The Ithaca Times’. However, we also need to consider the geographic distribution of the source. ‘The Ithaca Times’ will have greater influence over cities close to Ithaca, with decreasing influence the farther away a city is.

2.3.1 Estimating Source Influence

An analysis involving both ‘The New York Times’ and ‘The Ithaca Times’ must capture the *influence* relation between a news source and a location. This influence relation must take into account the distance of the source from the location, the circulation of the source, and other relevant features.

Constructing this influence function forms the basis of our heatmap model. The radius of influence of a news source depends on the circulation of the newspaper, and the population of nearby cities. Cities inside of this circle will be influenced by an amount depending on their distance from the center, with maximum influence at the newspaper's location. This is captured in the equation

$$influence(s,c) = \begin{cases} 0 & \text{if } dist(s,c) > r_{inf}(s) \\ f(distance(s,c)) \times \text{max-influence}(s) & \text{else} \end{cases}$$

where f is some decay function, in our case linear, and $r_{inf}(s)$ is the radius of influence of s. When f is linear, the influence of a newspaper can be thought of as a cone centered at the newspaper, with height the maximum-influence, and base the circle of influence. Cities outside of this circle will receive zero influence from the given source.

The maximum influence of a newspaper source is a combination of various circulation and ranking statistics of the source. In particular, we use a weighted combination of Alexa's reach per million web traffic analysis (www.alexa.org) and the weekday average circulation of the newspaper. Together they estimate the number of readers (both online and paper) of the newspaper. We estimate the online readership by multiplying the Alexa reach per million by the population of the U.S. in millions. We estimate the radius of influence supported by a given printed circulation through an estimate of the frequency with which people subscribe to newspapers. We model that 10% of the population covered by the radius of influence should equal the readership estimate.

2.3.2 Integrating Multiple Sources

Once the influence function of each source has been defined, the heat at a location can be calculated by a weighted average of the reference frequencies of the entity in each source, weighted by each source's influence on the location. (The reader

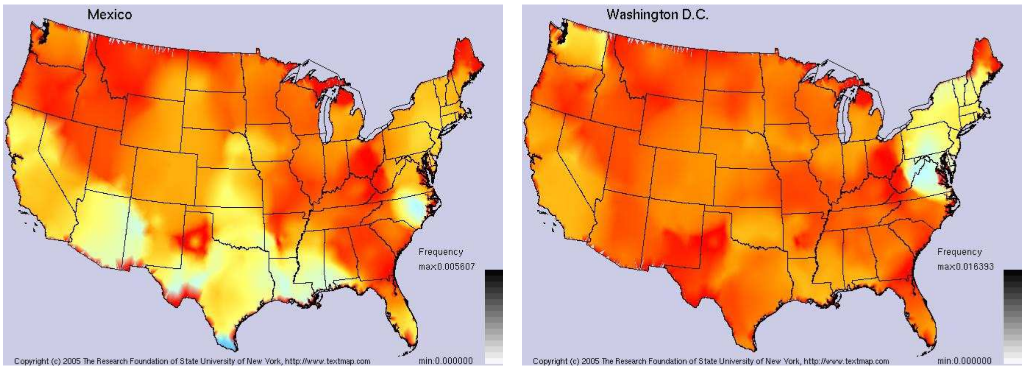

Mexico
Frequency
max0.005607
min:0.000000
Copyright (c) 2005 The Research Foundation of State University of New York, http://www.textmap.com
Washington D.C.
Frequency
max0.016393
min:0.000000
Copyright (c) 2005 The Research Foundation of State University of New York, http://www.textmap.com

Figure 10: Black body radiation heatmaps for Mexico and Washington, DC.

should be aware that we have overloaded the heat function. The final function we are trying to model is the heat function between cities and entities. We also make use of the heat between an entity and a source).

For our model, we discretize space into roughly 25,000 U.S. cities/towns of all sizes. For a given C of cities, S of news sources, and E of entities, we define the heat of an entity e at city c as:

$$\text{heat}(e,c) = \frac{\sum_{s \in S} \text{heat}(e,s) \times \text{influence}(s,c)}{\sum_{s \in S} \text{influence}(s,c)}. \tag{3}$$

2.4 Visualization Issues

To render our heatmaps, we use mesa/openGl graphics libraries. We know how to calculate the heat at cities, and openGl will interpolate heat between cities if given a polygon mesh. To get a polygon mesh from our set of cities, we used Jonathan Richard Shewchuk's C program ‘ triangle ’ [74], [6]. This creates a Delauney triangulation of the cities.

To get the coloring for the heatmaps, we set the scale such that the maximum heat value gets the highest red value, and the other values are scaled linearly. This

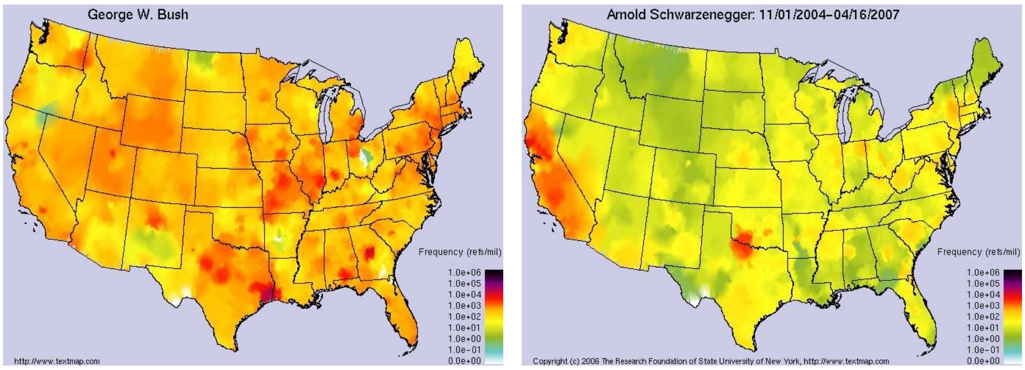


Figure 11: Comparing heatmaps for national-figure George Bush (l) and California Governor Arnold Schwarzenegger (r). Schwarzenegger displays a clear regional bias toward his home state.

makes 2 heatmaps incomparable, since they are on different scales. However, using absolute scales makes most heat effects unobservable, since very low values are imperceivable. This method ensures we will always have maximum contrast between the highest and lowest heat values.

Figure 10 show heatmaps using an alternate coloring, called black-body radiation. For this method, each heat value is scaled to Kelvin temperatures, and the color chosen is the wavelength of light a black body with that temperature emits. The black body coloring scheme makes it easier to observe the smaller fluctuations of heat while retaining the impact of the areas of large heat.

2.5 Identifying Significant Heatmaps

Once we have the ability to calculate a large number of entity heatmaps, we are faced with the problem of automatically screening which of these are interesting to look at, i.e. suggest significant geographic bias vs. entities of uniform national interest.

In this section we present methods for quantifying the geographic disparity of a heatmap. Geographic disparity does not have a precise definition, and proves difficult to quantify for a variety of reasons. Because the population density is highly non-uniform, the relative sizes of significant intensity proves misleading. For example. the area covered by “red” and “blue” states on the electoral map overstates the degree of red/Republican dominance due to their strength in the sparsely populated West. The news distributions resulting from our model prove to have large numbers of local optima. Do these represent distinct regional sources of elevated interest or are they modeling artifacts?

We consider two distinct classes of methods, based on variance analysis and connected component analysis respectively.

2.5.1 Variance Analysis

Statistical variance measures the deviation of data values from their mean. We would expect that heatmaps showing higher statistical variance of heat values more likely reflect regional biases, although similar variance measures can be derived from simple checkerboard patterns. That is, imagine a black/red checkerboard, versus a board with left half painted red, and the right half black. Both boards have the same frequency of red and black squares, but the checkerboard pattern has no spatial biases.

We define two scores reflecting this measure:

- *Variance* – Our heatmap construction gives us heat values for each of 25,374 cities. For this measure, we compute the variance of these 25,374 heat values.

- *Weighted Variance* – The variance measure will be biased by absolute heat. If we scale every value on a heatmap by some constant, the variance will also be scaled (by the square of the constant), although the underlying distribution is essentially the same. For this measure we divide the variance by the mean of the 25,374 values.

The first 2 scores we use are variance, and variance divided by mean. We expect a term with geographic bias to have 2 different distributions (1 for the area of interest, and 1 for the rest of the country) and thus have high variance. Variance alone however will usually weight a term that is overall more popular higher then a term that is overall less popular, but has more geographic bias. We correct for this in the second score by dividing by the mean.

2.5.2 Connected Component Analysis

Consider a heatmap with a very high geographic disparity. If we look at the ten cities with the highest heat values, we would expect them to be clustered close together. If the heatmap has no disparity, then the top 10 hot cities will probably be scattered all about the country. Suppose we continue to look at the top 20, 30, 40 cities and so on. Heatmaps with high disparity should have clusters of cities, while heatmaps with no disparity should remain scattered. This motivates the idea of the *connected component profile* for a heatmap.

Consider the graph G on the set of cities c induced by adding an edge between every pair of nearby cities, That is $(c_1, c_2) \in E$ if $distance(c_1, c_2) < d$. We can define a profile for a heatmap by counting the number of connected components in G only including cities above a certain heat value. The profiles for regional figure *Wayne Gretzky* and national term *America* are shown in Figure 12. We see the regional term has a long period of a small number of connected components, corresponding to the high concentration area.

We propose three different features of the profiles for scoring methods:

- *Largest Gap* – A large gap between a connected component change would suggest that the entity is drawn from two separate (national and local) probability distributions.

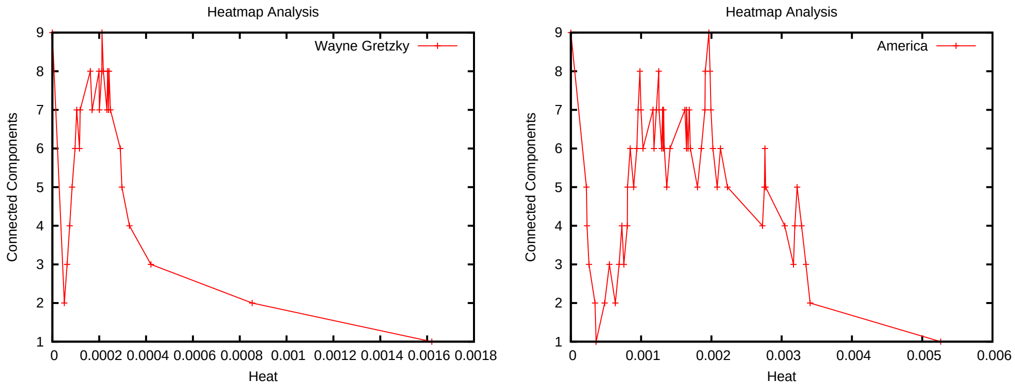


Figure 12: Connected Component Profiles for Wayne Gretzky and America.

- *Weighted Gap* – This method again uses the largest gap, but divides this number by the maximum heat. Thus terms that are generally popular won't be more heavily favored as they are in the first method.

- *Percentage Gap* – Finally we score based on the largest *percentage* heat change between component changes.

2.5.3 Results

To quantitatively evaluate our geographical bias measures, and avoid personal bias judgments as to the relative geographic disparity of heat maps, we conducted a large-scale experiment assessing how well these measures distinguish maps of regionally-biased entities from (1) maps of entities with presumed uniform national interest and (2) random maps generated under two different models. In the first model, the frequency of our imaginary entity in each news source is chosen from a uniform distribution, while in the second model it is chosen from a binomial distribution. Heatmaps generated under these models are shown in Figure 13. Uniform model has greater local variances, while the binomial distribution is more globally smooth because in a binomial distribution, values are centered around the mean.

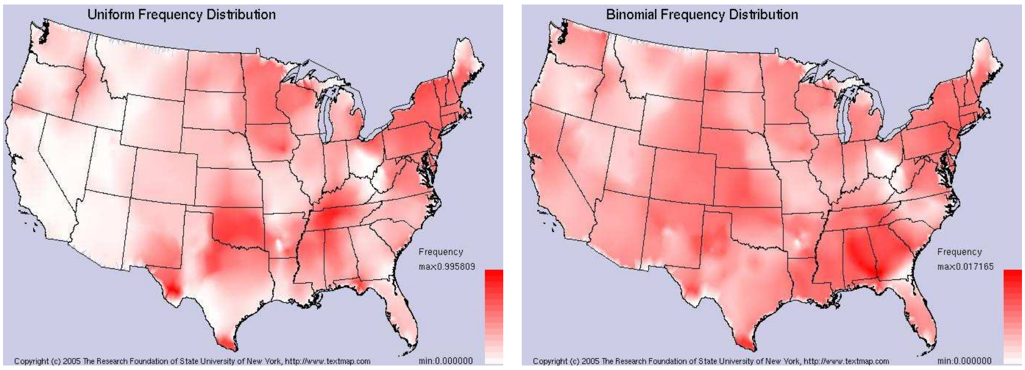

Uniform Frequency Distribution
Frequency
max:0.995809
min:0.000000
Copyright (c) 2005 The Research Foundation of State University of New York, http://www.textmap.com
Binomial Frequency Distribution
Frequency
max:0.017165
min:0.000000
Copyright (c) 2005 The Research Foundation of State University of New York, http://www.textmap.com

Figure 13: Random Heatmaps. The frequency of this imaginary entity in each source is given by a random uniform distribution, or a binomial distribution

We made two sets of real entity heatmaps for our experiments. Entities likely to be geographically biased include United States cities and local sports teams. Entities likely to have little bias include foreign cities, country names, national political figures, and entertainment terms. In total, we constructed 128 un-biased heatmaps, and 400 biased heatmaps. We also made 200 heatmaps each for the uniform and binomial distributions.

The results of our scoring method are presented in Tables 5 and 6. For all five methods we calculate the mean, median, max, and min on each set for the raw score, and for the ranks. From these experiments, we can conclude that the *weighted gap method* has the best results, since it in general scored biased maps higher than un-biased maps, and both higher then random maps. However, even though the other four scoring methods scored random maps above our biased heatmaps, they score biased higher then the real un-biased maps. Thus for actual maps, each of the scoring methods has some significance.

We are interested in distinguishing geographic bias among real world maps. Figure 14 shows the *Receiver Operating Curve* (ROC) for classifying the real world maps. we see that each scoring method is substantially above the 45-degree line

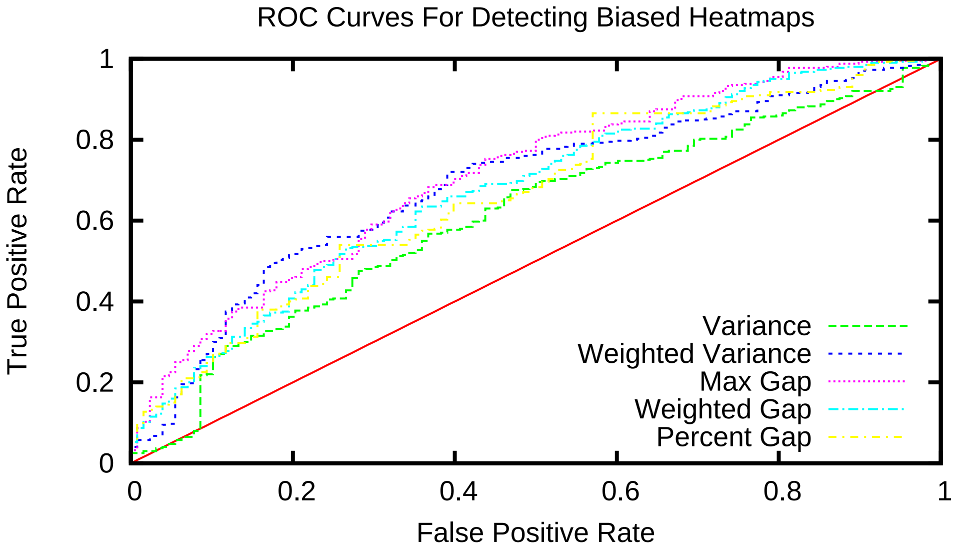


Figure 14: ROC Curves For Heatmap Classification

that indicates random guessing in an ROC curve. This, coupled with the fact that the methods are not perfectly correlated with each other (see Table 7) lead us to believe that a fusion method should do even better (A fusion method cannot improve when underlying different scoring methods are highly correlated, thus effectively all saying the same thing).

2.6 Conclusions

In this chapter we have presented a way of spatially modeling news references, based on references in newspapers. These visualizations, called ‘ heatmaps’, show the geographic popularity of an entity, and possible geographic biases. To help find heatmaps that display geographic bias, we have developed methods of scoring the maps, based on a spatial ‘connected components’ feature.

Weighted Gap	mean	min	median	max
biased	0.519	0.080	0.494	0.996
unbiased	0.367	0.053	0.323	0.947
uniform	0.080	0.035	0.070	0.208
binomial	0.098	0.037	0.088	0.254
Percent Gap	mean	min	median	max
biased	6.23	0.300	2.00	243.3
unbiased	2.08	0.273	1.25	18.0
uniform	7.55	0.294	1.68	967.6
binomial	2.53	0.411	2.49	4.55
Max Gap	mean	min	median	max
biased	1.66e-3	7.00e-6	5.12e-4	2.80e-2
unbiased	5.10e-4	6.00e-6	1.36e-4	9.34e-3
uniform	7.47e-2	6.55e-2	3.33e-2	2.05e-1
binomial	1.57e-3	5.19e4	1.33e-3	5.00e-3
Weighted Variance	mean	min	median	max
biased	6.60e-4	7.53e-6	2.30e-4	1.31e-2
unbiased	2.61e-4	3.50e-6	7.57e-5	3.23e-3
uniform	9.70e-2	5.58e-2	9.37e-2	1.58e-1
binomial	1.18e-3	7.56e-4	1.16e-3	1.94e-3
Variance	mean	min	median	max
biased	1.76e-7	1.80e-11	9.79e-9	9.14e-6
unbiased	9.35e-8	1.32e-12	3.36e-9	1.92e-6
uniform	3.36e-2	1.59e-2	3.22e-2	5.50e-2
binomial	8.13e-6	4.94e-6	8.00e-6	1.46e-5

Table 5: Performance of our 5 different scoring methods.

Weighted Gap	mean	min	median	max
biased	246	0	232	719
unbiased	354	21	352	867
uniform	747	450	770	927
binomial	685	408	690	925
Percent Gap	mean	min	median	max
biased	437	1	459	922
unbiased	589	25	660	927
uniform	506	0	553	925
binomial	395	134	367	911
Max Gap	mean	min	median	max
biased	576	200	632	926
unbiased	728	213	784	927
uniform	100	0	100	199
binomial	432	238	439	630
Weighted Variance	mean	min	median	max
biased	609	200	623	926
unbiased	728	216	769	927
uniform	100	0	100	199
binomial	367	238	367	498
Variance	mean	min	median	max
biased	649	245	641	926
unbiased	707	410	742	927
uniform	100	0	100	199
binomial	301	200	302	402

Table 6: Rank Performance of five different scoring methods. All 927 test maps are ranked according to each method. For each of the 4 categories of maps, the mean, min, median, and max ranks are shown. For example, the Weighted Gap score ranks biased maps a median of 232. This means half of the biased maps are ranked higher than 232. The highest ranked biased map is 0, the best rank possible.

	data	Wei. Gap	Max Gap	% Gap	Variance	Wei. Var.
Weighted Gap	bias	1	0.708	0.431	0.376	0.651
	unbias	1	0.722	0.856	0.621	0.738
Max Gap	bias	0.708	1	0.883	0.865	0.985
	unbias	0.722	1	0.966	0.910	0.931
Percent Gap	bias	0.431	0.883	1	0.984	0.928
	unbias	0.856	0.966	1	0.877	0.934
Variance	bias	0.376	0.865	0.984	1	0.919
	unbias	0.621	0.910	0.877	1	0.980
Weighted Var.	bias	0.651	0.984	0.928	0.919	1
	unbias	0.738	0.931	0.934	0.980	1

Table 7: Pearson Correlation Coefficients for five scoring methods.

Chapter 3

Identifying Co-referential Names Across Large Corpora [1]

3.1 Introduction

A single logical entity can be referred to by several different names over a large text corpus. For example, *George Bush* is often referred to as *Bush*, *President Bush*, *George W. Bush*, or *"W"*, even among polite company. However, morphologically-similar names like *George H.W. Bush* can refer to different entities. Accurately identifying the members of the *co-reference set* for a given entity is an important problem in text mining and natural language processing.

Our interest in identifying such co-reference sets arises in the context of our system *Lydia* [48, 47, 42, 53], which seeks to build a relational model of people, places, and things through natural language processing of news sources. Indeed, we encourage the reader to visit our website (*http://www.textmap.com*) to study our analysis of recent news obtained from over 500 daily online news sources. In particular, we display the members of each of the 100,000 co-reference sets we reconstruct daily (on a single commodity computer) from the roughly 150,000 entity

[1] **This chapter is an extended version of [49].**

names we currently track.

Our algorithm for identifying co-referring entity sets accurately and efficiently on a large scale involves optimizing our algorithm's three steps:

1. *Morphological Similarity* – The scale of our problem makes it infeasible to explicitly compare each pair of names for possible co-reference. First, we narrow our search space by identifying candidate pairs for analysis on a strictly syntactic basis via morphological hashing techniques.

2. *Contextual Similarity* – Next, we determine how similar a pair of names is based on the contexts in which they are used. The scale of our problem makes it infeasible to explicitly analyze all text references associated with each pair of candidate names. Instead, we propose methods using co-occurrence analysis to *other* entities to determine the probability that two entities are co-referent by context.

3. *Evidence Combination and Clustering* – Finally, we combine our measures of contextual and morphological similarity in order to cluster the names. The problem of clustering names is complicated by the vast difference in the number of references between popular and infrequently-used names. The strength of our contextual evidence is thus substantially weaker for unpopular names. We propose and evaluate methods for dealing with this problem.

Our problem is different from traditional cross-document co-reference analysis (see Section 3.2.1). In that problem, there is a set of documents that all mention the *same* name and the difficulty is clustering the documents into sets that mention the same entity. In our problem, there is a set of documents that mention many entities, each possibly with multiple names, and we want to cluster the names. This difference, combined with our need to manage the daily flow and scale of the news, presented challenges that separate us from existing techniques in the following ways: (1) the use of entity co-occurrence lists as the sole feature for contextual analysis,

(2) our high-speed dimension reduction techniques (based on *k*-means clustering and graph partitioning algorithms) to improve the quality of our contextual analysis and the efficiency of our algorithms, (3) our use of morphological similarity hashing techniques to avoid the need for pairwise-similarity testing of all name pairs, and (4) our use of *variable precision phonetic hashing* in order to tune the performance of our morphological similarity hashing.

The rest of this chapter is organized as follows. Section 3.2 surveys previous work on this and other problems. Section 3.3 discusses notions of morphological similarity, while Section 3.4 shows how we compute the probability that two names are co-referential from their respective co-occurrence lists. Section 3.5 discusses issues that arise in clustering. Experimental results are given in Section 3.6. We present our conclusions in Section 3.7.

3.2 Related Work

The problem of identifying co-reference sets has been widely studied in a variety of different contexts. In this section, we survey related work. In particular, two papers discussing projects with similar goals but different techniques are discussed below.

We now describe work on three related problems in the subsections below, namely, cross-document and in-document co-reference resolution in natural language processing and record linkage in databases.

Novak, Raghaven, and Tomkins [62] sought to find all of the aliases that correspond to the same person on Internet message boards. They cluster these aliases into groups based on extracted feature vectors. Their feature set includes general word usage, misspellings, punctuation usage, emoticons, and function word usage. Information retrieval similarity and KL-divergence are compared as potential distance measures over the feature space with KL-divergence working best. Agglomerative clustering is used to form the final clusters.

Li, Morie, and Roth [46] study a similar problem, which they call the robust reading problem. They present an unsupervised learning algorithm for this problem. First, they develop a probabilistic model for how entities and mentions are distributed throughout a corpus. Then they provide different relaxations of the model with corresponding inference algorithms. Finally they present an EM algorithm to learn these models in an unsupervised way and estimate the model parameters.

3.2.1 Cross Document Co-reference Resolution

The problem of cross-document co-reference has been examined fairly extensively.

Bagga and Baldwin [4] present an algorithm which extracts each sentence in each document that contains an ambiguous name and forms a summary of the document with respect to the entity. They then use the vector space model to compute the similarity of two such summaries. If the similarity of the two documents is above a threshold, then they consider the two documents to be referring the same person. They concluded that good results could be achieved by looking at the context surrounding the occurrences of the name and comparing documents using techniques from information retrieval.

Mann and Yarowsky [52] present a partially supervised algorithm for this problem. The algorithm takes as input either a small set of seed tuples for each of a small set of personal attributes from which it generates extraction patterns or a set of hand-crafted extractions for each of the personal attributes. Next, it uses these values along with other contextual clues as the feature vector for each document before using bottom-up centroid agglomerative clustering.

Gooi and Allan [30] study statistical techniques for cross-document co-reference resolution. Like Bagga and Baldwin, they use snippets of text around each mention of the ambiguous name. They compare *agglomerative clustering*, repeatedly merging the closest pair of clusters, with *incremental clustering*, either adding each point to an existing cluster or starting a new singleton cluster. They also compare KL-divergence as a distance function with cosine similarity. They conclude that agglomerative clustering performs better than incremental clustering, however incremental clustering is much more time efficient. They also conclude that cosine similarity performs better using KL-divergence.

3.2.2 Within-document co-reference resolution

The natural language processing community has extensively studied the problem of within-document co-reference resolution, finding chains of noun phrases that refer to the same things. For example, in a news article, *Dick Cheney* may later be referred to as *Vice President*, *he*, or *Mr. Cheney*. Published solutions all heavily rely on information that is only relevant within a document, such as the distance between the two phrases.

We note we perform such an analysis as part of our news NLP pipeline [48], but we do not discuss it further here.

Ng and Cardie [61] present a supervised machine learning-based algorithm for within-document co-reference resolution. They use a decision tree classifier to classify each pair of noun phrases in a document as either co-referring or not and a clustering algorithm to resolve conflicting classifications. They experiment with different feature sets, clustering algorithms, and training set selection algorithms. They conclude that linking a proper noun phrase to its most probable previously occurring co-referring phrase is a better way of clustering, that a training set selection algorithm that is designed for this clustering algorithm is superior, and while adding features can be helpful, too many can degrade performance.

Bean and Riloff [9] present an unsupervised approach to co-reference resolution that uses contextual role knowledge to determine if two noun phrases co-refer. First they identify easy-to-resolve co-referring pairs and use them as training data. Information extraction patterns are then used to generate information about the role each noun phrase plays in the text. The information extracted from the training data is used to help resolve the other pairs in the corpus. They show that this phase increases recall substantially with just a slight decrease in precision.

Luo, Ittycheriah, Jing, Kambhatla, and Roukos [50] present a supervised machine learning approach to co-reference resolution. They represent an instance of the co-reference resolution problem in terms of a *Bell Tree* where each leaf in the tree represents one possible solution and all possible solutions are given by a leaf in the tree, and cast the problem as trying to find the best root to leaf path in this tree. They train a maximum entropy model to assign probabilities to each edge in the tree and search the tree for the most probable leaf. Their feature set includes lexical features (same spelling, substring), distance features, and syntactic features (part-of-speech).

3.2.3 Record Linkage

Our co-reference set identification problem is similar to the record linkage problem from data mining. The problem arises when there is no shared, error-free key field to join on across databases. Consider two tables containing information about people from two different databases. Even if both databases used the person's name and address as the primary key, conventions concerning abbreviations and word usage may differ, and typos and misspellings may appear in either field. The goal is to identify which records correspond to the same entities.

Cochinwala, Dalal, Elmagarmid, and Verykios [16] present a survey of current techniques in record linkage. They identify three phases to the record linkage problem. First, the data is preprocessed. For example, a name field may be parsed to find the first and last names it contains. Then the database must be efficiently searched for potentially matching record pairs. Finally, potentially matching records are compared to determine whether or not to link them. Cochinwala et al. pose the record linkage problem as a classification problem and present a supervised machine learning-based algorithm to classify record pairs as either "should be linked" or “should not be linked”.

Hernandez and Stolfo [34] present two different techniques for large databases. The first approach sorts the data on some key and only considers two records for a merge if they are in a small neighborhood of each other. The second clusters the records in such a way that two records will be in the same cluster if they are potentially referring to the same entity. Finally, they propose taking the transitive closure of independent runs of the above algorithms, with independent key fields, as the final merge. They show that this multi-pass algorithm is superior to all the other algorithms that they consider.

Cohen and Richman [18] consider two problems: (1) taking in a pair of lists of names and determining which pairs of names in the different lists are the same and (2) taking in a single list of names and partitioning them into clusters that refer to the same entity. They propose adaptive learning-based matching and clustering methods to solve either of these problems. Their feature vector includes whether one string is a substring of the other and the edit distance between the two strings.

Bilenko and Mooney [10] propose two distance measures between strings. The first is a variation of edit distance. They view the edit distance between two strings as a stochastic process with a probability for each substitution, insertion and deletion. Then any two strings will have a probability associated with their editing. They train the parameters of this stochastic model on a set of strings so as to maximize the probability of the training pairs. The second measure is a variation on vector space similarity. Instead of using the traditional *term frequency-inverse document frequency* (tf-idf) weighting scheme, they propose using SVMs to learn weights for each term. They conclude that their distance metrics are very effective at solving this problem in the domain in which they are trained.

3.2.4 Gene Name Normalization

Noted biologist David Botstein likes to say that “biologists would rather share a toothbrush than a gene’s name”. Effectively mining the biomedical literature requires normalizing gene names by identifying appropriate synonym sets. Representative work includes [17, 84].

Cohen et al. [17] propose sets of gene-name transformations that yield synonyms, including removing the first character, the last character, the first word, the last word, mapping vowel sequences to a constant string, replacement of hyphens with spaces, normalizing capitalization, and the removal of parenthesized material. They conclude that these heuristics are useful in finding such synonym sets.

Yu and Agichtein [84] present a system to identify synonymous gene name pairs in a corpus. They use a combination of partially-supervised, supervised, and hand-crafted systems. The partially-supervised method is an iterative procedure that takes as input a small set of seed pairs of synonymous genes and uses them to identify a set of extraction patterns. Their supervised method uses an SVM to classify the contexts of a pair of genes as either synonymous or non-synonymous. The hand-crafted system uses a set of rules that were designed by domain experts to identify synonymous genes. Finally, a total confidence is calculated from the confidences of the three methods. They conclude that a combination of all three systems out-performs any of the single systems in isolation.

3.3 Morphological Similarity

With hundreds of thousands of names occurring in a large corpus, it is intractable to compare every pair as potentially co-referential. Further, most of these comparisons are clearly spurious, and thus would increase the possibility of false positives. We propose that most pairs of co-referential names result from the following set of morphological transformations:

- *Subsequence Similarity* – Taking a string subsequence of a name is one way of generating aliases of that name. For example, *Ford Motor Co.* is often referred to as *Ford* and *George W. Bush* is also called *George Bush*. To identify these pairs, we examine all 2^n possible string subsequences of each n-word name, hashing the name on each of its subsequences. Note that n, the number of words in a name, is bounded by about 10. Any subsequence matching another name implies potential morphological compatibility.

- *Pronunciation Similarity* – The Metaphone [64] algorithm returns a hash code of a word such that two words share the same hash code if they have similar English pronunciations. Here we say that two names are morphologically-compatible if they have the same Metaphone hash code. Metaphone is useful in identifying different spellings of foreign language names (e.g. *Victor Yanukovich* and *Viktor Yanukovych*) as possibly co-referential. In Section 3.3.1, we detail our methods for tuning the performance of this aspect of morphological similarity using variable precision phonetic hashing.

- *Stemming* – We use a Porter stemmer [65] to stem each word of each name and use the stem as a hash code for each name. A hash code collision means that two names have morphologically-compatible names. Stemming can be used to identify pairs like *New York Yankee* and *New York Yankees*.

- *Abbreviations* – If one name is an abbreviation of another, then we say that they are morphologically compatible. For example JFK and John F. Kennedy are both co-referential with John Fitzgerald Kennedy. To find all names that are abbreviations of a name, we check if any of the 2^n possible abbreviations of the name's n-words are also in our corpus.

We observe that there is a notion of degree of morphological similarity. For example, *George Bush* is more likely to be co-referential with *George W. Bush* than *U.S.* is with *Assistant U.S. Attorney Richard Convertino*. For each of our morphological transformations we have a different measure of the degree of similarity. For example, for pronunciation similarity, we model the generation of aliases as a stochastic "typing" process where the probability of a mis-type is a constant. Then we compute the probability that one name was "typed-in" when the other was intended.

3.3.1 Variable Precision Phonetic Hashing

Several phonetic hashing schemes have been developed to work well on a specific data set or for specific performance levels [64, 77, 11]. No methods exist that allow the hashing scheme to be parameterized to give different precision/recall tradeoffs. In this section we investigate phonetic hashing schemes that have an adjustable parameter giving a range of operating points with different precision/recall tradeoffs.

Given a query string, we envision a sequence of transformations from the query string to an empty or null string, where each transformation is a new version of the string that has had some tokenization or weakening applied to it. We can model the space of transformations on the universe of strings as a graph. For example the name 'Wright', is shown in Figure 15, with a possible transformation sequence.

The weight of each change is determined by how drastic it is. So the distance from 'Writ' to 'Rit' should be relatively small when compared with the distance from 'Rt' to 'R'. This tokenization path gives us different versions of the query name to use in different tolerances of the hashing function. We also see that the path for the name 'Rite' eventually joins the path of 'Wright'. The name 'Reston' similarly joins the path, but lower down; suggesting that 'Rite' and 'Wright' are closer to each other then to 'Reston'.

A particular tokenizer in this scheme specifies a set of n-gram substitution rules, along with weights for the rules. The rules are applied in a lowest cost rule first order. An example set of rules that could have generated Figure 15 is shown below.

$$\begin{array}{lcl} \text{Reston} & \longrightarrow & \text{Restn} \rightsquigarrow \text{Rst} \searrow \\ \text{Wright} & \longrightarrow & \text{Writ} \rightarrow \text{Rit} \rightarrow \text{Rt} \rightarrow \text{R} \rightarrow \\ & & \text{Rite} \nearrow \end{array}$$

Figure 15: Tokenization Path of the Name ’Wright’.

This rule set says the cheapest rule is substituting a ‘t’ for ‘ght’. The next cheapest is substituting an ‘r’ for ‘wr’ only if at the start of a query. Finally there are two deletion rules. The vowel deletion is considered less destructive, and is given a lower weight then the consonant deletion.

- ght → t;0.2
- _wr → r;0.3
- (a|e|i|o|u) → ;1
- (t|r) → ;5

The tokenizer also has a position vector. This vector weights the rules on their position in the string. This can be used to make rules applied at the beginning of a string more costly than rules applied toward the end of a string. Finally, ties are broken by applying the rule that changes the rightmost portion of the string.

To complete the definition of the hash function we must specify how to select the point on the tokenization path to operate at. Among the many candidates for these scoring methods, our experimentation showed that selecting the code that is a fixed distance from the null string works best.

- Distance From Query - We generate the string which lies an absolute distance from the query name. A potential problem is that the strings ‘Wright’ and ‘Rite’ would have different distances to ‘Rit’; and thus would not hash to the same consensus string.

- Distance From Null String - We select the point a certain absolute distance from the null string. This method has the property that any two strings that get hashed together stay together as we decrease the distance from null, and stay apart as we increase the distance. Since ‘Rit’ has only one distance to null, ‘Wright’ and ‘Rite’ would be hashed together if the distance was low enough to stop at or before ‘Rit’.

- Percentage Distance From Null String - We select a point a percentage distance from the null string. This method may at first seem reasonable, since more complex strings will have more complex hashed strings. However, we do not get the monotonicity properties we got with ‘Distance From Null String’.

- To the topmost code of a certain length - We select the code closest to the query of a certain string length. This method has the desired monotonicity properties; and it also makes the hashing scheme less sensitive to rule weights, since we always go to a certain length code word.

- To the lowest code of certain length - Same as the above scheme, but we take the code closest to Null.

Table 8 shows how we can vary the precision and recall of our hashing algorithm to get different tradeoffs. For a hand-created set of names extracted from our test set (see Section 3.6), we measured the precision and recall of our hashing algorithm at a range of its operating points. For comparison, we also show the precision and recall of three other phonetic hashing algorithms. It shows how we can use our algorithm to dial in the precision and recall of our notion of pronunciation similarity.

Our initial attempts at a hash function did not focus on developing the rule set. Rather hand coded rule sets were used that attempted to emulate an already well known tokenization method (Soundex, NYSIIS, Caverphone, Metaphone). For

Code Weight	Precision	Recall
0	0.002	1
120	0.150	0.909
121	0.139	0.818
141	0.157	0.727
146	0.293	0.636
167	0.360	0.545
172	0.442	0.454
187	0.662	0.363
229	1.000	0.090
Metaphone	0.715	0.732
Soundex	0.468	0.797
NYSIIS	0.814	0.672

Table 8: Precision and recall for our variable precision phonetic hashing and fixed precision hashing

instance, Soundex treats the letters ’b’,’f’,’p’, and ’v’ as the same token; So the Soundex-like rule set contained the rule

- (b|f|p|v) → 1;2

We also see with these rules that the alphabet for the rules may be different from the alphabet for the names (i.e., we have the character ’1’).

3.4 Contextual Similarity

Our mental model of where an entity fits into the world depends largely upon how it relates to other entities.

We predict that the co-occurrences associated with two co-referential names (say *Martin Luther King* and *MLK*) would be far more similar than those of morphologically-similar but not co-referential pairs (say *Martin Luther King* and

Martin Luther). Thus we use the vector of co-occurrence counts for each name as our feature space for contextual similarity.

We identified two primary technical issues in determining contextual similarity using this feature space: (1) dimension reduction and (2) functions for computing the similarity of two co-occurrence lists. Each of these will be described in the following subsections.

3.4.1 Dimension Reduction

In the experimental run of $88,097$ newspaper days of text we used throughout our experiments (details in Section 3.6), we encountered $174,191$ different name strings that occurred more than 5 times. This implies an extremely sparse, high-dimensional feature space – large because each additional entity name represents a new dimension, and sparse because a typical entity only interacts with a few hundred or so other entities even in a large text corpus.

Our experiments show that simple techniques which hunted for identical terms among the 100 or so most significant entries on each co-occurrence list failed, because the most significantly co-occurring terms for an name were highly unstable, particularly for low frequency names. Much more consistent were "themes" of co-occurring terms. In other words, while the most frequent associations of *George Bush* and *"W"* might have relatively few names in common, both will be strongly associated with "Republican" and "Texas" themes.

Dimension-reduction techniques provide a way to capture such themes, and can improve both recognition accuracy and the computational efficiency of co-reference set construction. We examined two different dimension-reduction techniques based on creating crude clusters of names, then project our co-occurrence lists onto this smaller space.

- *K-means clustering* – This widely-used clustering method is simple and performs well in practice. Beginning with *k* randomly selected names as initial cluster centroids, we assign each name to its closest centroid (using cosine similarity of co-occurrence lists) and recompute centroids. After repeating for a given number of iterations (5, in our case) we assign each name to its closest centroid and take this as our final clustering.

- *Graph partitioning* – The problem of graph partitioning seeks to partition the vertices of a graph into a small number of large components by cutting a small number of edges. Such components in a graph of co-occurrences should correspond to "themes", subsets of terms which more strongly associate with themselves than the world at large. Thus we propose graph partitioning as a potential dimension reduction technique for such relational data – the names in each component will collapse to a single dimension.

  Although graph partitioning is NP-complete [25], reasonable heuristics exist. In particular, we used METIS[39], a well-known program for efficiently partitioning large weighted graphs into k high-weight subgraphs, with k being a user-specified parameter. Our graph contains a node for every name and an edge between every pair of nodes (x,y) if they co-occur with each other at least once. The weight assigned to each edge is the cosine similarity between the co-occurrence lists of x and y.

3.4.2 Measuring Contextual Similarity

Given two names, with their co-occurrence lists projected onto our reduced dimensional space, we now want a measure of how similar they are. We consider two different approaches: (1) they can be viewed as probability distributions and be compared by *KL-divergence* or (2) they can be viewed as vectors and compared by the cosine of the angle between the vectors. We detail each of these potential measures here.

3.4.2.1 KL-Divergence

The KL-Divergence is an information theoretic measure of the dissimilarity between two probability distributions. Given two distributions, the KL-Divergence of them is defined by

$$KL(p,q) = \sum_{x \in X} p(x) \log \frac{p(x)}{q(x)}$$

To use this measure, we turn each co-occurrence list into a probability distribution for each name *i*,

$$\hat{p}_i(j) = \frac{\text{number of co-occurences between i and j}}{\text{total number of co-occurrences for i}}$$

As a discounting method for probability-0 pairs, we do linear smoothing of all probabilities with the background distribution setting

$$p_i(j) = \alpha \hat{p}_i(j) + (1-\alpha) bg(j)$$

where

$$bg(j) = \frac{\text{total occurrences of names in cluster j}}{\text{total number of entity occurrences in corpus}}$$

3.4.2.2 Cosine Similarity

A standard way of comparing contexts views the two contexts as vectors in a high dimensional space and computes the cosine of the angle between them. [4] proposed this technique for the similar problem of personal name disambiguation. We use the *term frequency-inverse document frequency* of each vector position, i.e. we weight each term in the vector by the inverse of the number of occurrences it has in the corpus. Letting *N* be the number of sentences in the corpus, our score is

$$d(x,y) = \sum_{i=0}^{k} jp_x^*(i) \cdot jp_y^*(i)$$

where $jp_x(i)$ the number of co-occurrences between i and x, weighted by $\log(\frac{N}{\text{number of occurrences of i}})$, and

$$jp_x^*(i) = \frac{jp_x(i)}{\|jp_x\|}$$

3.5 Issues in Clustering

Now that we know which pairs of names are morphologically-similar and their degrees of morphological and contextual similarity, we need: (1) a way of combining morphological and contextual similarities into a single probability that two names are co-referential and (2) a method to cluster names into co-reference sets. We discuss each problem below.

3.5.1 Combining Notions of Similarity

For each pair of morphologically-related names, we have measures of their morphological and contextual similarities. We need a way to combine them into a meaningful probability that the two names are co-referential.

For each measure of contextual similarity and for edit distance, we computed the precision curve on our experimental corpus (see Section 3.6). Since the precision at a measure of similarity is the probability that a pair from the test set with this amount of similarity were co-referential, we use these curves to turn each of our notions of similarity into a probability estimate. Assuming that these two probabilities are independent, we now can estimate the probability that these two names are co-referential by multiplying the probabilities given by their morphological and contextual similarities. This gives us a single probability estimate that we can use as a similarity score in the clustering algorithms.

3.5.2 Clustering Algorithms

Once we have probabilities associated with each pair of morphologically related names, we need to group them into co-reference sets. Because our system must be able to handle large numbers of names, we must be careful what kind of clustering algorithm we choose. We experimented with two algorithms:

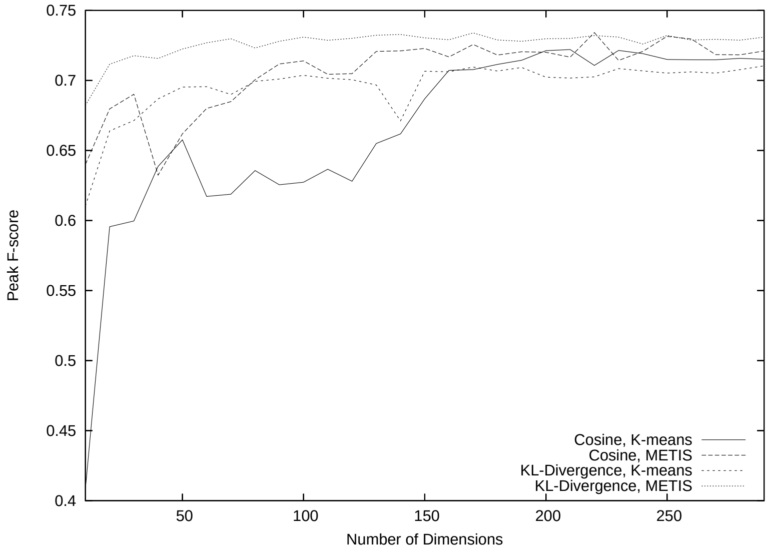


Figure 16: Number of Clusters vs. Peak F-score for our dimension reduction algorithms and distance measures

- *Single link* – Here we merge the clusters that two names are in if the probability that they are co-referential is above a threshold.
- *Nearest neighbor* – Each name is linked with its nearest neighbor if the probability that they were co-referential is above a threshold.
- *Average link* – Our algorithm merges two clusters if the weighted average probability between names in each of the clusters is above a threshold.

3.6 Experimental Results

In order to optimize various parameters, decide which methods work best, and verify our techniques, we ran a set of experiments against the same test set that was used to produce the precision curves used in section 3.5.1. Each of these experiments is described below.

All of the experiments in this chapter where conducted on a test set of $88,097$ newspaper-days worth of text, partitioned among 604 distinct publications. These were taken from spidering that was performed between April 11, 2005 and November 5, 2005. We used a hand-crafted set of roughly 320 co-reference sets from the entities in this corpus.

In Section 3.6.1, precision is given by $\frac{tp}{tp+fp}$, recall by $\frac{tp}{tp+fn}$, and f-score by $\frac{1}{\alpha\frac{1}{P}+(1-\alpha)\frac{1}{R}}$ where $tp =$ true positives, $fp =$ false positives, and $fn =$ false negatives. In Section 3.6.3 these measures are given by the B-cubed algorithm introduced in [4]. For each name

$$\text{Precision} = \frac{||\text{intersection of propsed set and true set}||}{||\text{proposed set}||}$$

$$\text{Recall} = \frac{||\text{intersection of proposed set and true set}||}{||\text{true set}||}$$

and overall precision and recall are the averages of these values.

3.6.1 Optimizing Contextual Similarity Measure

Optimizing our contextual similarity phase involves the proper choice of (1) dimension reduction algorithm, (2) number of dimensions, and (3) contextual similarity measure. For both of the dimension reduction algorithms (*k*-means, METIS) and both of the distance measures (KL-Divergence, Cosine similarity), we recorded the peak F-score as a function of number of dimensions from 10 to 290.

Figures 16 shows this plot. It shows that while the peak performance of all four combinations is comparable, KL-Divergence with METIS dimension reduction is to be the most robust to changes in *k*. For the rest of the analysis in this chapter, we used KL-divergence, METIS dimension reduction, and 150 dimensions.

3.6.2 Morphological Similarity

We hypothesized that using a value different than 1 for the cost of substituting a vowel for a vowel might lead to more meaningful notions of edit distance similarity. In order to find the best notion of edit distance, we looked at the peak F-score obtained at values of the parameter from 0 to 3 in increments of .1. Our results show that the value of the vowel parameter do not affect either the subsequence morphological similarity class nor the stemming class of morphological similarity. However, for the Metaphone class of morphological similarity, it is best to use a vowel parameter of about 2.

3.6.3 Clustering Methods

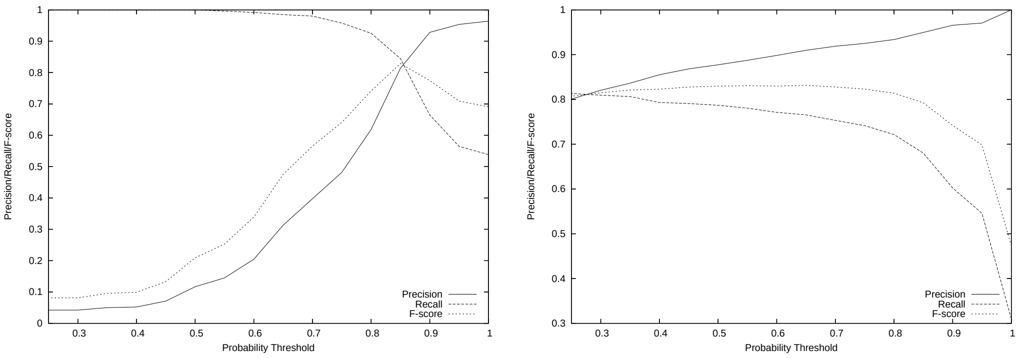
The first clustering algorithm that we tried was simple single link clustering. Figure 17(a) shows that it has decent peak performance, but is not very robust to the setting of the threshold. Further, manual evaluation of the clusters that are produced shows that it tends to create very long clusters, putting many things into the same cluster that should not even be considered. For example, the sequence *George Bush* → *Bush* → *Bush-Cheney* → *Cheney* → *Dick Cheney* leads to *George Bush* and *Dick Cheney* being called co-referential.

The next simple clustering algorithm was nearest neighbor. While this is an improvement over single link clustering, it still does not perform very well.

The next clustering algorithm that we tried was weighted-average link. Figure 17(b) shows that this has slightly better peak performance than single-link clustering, and is much more robust in the setting of the threshold.

3.7 Conclusion

In this chapter we presented an algorithm to find sets of co-referential names. We introduced the idea of morphological similarity, the notion that two names are potentially co-referential based on the text that comprises the name. Then we discussed the issues surrounding computing the contextual similarity of two names and gave two different measures. Clustering names given their morphological and contextual similarities was discussed and we presented experimental results for our



(a) Single-link clustering

(b) Average-link clustering

Figure 17: Threshold vs. Precision, Recall, and F-score for our clustering algorithms

system.

Chapter 4

Improving Usability Through Password-Corrective Hashing [1]

4.1 Introduction

The design of any password authentication system requires a tradeoff between security and usability. For example, mandating longer passwords in a system improves security while complicating the user's ability to remember passwords and enter them correctly.

The data entry problem is by no means trivial. Empirical studies of typing accuracy [31, 51, 63] suggest that typists make data-entry errors roughly once every 30 keystrokes on typical English text. Assuming ten-character passwords, this implies that roughly one out of every three login attempts by legitimate users fail due to data entry errors. Indeed, typing error rates are presumably even higher on the cryptic, case-sensitive, punctuation-intensive strings recommended for high-security passwords. An inspiration for this chapter was the painful memory of repeatedly typing a 128-bit wireless encryption WEP key (consisting of 26 hexadecimal-characters) until achieving the required perfect fidelity. Finally, users juggling passwords for

[1] **This chapter is an extended version of [54].**

several different systems can easily confuse typing errors with recalling the wrong password [70]. Subsequent cycling through passwords on other systems may result in users getting locked out, with a subsequent need for a password reset.

In this chapter, we propose a way to increase the usability of password authentication systems by correcting for two common classes of data entry errors, namely transposition and substitution errors. Transpositions and substitutions can arise from physical input errors or from partial password recall. We show how to identify and correct for these errors with low false positive rates (i.e., low probability that an arbitrary string will be accepted as the password for authentication). Thus our techniques increase usability with provably little loss of security. Indeed, they may arguably even *increase* security in practice, because users benefiting from our correcting schemes will be more inclined to choose strong passwords, and not resort to insecure practices such as writing down a password.

Some naive approaches to this problem suggest themselves. The first would involve explicitly comparing an entered string to the password on file to check for equivalence modulo single transpositions or substitutions. However, this requires that the password file be stored as plain-text instead of being encrypted, which is clearly a bad idea for security. The second approach involves automatic repeated login attempts using explicit transformations of the entered string. Indeed, SAMBA appears to employ such a method to relax sensitivity to password case and character order [79]. However, such methods quickly get expensive, as there are $n-1$ possible transpositions and nm possible substitutions on an n-character password defined on an alphabet of size m. Finally, one might generate all variants of a password, and then store these encrypted. A login would then check each possibility. Not only would this increase the size of the password file, but it may also make malicious decrypting easier if it is known that a set of encrypted keys differ by only a transposition.

Instead, we propose a simple technique of applying a single *password-corrective* hash function to each entered password attempt. That is, this hash function is applied to the entered password, and the resulting key is then encrypted and stored. The important property required of the hash function is that two strings differing by a single data entry error (i.e. one transposition or substitution) be likely to be hashed to the same key, while more substantially differing strings are hashed to different keys.

In this chapter, we study the efficiency of a variety of hash functions in correcting single transposition and substitution errors. We rigorously analyze the recall / false positive rate tradeoffs for each class of hash functions to determine the most appropriate choice for common password applications. In particular:

- We develop precise analytical formulae for the precision/recall tradeoffs for correcting transposition errors using sorting-network and block-sorting hash functions. These functions contain parameters trading off security for usability; tradeoffs which are made explicit through our analysis.

- We do the same for two classes of alphabet-weakening hash functions, which correct for substitution errors. These alphabet-weakening schemes can be composed with the permutation-based functions described above, yielding a function which can simultaneously correct for transposition and substitution errors.

- We prove the curious property that the limiting case of both of our permutation-based methods (character sorting) has the highest precision among all perfect-recall methods for correcting transposition errors.

- The explicit precision / recall performance of these methods is very sensitive to the length and alphabet size of the associated keys. Therefore, we evaluate these tradeoffs at parameter values reflecting common classes of keys/passwords (including system passwords, social security numbers, WEP passwords, and credit card numbers) to identify the most desirable hash functions and parameters for each.

- Finally, we evaluate these schemes using a popular crack-list (dictionary) of 680,000 common words. We show that we can correct for *all* user transposition errors while reducing the computational cost of a crack attack by only 13%.

This chapter is organized as follows. Previous work on password system usability and corrective hashing techniques is reviewed in Section 4.1.1. We introduce the notion of password-corrective hashing in Section 4.2. The next two sections present our analysis of hashing techniques against transposition and substitution errors, respectively. Finally, Section 4.5 details our experiments using standard crack-lists.

4.1.1 Previous Work

The importance of user interaction in password authentication is well known. Basic facts about human memory are in conflict with most password policies. Sasse and Adams [1] stress human factors in developing security. Sasse, Brostoff and Weirich [70] note usability problems with password authentication, such as the number of passwords a user must remember, strict password policies, varying systems, and memory demands. Their studies found that users rarely completely forget a password. Instead, users often partially recall a password or recall the wrong password (typically from another system the user is enrolled in). They note that the user cannot know which of these two reasons apply after a failed authentication attempt.

There have been many human factor studies of data entry methods. Grudin [31] investigated error rates by typists, discovering that novice typists (20 wpm) had per-character error rates of about 3.2%, while experts (60-90 wpm) had error rates of 0.4% – 1.9%. Mackenzie [51] sought to partition these errors by type, identifying per-character substitution error rates of 0.962%, insertion error rates of 0.218%

and deletion rates of 1.045%. Peterson [63] found that transpositions represented between 2.6% and 13.1% of all data-entry errors, while substitutions accounted for between 26.9% and 40.0% of all errors.

There has been some previous work in developing password recovery schemes that tolerates errors. Frykholm and Juels [24] require users to supply answers to personal questions for authentication, but the answers are not required to be entirely correct. Spector and Ginzberg [76] propose a pass-phrase scheme that matches phrase semantics, and is flexible on syntax and actual words used.

Cranor and Garfinkel [19] suggest a system require more than 10^{12} different potential passwords for effective security. While most systems in theory allow this many, users restricting themselves to dictionary words use only about 10^6 different password keys. Our methods do reduce the theoretical number of potential passwords for a system; however for added security, password length can be made longer. The convenience offered by our system should make longer passwords less of a burden. Our proposed hashing methods reduce the effective space of potential keys. However modest assumptions of key length and alphabet size leave more than enough potential effective passwords to satisfy this concern.

The problem of determining when two different strings in fact refer to the same entity has several applications. When the strings refer to entities (people, places, things) this is known as the *co-reference resolution problem*, which has been well-studied within the natural language community [4, 9, 30, 52, 61]. Two names that should be spelled identically could possibly differ due to a typing error, a common misspelling, an alternate spelling, or in the case of foreign names differing transliteration methods (Osama / Usama). This use is leveraged in our system for co-reference resolution (Chapter 3. It also arises in the record linkage problem from data mining [16, 18, 34], The problem arises when there is no shared, error-free key field to join on across databases. Consider tables containing information about people from two different databases. Even if both databases used the person's name and address as the primary key, conventions concerning abbreviations and word usage may differ, and typos and misspellings may appear in either field. The goal is to identify which records correspond to the same entities.

4.2 Password Correction Hashing

In general, it is logical to assume that a minor difference between an entered password and the version on file still represents an authorized attempt to access an account. We will evaluate different hashing schemes to withstand substitution and transposition errors. These schemes transform an input string into a generalized representation; typically similar size to the original string. We will evaluate how these generalized representations correctly fix transposition and substitution errors (recall) vs. how often they induce random strings to collide (false positive rate).

4.2.1 Preliminaries and Notation

A transposition error is one in which two consecutive characters of a password are switched. If the password is $c_1c_2\dots c_n$, then a transposition is $c_1c_2\dots c_{i+1}c_i\dots c_n$. A substitution is when any single character is replaced by another. Thus for any $b \in \Sigma$, $c_1c_2\dots c_{i-1}bc_{i+1}\dots c_n$.

In dealing with the password correcting systems, it is necessary to distinguish the types of errors the system makes. A system that makes the error of not allowing authorized access is preferable to one that allows unauthorized access.

We denote a pair of different strings which are considered equivalent to be *real positive*. For equivalence under transposition, the pair “12345” and “12435” represent a real positive. Similarly, “12345” and “67890” are a real negative, since they are not equivalent. The *true positives* are the real positives that the hashing scheme correctly hashes to the same representative string. The false positives are the real negatives that the scheme incorrectly hashes to the same representative string. The definition is symmetric for true negatives and false positives.

We have the following relationship

$$\text{true positives} + \text{false negatives} = \text{real positives} \tag{4}$$

$$\text{true negatives} + \text{false positives} = \text{real negatives} \tag{5}$$

We will survey hash schemes on strings of length n over an alphabet Σ of size m. *Recall* is defined as

$$\text{recall} = \text{true positives}/(\text{real positives})$$

that is it is the fraction of positives the scheme correctly identifies as positive. Higher recall means easier access to the system, whereas lower recall is less flexible on the errors in the password. The *False Positive Rate* is defined as,

$$\text{False positive rate} = \text{false positives}/(\text{real negatives})$$

i.e. the frequency an unauthorized access (negative) is incorrectly called a positive. The lower this value, the more secure a system is.

4.3 Correcting Transpositions

In analyzing transposition errors, we note that the number of different positives and negatives depends on n and m. Let $P_{trans}[n,m]$ be the number of transposition positives, and $N_{trans}[n,m]$ be the number of negatives. The positives are counted as follows. Choose among the $n-1$ possible spots for a transposition. Then choose among the m characters for the $n-2$ spots not in the transposition. Finally we must choose from the m characters, 2 different characters that are in the transposition spot. We must choose different characters, since choosing the same character results in a transposition that gives back the original string. Thus

$$P_{trans}[n,m] \quad = \quad (n-1)m^{n-2}\binom{m}{2} \tag{6}$$

Since there are m^n different strings,

$$P_{trans}[n,m] + N_{trans}[n,m] = \binom{m^n}{2}$$

and thus

$$N_{trans}[n,m] = \binom{m^n}{2} - P_{trans}[n,m] = (m^n/2)((m^n-1)-(n-1)\frac{m-1}{m}) \tag{7}$$

4.3.1 Character Sorting

Sorting is a natural choice for trying to eliminate transposition errors, since sorting will tend to impose its own order on a string. Sorting the input sequence renders the original order inconsequential, so character distribution is the only distinguishing feature of a sequence. Thus all transpositions will be caught and hence

$$\text{recall}_{sort} = 1 \tag{8}$$

To count the false positives associated with character sorting, we first count the true positives plus false positives. Any pair of strings with the same character distribution will be hashed together to a true or false positive. So we count pairs of strings with the same character distribution:

$$\text{tp}_{sort} + \text{fp}_{sort} = \sum_{k_i \geq 0} \binom{\binom{n}{k_1 \ldots k_m}}{2} = \frac{1}{2}(\sum_{k_i \geq 0} \binom{n}{k_1 \ldots k_m}^2 - m^n) \tag{9}$$

Since we know the recall and total positives (from the previous section), we can use the formula for recall to solve for true positives. We then subtract this from the above result to get false positives, and divide by negatives to get

$$\text{fp-rate}_{sort} = \frac{X/m^n - 1 - n(m-1)}{m^n - 1 - n(m-1)}; X = \sum_{k_i \geq 0} \binom{n}{k_1 \ldots k_m}^2 \tag{10}$$

In fact, character sorting offers the highest precision way of correcting all single transposition errors:

Theorem 1 *Character sorting has perfect recall for single transpositions, and has the lowest false positives of any method that does so.*

Proof: Character sorting must have perfect recall, since any two strings differing by a single transposition must have the same character set. Now consider another method M which also has perfect recall but fewer false positives. There must be two strings S and T that are a false positive under character sorting, but not in the new method. S and T are hashed together under character sorting, so they have the same character set. Thus there is a sequence of strings $S, s_1, s_2, \ldots s_j, T$ where each consecutive strings differ by a single transposition. Since S and T are not hashed together under M, there must exist consecutive strings s_i, s_{i+1} in the sequence that are not hashed together under M. Since s_i and s_{i+1} differ by a single transposition, this contradicts the assumption that M had perfect recall. Therefore character sorting has the best performance of any perfect recall method. □

4.3.2 Even-Odd Transposition Sorting Networks: Single Stage

We now consider weakening (hashing) a string by sending it through k stages of an even-odd sorting network [43]. A sorting network is a computation graph. In an odd/even sorting network, at each stage adjacent entries can be swapped or left alone. Even pairs may be swapped in the even stages, and odd pairs in the odd stages. We assume the first stage is an even stage. The following example illustrates a string as it is transformed through each stage of the network:

$$14572463 \rightarrow 14572436 \rightarrow 14527346 \rightarrow 14253746 \rightarrow \cdots \rightarrow 12344567$$

We first consider the case of a single stage sorting network. After the first stage of an odd-even network, all even transpositions will be corrected, but odd transpositions will remain, so

$$\begin{aligned}\text{recall}_{1-stage} &= \frac{\text{even transpositions}}{\text{total transpositions}} \qquad (11)\\ &= \frac{\lfloor n/2 \rfloor}{n-1} \approx 1/2 \qquad (12)\end{aligned}$$

To calculate the false positives (fp) and the fp-rate, we first calculate the sum of false positives and true positives and then subtract the true positives (tp $=$ recall $*$ positives). To determine tp $+$ fp, we consider a string with k possible even transposition locations (i.e. $n-k$ characters are repeats, so no real transposition is possible). There are $\binom{\lfloor n/2 \rfloor}{k}$ ways to choose the k transposition locations; $m^{\lfloor n/2 \rfloor - k}$ ways to choose the characters for the repeated character transposition locations; $\binom{m}{2}^k$ ways to choose the characters for the k transposition locations; and finally 2^k ways to order the characters involved in the transposition locations. Each of these 2^k strings differs only in even transpositions, so all will get hashed together giving $\binom{2^k}{2}$ colliding pairs. Summing over k gives

$$\begin{aligned}\text{tp}_{1-stage} + \text{fp}_{1-stage} &= \sum_{k=0}^{\lfloor n/2 \rfloor} \binom{\lfloor n/2 \rfloor}{k} \binom{m}{2}^k m^{\lfloor n/2 \rfloor - k} \binom{2^k}{2} \qquad (13)\\ &= \frac{1}{2} m^{\lceil n/2 \rceil} ((2m-1)^{\lfloor n/2 \rfloor} - m^{\lfloor n/2 \rfloor}) \qquad (14)\end{aligned}$$

Then we solve for false positives.

$$\begin{aligned}\text{fp}_{1-stage} &= (\text{tp}_{1-stage} + \text{fp}_{1-stage}) - \text{tp}_{1-stage} \qquad (15)\\ &= \frac{1}{2} m^{\lceil n/2 \rceil} ((2m-1)^{\lfloor n/2 \rfloor} - m^{\lfloor n/2 \rfloor} - \lfloor n/2 \rfloor (\frac{m-1}{m}) m^{\lfloor n/2 \rfloor} \qquad (16)\end{aligned}$$

Finally,

$$\begin{aligned}\text{fp-rate}_{1-stage} &= \text{fp}_{1-stage}/\text{N[n,m]} = \frac{(2-1/m)^{\lfloor n/2 \rfloor} - 1 - \lfloor n/2 \rfloor (\frac{m-1}{m})}{m^n - 1 - (n-1)(\frac{m-1}{m})} \qquad (17)\\ &\approx m^{-n}(2^{\lfloor n/2 \rfloor} - 1 - n) \qquad (18)\end{aligned}$$

4.3.3 2-stage Sorting Networks

By adding an extra stage of sorting some odd transposition errors will now be caught, depending on whether the first stage moved the characters involved in the transposition. Consider the string fragment *abcd*; The odd transposition (*acbd*) will be corrected in the second step when $a \le b, c \le d$. This gives $\binom{m+2}{4}$ corrected transposition errors from $\frac{1}{2}m^3(m-1)$ possible transposition errors. Odd length strings have an extra odd transposition, not surrounded by 4 characters, only 3, as in *abc*. In this case, there are $\binom{m+1}{3}$ transpositions that get corrected from $\frac{1}{2}m^2(m-1)$ possible errors.

$$\begin{aligned} \text{recall}_{2stage} &= (\lfloor n/2 \rfloor + \frac{(\lceil n/2 \rceil - 1) * \binom{m+2}{4}}{(\frac{1}{2}m^3(m-1))} + [nodd]\frac{\binom{m+1}{3}}{(\frac{1}{2}m^2(m-1))}) \times (\frac{1}{n-1}) \qquad (19) \\ &\approx (1/12) + (11/12)(\frac{\lfloor n/2 \rfloor}{n-1}) \qquad (20) \end{aligned}$$

Where $[nodd]$ evaluates to 1 if n is odd, and 0 otherwise. We do not have analytical results for the false positive rate of 2-stage sorting, so we instead ran simulations to get results. See Section 4.3.5 for these results.

4.3.4 Block Sorting Methods

With block sorting, we divide the string into fixed-size blocks, and completely sort each block. The following example illustrates the transformation for blocks of size 4:

$$1738|5901|9874|3509|1237 \rightarrow 1378|0159|4789|0359|1237$$

The only transposition errors not matched by such a scheme are those that occur across block boundaries.

We first consider the case of 2 blocks. The string is broken up into a block of size n_1 and one of size $n - n_1$, typically $n_1 = n/2$. Only a transposition that crosses over the block boundary will not be caught, so

$$\text{recall}_{2-block} = (n-2)/(n-1)$$

regardless of the block sizes. Now consider true positives. Consider the true positives that result from a single transposition spot. We can choose among all m characters for every digit, except the transposition digits must be different. Thus the $m^{n-1}(m-1)$ term. There are $(n-2)$ transposition spots (since we cant match across the blocks). Finally, each match is counted twice, so we divide by 2.

$$\mathrm{tp}_{2-block} = (1/2)m^{n-1}(m-1)(n-2)$$

Let $\mathrm{fptp}_{\mathrm{CS}}(k)$ be the true positives plus false positives for complete sorting a string of length k. We get the fp-rate by using the results from complete sorting. Since within a block, the contents are completely sorted, we have

$$\mathrm{fp}_{2-block} = \mathrm{fptp}_{\mathrm{CS}}(n_1) \times \mathrm{fptp}_{\mathrm{CS}}(n-n_1) \tag{21}$$

We can generalize for an arbitrary number of k blocks. The only transpositions not found are still ones occurring across block boundaries, so

$$\mathrm{recall}_{k-blocks} = (n-k)/(n-1)$$

We again get the fp-rate by using results from complete sorting.

$$\mathrm{fp}_{k-blocks} = \prod_{j=0}^{k} \mathrm{fptp}_{\mathrm{CS}}(n_j) \tag{22}$$

4.3.5 Evaluation

In this section, we evaluate these transposition correction methods on a variety of alphabet sizes and string length pairs corresponding to important classes of passwords/keys. In particular, we consider:

- *System Passwords* – Typical online account passwords. We consider three cases: the full English alphabet with case, digits, underscore and period ($m = 64$), a smaller case-independent alphabet of size 32, and binary passwords on typical lengths.

- *WEP Keys* – Wireless encryption (WEP) keys. We consider hexadecimal WEP keys for 64 and 128-bit WEP ($n = 10, 26$, $m = 16$).

- *Social Security Numbers* – Nine digit identification numbers, ($n = 9$, $m = 10$).

- *Credit Cards* – Credit cards numbers comprise 16 digit numbers, so ($n = 16$, $m = 10$).

- *Proper Names* – The first/last names of people average about seven characters on the case-insensitive English alphabet, so ($n = 7$, $m = 26$).

- *System Passwords* – These represent passwords used for typical online account access. We consider three cases: the full English alphabet with case, digits, underscore and period ($m = 64$), a smaller case-independent alphabet of size 32, and binary passwords on typical lengths.

- *WEP Keys* – Wireless encryption (WEP) keys are widely used for encrypting wireless communications. We consider hexadecimal WEP keys for 64 and 128-bit WEP ($n = 10, 26$, $m = 16$).

- *Social Security Numbers* – These identifier numbers are widely used as database keys to identify individual Americans. Each identifier is a nine digit number, so ($n = 9$, $m = 10$).

- *Credit Cards* – Web transactions often employ credit card numbers as identifiers after they have been previously entered for payment. They comprise 16 digit numbers, so ($n = 16$, $m = 10$).

- *Proper Names* – The first/last names of people average about seven characters on the case-insensitive English alphabet, so ($n = 7$, $m = 26$).

Table 9 compares the performance of k-stage sorting networks, and k block sorting for correcting transposition errors. Most of the results were calculated exactly using formulas 4 - 22, but some were approximated. The X summation, $\sum_{k_i \geq 0} \binom{n}{k_1 \ldots k_m}^2$, that appeared in the character sorting analysis was computationally evaluated for the different applications; except for the cases ($m = 64$,$n = 8$) and ($m = 16$,$n = 26$), for which it was approximated. The maximum of the $\binom{m+n-1}{n}$ terms in the summation is $(\frac{n!}{\lfloor n/m \rfloor !^m \lceil n/m \rceil !^{(n-\lfloor n/m \rfloor m)}})^2$. Thus

$$X \approx \binom{m+n-1}{n}(n!/\lfloor n/m \rfloor !^m \lceil n/m \rceil !^{(n-\lfloor n/m \rfloor m)})^2 \qquad (23)$$

For the two-stage sorting network, the false positive rate is estimated by taking random samples and doing a 2-stage sorting network, and then doing a reverse 2-stage sorting network on the result to determine the set of starting strings that could have generated the result code. This gives us the ratio of false positives to true positives, which is then used to calculate the false positive rate.

We see for most schemes, good recall is achieved at reasonably low false positive rates. High recall and low false positive rate will ensure that the added convenience of a system does not come with a loss of security. *The 2-Block scheme offers the best balance between high recall and low false positive rates, and is recommended.* It should be noted that these schemes do become risky on small alphabets, as the row for $m = 2$ indicates. Fortunately secure systems use large alphabet sizes, so this will not be a problem.

4.4 Correcting Substitution Errors

Another common class of entry errors is substitution errors, where one character gets replaced by another character. We now consider two classes of hash functions that weaken the alphabet by making distinct characters the same. Such schemes can overcome substitution errors, i.e. two strings should be hashed together if they differ by only a single substitution. For substitutions, we have

$$P[n,m]_{subs} = \frac{1}{2} m^n n(m-1) \qquad (24)$$

		Algorithm									
Application		1-Stage		2-Stage		Complete		2-Block		3-Block	
n	m	Rec	fp-rate	Rec	fp-rate	Rec	fp-rate	Rec	fp-rate	Rec	fp-rate
Passwords											
8	64	0.571	3.75e-14	0.609	3.92e-13	1	2.17e-10	0.857	6.37e-13	0.714	2.07e-14
10	32	0.556	2.11e-14	0.596	3.64e-13	1	8.20e-11	0.889	4.23e-12	0.778	1.03e-13
20	2	0.526	4.93e-5	0.645	0.0110	1	0.125	0.947	0.0146	0.894	0.002135
WEP Key											
10	16	0.556	1.97e-11	0.600	3.30e-10	1	7.67e-7	0.889	3.08e-9	0.778	8.48e-11
26	16	0.520	2.67e-28	0.568	3.57e-28	1	2.39e-5	0.960	5.96e-15	0.920	2.58e-18
SSNs											
9	10	0.500	8.43e-10	0.587	1.60e-7	1	5.48e-5	0.875	5.93e-7	0.750	1.77e-8
Credit Cards											
16	10	0.533	1.62e-14	0.585	1.45e-12	1	4.12e-6	0.933	4.30e-9	0.867	4.06e-11
Names											
7	26	0.500	1.75e-11	0.598	5.96e-9	1	4.15e-7	0.833	6.43e-9	0.667	1.34e-10

Table 9: Recall and False Positive Rate for correcting transposition errors for common password/key lengths

$$N[n,m]_{subs} = \binom{m^n}{2} - P[n,m]_{subs} = \frac{1}{2}m^n(m^n - 1 - n(m-1)) \tag{25}$$

4.4.1 High-Low Weakening

In this scheme, we partition characters in the alphabet Σ as being either high or low. This reduces the input key to a binary string. For example, considering the digits $0-4$ as low ('l') and $5-9$ as high ('h') transforms:

$$17385901987435091237 \rightarrow lhlhhhllhhhllhlhlllh$$

A substitution error is found whenever the substituted characters map to the same symbol. Let k be the size of the low set (and thus $m-k$ the size of the high set). The true positives follows since there are n character positions to perform a substitution, the other $n-1$ characters can be anything.

$$\mathrm{tp}_{high-low} = nm^{n-1}(\binom{k}{2} + \binom{m-k}{2}) \tag{26}$$

We divide this by the number of positives to get

$$\mathrm{recall}_{high-low} = \frac{k(k-1) + (m-k)(m-k-1)}{m(m-1)} \tag{27}$$

To determine the false positives, we first calculate $\mathrm{tp}+\mathrm{fp}$. We sum over j where j is the number of characters in the string belonging to the low set.

$$\begin{aligned}\mathrm{tp}_{high-low}+\mathrm{fp}_{high-low} &= \sum_{j=0}^{n}\binom{k^j(m-k)^{n-j}}{2}\times\binom{n}{j} && (28)\\ &= \frac{1}{2}((2k^2+m^2-2mk)^n-m^n) && (29)\end{aligned}$$

We then subtract the true positives and divide by negatives to get the false positive rate.

4.4.2 Weak Set Methods

In this scheme, a set of k ‘weak’ characters get replaced by a single character, while the other characters remain the same. For example, defining the weak set as consisting of all non-alphabetic characters and replacing them with the weak symbol (’.’) yields the transformation

$$L1saS!mps0n \rightarrow L.saS.mps.n$$

This leaves an alphabet of size $m-k+1$. Only substitutions among these k characters are found, so

$$\mathrm{recall}_{weak-set}=\frac{k(k-1)}{m(m-1)} \qquad (30)$$

We get the false positives by first calculating $fp+tp$.

$$\mathrm{tp}_{weak-set}+\mathrm{fp}_{weak-set}=\sum_{j=0}^{n}\binom{k^j}{2}(m-k)^{n-j}\binom{n}{j}=\frac{1}{2}((m-k+k^2)^n-m^n \qquad (31)$$

Solving for fp-rate gets

$$\text{fp-rate}_{weak-set}=\frac{(1-k/m+k^2/m)^n-1-n/m(k)(k-1)}{m^n-1-n(m-1)}\approx(\frac{1}{m}-\frac{k}{m^2}+\frac{k^2}{m^2})^n \qquad (32)$$

		Algorithm			
Application		High-Low		Weak Set	
n	m	Rec	fp-rate	Rec	fp-rate
Passwords					
8	64	0.492	0.00391	0.246	1.95e-5
10	32	0.484	9.77e-4	0,242	1.75e-6
WEP Key					
10	16	0.467	9.77e-4	0.233	3.10e-6
26	16	0.467	1.49e-8	0.233	4.75e-15
SSNs					
9	10	0.444	0.00195	0.222	1.97e-5
Credit Cards					
16	10	0.444	1.53e-5	0.222	4.30e-9
Names					
7	26	0.480	0.00781	0.240	1.03e-4

Table 10: Recall and False Positive Rate for correcting substitution errors on common password/key lengths.

		Algorithm							
Application		k=4		k=m/8		k=m/4		k=m/2	
n	m	Rec	fp-rate	Rec	fp-rate	Rec	fp-rate	Rec	fp-rate
Passwords									
8	64	0.003	5.17e-15	0.014	5.14e-13	0.060	9.20e-10	0.246	1.95e-5
10	32	0.012	1.72e-14	0.012	1.72e-14	0.056	2.20e-11	0.242	1.75e-6
WEP Key									
10	16	0.050	2.37e-10	0.008	9.07e-13	0.050	2.37e-10	0.233	3.10e-6
26	16	0.050	1.03e-25	0.008	8.44e-31	0.050	1.03e-25	0.233	4.75e-15
SSNs									
9	10	0.133	1.95e-6	-	-	0.022	2.36e-9	0.222	1.97e-5
Credit Cards									
16	10	0.133	3.01e-11	-	-	0.022	1.43e-15	0.222	4.30e-9
Names									
7	26	0.018	1.25e-9	0.009	2.07e-10	0.046	2.56e-8	0.240	1.03e-4

Table 11: Weak Set performance in correcting substitution errors for different weak set sizes.

4.4.3 Evaluation

Tables 10 and 11 present our results for correcting single substitution errors using alphabet weakening. Table 10 shows High-Low weakening results for equal-sized high and low sets; and 'Weak set' weakening for a weak set size of $m/2$. Table 11 gives the tradeoffs for 'Weak Set' weakening as we change the size of the weak set. Since an alphabet of size 2 cannot be further weakened, applications with $m = 2$ are not shown.

Our results show a clear recall / false positive rate tradeoff; and the false positive rates are more problematic than we obtained for transposition error correction in Table 9. The weak set results of Table 11 has more acceptable false positive rates, but very little recall gain. The figures below show the performance for varying the k parameter (the size of the weakened sets).

We see that substitution errors appear more difficult to correct than transposition errors. The loss of information of the type of character makes any scheme capable of high recall also have high false positive rates.

4.5 Resistance to Crack-List Attacks

Users usually choose passwords from a much smaller key space than that offered by the system. For instance, although systems typically allow arbitrary character strings many users stick to dictionary words or common names. One failing of our analytical results is that we assumed a uniform distribution of passwords over the space of possible keys. Also, we assumed that all keys are the same length, which is not true in many domains.

To get a more complete sense of the performance of correction schemes, we tested on them a crack list of dictionary words and common names. We used the lists from ftp://ftp.cerias.purdue.edu/pub/dict/dictionaries, which includes dictionaries in English, German, Italian, Swedish, Norwegian, and Dutch; as well as

Block Sorting				Sorting Network			
Blocks	Recall	Unique Codes	False Pos. Rate	Stages	Recall	Unique Codes	False Pos. Rate
1	1	593347	1.42e-06	Inf	1	593347	1.42e-06
2	0.89	656475	4.79e-07	9	0.93	596026	1.41e-06
3	0.79	670265	3.16e-07	8	0.89	600209	1.39e-06
4	0.68	676146	2.68e-07	7	0.85	607036	1.33e-06
5	0.57	678491	2.51e-07	6	0.80	618944	1.21e-06
6	0.47	679395	2.42e-07	5	0.75	632101	1.03e-06
7	0.38	679737	2.39e-07	4	0.70	648115	7.61e-07
8	0.30	679873	2.38e-07	3	0.66	658338	5.74e-07
9	0.24	679932	2.37e-07	2	0.60	668050	3.80e-07
10	0.18	679972	2.36e-07	1	0.55	672544	3.17e-07
Inf	0	680000	0	0	0	680000	0

Table 12: Performance of Transposition Correcting Methods on Dictionary Data.

lists of common names, organizations, abbreviations, popular movie and TV names, common slang, Internet words, famous people, and a few other popular terms that appear in passwords. Combined, these lists had 680,000 unique terms.

Table 12 shows the ability of block sorting and sorting networks to correct transpositions on the crack lists. We see that in the case of a complete sort, the number of unique keys is now only 593,347. That is, a cracker whose initial crack-list of 680,000 words could now get by with a list of 593,347 words; this is about 13% shorter. This is not much to pay for eliminating all transposition errors. For more extreme security, 5-block sorting still has over 50% recall, yet allows a reduction of only 1509 names off the crack list, or 0.22%. The performance of sorting networks is not quite as good, though still reasonably effective. Figure 18 illustrates the false positive-recall tradeoffs of the two methods.

Figure 19 shows the distribution of the size of the equivalence classes for high-low and weak set, with a split point of 248. The x-axis is the size of the code set. That is, the value 3 means that 3 different words get hashed to the same code.

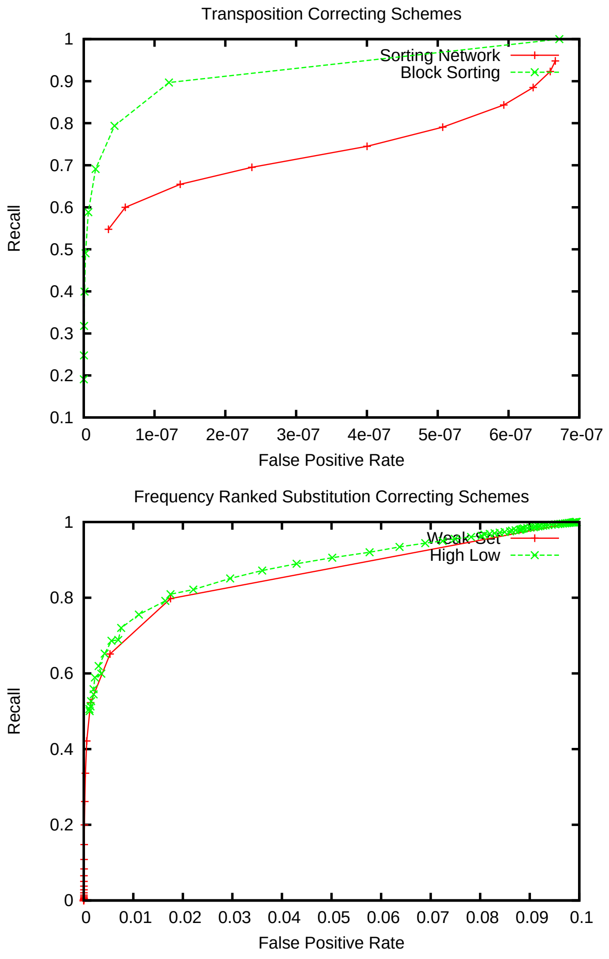


Figure 18: False Positive-Recall Tradeoffs for Transposition (l) and Substitution (r) Methods

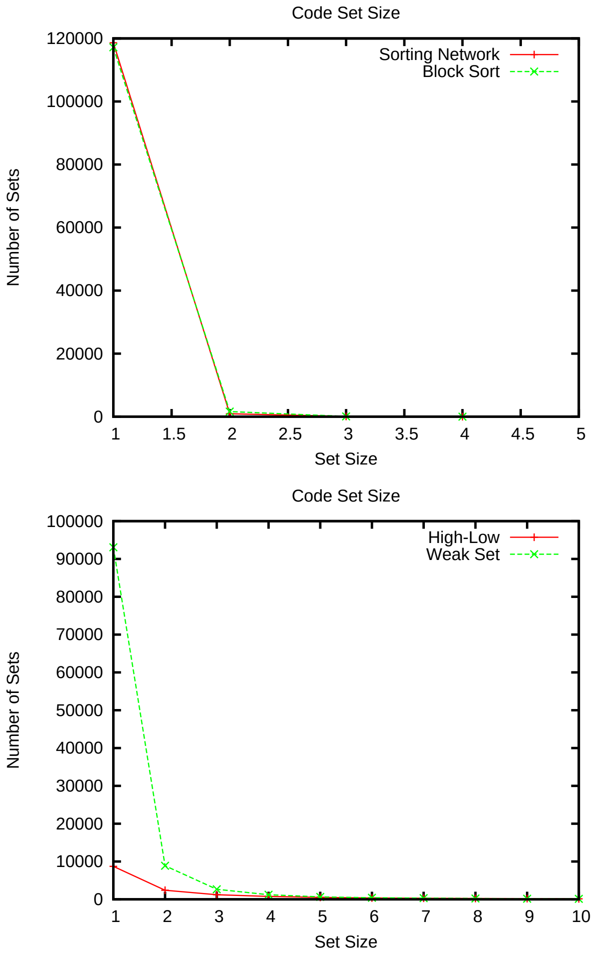


Figure 19: Number of Words per Hash Key, for Transpositions (l) and Substitutions (r)

Chapter 5

Implied Relations and Edge Pruning

5.1 Introduction

On-line news sources provide a large and comprehensive corpus of world events and news entities. The Lydia project (www.textmap.com) analyzes over a thousand on-line newspapers every day to discover news trends, sentiments, and geographic biases. The aim of the project is to deliver news analysis on a scale of content that would be impossible for a person to read, and to mine the data to discover world facts that a human analyst would be unable to realize. One technique to analyze the data is leveraging network science on a co-occurrence network of named entities. Edge scores in this network are frequency of co-occurrence between two entities.

But real world network data is often noisy. Edge scores are also often biased for popular entities. We would like to develop an edge score that is a more accurate reflection of actual link strength. This will give a better understanding of the network, as well as enable us to prune low strength relations. Since our network is very large, this will enable otherwise intractable analysis.

We want to score pairs of entities based on how related they are. Currently, we have a co-occurrence count as an edge score, but as we will see, there are shortcomings to relying on this measure alone. We have developed another notion of an ‘implied relation score’ that corrects for implied co-occurrences, and normalizes for overall entity popularity. We can also consider other measures such as mutual information (or point-wise mutual information), and measures that consider the profiles of high frequency words appearing with entities. This will help us distinguish entities with high co-occurrences from their frequent appearance with a common (most likely popular) neighbor, such as ‘Dick Cheney’ and ‘Laura Bush’. Both of these people occur frequently with ‘George Bush’, so they often co-occur with each other due to the common friend, but they themselves share no real connection. A second-order relationship exists between Laura Bush and Dick Cheney which is almost completely explained by interactions between George Bush and Dick Cheney. Second-order relationships within a network are in general less interesting than primary relations. On the other hand, entities which are overall not mentioned frequently will have a low number of co-occurrences (because they have a low number of occurrences), although they may be involved in an important relation. We propose a model to identify second order relationships within a news network. We show that it enables us to prune a news network of less interesting relationships with minimal impact on network structure. These prune-able edges are in essence ‘implied’ edges; existing only as an artifact of other relations.

5.2 Previous Work

5.2.1 The Lydia Project

The Lydia project has been discussed in Chapter 1. It is mentioned again here to make this chapter self-contained. The Lydia system automatically builds an entity database from online U.S. newspapers downloaded on a daily basis. The techniques used for entity identification include part-of-speech tagging, templates and gazetteers, as well as clustering for co-reference resolution. The architecture of the Lydia system is described in detail in [48]. The main components of Lydia are spiders to download the news sources, named entity extraction and classification, including co-reference detection [49], and various derivative analysis based on the named entity recognition.

The system can also be extended to other sources such as pub-med databases, financial reports, and blogs [47]. Some of these analysis include question answering [42], spatial analysis and geographic bias (‘heatmaps’) [53], and sentiment analysis [29]. Searching our database is also discussed in [8].

5.2.2 Network Links

Motivated by clustering, work has been done on assigning similarity scores to vertices of a network, as a measure of their structural equivalence. These include Euclidean distance, Pearson correlation, and mutual information [58]. While these measures may be useful in determining vertices’ similarity (or dis-similarity), they have classically been used as metrics for clustering algorithms.

Clauset, Moore, and Newman [15] describe a method of predicting *missing* links in networks, the opposite problem we are looking at. Their method is based on first finding a hierarchical structure, and using this probabilistic hierarchy to decide which non-existent links are most likely to be in the network. This same idea could in principle be applied to deciding which existing links the model is most surprised by, but this was not studied.

5.3 Identifying Implied Relationships

The nature of the way news articles are written, and the difficulty of NLP processing tends to lead to networks that are larger than necessary. Authors feel free to make off-cuff and unrelated comments on any subject, and in general the real world aspect of article data leads to noise. We describe a technique for calculating the number of ‘noisy’ edges in such a network. In particular, we develop an ‘implied relation score' for each edge that indicates how certain we are the edge represents a real relationship between entities, or if the edge is an artifact of noise (i.e. it can be explained away by other relations in the network). (These edges were originally termed *spurious*, but this notion didn't accurately describe our methods; however, due to this legacy 'spurious score', and 'implied relations score' are occasionally used interchangeably)

The purpose of determining the *implied* edges is two-fold. First, pruning implied edges from our graph can make existing network analysis techniques tractable. Alternatively we can use the score itself as the basis for analysis. We can then use this score as a similarity score for, say, clustering.

5.3.1 Implied/Meaningful Juxtapositions

From our news analysis, we get a juxtaposition count for each pair of entities. A juxtaposition (co-occurrence) is defined as a pair of entities appearing in the same sentence. Although correlated, this count does not tell us how meaningful a relationship exists between the two entities. For example, consider the two pairs of entities (George Bush, United States) and (Stephen Harper, Canada). Clearly we would expect many more juxtapositions for (George Bush, United States), based on the relative popularity. However, the relation between the two pairs is essentially the same. Also consider Laura Bush and Dick Cheney. The juxtapositions between these two entities will be artificially high, simply because they both get mentioned frequently with a third entity, namely George Bush .

We determine a normalized meaning-fullness by modeling juxtapositions as being of three basic types:

- *Popularity* - Some juxtapositions will occur simply because the entities involved are overall extremely popular.

- *Neighbors* - Some juxtapositions will occur because two entities have a common neighbor. No strong relation exists between the two entities, but their juxtapositions are fairly high due to a common friend.
- *Significant Relationships* - Juxtapositions occurring from actual relations and interactions.

We model these types of juxtapositions by taking a naive assumption of independence. We estimate the first two types of juxtapositions, and subtract these from the observed number of juxtapositions to get the third type.

5.3.1.1 Entity Popularity

If we assume an entity is equally likely to participate in any juxtaposition pair, regardless of the other entity in the pair, than the expected number of juxtapositions between two entities depends only on their overall frequencies. That is,

$$\text{Juxts}_{pop}(e_1, e_2) = \text{Juxts}_{total} * f(e_1) * f(e_2)$$

where Juxts_{pop} are the estimate of juxtapositions occurring from popularity, $f(e)$ is the total frequency of an entity e in the corpus, and Juxts_{total} are the total number of juxtapositions in the corpus.

5.3.1.2 Neighborhood

Consider now two entities e_1, e_2 that have a common neighbor z. We can again assume that given the occurrence of z, the occurrence of e_1 and e_2 are independent. Under these assumptions e_1 and e_2 have a frequency of appearing with z given by,

$$f(e_1|z) = \text{Juxts}(e_1, z) / \sum_e \text{Juxts}(e, z).$$

and the estimate is then the same as the popularity estimate,

$$\begin{aligned}\text{Juxts}_{neighbor}(e_1,e_2|z) &= \text{Juxts}_z * f(e_1|z) * f(e_2|z) &(33)\\ &= \text{Juxts}(e_1,z) * \text{Juxts}(e_2,z) / \sum_e \text{Juxts}(e,z). &(34)\end{aligned}$$

To get the total neighborhood contribution we sum over all neighbors z.

5.3.1.3 Implied Relation Score

From the estimates of Neighborhood and Popularity juxtapositions, we get the significant juxtapositions by subtracting from our empirical count. To normalize for overall popularity, we then convert to a score by calculating the probability that the significant juxtaposition count could occur by chance. That is, two very popular entities will be more likely to have a juxtaposition count much greater than what we estimate, so the apparent large number of significant juxtapositions may be due to chance fluctuations. By converting to a probability we have a score that is comparable across all pairs of entities.

This probability is a computationally expensive operation. An alternative score is to use the ratio of real edges to implied edges. Since this ratio would be extremely high for low weight edges, we multiply by the log of the real edges, giving

$$\text{implied relation score} = (\text{real edges}/\text{implied edges}) * \log(1 + \text{real edges}).$$

5.3.2 Evaluation of Implied Relation Score

We evaluate the effectiveness of our implied relation score by seeing how properties of the graph change as implied edges are removed in order of score. We expect our network to be a small-world network and thus have short paths between any two connected vertices. This can be seen in Figure 20. Since calculating all shortest path is computationally intensive, we compute the average shortest paths on random

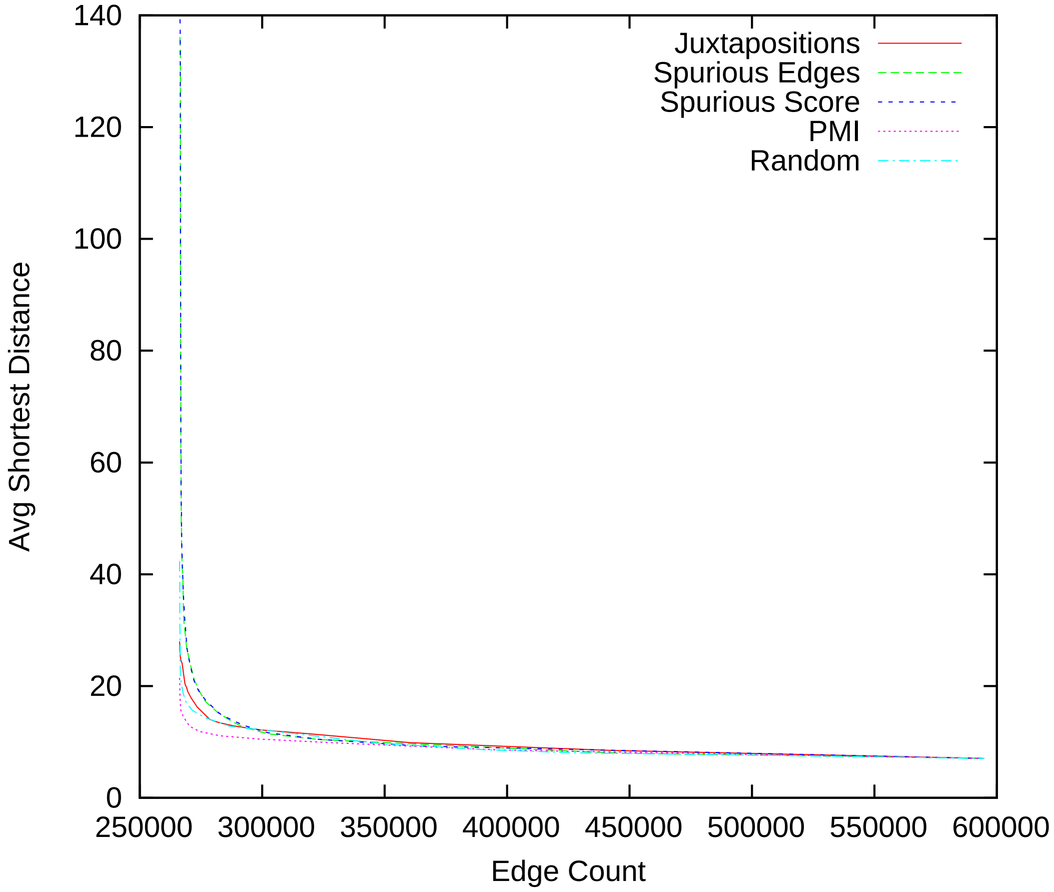


Figure 20: Average shortest path distances of pruned networks. Edges are added according to pointwise mutual information, implied relation score, or randomly; starting from a minimum spanning tree. As edges are added, the average distance quickly decreases, approaching the complete graph.

samples of the vertices. The graphs also show what happens if we remove edges in reverse order (Inv. Implied Score), by simple co-reference count (Juxt Count), and randomly (Random Score).

Another property we examine is articulation points (Figure 21) and leaves (Figure 22. We see that as we remove implied edges, the number of articulation points grows very slowly, but as more important edges are removed, the number of articulation points begins to grow rapidly. This effect needs to be studied more rigorously, but we believe that removing the most implied edges leaves tight clusters, and thus many bridges between clusters. Thus once the most implied edges are removed, and then inter-cluster edges, there will be many articulation points.

One problem in evaluating our score is that there is no ground truth for what a *real* relation is. We can approximate a ground truth by using a collection of human knowledge, wikipedia. We assume that there is a real relation between two entities if one of them appears on the other's wikipedia page. We then use the implied relations score as a threshold for pruning, and can get precision and recall for how well the pruned graph approximates the wikipedia graph. We use 2 different sets of wikipedia dumps for our ground truth. One is a small dump from 5 November 2005; and the other is a much larger dump from 18 October 2007 (After Wikipedia gained significant popularity.) The average and median positive and negative score for each scoring method is shown in Tables 13 - 16. The precision / recall graphs are shown in Figures 23 - 24. Figure 23 shows the precision of above a certain rank for our different scoring methods. Figure 24 shows the precision / recall curves for each scoring method. The curves are obtained by considering all possible thresholds for each method. The precision / recall graphs are shown in Figures 23 - 24.

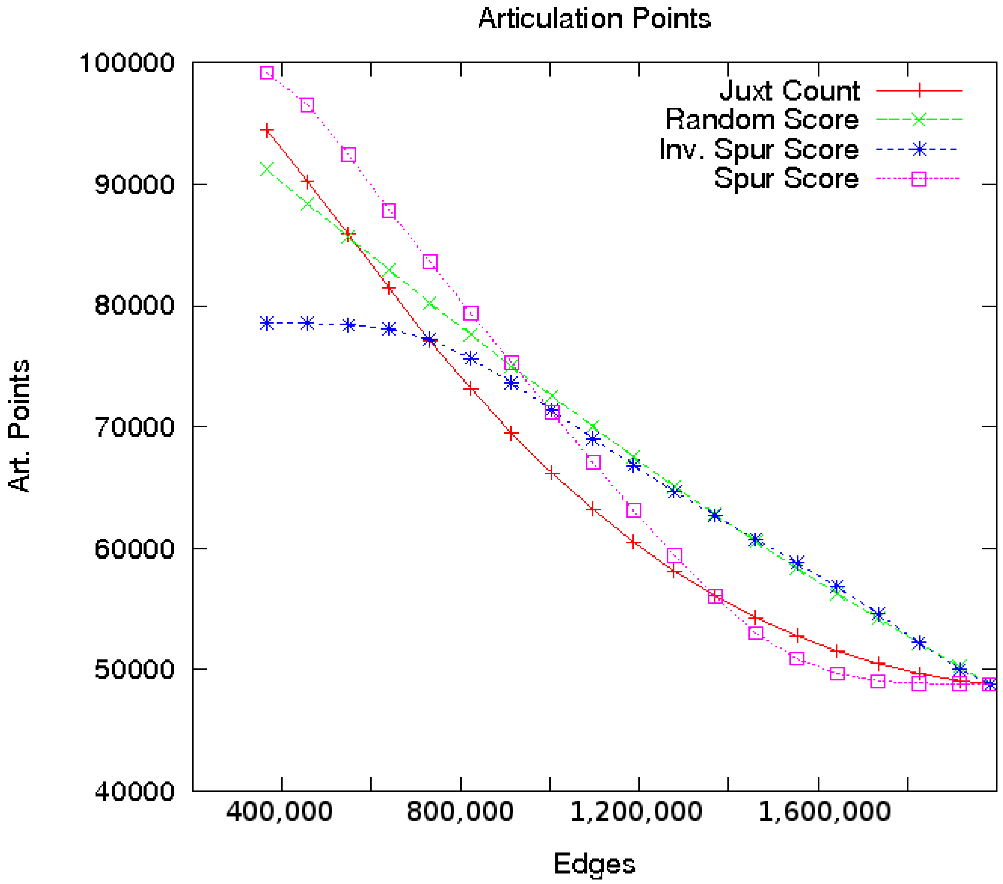


Figure 21: Number of articulation points.

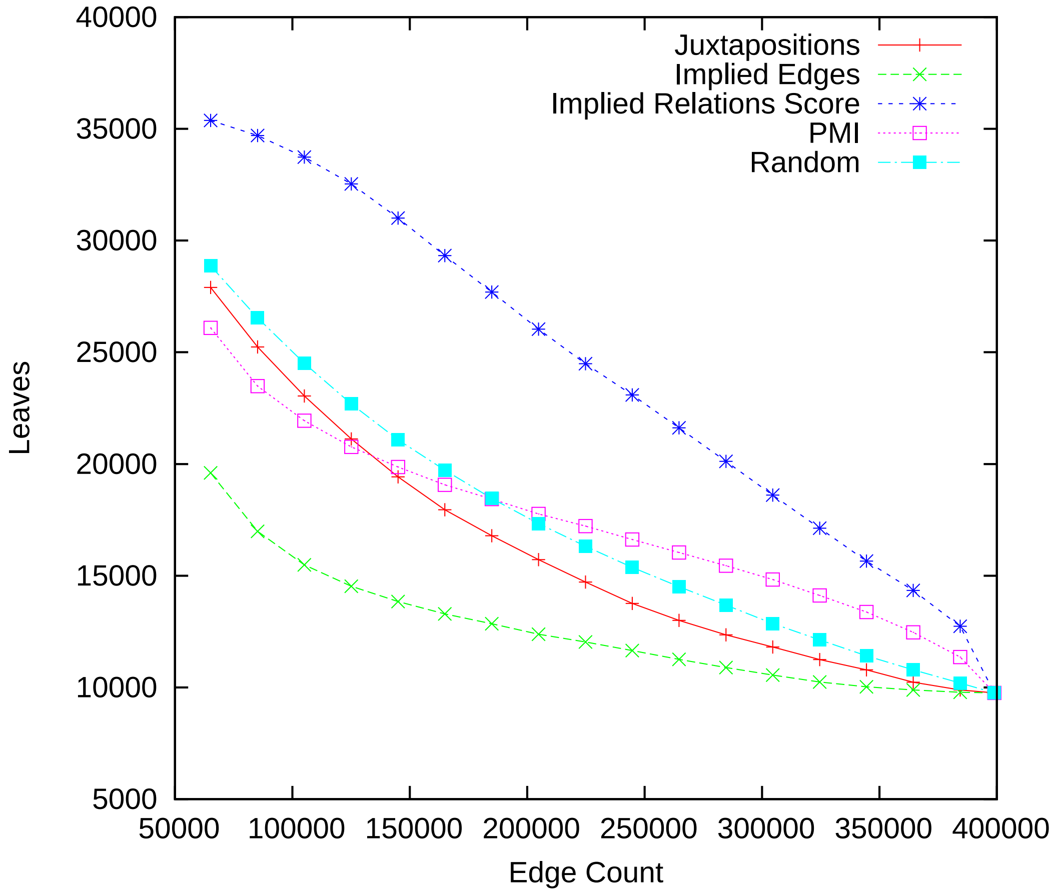


Figure 22: Number of leaves

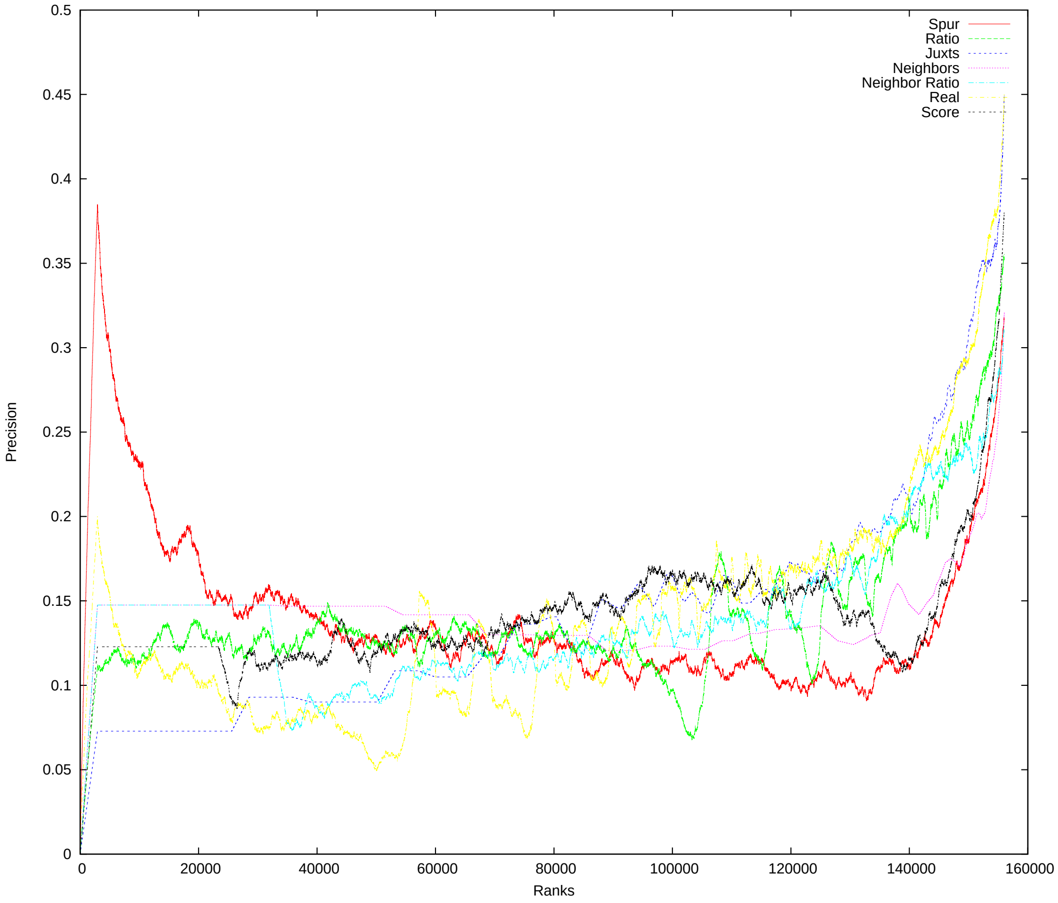


Figure 23: Precision/Recall tradeoffs for different pruning methods.

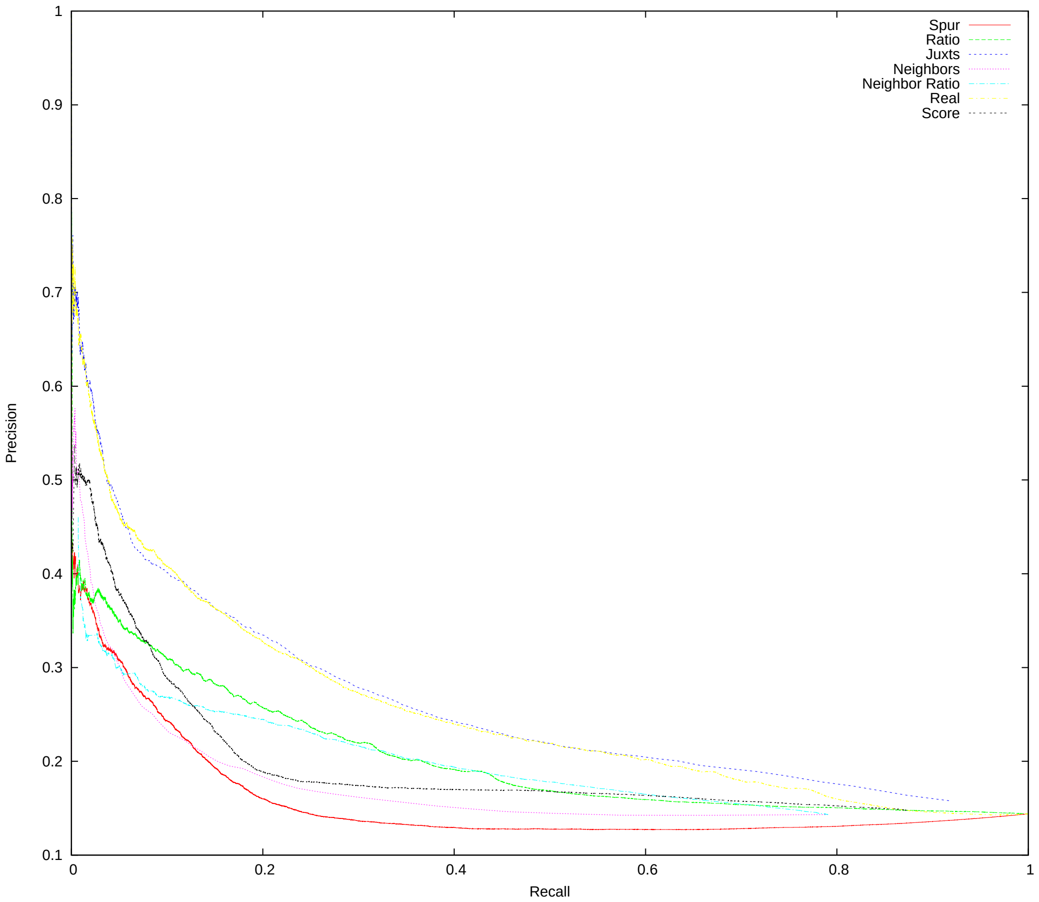


Figure 24: Precision/Recall tradeoffs for different pruning methods.

Method		medline ALL 20071018	dailies PERSON 20071018	dailies ALL 20071018	goodnews PERSON 20071018	goodnews ALL 20071018
Score	Pos.	4,300.11	48,384.40	86,039.20	65,302.80	152,563.00
Score	Neg.	219.62	11,159.10	11,135.90	8,731.04	21,276.00
Juxts	Pos.	94.68	85.09	78.83	39.13	38.57
Juxts	Neg.	48.64	30.12	30.87	17.14	20.14
Real	Pos.	34.95	52.88	37.59	26.11	15.17
Real	Neg.	-41.61	16.05	5.39	9.73	5.18
Implied	Pos.	59.73	32.21	41.23	13.03	23.40
Implied	Neg.	90.26	14.06	25.48	7.41	14.96
Ratio	Pos.	4.03	7.95	5.09	6.46	4.19
Ratio	Neg.	1.11	4.39	3.28	3.78	3.28
Nbrs	Pos.	36.46	8.96	51.89	6.13	38.98
Nbrs	Neg.	60.50	7.12	44.44	6.09	33.34
NRatio	Pos.	0.16	0.11	0.07	0.11	0.08
NRatio	Neg.	0.12	0.08	0.04	0.08	0.05

Table 13: Network averages for positive and negative examples on the 2007 Wikipedia data. Except for some medline exceptions, positive scores are higher than negative scores.

5.4 Future Work

Network science research is often limited by the availability of good network data. Data such as citation networks [3], karate clubs [58], and email/blog networks [44] have proven useful for exploration, and confirming theory. Our data then makes for interesting study, because it is both large scale and comprehensive of world events.

Method		medline ALL 20071018	dailies PERSON 20071018	dailies ALL 20071018	goodnews PERSON 20071018	goodnews ALL 20071018
Score	Pos.	5.70	13.97	9.18	20.16	9.94
Score	Neg.	-1.00	8.12	5.89	8.33	6.85
Juxts	Pos.	27.00	20.00	17.00	13.00	11.00
Juxts	Neg.	17.00	10.00	8.00	9.00	8.00
Real	Pos.	12.48	12.00	9.47	8.99	6.84
Real	Neg.	-0.88	5.97	4.65	5.99	5.51
Implied	Pos.	11.92	4.58	5.22	2.26	2.87
Implied	Neg.	22.21	3.01	3.02	2.31	2.51
Ratio	Pos.	1.63	3.18	2.36	4.27	2.66
Ratio	Neg.	-0.05	2.26	1.82	2.42	2.10
Nbrs	Pos.	19.00	4.00	15.00	2.00	8.00
Nbrs	Neg.	41.00	4.00	15.00	3.00	11.00
NRatio	Pos.	0.14	0.07	0.05	0.06	0.04
NRatio	Neg.	0.11	0.05	0.03	0.06	0.03

Table 14: Network medians for positive and negative examples on the 2007 Wikipedia data.

Method		medline ALL 20051105	dailies PERSON 20051105	dailies ALL 20051105	goodnews PERSON 20051105	goodnews ALL 20051105
Score	Pos.	6,114.22	42,952.30	59,024.40	81,593.00	174,486.00
Score	Neg.	125.31	6,738.93	5,609.93	6,675.41	13,821.30
Juxts	Pos.	136.14	87.06	113.22	40.98	48.51
Juxts	Neg.	55.67	39.04	31.44	21.63	21.02
Real	Pos.	43.87	57.60	54.27	29.64	22.94
Real	Neg.	-54.37	19.61	5.64	10.89	4.71
Spur	Pos.	92.27	29.46	58.95	11.34	25.57
Spur	Neg.	110.03	19.43	25.80	10.74	16.31
Ratio	Pos.	4.22	9.66	5.22	7.65	4.65
Ratio	Neg.	0.86	4.47	2.98	3.59	3.05
Nbrs	Pos.	40.23	7.35	48.80	4.37	31.77
Nbrs	Neg.	64.11	6.17	45.93	6.00	33.18
NRatio	Pos.	0.18	0.09	0.08	0.10	0.08
NRatio	Neg.	0.14	0.08	0.05	0.10	0.06

Table 15: Network averages for positive and negative examples on the 2005 Wikipedia data. Except for some medline exceptions, positive scores are higher than negative scores.

Method		medline ALL 20051105	dailies PERSON 20051105	dailies ALL 20051105	goodnews PERSON 20051105	goodnews ALL 20051105
Score	Pos.	6.12	17.77	9.48	39.51	12.91
Score	Neg.	-1.00	7.48	4.57	6.97	5.73
Juxts	Pos.	33.00	19.00	22.00	13.00	12.00
Juxts	Neg.	18.00	11.00	9.00	9.00	8.00
Real	Pos.	13.96	12.13	11.72	9.24	7.83
Real	Neg.	-4.20	5.99	4.31	5.96	5.26
Spur	Pos.	13.05	3.6	6.36	1.3	2.67
Spur	Neg.	27.97	3.48	3.73	2.82	2.89
Ratio	Pos.	1.69	3.71	2.38	5.00	3.14
Ratio	Neg.	-0.18	2.10	1.52	2.11	1.86
Nbrs	Pos.	19.00	3.00	15.00	1.00	6.00
Nbrs	Neg.	45.00	3.00	17.00	3.00	12.00
NRatio	Pos.	0.17	0.06	0.05	0.04	0.05
NRatio	Neg.	0.13	0.05	0.03	0.06	0.03

Table 16: Network medians for positive and negative examples on the 2005 Wikipedia data. Except for some medline exceptions, positive scores are higher than negative scores.

Chapter 6

Discovering Entity Communities

6.1 Introduction

To understand the world, we must learn about its communities. There is a two-way path of knowledge; we learn about an entity by learning what communities it is a member of, and we learn about communities by examining its members. We seek to discover communities from news entities mined from the Lydia project (www.textmap.com).

Our approach to community discovery uses techniques from network science and social network analysis. The information needed to discover communities is contained in the co-occurrence network of the Lydia data. The co-occurrence network has named entities as vertices, and an edge between two named entities if they co-occur in a sentence in a news article. The weight of the edges is the frequency of co-occurrence. A pruned example of the network is shown in Figure 25 (Edge weights are not shown).

Typically, a few members of any community are known. Ask the average person to name 10 baseball players, and they could run them off without pause. Think now how their response would differ had you asked them to name *every* baseball player. Our version of the community discovery problem attempts to leverage this partial

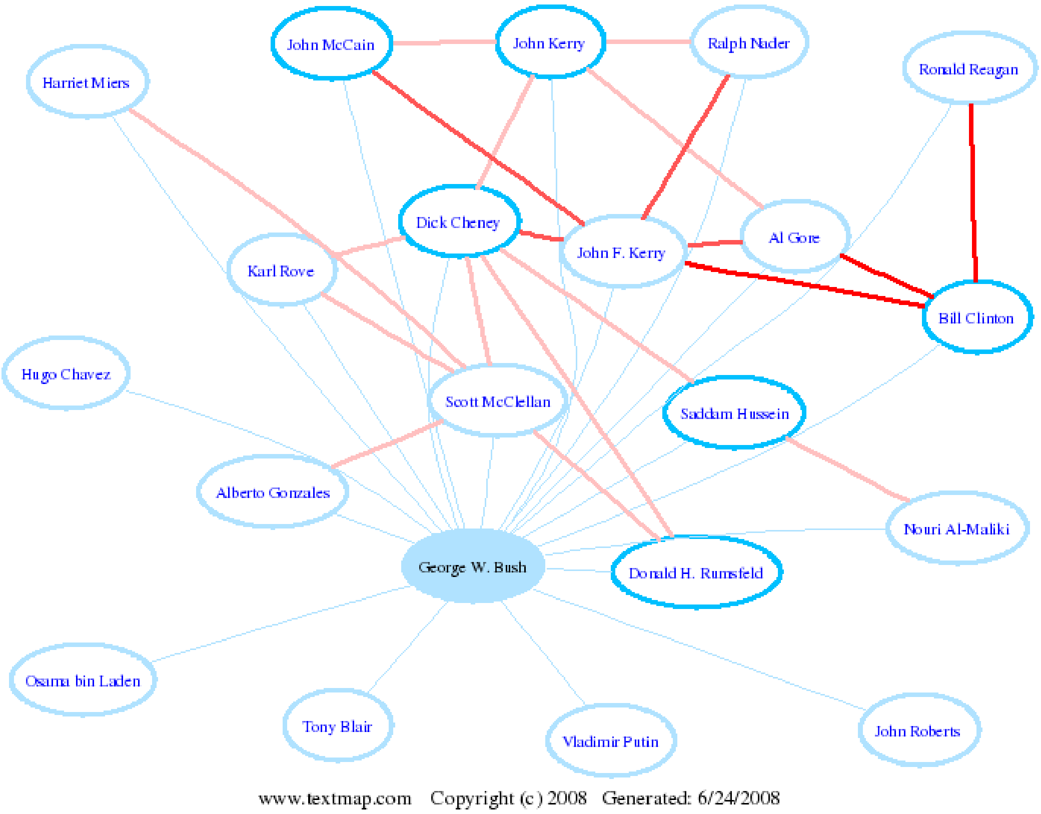


Figure 25: A partial view of a co-occurrence network.

knowledge of a community. Starting with an initial seed set that is a sampling of a community, we seek to discover as much of the full community as possible.

Community discovery is a fundamental classification problem in social network analysis. Once communities are discovered, interesting analyses result. Knowledge of communities can assist classic NLP tasks such as information retrieval and question answering. The communities can also themselves be a topic of study. The evolution and growth of a community can be predicted. The interaction between communities can be studied. Ideology of a community can be observed, and its movement predicted. In addition, the flow of ideas between communities can be studied. In the context of Lydia, once we have known communities and we can observe community-wide sentiment, we can predict how sentiment of individuals interacts with community sentiment and how a communities sentiment effects one of its members.

In its most general form, community discovery can be considered a clustering problem; and as such is as hard as clustering. In our context, several heuristics present themselves, but none offer a completely satisfactory solution:

- *Reference lists / gazetteers.* One solution that offers itself is the use of reference lists (say from Wikipedia). Most curated lists, such as those found on Wikipedia, will be incomplete; or at least lag in completeness relative to the speed of news. Certain lists may also just not exist. Others may contain a preconception, or bias. Such would be the case where members are self-identified. The reality of an individual's company may differ from how they perceive themselves (or want themselves to be perceived). Finally, the names used from an external list may not be consistent with the names used in a network, adding an extra layer of complexity.

- *Relation Extraction.* Another means of discovering communities could be to read them from the text. The text says “Democrat Bill Clinton announced yesterday..”, and thus we know Bill Clinton is in the community ‘Democrat’.

This method may be possible but it just shifts the hardness from a hard discovery problem to a hard NLP problem. Determining Bill Clinton is a Democrat from reading text is a difficult NLP problem. Even if it could be done, not all communities are mentioned as explicitly as our example; particularly for low volume entities.

General solutions to our problem also have many hurdles to overcome:

- *Network Noise.* Analytical methods don't start with perfect data. Our input data was computed from imperfect named entity recognizers, classifiers, co-reference resolution, and other NLP tasks. The network also contains noise, such as spurious relations, from the way news stories are written.

- *Multiple Membership.* A problem also arises with people that bridge communities. Say we are trying to discover an entertainment community, and have Arnold Schwarzenegger as a member. His political friends may start to creep into the community. Eventually, if enough political entities creep in, we will be growing the wrong community.

- *Entity Disambiguation.* A similar problem arises when different people share the same name, such as with John Edwards, which is the name of both a Carolina congressman, and an Indiana Pacers Center. This will cause the two communities to be bridged.

- *Community Granularity.* Specific groups can also be difficult to discover. Trying to identify 'Baseball Players' will often result in identifying all people related to baseball (managers, agents, owners, etc.)

Our contribution is as follows

- We give a general method for community discovery from seeds.

- We demonstrate our methods work well on real world networks, even given very small seed groups (20 - 400 members).

- Our methods are fast, and incremental.

- Our methods allow local community discovery.

- We evaluate parameter optimization to maximize performance.

Our method can be divided into two tasks. First, as an incremental method, given a set of members of a community, find which vertex is most likely to also be a member. Second, determine when the next member to add is likely *not* to be in the community. To put it simply, we have one phase to determine the next member to add, and another phase to determine when to stop adding members. As can be seen in Figure 26, identifying this stopping phase is just as important as the growing phase. This figure shows what happens as a grower tries to expand a seed set. The x-axis shows the number of members in the expanded community; as the grower is expanding the community, we move along the x-axis. The 3 shaded regions represent the size of the seed set, the number of correctly identified baseball players, and the number of falsely identified baseball players. Starting from the seed set, we see initially that almost all added members are correctly identified baseball players. For the first 1000 members, we see almost no false positives. For the next 1000 members, we start to see non-baseball players creeping into the community (The darkest shaded region), but still a significant number of baseball players being added. Eventually, once the community gets too large, the grower has lost all ability to distinguish baseball players, and is adding correct members at a rate no better than random. Identifying where this breaking point happens is an essential component of reliably recognizing communities.

We also notice that even at this breaking point, only about a third of the community has been found. The community appears to have an easily discover-able core of members, and many harder to distinguish outliers. (As we will see later, the

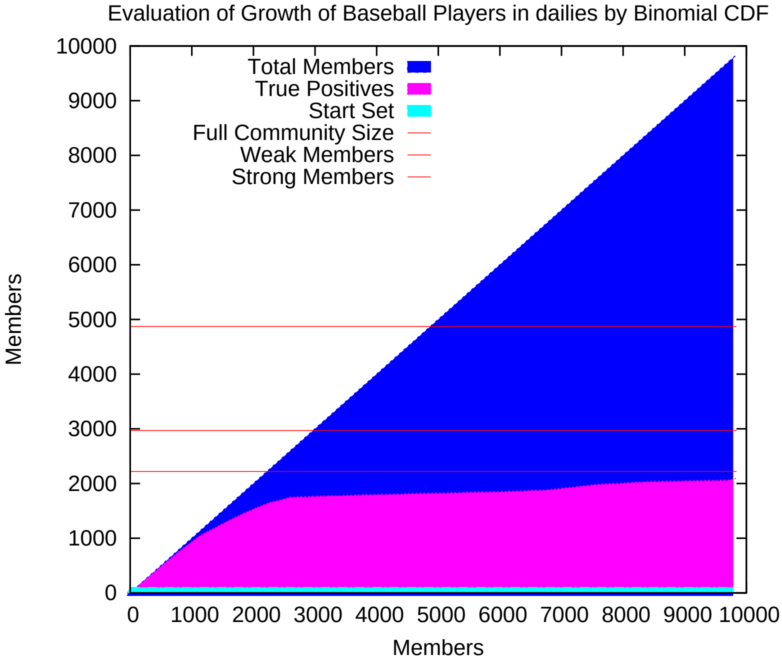


Figure 26: Growing the community ‘Baseball Players’. The first thousand members added to our seed group of 100 are almost exclusively baseball players. The fraction of baseball players in the next thousand members start to fall off. After about two thousand insertions the grower is adding members seemingly at random.

problem isn’t as bad as it appears, and can partially be explained by difficulties in evaluation techniques).

6.2 Related Work

The two main problems with communities are discovery and identification. Discovery is concerned with finding a group of entities that are members of a community, while identification seeks to identify what a community is given its members. Our problem is purely a discovery problem, since it is assumed that the seed community’s identity is already known.

6.2.1 Communities

Our notion of a community is external from the network. A community is a group existing in the real world. Although real world communities should have properties that translate to the network representation, they are not defined by the network.

Typical definitions of communities however depend solely on network properties. Generally communities are groups of vertices that are better connected within the community than outside of it [58], as in ‘ Web Communities’, which are defined as a set of vertices each with more neighbors in the community as out of it [22, 27]. It is expected that a definition of a community within the network will necessarily produce a real-world community.

Web Communities are a specialization of graph alliances [21]. Specifically, a defensive alliance is a set of vertices where each has at least as many neighbors in the alliance, than out of it (a *strong* defensive alliance has strictly more neighbors in the alliance). Offensive alliances are a set of vertices where the vertices’ neighbors each have more members in the alliance than outside of it; and a global alliance is both offensive and defensive. The complexity of finding alliances of a given size k is NP-complete [13, 20, 45, 73], but also shown FPT (fixed parameter tractable) [21].

6.2.2 Discovery Methods

Community discovery aims to find groups in the network that have strong connections. Among members of a group, there will be lots of connections (high density), while there will be much fewer connections between different groups. Discovering communities is typically viewed as a clustering problem, with specific techniques being more applicable to social networks. There have also been real world studies, such as Kossinets’ and Watts’ study of the Yale email network [44]. A large class of methods deal on a global scale. The output of these algorithms is to assign every single vertex to a community. An overview of these methods follows.

6.2.2.1 Bisection Techniques

Bisection techniques attempt to partition the network by repeated bisections. Most methods perform well for a single bisection, but not always so well for more than two groups. Also, an external decision needs to be made to indicate when to stop bisecting (how many groups to stop with) [58].

- *Max Flow - Min Cut.* Min Cut algorithms can produce good bisections, but make no guarantees on keeping both groups of similar size. Flake, Lawrence, and Giles [23] give a min-cut algorithm based on min-cut trees. This algorithm is actually able to produce an arbitrary number of clusters, and can be expanded to produce a hierarchical clustering.

- *Spectral Bisection.* Spectral bisection techniques partition a graph based on the eigenvectors of its Laplacian. The Laplacian Q of a graph G is defined as $Q = D - A$ where D is an $n \times n$ diagonal matrix with $d_{v,v} = d(v)$, and A is the *nxn* adjacency matrix of G. Since all rows and columns of Q sum to 0, Q has 1^n as an eigenvector, with eigenvalue $\lambda_1 = 0$. If the graph had two connected components, there would be eigenvectors with value 1 for vertices in the component, and 0 otherwise. For a real network where the two communities aren't completely disconnected, there will be a small eigenvalue with eigenvector a linear combination of eigenvectors of the perfect group splits (the vector of 1's for members of the groups, and 0 otherwise). The spectral bisection method finds the eigenvector corresponding to the second smallest eigenvalue λ_2, and bisects the graph on weather the eigenvector entry for a vertex is positive or negative. λ_2 is also called the algebraic connectivity of a graph. A smaller value indicates a better split into 2 groups [66, 58]. This method relies on calculation of eigenvectors. Using the Lanczos method, this can be calculated in $O(m/(\lambda_3 - \lambda_2)$. Also, the median entry of the eigenvector can be used to split the vertices into more equally sized sets if 0 produces a poor split.

- *Kernighan-Lin Algorithm.* The Kernighan-Lin algorithm [41] is a heuristic algorithm that attempts to greedily minimize the "external cost" of a partition, which is the sum of the cost of inter-partition edges. It starts with an initial partition n (possibly random), and determines the pair of vertices whose swap would produce the largest decrease in cost. In then swaps these, and determines the next pair, not considering already swapped vertices. This gives a sequence of vertex swaps which is then scanned to find the minimum. The procedure is then repeated with the new partition as the starting point, until convergence on a local minimum is achieved. Sizes of the two partitions must be given, although ranges can be specified by adding dummy nodes (vertices with all edge costs 0). The run time of the described algorithm is $O(n^2)$, but additional heuristics on pair selecting can be used to speed up the heuristic.

6.2.2.2 Hierarchical Clustering

Hierarchical clustering produces clusters of various degrees of similarity, often represented as a dendrogram. The techniques for hierarchical clustering are driven by a similarity measure between the vertices of a network [72], which is usually application specific.

- *Agglomerative.* Each vertex initially belongs to its own cluster. Edges are considered in order of similarity to merge clusters into larger clusters. (See the section below on similarity measures). In *Single Linkage*, clusters are formed whenever two components become connected. For *Complete Linkage*, clusters are formed whenever all links between two clusters are added.

  Newman [58] gives an algorithm based on *modularity* Q. Given a partition of the vertices, define a matrix e where e_{ij} is the fraction of edges in G between components i and j. Then Q is defined as

$$Q = \Sigma_i e_{ii} - \Sigma_{ijk} e_{ij} e_{ki} = Trace(e) - ||e^2||.$$

At each step, choose to merge the two clusters that causes the greatest increase in Q. The running time is $O(mn)$.

Agglomerative clustering methods don't find peripheral members reliably [59]. There is also an added level of complexity for determining at which level of the hierarchy gives the best communities.

- *Divisive.* In divisive hierarchical clustering, the entire graph G is started with as one cluster, and edges are removed to break the cluster into smaller clusters (as opposed to agglomerative where clusters are joined to larger clusters.

  Girvan and Newman [28, 59] give an algorithm based on edge betweenness centrality. The edge with highest betweenness centrality is removed from the graph until no edges remain (centrality is recomputed after each removal.) Edge betweenness can be calculated in $O(mn)$, giving a total computation time of $O(m^2n)$. The running time was improved by Tyler et al. by randomly sampling the vertices to compute betweenness on.

Clauset, Moore, and Newman [15] go on to state that hierarchical structure is actually a defining component of social networks; sufficient for power law degree distributions, high clustering coefficients, and short path lengths (small world). The *hierarchical random graph* model is a dendrogram, with probabilities at internal nodes. The probability of an edge between two leaves is equal to the value in their lowest common ancestor. This model produces networks exhibiting the properties of small-world networks. They also give a statistical based algorithm for inferring the most likely hierarchical random graph model from a given network.

6.2.2.3 Other methods

Not all community discover methods seek to partition the network. Hopcroft, Khan, Kulis, and Selman [36] give a method of using agglomerative clustering to find ‘natural communities’. The idea is that not all individuals will naturally fall into communities. A ‘natural community’ is a stable community; a cluster that should still exist if the network is slightly perturbed. By perturbing the network, and seeing which clusters consistently are found, the ‘natural’ and ‘stable’ groups can be found. Efficiently perturbing networks and stability of clustering algorithms (HITS and PageRank) can be found in [60]. This work is also extended in [37], where the goal is then to track these ‘natural communities’ over time. Other methods include:

- *Resistor Networks.* Wu and Huberman [83] give a clustering method based on considering the network as a resistor network and clustering vertices based on similar electrical potential. This method scales to extremely large graphs (linear run time), and can also be modified to extract a single community from a single node (i.e. a single seed).
- *Core Collapse Sequence.* A k-core is a component of a graph G where each vertex has degree k or larger. The core collapse sequence looks at the sequence of cores for $k = 1, 2, \ldots n-1$. [72]

6.2.3 Similarity Measures

Two entities are structurally equivalent if they have the same set of neighbors. Various measures have been proposed for measuring the degree of equivalence for two entities that are not completely equivalent, including Euclidean distance and Pearson correlation. Euclidean distance is the distance between the two vertices adjacency vectors. The distance between vertices i and j is

$$x_{ij} = \sqrt{(}\sum(a_{ik} - a_{jk})^2)$$

where a_{ik} are the entries of the adjacency matrix A [12]. Pearson correlation is defined as

$$C_{ij} = \frac{(1/n)\sum_k (a_{ik} - \mu_i)(a_{jk} - \mu_j)}{\sigma_i \sigma_j}$$

where $\mu_i = deg(i)/|V|$. Other measures can also be defined, the reader is referred to [12].

Much study has been done on co-citation networks. Classes of similarity metrics derived from this application are termed Bibliometric. Balakrishnan and Deo [5] give a similarity metric inspired by Bibliometrics.

$$\frac{|N(u) \cap N(v)|}{min(d_u, d_v) + 1}$$

where $N(u)$ is the neighborhood of u.

6.2.4 Growing From Seeds

Growing communities from seeds is done on a smaller scale with Google Sets (http://labs.google.com/sets) [14]. A user can enter up to five items, and the set is expanded to 15 or 30 items. Inspired by this, Ghahramani and Heller [26] developed the idea of Bayesian Sets. Using a statistical model of sets (communities), and Bayesian inference, complete sets could be grown from seeds on a similar scale to that of Google Sets. Our problem differs from Google sets in that we are attempting to grow much larger communities (up to thousands of members); but also rely on larger seed sets (tens or hundreds of members).

Flake, Lawrence and Giles [22] examined the problem of finding web communities, where a community was defined as a set of sites, each with more neighbors in the community than out. They found that communities can be efficiently discovered as a max-flow / min-cut problem, if the source set contained members of the community, and the sink contained non-members. Thus with seeds for the community, and seeds for non-community members, the community could be discovered. They also show approximation algorithms that work on just a local view of the network.

Thelen and Riloff [78] give a method for learning semantic lexicons from seed sets. Their method is NLP based (not network based), using pattern matching such as ‘A was arrested’, to learn new members. They also explore the benefits of simultaneously growing classes.

Sarmento et al. [69] grow entity classes from very small seed sets. They seek to estimate the membership function $\mu(S,e)$; which is a measure of if entity e belongs to the same classes as the seed set S. This is done by a cosine similarity score on the co-occurrence vectors; where a co-occurrence means the entities appear in a list structure (such as “A, B, and C”).

Our problem of growing the communities from seeds resembles minimally supervised learning, and bootstrapping. Supervised learning uses large amounts of training data to construct a classifier. Unsupervised learning seeks to construct a classifier without training data. Since this is usually an extremely hard (sometimes impossible task), minimally supervised learning attempts to construct a classifier using a very small amount of training data. These techniques are useful for quickly constructing classifiers on lesser known domains, where a large amount of training data is unavailable.

6.2.5 Temporal Growth

Our problem attempts to grow a community from a partial view (seed set). The communities formed by entities are not themselves static in time. A related problem would be to predict temporal changes from a complete view. Members will come and go as they please, and groups will grow or shrink in size. For instance, we would like to know the probability p of a particular entity joining a particular group. Current research has shown it is possible to predict these changes in a community based on its current structure. The change in a community is usually a very small fraction of the community itself. Thus a very small number of members are being predicted, starting from a very large number of members; as opposed to our problem of finding a large number of members from a small number. Because only a very few members will join, there is large prior probability of *not* joining. Thus even though the methods have some predictive power, accurately predicting individual membership remains elusive. Predicting overall size change is somewhat more attainable.

Backstrom et al. [3] worked on predicting changes from current network properties. They used a decision tree technique,where they predetermined properties of the network that would be used as a feature vector. To train, they took snapshots of LiveJournal and DBLP co-authorship networks at different points in time. They found that the most important feature determining membership is not just who your neighbors are, but how your neighbors are connected.

Sarkar et al. [68] tracked group dynamics by first reducing entities to a latent space model. This reduced dimension allow entities to be considered as spatially separated only, and Markov chain models could be used to predict movement.

6.2.6 Real World Networks

The Internet can be cited as the reason network science has recently begun to be studied [7]. Barabasi explains that before the Internet, there was simply no data available for large and reliable networks. Internet structure has revealed many properties that are then found in other networks.

Gibson, Kleinberg, and Raghavan [27] examined the link topology of the world wide web. They discovered that communities exist on the web. These communities have “authoritative” pages, and are linked together by “hub” pages.

Tyler, Wilkinson, and Huberman [80] discovered organizational community structure through examining emails. The network they considered was formed by the to/from pairs of email. They then used a divisive betweenness based technique for discovering communities. The sample contained 485 HP Lab employees, and 185,773 emails.

6.3 Expanding a Community

Our task of expanding communities lies somewhere between discovering communities, and predicting growth. Discovery tasks typically produce a clustering of sorts, based entirely on network structure. Our task differs in that we know initial seed members. This gives both a starting point for a community, and also limits discovery to a single community. Also, we know what community is being looked for; as opposed to a discovery task that outputs a group of members that belong to some community, but gives no indication of what that community is. Also, community discovery is a global problem, seeking to partition vertices into communities. Our version can operate on a local level, discovering just a single community if wanted.

Prediction tasks are concerned with how the community is changing over time. Given the complete membership of a community, it seeks to find which people are most likely to join in the future, and which members are likely to leave. Our task is to take an incomplete membership of a community, and predict what the complete membership is, but at a static time period. Usually the members of communities in a prediction task are self-identified; as in say a Myspace group. It is possible that the new members really are already functioning as a community member, but just have not identified themselves as such yet, and as such the self-identified communities aren't a completely accurate representation of the real world.

6.3.1 Selecting The Most Likely Next Member

The essential function of a community grower is to choose the next member to add to the community. This is achieved by assigning a score to all entities of the network. Below we describe different criteria for identifying the most likely next community member.

- *Neighbor Count* - The most obvious way of finding new members is to see who has a lot of neighbors in the community. Basketball players will be neighbors with other basketball players, musicians with other musicians, etc.

- *Juxtaposition Count* - One drawback of using a neighbor count is that each neighbor is given the same weight, regardless of the strength of the relation. The edge weights on our network are co-occurrence frequencies, also termed *juxtapositions* in the Lydia project. Using juxtaposition weight instead assigns more importance to neighbors that appear more frequently with an entity.

- *Neighbor Ratio* - A failing of the counting scores is that the statuses of ubiquitous entities get artificially elevated. An extremely popular entity like “George Bush” will have neighbors from many communities, since he has over a thousand neighbors. Perhaps six of these neighbors are chemists, compared to John Dalton, an entity that has only 8 neighbors (5 of which are chemists). The raw neighbor count score would identify George Bush as more likely to be a chemist. But if we consider an entity’s total popularity, and use a ratio, Dalton is promoted to the most likely chemist.

- *Juxtaposition Ratio* - The same bias to ubiquitous entities found with neighbor counts is also present in juxtaposition counts. All of the edges to “George Bush” will have a high weight, simply because of the total popularity of “George Bush”. Using a ratio helps account for high frequency vertices. However, this ratio (and juxtaposition measures in general) suffer from being one sided. The edge weights on very rare entities will be much larger than on edges to popular entities.

- *Binomial Probability* - Using ratios eliminates the problem of popular entities being elevated, but at the cost of elevating extremely unpopular entities. If an entity had 100 neighbors, 60 of which are chemists, they would have a neighbor ratio of 0.6. Compare this to an entity that has 1 neighbor who is a chemist, for a ratio of 1. What if we consider an entity whose neighbors are chosen randomly. If the entity has n neighbors, the probability that at least k of these are chemists is

$$\sum_{i=k}^{n} \binom{n}{i} * p^i * (1-p)^{n-i}.$$

where p is the fraction of known chemists in the network. If this probability is extremely low, than it might be reasoned that the neighbors were not chosen randomly, and the entity is a member of a community.

6.3.2 Complexity of Adding New Members

Our algorithm for expanding a network is logically very simple. Our initial guess of the community is the seed set. We then continually grow the community, at each step adding the highest scoring non-member.

If done naively, adding a member costs $O(|E|)$ at each iteration. For each vertex, we need to check which of its neighbors are in the community to get the score. This is because all of our scoring methods depend solely on neighbors. But this dependence leads to a performance improvement. Thus when a vertex is added to the community, the only vertices whose scores will change are the neighbors of the added vertex. In addition, the vertices can be kept in a priority queue to easily identify the highest scoring member. Since a vertex can only be added once, each edge is only considered once. Combined with the cost of maintaining a priority queue, we get a complexity of $O(|E|log(|V|)$ (The priority queue is on vertices so has at most $|V|$ elements). Our algorithm then, whenever a vertex v is added to the community, only updates the scores of the neighbors of v.

6.4 Validating a Community

The methods for growing a community described in the previous section are useful for adding members to a community, but do not provide a clear answer on when to stop adding members. Figure 26 shows what happens when a grower is allowed to continuously add new members. Initially, most of the members added are indeed members of the community. Eventually, at around 700 members, we see a shift in composition , and most of the new members added do not belong to the community. The grower appears to have expanded outside the community. If we could stop the growth before it enters into this second phase, we would optimize the quality of the expanding community.

6.4.1 Stopping Rules

If we knew what Figure 26 looked like, properly terminating the growth would be easy. However, all we are given is a (small) subset of the community, and no other validation information as to which insertions are invalid. If we had a subset of the community, we could monitor how frequently these members are being added by the grower. In the first phase, when nearly everything being added is correct, we expect to add a new validation member with frequency equal to the percent of the community the validation set is (for example if our validation set is 5% of the total community, we would expect to see a validation member once about every 20 inserts). Once we shift into adding arbitrary entities, we expect to see validation members with frequency equal to the percent of the entire network composed of the validation set, which is much much less than the first percentage.

To precisely find the correct stopping point, we find the point that best splits the validation intervals into two groups. Since all of the intervals on one side of the ideal split should be about the same, we expect the deviation of the intervals to be small. We then find the split that minimizes the absolute deviation. The stopping point s is

$$\text{stopping point} = argmin_k(\sum_{i=0}^{k}(|x_i - \mu(x_0,\ldots,x_k)|) + \sum_{i=k+1}^{n}(|x_i - \mu(x_{k+1},\ldots,x_n)|)).$$

where the function μ is the arithmetic mean of its arguments. For example, suppose the validation members are found on inserts 3, 7, 10, 11, 16, 18, 21, 120, 203, 290, 387, 506. This yields an interval sequence of $\{3,4,3,1,5,2,3,99,83,87,97,119\}$. The optimal stopping point is after the seventh insert, which gives absolute deviations of

$$\begin{aligned}
\sum(|x_i - \mu(3,4,3,1,5,2,3)|) + \sum(|x_i - \mu(99,83,87,97,119)|) \\
= \sum(|x_i| - 3) + \sum(|x_i| - 97) \\
= |3-3| + |4-3| + |3-3| + |1-3| + |5-3| + |2-3| + |3-3| \\
+ |99-97| + |83-97| + |87-97| + |97-97| + |119-97| \\
= 54
\end{aligned}$$

which is minimal for this sequence. Figure 27 shows actual intervals for various validation set sizes. We see that the intervals do start out small, and take a sudden and dramatic spike. At the 90th validation member found, the intervals suddenly become orders of magnitude larger.

6.4.2 Boosting

Our grower is sensitive to which vertices are chosen as validation vertices, and which as seed vertices. The algorithm is designed to use very small given sets, and this causes a high degree of sensitivity to which vertices get chosen for seeding and which for validation.

To improve the performance, we use a boosting technique. We run our growing algorithm multiple times, each time using a different partition of seed and validation members. Each vertex then accumulates a number of ‘votes’ for how often it is identified as part of the community. Figure 28 demonstrates the value of boosting. The x-axis represents vertices receiving at least that many votes. The blue shaded region represents false-positives; vertices that we incorrectly added to the community. As the number of votes is increased, the precision of the community members is increased. The reason being is that for any number of votes, the number

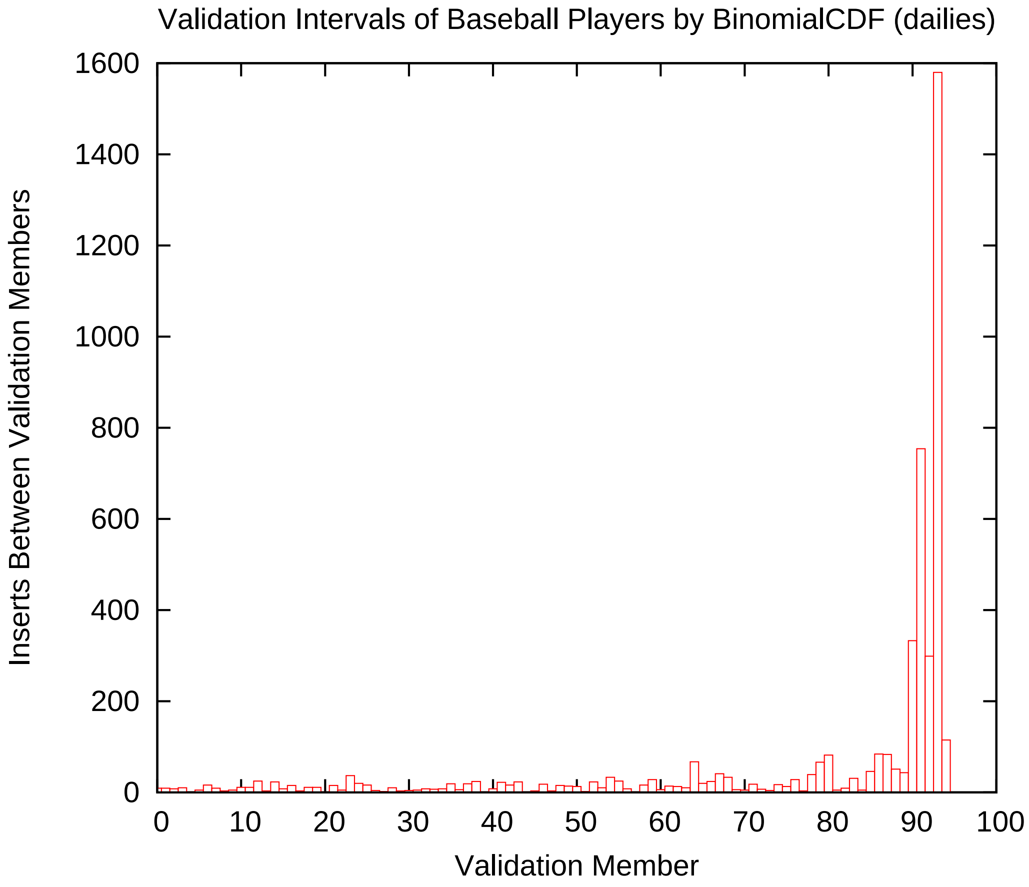


Figure 27: Validation Intervals. The height of the bar at position i represents the number of members added between finding the (i-1)st validation member, and the ith member.

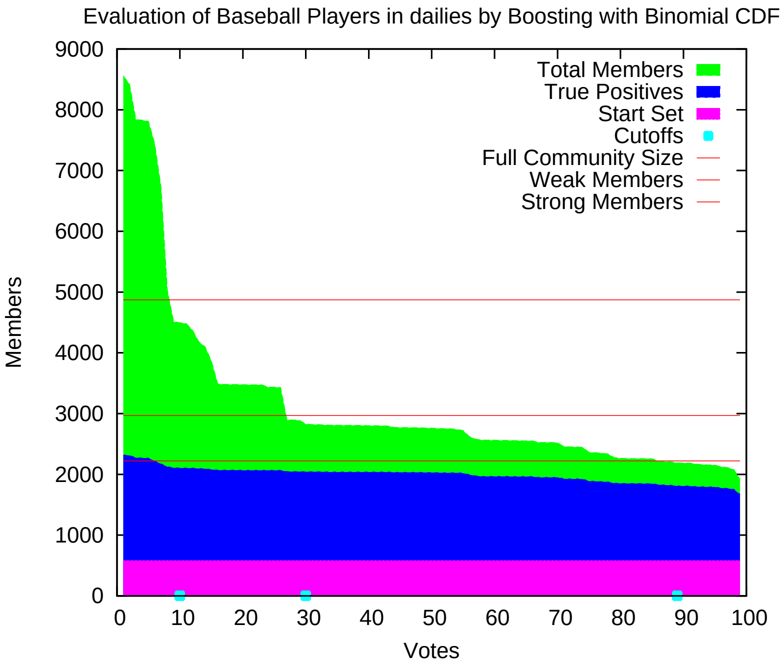


Figure 28: Boosting Performance. At around 30 votes we are finding nearly as many members as 0 votes, with much higher precision. For even higher precision, we can operate at 90 votes, with only a slight loss in recall.

of true positives is fairly constant. So the bulk of the true positives are being seen in nearly every boosting run. However, as the vote requirement is increased, many of the false positives drop out, since they are only appearing in a small number of the boosting runs.

6.4.2.1 Precision / Recall Tradeoffs

It now remains to determine what boosting cutoff to use. We achieve this by using our validation members to estimate precision and recall. The given members are divided into seed and validation for each boosting run. We use these validation members, and track how many votes they get. If we make the leap of faith that any vertex that appears in 100% of the boosting runs is indeed a true member of the community, we can estimate the percentage of the community that comprises the validation set as

$$f_{val} = \text{\#validation members with 100\%}/\text{\#vertices with 100\%}$$

Then at any vote cutoff we can estimate precision and recall by looking at validation precision and recall, and adjusting by this frequency. For a given number of votes v, the precision pre_v is estimated as

$$pre_v \approx val(v) * f_{val}/mem(v)$$

where $val(v)$ is the number of validation members with at least v votes, and mem(v) is the number of all members with at least v votes. The estimated recall is

$$rec_v \approx val(v)/val(0)$$

where $val(0)$ is simply the size of the entire validation set.

Figures 29 shows examples of estimating the precision and recall, compared to the actual precision and recall. This approximation yields an approximation of f-score, which can be maximized to set a cutoff. Going even further, we can maximize the general F-measure for any β.

$$F_\beta = (1+\beta^2) * (precision * recall)/(\beta^2 * precision + recall)$$

This now gives us a knob to turn to get precision / recall tradeoffs. If the precision and recall estimates are sufficiently accurate, we can maximize any general F-measure.

6.5 Parameter Optimization

To maximize the performance of a community growing scheme, the parameter space needs to be optimized:

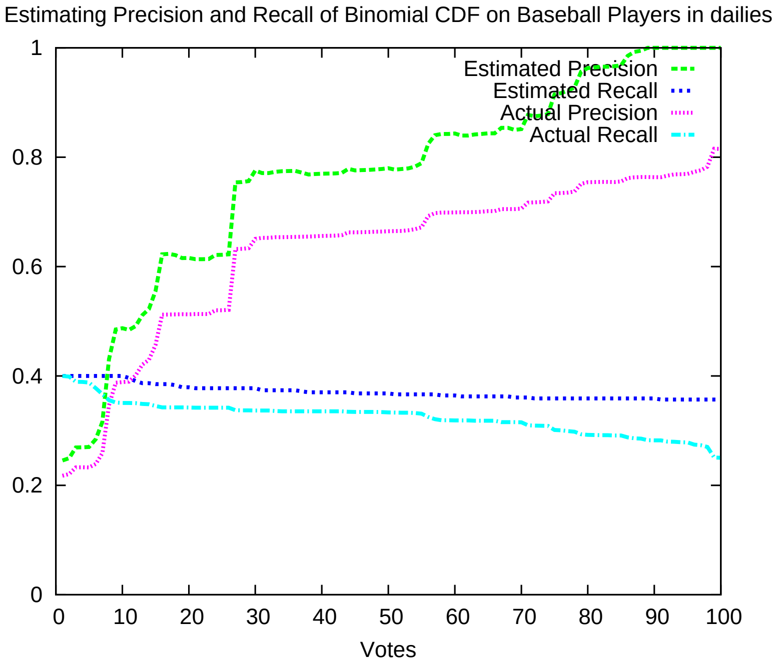


Figure 29: Estimating Precision and Recall. The estimated precision and recall curves closely shadow the actual precision and recall.

- *Neighbor Selection Method.* We compare five different scoring methods Neighbor Count, Juxtaposition Count, Neighbor Ratio, Juxtaposition Ratio, and Binomial CDF. The default grower is Neighbor Count.

- *Validation Set Size.* We compare the effects of the number of members we reserve for validation. Too few members, and there won't be enough to validate with. Too many members and we are taking away seed members. The default validate percentage is 50%.

- *Number of Boosters.* We evaluate the effect of the number of boosters. More boosters being better is our general expectation, but we ensure there is no fall-off with too many. Also, more boosters means higher computation costs. If there is no difference in performance between two sizes, we prefer the smaller size. Default value is 100 boosters.

- *Beta Knob.* We evaluate the effects of our beta knob, and ensure that a useful precision / recall tradeoff is achieved. Default value is 1.0.

- *Given elements.* In applications of our grower, the number of given elements may not be flexible. However, we evaluate our methods with different sizes of given elements to see how robust our methods are to extremely small given sets. We evaluate at 20, 200, and 400 given members.

Globally optimizing the parameter space would be very computationally intensive. Instead, each parameter is optimized individually, with reasonable choices for the other parameters. Our initial setup uses default values mentioned above. For each data point, the grower was run on five different randomly assigned given sets, and the results macro-averaged.

6.5.1 Difficulty with Evaluation

It should be pointed out that our evaluation methods are not perfect. To begin with, the assumed truth communities are not 100% accurate. The truth set for baseball for example was taken from the baseball databank (http://www.baseball-databank.org/). The completeness of the members may be lagging from real world. Also, there is a problem of entity co-reference. The named entry in the baseball databank may differ from our name. For instance, the databank may list ‘Larry Jones’ as a baseball player, while we only see his preferred name of ‘Chipper Jones’ in the data. Mirroring the co-reference problem is the disambiguation problem. There exist different people that share the same name. Wikipedia has over 30 entries for the name ‘John Edwards’. While current news is no doubt dominated by the former Carolina congressman, there is also a current NBA player named ‘John Edwards’, as well as a 1960s baseball player. Since our evaluator isn’t equipped with the means of disambiguating references, performance will be hurt. The baseball player grower will be penalized for not discovering ‘John Edwards’, even though none of the mentions of ‘John Edwards’ in the data was for the baseball player.

Finally, there are problems with the named entity recognition of the Lydia system that are propagated. Lydia may get the segmentation wrong, and tag ‘Outfielder Carlos Beltran’ as an entity. When this entity gets included in the community, it is evaluated as a false positive. Lydia may also get categorization wrong. If the term ‘New York Mets’ gets incorrectly classified as a Person, it will show up in the community. The community grower will call this a baseball player.

Finally, a concept such as ‘baseball players’ may be too specific for our growers to distinguish; instead the neighborhood may reflect a general ‘baseball’ category, including owners, managers, and agents.

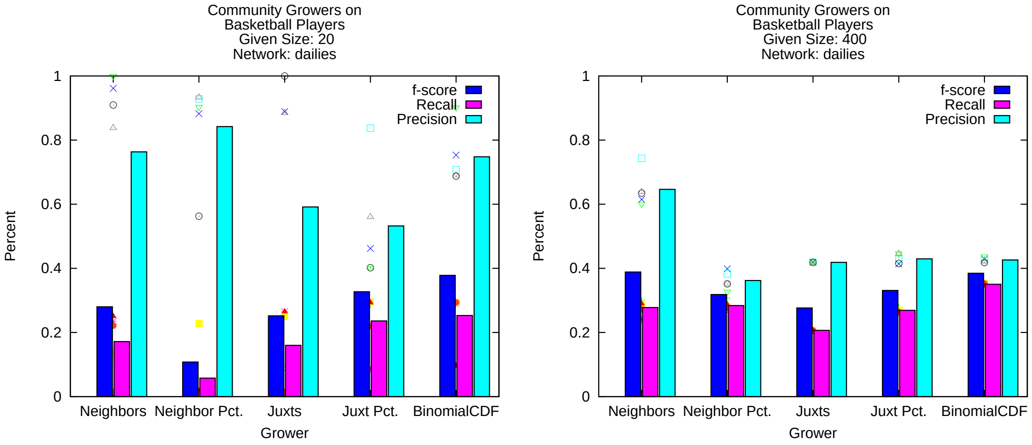


Figure 30: Evaluation of Community Growers on Basketball Players, for starting sizes of 20 and 400.

6.5.2 Starting Size

Our parameter optimization experiments were done on given set sizes of 20, 200, and 400. Since our method needs to be seeded, there may be instances where the user can only supply a very small starting set. Figure 26 shows the results for a single (no boosting) grower run, on different sized random given sets. There is some variance, since each given set is randomly generated, but we find that larger given sets do not seem to increase performance, especially recall. This gives us some insight into the structure of the community. Remember that the truth set for our community came from an external source. Within the network it appears that there is a component of the community that is closely connected, with typical community properties, and another component that is only randomly connected. The closely connected members are easily discovered with either small or large seed sets, and the other members are hard to find with both small and large seed sets.

6.5.3 Comparing Growers

Figure 30 show results of evaluating the different growers. The Binomial CDF based score consistently has the highest f-measure, and highest recall. The Neighbors count method will sometimes have a higher precision, but at much lower recall. Thus we choose Binomial CDF as our default grower.

6.5.4 Effect of Validation Set Size.

Figure 31 shows results of validation percentage evaluations. The fraction of the given set to use for validation was varied from 10% to 90% in increments of 10%, with a minimum size of three for the validation set. For a given set size of 20, there is a lot of variance. This is due to the nature of our stopping criteria. We need intervals to determine a stopping criteria. With few validation members, we get few intervals, and thus a very volatile method. With large sets, there seems to be very little difference in the choice of validation size. With too low a percentage, there are not enough members to validate with. Too high, and there aren't enough seed members to grow from. A choice of 50% offers the best balance between seed and validation.

6.5.5 Effects of Boosting

We seek to find what kind of performance gains we get from boosting. The more boosting runs we use, the better results we expect, but with higher computation cost. We see the results in Figure 32. For given sets of 200 and 400, we see very little difference in performance. For a given size of 20, we again see erratic behavior. Since there is no difference in performance, we choose 100 runs, since this will save computation time.

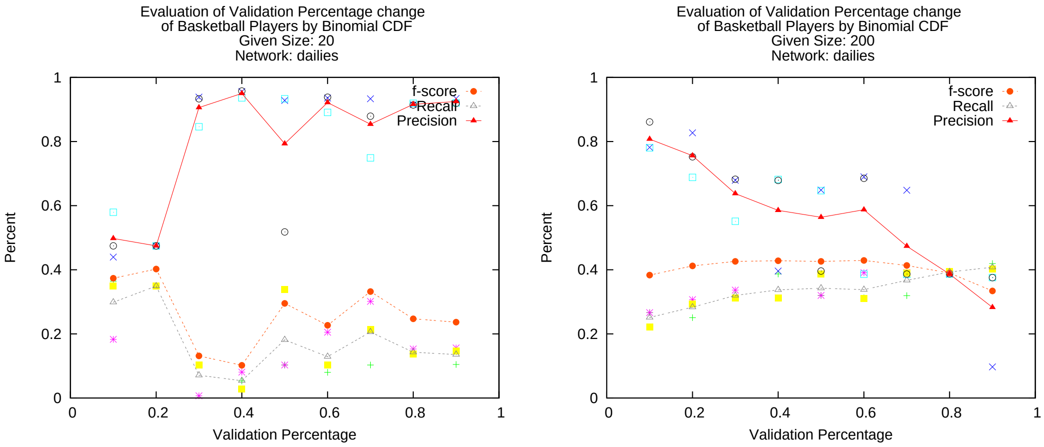


Figure 31: Evaluation of Validation Percentage on Basketball Players, for starting sizes of 20 and 400.

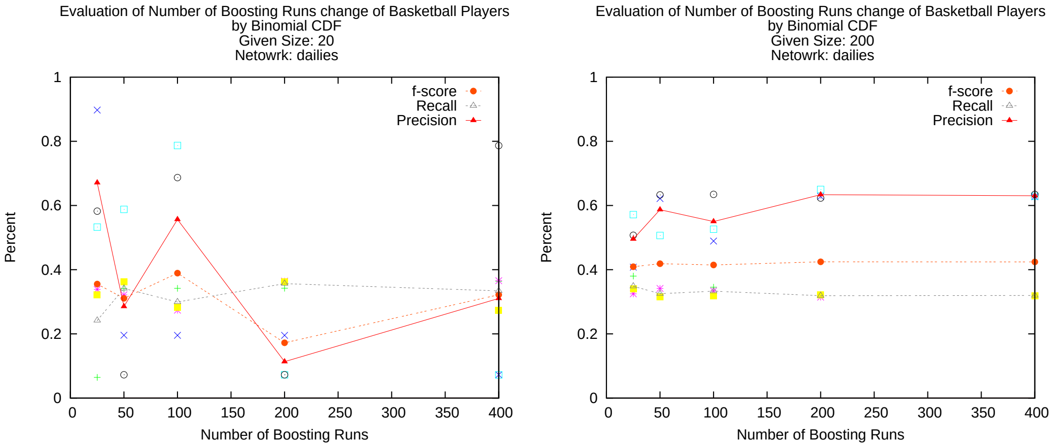


Figure 32: Evaluation of Number of Boosting runs on Basketball Players, for starting sizes of 20 and 400.

6.5.6 Effects of Beta Parameter.

As explained in section 6.4.2.1, we have a knob F_β for precision/recall tradeoff. Figure 33 shows results of turning this knob for different domains (Baseball players, Basketball players, Football players, and Movie Stars). F_β measure typically has β in the range $(0, \inf)$; with the interpretation that β represents how much more recall is weighted over precision. Thus $\beta = 1$ is an equal weighting, $\beta = 2$ means recall is weighted twice as much, and $\beta = 0.5$ means recall is weighted half as much. We make the operating points favoring precision have the same range as those favoring recall. We do this by the transformation, $\beta' = \beta - 1$ for $\beta > 1$, and $\beta' = (-1/\beta) + 1$ for $\beta < 1$.

Turning the knob positive weights recall higher than precision, and negative weights precision higher. We see the actual results are consistent with this notion. We also observe however that the growers seem to have a few optimal operating points, and not a continuum of points; as different values of beta all operate at the same point (the vote cutoff point is the same for a large range of beta values). The optimal points tend to be at the right, bottom ends of ‘cliffs’. That is, looking at the graph we see some points of sharp decline in false positives. It is clearly better to use the points just after this decline, as there is little or no difference in recall immediately before or after the decline.

6.6 Experiments in Community Discovery

The evaluations for parameters were conducted on 4 different communities, baseball players, basketball players, football players, and movie stars. In this section we discuss the setup and evaluation for each community; and we also look closely at the community growth.

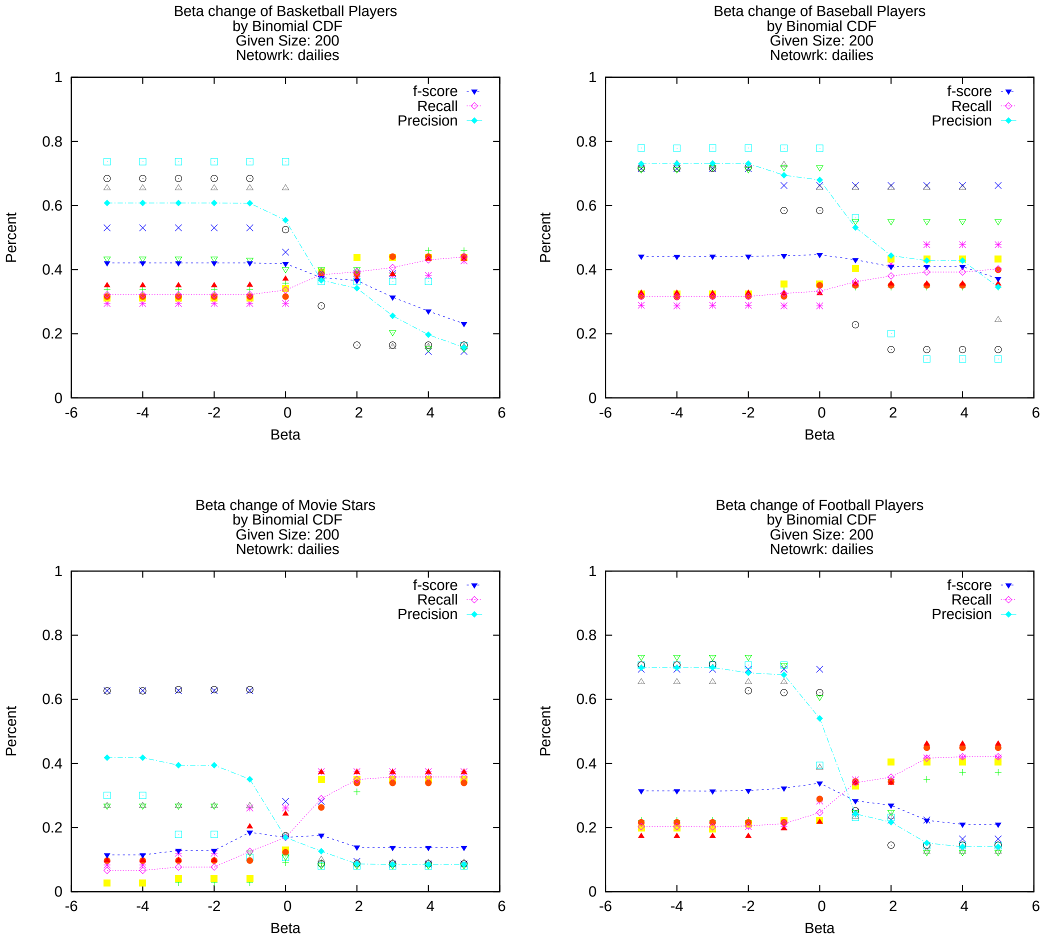


Figure 33: Evaluation of Number of Beta Knob on different domains, Baseball, Basketball, Movies, and Football.

	False Positives		False Negatives
	Laci Peterson	*	John Edwards
+	Roy Williams	*	Michael Jackson
+	Madison Square Garden		Shaquille O’Neal
+	Mike Krzyzewski	*	Bob Riley
+	Mark Cuban	*	Michael Phelps
+	Van Gundy	*	Steve Smith
+	Greg Oden	*	Mel Gibson
+	Rick Pitino		Billy Donovan
+	Gregg Popovich		Pat Riley
+	David Stern	*	Jim Davis
+	Mike Montgomery	*	Greg Anderson
+	Jim Calhoun	*	Michael Young
+	Bernie Bickerstaff	*	Bernie Williams
+	Flip Saunders	*	Mike Davis
+	Jerry Buss		Larry Johnson
+	Rick Barnes	*	Aaron Brooks
+	Van Horn		J.J. Redick
+	Paul Hewitt	*	Mike Williams
+	John Calipari	*	John Chaney
+	Lawrence Frank		Jayson Williams

Table 17: Incorrectly classified basketball players. Names labeled with a ’+’ are associated with basketball, but not necessarily players. Names labeled with a ’*’ are people that share a name with lesser known basketball players. The frequency of these labels indicates that the growers are actually growing more reasonable communities than the performance scores indicate.

6.6.1 Basketball

Truth data for basketball players is taken from basketballreference.com. To get a sense of how the community is grown, we look at the most popular entities that are mis-classified. Table 17 shows the top 20 false positives (left) and false negatives (right). The popularity is the number of references the entity has in our data. Looking at the false negatives, we see that the grower is often the victim of an obscure basketball player having the name of someone more famous. John Edwards, Bob Riley, Mike Davis, and John Chaney, most famously politicians; Michael Jackson, most famously a musician; Mel Gibson, most famously a film actor; Michael Phelps, most famously a swimmer; Steve Smith; Jim Davis, most famously a cartoonist; Greg Anderson, most famously a personal trainer; Michael Young and Bernie Williams most famously baseball players; Aaron Brooks, most famously a football player; Mike Williams, two different football players - are all also the names of basketball players. We also see on the false positive side many people that are not quite basketball players, but basketball related, such as NBA commissioner David Stern, Owners Mark Cuban and Jerry Buss, coaches Mike Krzyzewski, Van Gundy, Rick Pitino, Gregg Popovich, Mike Montgomery, Jim Calhoun, Bernie Bickerstaff, Flip Saunders, Rick Barnes, Paul Hewitt, John Calipari, and Lawrence Frank. We also see a basketball arena, Madison Square Garden, a result of poor categorization (most likely caused by ‘Madison’ being considered a female first name). Greg Oden being called a false positive is an example of the incompleteness of basketballreference.com. Oden is currently an NBA player, signed under the Portland Trail Blazers. Injuries cause Oden to miss his entire first season, and as such, he has never played in an official NBA game, so he is not yet listed as a player in basketballreference.com. Examining these lists leads us to believe that our growers are actually performing better than reported.

6.6.2 Baseball

Truth data for baseball players is taken from the baseball databank (baseball-databank.org). We again ran our grower, and examined the most popular false positives and false negatives. The results are shown in table 18. We see many of the same phenomenon we saw for basketball players. Many of the false positives are not baseball players, but people most associated with baseball. 'Winter Haven' is a classification error, being the name of the city where the Indians and Red Sox have spring training. 'League Baseball' is also an NLP error, our pipeline mistakingly thinking the 'Major' in 'Major League Baseball' is a military title. Bud Selig is the commissioner of baseball. George Steinbrenner is the owner of the Yankees. Brian Cashman, Theo Epstein, and Jim Hendry are general managers. Tony La Russa (also appearing as 'La Russa', an error in co-reference) is a manger. Scott Boras is a notorious agent. There also appear to be a number of non-baseball people who are linked to baseball through the recent steroid scandals. George Mitchell, a U.S. senator, and never previously involved with baseball, is now most famous for his 'Mitchell Report', an investigation on steroids sanctioned by Major League Baseball. Also included in the community via a steroid connection is congressman Henry Waxman (part of the congressional hearings on steroids), defamed trainer Greg Anderson who supplied many athletes with steroids, and non-baseball athletes who have been involved with performance enhancing drug scandals including: Floyd Landis who was stripped of the Tour de France for a positive drug test, Marion Jones who was forced to return 5 Olympic medals after admitting steroid use, and Tim Montgomery who was stripped of the 100m record for involvement with performance enhancing drugs.

On the false negative side, we again have a disambiguation problem. Larry Brown appears in our corpus, but always in reference to the basketball coach. Unfortunately, out evaluation only sees the name 'Larry Brown', and assumes it belongs to the middle infielder that played for the Indians in the late 60s. The same problem causes the evaluation of the grower to have false negatives for Mike Tyson, most famously a heavyweight boxer; George Washington, most famously first president of the United States and Commander in Chief of the Continental Army; Bill Richardson, most famously governor or New Mexico; Bill Nelson, most famously a senator from Florida; Paul Martin, the 21st Prime Minister of Canada; Michael Brown, most famously the former director of FEMA; Jim Davis, creator of Garfield; John Warner, a senator from Virginia; Tommy Thompson, former presidential candidate and Governor of Wisconsin; Paul O'Neil, former Secretary of the Treasury; Larry Johnson, a former NBA player; and John Fox, most famously a comedian.

Once again, inspection of what the grower is getting wrong leads us to believe that it will perform better in practice than the evaluations.

6.6.3 Football

Football players were taken from http://www.pro-football-reference.com/. Table 19 shows mis-classified football players. As usual, there is a disambiguation problem with evaluating false negatives. The false positives have accumulated many basketball players. While still in the 'athlete' category, basketball players are not otherwise related to football players. If we look closer at the actual vote count, we see that many of the basketball related entities received lower votes than football related entities.

6.6.4 Movie Stars

Truth data for movie stars was taken from the Internet Movie Database (imdb.com). The problem with directly using IMDB data is that they are often too complete. Nearly everyone of any fame has an entry in IMDB, even Bill Clinton. The difference is that non movie stars usually appear in documentaries, or appear as themselves, or have only appeared in a couple of films. To find a more classical set of

	False Positives		False Negatives
#	Lance Armstrong	*	Larry Brown
	Scott Peterson	*	Mike Tyson
+	Winter Haven	*	George Washington
+	Bud Selig	*	Bill Richardson
+	League Baseball		David Wells
#	Floyd Landis		Felipe Alou
	Eli Manning		Miguel Tejada
+	George Steinbrenner	*	Bill Nelson
#	Marion Jones	*	Paul Martin
#	Greg Anderson		Mike Brown
+	Brian Cashman		Chris Carpenter
#	George Mitchell	*	Jim Davis
+	Tony La Russa	*	John Warner
#	Tim Montgomery		Mark Mulder
+	La Russa		Mike Lowell
	Bode Miller	*	Tommy Thompson
*	Scott Boras	*	Mike Davis
#	Henry Waxman	*	Paul O'Neill
+	Theo Epstein	*	Larry Johnson
+	Jim Hendry	*	John Fox

Table 18: Incorrectly classified baseball players. Names with a '+' are people most associated with baseball, but who are not players. Names with a '#' are people associated with performance enhancing drug scandals, of which baseball was a large part of. Names marked with a '*' are people who share the same name with a lesser-known baseball player.

	False Positives		False Negatives
	Kobe Bryant	*	Michael Jackson
	Scott Peterson	*	Tony Stewart
	Shaquille O’Neal		Jimmie Johnson
	Michael Jordan	*	Randy Johnson
	Laci Peterson	*	Bob Riley
	Allen Iverson		Reggie Bush
	Richard Nixon	*	Michael Moore
	LeBron James	*	Michael Brown
	Barry Bonds	*	Bill Nelson
+	Bill Belichick		Matt Hasselbeck
+	Bill Parcells	*	George Allen
	Dwyane Wade	*	Tommy Thompson
	Dirk Nowitzki	*	Frank Robinson
	Phil Jackson	*	Michael Young
	Mike Tyson	*	Kevin Brown
	Arthur Andersen	*	Ted Williams
	George Washington	*	Gordon Brown
+	Nick Saban	*	Dan Brown
	Jason Kidd	*	Luis Castillo
	Steve Nash	*	Tim Johnson

Table 19: Incorrectly classified football players. Names labeled with a ’+’ are associated with football, but not necessarily players. Names labeled with a ’*’ are people that share a name with lesser known football players.

movie stars, we filtered IMDB's data to remove all actors whose movie list was over 25% documentaries, or who appeared in less than 3 other movies. The most popular misclassified entities were examined, and results did not look very promising. Inspection of the false negatives gives some insight to the problems. Like the other communities, movie stars has a problem with disambiguation; that is multiple people sharing the same name. For instance, in the basketball community, we saw John Edwards was the name of a basketball player. When he is not included in the community, recall gets penalized, but the community is otherwise unaffected. However, if John Edwards happened to be included in the seed set, community growth would be influenced and directed towards political members. Movie stars seems to have a high number of name clashes. Since the seed sets in our evaluators are chosen randomly, bad members could get in the seed set, and affect all growth. Instead, a hand-crafted seed set of 50 popular movie stars was constructed and used as seeds. The results of using these seeds is shown in Table 20.

These results are more reasonable than before, and can be explained better. Nearly all of the false positives are entertainment related people, movie directors, television people, or misclassified entertainment related entities (Warner Bros., Beverly Hills). Even David Beckham can be explained by either the movie with his name in the title "Bend it like Beckham", or his elevated status as a celebrity, and not just a soccer player. Lance Armstrong also has a celebrity status.

On the false negative side, there are the disambiguation problems we have seen in other communities, and also people who are most famous for something other than movies, but also have appeared in some movies, like Shaquille O'Neal, most known as an NBA player, but who also appeared in the films "Blue Chips", "Kazaam", "Steel", "Freddy Got Fingered", and "The Wash". So while he can definitely claim movie star status, in the public mind, he is a basketball player and not considered a movie star. Elvis Presley is also in this category, being more associated with music than acting.

	False Positives		False Negatives
	Michael Jackson: 113931	⊕	Shaquille O'Neal: 54253: 0
	Lance Armstrong: 77978	*	Robert Blake: 24985: 0
	Martha Stewart: 59851	*	David Wells: 21368: 0
	" Friends ": 47218	⊕	Elvis Presley: 16273: 95
	Britney Spears: 41652	*	John Howard: 16065: 61
	Donald Trump: 38343	*	Richard Hamilton: 14227: 0
+	Bob Dylan: 21038	*	Adam Scott: 12927: 0
	Beverly Hills: 18792		Bill Cosby: 11685: 80
	David Beckham: 18617	*	John Lynch: 9775: 32
	Warner Bros: 17713	⊕	Rosie O'Donnell: 9388: 82
+	Paris Hilton: 17294		Willie Nelson: 8781: 94
	David Letterman: 17014	*	Chris Young: 7440: 0
+	Steven Spielberg: 16695	*	Eddie Jones: 6906: 0
	Paul McCartney: 16561		Woody Allen: 6906: 61
	Katie Couric: 16172	*	Vernon Wells: 6851: 0
+	Ray Charles: 14609		Tim McGraw: 6435: 96
	Oprah Winfrey: 14545	*	Mike Smith: 6314: 0
+	Martin Scorsese: 12161		John Wayne: 6122: 87
	Elton John: 11397		Jane Fonda: 5995: 92
	Simon Cowell: 10643	*	John Abraham: 5660: 0

Table 20: Incorrectly classified film actors, grown from manually set seeds. People marked with a '+' are movie related, if not primarily actors (Dylan and Charles the subjects of recent films). The other false positives are entertainment related, but not movie actors. People marked with a '*' have a name clash with non movie stars. People marked with a '⊕' have been in enough films for IMDB to call them an actor, but in everyday news are primarily associated with some other community (O'Neal with sports, Presley with music, and O'Donnell with daytime television).

Robert Blake is most famously a hockey player; David Wells is most famously a baseball pitcher; John Howard is the 25th Prime Minister of Australia; Richard Hamilton is most famously a basketball player; Adam Scott a professional golfer; John Lynch a football player; Chris Jones a baseball player; Eddie Jones a basketball player; Vernon Wells a baseball player; Mike Smith a hockey player; John Abraham a football player.

	Network	Baseball	Basketball	Football	Movie Stars
Vertices	144,851	3,587	1,276	4,699	2,171
Edges	265,779	21,565	5,879	11,256	5,718
Intra-Comm. Deg.	3.67	12.02	9.21	4.79	5.27
Density	.000025	.003353	.007227	.00102	.00243
Bridges	-	16,510	9,787	24,181	14,921
Bridge Deg.	-	4.6	7.67	5.15	6.87
Strong Vertices	-	2,020	639	2,206	1,008
Weak Vertices	-	339	203	761	482
Isolated Vertices	-	1,228	434	1,732	681

Table 21: Community properties in goodnews data. The community regions are higher density, and vertices will typically have more neighbors inside the community than outside. Isolated vertices are detrimental to the structure of the community, and are possibly an artifact of bad evaluation lists.

	Network	Baseball	Basketball	Football	Movie Stars
Vertices	299,486	4,872	1,653	6,514	2,703
Edges	594,884	36,509	10,358	16,745	8,081
Intra-Comm. Deg.	3.97	14.98	12.53	5.14	5.98
Density	.000013	.003077	.007586	.00079	.00221
Bridges	-	39,107	21,553	52,458	25,355
Bridge Deg.	-	8.03	13.04	8.05	9.38
Strong Vertices	-	2,221	735	2,491	1,161
Weak Vertices	-	748	348	1,584	772
Isolated Vertices	-	1,903	570	2,439	770

Table 22: Community properties in dailies data. Community regions are of higher density. Isolated Vertices still corrupt our communities.

6.6.5 Summary of Communities

A summary of community properties in the network is shown in Table 21 and 22. These are the network properties of the gold standard community, not grown communities (The vertex set is the gold standard set). The columns show properties for each of the four communities, with the first column showing properties of the entire network. The first row, *Vertices* is the number of members of the community. The second row, *Edges* is the number of intra-community edges (edges between two members of the community); as opposed to the *Bridges* row which is the number of edges with exactly one end in the community. Average Intra-Community Degree gives the average number of neighbors a community member has in the community, and average bridge degree gives the number of neighbor outside of the community. To further inspect the communities, we classify the vertices as either being 'Strong', 'Weak', or 'Isolated'. A Strong vertex is a vertex that has at least half of its neighbors in the community. An Isolated vertex is one that has none of its neighbors in the community. A Weak vertex is one that is neither Strong nor Isolated (Less than half of its neighbors are in the community, and at least one of its neighbors is a community member).

All of the communities have densities of about 2 orders of magnitude greater than the network as a whole. This is a typical property of communities. Many general definitions of communities describe them as regions of higher density.

As we have discussed, we believe part of our evaluation problems are caused by imperfect gold standards. The large number of Weak and Isolated vertices indicates that this is not a trivial problem. Except for baseball players in goodnews, all of the communities have over half of their vertices weak or isolated. This means that most vertices have more neighbors outside the community than inside (also reflected in the average degrees). When these vertices are included in seed sets, there can be negative results, highlighted by the movie star example of the previous section.

Not surprisingly, the recall for our growers tops out at around the same value as the number of Strong vertices. Isolated vertices should not be expected to be found during growing. In fact, since our growers only consider neighbors, an isolated vertex can only be added if incorrect members are added before it. Thus a high precision grower will miss nearly all isolated vertices, over a third of the community in most cases.

Chapter 7

Conclusions and Future Work

The Lydia project provides a unique large scale network of real world entities. The addition of community information will promote deeper understanding of the world, and allow further analysis on the data. Topics for future work are outlined below.

7.1 Future Work

7.1.1 NLP Scored Edges

Working with a network allows us to perform analysis we could not otherwise accomplish. However, we can still leverage some NLP tools, especially in creating edge similarity scores. One method we would like to explore is a similarity score based on entity word profiles; That is, what is the frequency of words that appear in the same article (or sentence, or window) with an entity. Babe Ruth will have words such as ”home”, ”run”, ”baseball”, ”world”, ”series”, ”Yankees”, which would be very similar to the profile of other baseball players. George Bush on the other hand will have ”President”, ”Iraq”, and ”administration” as high frequency words, and be very dis-similar to Babe Ruth. Michael Jordan may have his mid-frequency words

”play”, ”win”, ”champion”, ”great” in common with Babe Ruth, giving them a similarity in the middle.

7.1.2 Changes in Communities/Network Over Time

The previous problem concerned itself with a snapshot of our network at a specific time. We would also like to know how the communities evolve as time progresses. We want to be able to predict new membership of communities, how much a community will grow or shrink, and predict the emergence of new communities.

The communities formed by entities are not themselves static in time. Members will come and go as they please, and groups will grow or shrink in size. For instance, we would like to know the probability p of a particular entity joining a particular group. Current research has shown it is possible to predict these changes in a community based on its current structure.

7.1.3 Discover Dominating and Sibling Entities

We often notice in our network that some entities have nearly all the same neighbors (structural equivalence.) For instance, ‘Bill Clinton’ and ‘William Jefferson Clinton’ (both strings refer to the same person) would have many of the same neighbors, but so to would ‘David Ortiz’ and ‘Manny Ramirez’ (both play for the same baseball team.) Another situation is when the neighbors of one entity are nearly a subset of the neighbors of another entity. For example ‘David Ortiz’ and ‘Red Sox’, when one entity is part of the other entity. This can also occur when an entity has a representation that is very infrequent, and usually reserved for a specific setting. Finding dominating and sibling entities can be used to improve co-reference, and discover hierarchies and groups.

7.1.4 Classify Entity Type

The domain of entities of news articles is extremely large, and as a result classification of entities leads to many UNKNOWN entities. Currently, Lydia uses hand made rules, lists, and statistical classification. We can also use position in the network to classify some of these UNKNOWN entities. Vertices of the same type will likely display similar network properties (i.e. cluster coefficient), and similar link frequencies.

7.1.5 Co-reference and Disambiguation

Co-reference (entities which can be referred to by different strings, e.g. 'IBM' and 'International Business Machines') can be done by using various network properties (including the sibling relation mention above.)

Often, a single string can possibly refer to multiple different people. For example, Wikipedia lists 11 entries for the name 'Adam Smith'. Lydia currently has no methods for disambiguating a mention of 'Adam Smith' in an article. The network would be able to help in determining which 'Adam Smith' a particular mention is referring to. This would involve clustering the mentions of entities. We could then classify new mentions by determining which cluster they best fit into. We would also periodically attempt to re-cluster to learn if new meanings of an entity are emerging (say a 12th Adam Smith).

7.1.6 Predict Sentiment

Lydia assigns sentiment scores to entities [48]. We would want to learn what the interaction of group sentiment is with individual sentiment. In particular, how does individual member's sentiment affect a group's sentiment (the group as a single unit, as well as the sentiment of the other members of a group.) For example, if the sentiment of ‘George Bush’ goes up or down, how does the sentiment of ‘Republicans’ change, and how do other republicans’ (say ‘Dick Cheney’) sentiment change. An essential question here is: given a network and the sentiment of every individual save one, how accurately can we predict that individual’s sentiment?

There has been much work done on the spread of influence in a network [67, 40, 35]. Viral marketing aims to target the smallest set of vertices that will cause a cascade. A customer has both intrinsic value (the value of himself buying a product), and network value (the value of his influence). It may be worthwhile to market to an individual with negative intrinsic value if his network influence is positive. On the other hand it may be bad to market to an individual with positive intrinsic value if he will negatively influence the network [67]. Richardson et al [67] describe a model for viral marketing. First they mine networks from data (from knowledge sharing sites), then build a probabilistic model, and finally develop a marketing plan. Their model includes continuous marketing functions, and also models costs for discovering network structure. Spread of epidemics is also modeled on networks [81]. These are slightly different angles than what we are trying to model, the spread of sentiment.

Bibliography

[1] A. Adams and M. Sasse. Users are not the enemy. *Commun. ACM*, 42(12):40–46, 1999.

[2] P. W. Anderson. More is different. *Science*, 177:393–396, 1972.

[3] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 44–54, New York, NY, USA, 2006. ACM Press.

[4] A. Bagga and B. Baldwin. Entity-based cross-document coreferencing using the vector space model. In *Proceedings of the Thirty-Sixth Annual Meeting of the Association for Computational Linguistics and Seventeenth International Conference on Computational Linguistics*, pages 79–85, San Francisco, California, 1998. Morgan Kaufmann Publishers.

[5] H. Balakrishnan and N. Deo. Discovering communities in complex networks. In *ACM-SE 44: Proceedings of the 44th annual Southeast regional conference*, pages 280–285, New York, NY, USA, 2006. ACM.

[6] H. Bao, J. Bielak, O. Ghattas, L. F. Kallivokas, D. R. O'Hallaron, J. R. Shewchuk, and J. Xu. Large-scale simulation of elastic wave propagation

in heterogeneous media on parallel computers. *Computer Methods in Applied Mechanics and Engineering*, 1998.

[7] A.-L. Barabasi. *Linked*. Penguin Books Ltd., 2003.

[8] M. Bautin and S. Skiena. Concordance-based entity-oriented search. In *WI '07: Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*, pages 586–592, Washington, DC, USA, 2007. IEEE Computer Society.

[9] D. Bean and E. Riloff. Unsupervised learning of contextual role knowledge for coreference resolution. In D. M. Susan Dumais and S. Roukos, editors, *HLT-NAACL 2004: Main Proceedings*, pages 297–304, Boston, Massachusetts, USA, May 2 - May 7 2004. Association for Computational Linguistics.

[10] M. Bilenko and R. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proc. Ninth ACM SIGKDD Conf. on Knowledge Discovery and Data Mining*, pages 39–48, 2003.

[11] C. L. Borgman and S. L. Siegfried. Getty's synoname and its cousins: A survey of applications of personal name-matching algorithms. *Journal of the American Society for Information Science and Technology*, 43(7):459–476, 1992.

[12] G. Caldarelli. *Scale-Free Networks*. Oxford University Press, 2007.

[13] A. Cami, H. Balakrishnan, N. Deo, and R. Dutton. On the complexity of finding optimal global alliances. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 58:23–31, 2006.

[14] J. Cirasella. Google sets, google suggest, and google search history: Three more tools for the reference librarian's bag of tricks. *Reference Librarian*, 48.1, 2007.

[15] A. Clauset, C. Moore, and M. E. J. Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, 453(7191):98–101, 2008.

[16] M. Cochinwala, S. Dalal, A. K. Elmagarmid, and V. S. Verykios. Record matching: Past, present and future. Technical report, PR-OWL: A Bayesian Ontology Language for the Semantic Web. Workshop on Uncertainty Reasoning for the Semantic Web, International Semantic Web Conference, 2001.

[17] K. B. Cohen, A. Dolbey, G. Acquaah-Mensah, and L. Hunter. Contrast and variability in gene names. In *Proceedings of the Workshop on Natural Language Processing in the Biomedical Domain*, pages 14–20, Philadelphia, Pennsylvania, USA, 2002.

[18] W. W. Cohen and J. Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *KDD '02: Proceedings of the Eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 475–480, New York, NY, USA, 2002. ACM.

[19] L. F. Cranor and S. Garfinkel. *Security and Usability*. O'Reilly, Sebastopol, CA, 2005.

[20] O. Favaron, G. Fricke, W. Goddard, S. M. Hedetniemi, S. T. Hedetniemi, P. Kristiansen, R. C. Laskar, and D. Skaggs. Offensive alliance graphs. *Discussiones Mathematicae - Graph Theory*, 2002.

[21] H. Fernau and D. Raible. Alliances in graphs: a complexity-theoretic study. In J. van Leeuwen, G. F. Italiano, W. van der Hoek, C. Meinel, H. Sack, F. Plasil, and M. Bielikov, editors, *SOFSEM (2)*, pages 61–70. Institute of Computer Science AS CR, Prague, 2007.

[22] G. W. Flake, S. Lawrence, and C. L. Giles. Efficient identification of web communities. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 150–160, New York, NY, USA, 2000. ACM.

[23] G. W. Flake, R. E. Tarjan, and K. Tsioutsiouliklis. Graph clustering and minimum cut trees. *Journal of Internet Mathematics*, 1:385–408, 2004.

[24] N. Frykholm and A. Juels. Error-tolerant password recovery. In *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 1–9, New York, NY, USA, 2001. ACM Press.

[25] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the theory of NP-completeness*. W. H. Freeman, San Francisco, 1979.

[26] Z. Ghahramani and K. A. Heller. Bayesian sets. In *Proceedings of NIPS*, 2005.

[27] D. Gibson, J. Kleinberg, and P. Raghavan. Inferring web communities from link topology. In *HYPERTEXT '98: Proceedings of the ninth ACM conference on Hypertext and hypermedia : links, objects, time and space—structure in hypermedia systems*, pages 225–234, New York, NY, USA, 1998. ACM.

[28] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *In the National Academy of Sciences USA*, 99:7821–7826, 2002.

[29] N. Godbole, M. Srinivasaiah, and S. Skiena. Large-scale sentiment analysis for news and blogs. *Proceedings of Int. Conf. on Weblogs and Social Media (ICWSM 2007), Denver CO*, March 26-28 2007.

[30] C. Gooi and J. Allan. Cross-document coreference on a large scale corpus. In *Proceedings of Human Language Technology Conf. North American Chapter Association for Computational Linguistics*, pages 9–16, Boston, Massachusetts, USA, May 2004.

[31] J. Grudin. Non-hierarchic specification of components in transcription typewriting. *Acta Psychologica*, 54:249–262, 1983.

[32] K. Hansen. Head-banging: robust smoothing in the plane. *IEEE Transactions on Geoscience and Remote Sensing*, 29:369–378, 1991.

[33] K. Hemenway and T. Calishain. *Spidering Hacks*. O'Reilly and Associates, Sebastopol CA, 2003.

[34] M. Hernandez and S. Stolfo. The merge/purge problem for large databases. In *The 1995 ACM SIGMOD International Conference on the Management of Data*, pages 127–138, San Jose, California, USA, 1995.

[35] P. Holme and M. E. J. Newman. Nonequilibrium phase transition in the coevolution of networks and opinions. *Physical Review E*, 74:056108, 2006.

[36] J. Hopcroft, O. Khan, B. Kulis, and B. Selman. Natural communities in large linked networks. In *KDD '03: he 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 541–546, New York, NY, USA, 2003. ACM Press.

[37] J. Hopcroft, O. Khan, B. Kulis, and B. Selman. Tracking evolving communities in large linked networks. *Proc Natl Acad Sci U S A*, 101 Suppl 1:5249–5253, April 2004.

[38] R. Karp and M. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Research and Development*, 31:249–260, 1987.

[39] G. Karypis and V. Kumar. METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. http://www-users.cs.umn.edu/ karypis/metis, 2003.

[40] D. Kempe, J. Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *KDD '03: The ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146, New York, NY, USA, 2003. ACM Press.

[41] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(1):291–307, 1970.

[42] J. Kil, L. Lloyd, and S. Skiena. Question answering with lydia. In Proceedings of 14th Text Retrieval Conference (TREC 2005), 2005.

[43] D. E. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley Publishing Company, Reading, MA, 1973.

[44] G. Kossinets and D. J. Watts. Empirical analysis of an evolving social network. *Science*, 311:88–90, dec 2005.

[45] A. M. L. Jamieson, S. T. Hedetniemi. The algorithmic complexity of alliances in graphs. *J. Combin. Math. Combin. Comput.*, 2002.

[46] X. Li, P. Morie, and D. Roth. Robust reading: Identification and tracing of ambiguous names. In *Proceedings of Human Language Technology Conf. of the North American Association for Computational Linguistics*, pages 17–24, Boston, Massachusetts, USA, May 2004.

[47] L. Lloyd, P. Kaulgud, and S. Skiena. Newspapers vs. blogs: Who gets the scoop? In *Proceedings of Computational Approaches to Analyzing Weblogs (AAAI-CAAW 2006)*, volume AAAI Press, Technical Report SS-06-03, pages 117–124, 2006.

[48] L. Lloyd, D. Kechagias, and S. Skiena. Lydia: A system for large-scale news analysis. In *Proceedings of String Processing and Information Retrieval (SPIRE 2005)*, volume Lecture Notes in Computer Science, 3772, pages 161–166, 2005.

[49] L. Lloyd, A. Mehler, and S. Skiena. Identifying co-referential names across large corpora. In *Proc. Combinatorial Pattern Matching (CPM 2006)*, 2006.

[50] X. Luo, A. Ittycheriah, H. Jing, N. Kambhatla, and S. Roukos. A mention-synchronous coreference resolution algorithm based on the bell tree. In *Proceedings of Meeting Ass. of Computational Linguistics*, pages 135–142, 2004.

[51] I. MacKenzie and R. Soukoreff. A character-level error analysis technique for evaluating text entry methods. *Proceedings of the second Nordic Conference on Human-Computer Interaction*, pages 241–244, 2002.

[52] G. Mann and D.Yarowsky. Unsupervised personal name disambiguation. In *Proceedings of CoNLL*, pages 33–40, Edmonton, Alberta, Canada, 2003.

[53] A. Mehler, Y. Bao, X. Li, Y. Wang, and S. Skiena. Spatial analysis of news sources. *IEEE Trans. Visualization and Computer Graphics 12*, pages 765–772, 2006.

[54] A. Mehler and S. Skiena. Improving usability through password-corrective hashing. In *Proceedings of String Processing and Information Retrieval (SPIRE 2006)*, pages 193–204, 2006.

[55] S. Milgram. The small world problem. *Psychology Today*, 2:60–67, 1969.

[56] H. Miller and J. Han. *Geographic Data Mining & Knowledge Discovery*. CRC, 2001.

[57] M. Munigole, L. Pickle, and K. Simonson. Application of a weighted head-banging algorithm to mortality data maps. *Statistics in Medicine*, 18:3201–3209, 1999.

[58] M. E. J. Newman. Detecting community structure in networks. *The European Physical Journal B*, 38:321–330, March 2004.

[59] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69:026113, 2004.

[60] A. Y. Ng, A. X. Zheng, and M. I. Jordan. Link analysis, eigenvectors and stability. In *Proceedings of IJCAI*, pages 903–910, 2001.

[61] V. Ng and C. Cardie. Improving machine learning approaches to coreference resolution. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 104–111, Philadelphia, Pennsylvania, USA, 2002.

[62] J. Novak, P. Raghavan, and A. Tomkins. Anti-aliasing on the web. In *Proc. 13th Int. Conf. on World Wide Web*, pages 30–39, New York, New York, USA, 2004.

[63] J. L. Peterson. A note on undetected typing errors. *Communications of the ACM*, 29(7), July 1986.

[64] L. Philips. Hanging on the Metaphone. *Computer Language*, 7(12):39–43, 1990.

[65] M. F. Porter. An algorithm for suffix stripping. *Readings in information retrieval*, pages 313–316, 1997.

[66] A. Pothen, H. D. Simon, and K.-P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 11(3):430–452, 1990.

[67] M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 61–70, New York, NY, USA, 2002. ACM Press.

[68] P. Sarkar and A. Moore. Dynamic social network analysis using latent space models. *SIGKDD Explorations: Special Edition on Link Mining*, 2005.

[69] L. Sarmento, V. Jijkuon, M. de Rijke, and E. Oliveira. ”more like these”: growing entity classes from seeds. In *CIKM '07: Proceedings of the Sixteenth ACM Conference on information and knowledge management*, pages 959–962, New York, NY, USA, 2007. ACM.

[70] M. A. Sasse, S. Brostoff, and D. Weirich. Transforming the weakest link ? a human/computer interaction approach to usable and effective security. *British Telecom Technology Journal*, 19(3):122–131, 2001.

[71] S. Schleimer, D. Wilkerson, and A. Aiken. Winnowing: Local algorithms for document fingerprinting. In *Proceedings of 22nd ACM SIGMOD International Conference on Management of Data / Principles of Database Systems*, pages 76–85, San Diego, California, USA, 2003.

[72] J. Scott. *Social Network Analysis: A Handbook.* Sage Publications, 2000.

[73] K. H. Shafique. *Partitioning a graph in alliances and its application to data clustering*. PhD thesis, School of Computer Science University of Central Florida Orlando, 2001.

[74] J. R. Shewchuk. *Delaunay Refinement Mesh Generation*. PhD thesis, Carnegie Mellon University, 1997.

[75] T. A. Slocum, R. B. McMaster, F. C. Kessler, and H. H. Howard. *Thematic Cartography and Geographic Visualization*. Pearson Prentice Hall, 1999.

[76] Y. Spector and J. Ginzberg. Pass-sentence? a new approach to computer code. *Comput. Secur.*, 13(2):145–160, 1994.

[77] R. L. Taft. Name search techniques. *New York State Identification and Intelligence Systems, Special Report No. 1, Albany, New York.*, 1970.

[78] M. Thelen and E. Riloff. A bootstrapping method for learning semantic lexicons using extraction pattern contexts. In *Proceedings of 2002 Conf. on Empirical Methods in Natural Language Processing*, 2002.

[79] J. T's, R. Eckstein, and D. Collier-Brown. *Using Samba*. O'Reilly Media, 2003.

[80] J. R. Tyler, D. M. Wilkinson, and B. A. Huberman. Email as spectroscopy: automated discovery of community structure within organizations. *Communities and Technologies*, pages 81–96, 2003.

[81] D. J. Watts. *Six Degrees: The Science of a Connected Age*. W. W. Norton & Company, 2003.

[82] D. J. Watts, P. S. Dodds, and M. E. J. Newman. Identity and search in social networks. *Science*, page 1302, May 2002.

[83] F. Wu and B. A. Huberman. Finding communities in linear time: a physics approach. *European Physical Journal B*, 38:331–338, Mar. 2004.

[84] H. Yu and E. Agichtein. Extracting synonymous gene and protein terms from biological literature. *Bioinformatics*, 19:i340–i349, 2003.