UNIVERSITY OF CALIFORNIA, SAN DIEGO

Name Disambiguation in Web People Search

A Thesis submitted in partial satisfaction of the
requirements for the degree
Master of Science

in

Computer Engineering

by

Harshit N. Chitalia

Committee in charge:

Professor Y. Y. Zhou, Chair
Professor Pankaj Das, Co-Chair
Professor Sujit Dey
Professor Clark C. Guest

2011

The dissertation of Harshit N. Chitalia is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____
Co-Chair

_____
Chair

University of California, San Diego

2011

DEDICATION

To my Parents for the love and support they have always given me.

EPIGRAPH

The purpose of life is a life of purpose
—Robert Byrne

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank Prof. YY Zhou who has been a great source of inspiration for me. She has helped me a lot and has been a guide throughout my masters.She has treated me as her child and I also consider her as a parent. There are a lot of things, academic and non-academic that I have learnt from her and it has been a great journey throughout. If given an opportunity, I would definitely like to be back at the OPERA group.

I would also like to thank my teammates of the project, Weiwei Xiong and Zhuoer Wang who have significantly contributed to the project. The innumerable discussions, meeting's and feedback have helped us build the current system. I would also like to thank the team members of the OPERA GROUP as it has been a great learning experience in reading groups and other discussions throughout the quarters.

The thesis is part of the project which would assist in disambiguation of people search on the web.

VITA

2005-2009	B. Eng. in Electronics and Telecommunication *cum laude*, University of Mumbai, India
2010	Engineering Intern, Cisco Systems Inc. , Data Center Switching Technology Group.
2009-2011	Graduate Research Assistant, University of California, San Diego
2009-2011	M. S. in Computer Engineering, University of California, San Diego

ABSTRACT OF THE THESIS

Name Disambiguation in Web People Search

by

Harshit N. Chitalia

Master of Science in Computer Engineering

University of California, San Diego, 2011

Professor Y. Y. Zhou, Chair
Professor Pankaj Das, Co-Chair

Searching for people on the web is a ubiquitous activity. We search for people on web search engines, social networks, publication websites and various other websites. Although the quality of web search has improved over the decade, finding information regarding people has become increasingly difficult. One of the major issues relating to it is with regards to different people having the same name. Name disambiguation can occur when trying to find person in a large web space. We have build an efficient integrative framework to solve the problem of name disambiguation across not just one but several domains. The thesis presents an algorithm to aggregate and cluster people from various domains using their rich feature space of biographic facts and their connections. We demonstrate the

effectiveness of our approach by testing the efficacy of the disambiguation algorithm and its impact on person search.

Chapter 1

Introduction

A large amount of data on the web is structured which can be embedded in natural language text, relational tables, and other forms. Although it promises new and compelling applications, this Structured Web of data is huge,completely non-uniform and difficult to standardize. Most search engines are extremely powerful at document finding, and use page ranking to give relevance results. But the existing tools such as search engines and relational databases ignore Structured Web data entirely. The sheer amount of Web data makes it an enticing subject for reuse and recombination. We looked into one form of this data, data relating to people . We wanted to collect information about people which is already present on the web, integrate it and present it in a manner which was intuitive, easy to follow and relevant. Collecting information about people gave rise to a unique problem of differentiating and integrating people from various different sources.

Searching for people in the web is a ubiquitous activity. Searching for web pages related to a person accounts for more than 5 percent of the current Web searches. Currently, it is done using keywords. We try to retrieve information about a person by typing his name and the search engine would typically return thousands of webpages which contain the keywords i.e. name. A search engine such as Google or Yahoo! returns a set of web pages, in ranked order, where each web page is deemed relevant to the search keyword entered (the person name in this case). A search for a person such as say "Steve Jobs" will return pages relevant to any person with the name "Steve Jobs".This name could refer to the same person

but most likely it refers to many people as names are highly ambiguous . According to the US Census Bureau approximately 90,000 names are shared by 100 million people. This gave rise to the need of name disambiguation. Name disambiguation is desired in many cases: e.g., evaluating faculty publications, calculating statistics of social network and author impacts, etc. Typically word senses and translation ambiguities may have 2-20 alternative meanings that must be resolved through context, a personal name such as "John Smith" may potentially refer to hundreds or thousands of distinct individuals. Each different referent typically has some distinct contextual characteristics. These characteristics can help distinguish, resolve and trace the referents when the surface names appear in online documents. A search of Google shows 10,200,000 web pages mentioning John Smith, of which the first 10 possible referents are:

1. John Smith - English soldier, explorer, and author
2. John Smith - Actor
3. John Smith - Professor
4. John Smith - Artist
5. John Smith - Fire Fighter
6. John Smith - Car Salesman in Kansas
7. John Smith - Fishing Instructor in Canada
8. John Smith - Computer Science student in NewYork
9. John Smith - Race car driver from Scotland
10. John Smith - Gun Dealer in Louisiana

The above define webpages which have John Smith , but ignore ones which have J. Smith or John S. , so due to name abbreviations, identical names, name misspellings, the results are not completely correct.To make it worse people have pseudonyms in publications, social networks and public records. A next generation search engine can provide significantly more powerful models for person search. Ideally we would have required the search engine to be smart enough to identify the real entity (i.e. which John Smith) any given page refers to and then cluster them. This can go further to rank individual clusters based on the aggregate rank of the individual pages in each cluster. Thus the results are clustered by associating

each cluster to a real person,with each cluster, we could also provide a summary description that is representative of the real person associated with that cluster (for instance, in this example, the summary description may be a list of words such as "computer science, machine learning, and professor"). The user can hone in on the cluster of interest to him and get all pages in that cluster, i.e., only the pages associated with that John Smith. Such cluster-based people search could potentially be very useful.But on looking further we found that most people were interested in finding out this summary information itself, which aggregates all the information from different sources and presents all the possible information.Hence we decided to build a framework which would display all the information we have and also gave a link to our original source. Imagine searching for the web page of "Bill Clinton" who used to live in your neighborhood in Champaign, Illinois, using Google today. This is virtually impossible (or at least very tiring) since the first 20-30 pages of a Google search of "Bill Clinton" returns pages only about the President. In our approach all of the President's pages information will be folded into a single row, giving his namesakes an opportunity to be displayed in the first page of search results. One might argue that the use of context could improve the results of the standard search engines today, and thus, there is no need for aggregation approaches. However, this is not the case if you have very little knowledge about the person you are searching for. For example, assume that we are searching for "Tim Riddle, the psychology professor" with his name and keywords "psychology" and "professor." The search engine, e.g., Google, returns more than two different people . Hence, the task of clustering the pages related to different people is still difficult even for the queries that include context.

We will go further into understanding what is Structured Data?

Researchers in the database field continuously refer to their data as either structured, semi-structured, or unstructured. Although it is difficult to differentiate precisely, generally, structured data refers to data expressed using the relational model; semi-structured data means XML data; and unstructured data refers to documents such as text, Web pages, spreadsheets, and presentations. Wikipedia defines semi-structured data as "Semi-structured data is a form of structured data

that does not conform with the formal structure of tables and data models associated with relational databases but nonetheless contains tags or other markers to separate semantic elements and hierarchies of records and fields within the data.".It is somewhat believed these terms refer to whether the data is intended for machine use (structured) or human consumption (unstructured) or somewhere in between (semi-structured). This usage is probably accurate to some extent - unstructured documents are often meant for human consumption. But a row in a structured relational database can be very easy for a person to read, and unstructured spreadsheets often contain abstruse statistical data. In our framework we tried to bring all the data from social networks, public records, publications and other sources into one format which is easy for human consumption. We aggregated the results and ran our algorithm for clustering and finally displayed the end result. The next section talks about the Software Architecture and Section 3. talks about the Algorithm. Section 4. gives account of our Results and Evaluation. Section 5. talks about Related Work. Finally Section 6 talks about the Conclusion and Future Work.

Chapter 2

Software Architecture

The software architecture can be divided into four parts.

1. Getting the data
2. Storage of data
3. Analysis of the data

We would describe each part and the software's architectures used and also the decisions why certain architectures were preferred over others.

2.1 Getting the data

Getting the data was the most certainly not the easiest part of the project. We had to crawl a large number of webpages restrict ourselves to various bandwidth requirements of different Websites. The sheer amount of data we planned to crawl required a robust and scalable crawling framework. We got information from websites like facebook,DBLP,linkedin,crunchbase,patent. We also got certain information from newspapers, college department webpages. For the crawling , we wrote individual spiders (crawlers) for each website. Main part of the crawling was done using Scrapy 0.8 a python based framework for crawling websites. Some of the other packages used include Httrack, Sphider, Djanjo, XAMPP, Firebug.

2.1.1 Scrapy

Scrapy [3] was used because it provided a highly programmable and easy framework to crawl websites. It is a fast high-level screen scraping and web crawling framework, used to crawl websites and extract structured data from their pages. It can be used for a wide range of purposes, from data mining to monitoring and automated testing. It has been used in production crawlers to completely scrape more than 500 retailer sites daily, all in one server. Also it is extensible, it was designed with extensibility in mind and so it provides several mechanisms to plug new code without having to touch the framework core. Finally and most important of all it was Open Source and 100% Python, which makes it very easy to hack. Also to mention that it is actively developed and well-tested. It has an extensive test suite with very good code coverage.

2.2 Storing the data.

Storage of vast amounts of data was a crucial aspect in the design of our overall architecture. Currently we have data from facebook records and linkedin records. All the patents granted , and other information from different sources. Initially all the data was in the raw format, we have separated the process of crawling from parsing because of system efficiency. Parsing is a high I/O job and will affect performance if done simultaneously with crawling. We decided to parse all the structured information from various sources and the best solution to structured data is a relational database. Different relational databases were considered and eventually we used MySQL as it is highly evolved and also gives ACID performance. Furthermore most researchers were acquainted with MySQL and hence development time would be less as compared to other databases. Also MySQL is a popular choice of database for use in web applications, and is a central component of the widely used LAMP web application software stack. LAMP is an acronym for "Linux, Apache, MySQL, Perl/PHP/Python". MySQL is used in some of the most frequently visited web sites on the Internet, including Flickr, Nokia.com YouTube and Wikipedia, Google and Facebook. It is still most commonly used in small to

medium scale single-server deployments, either as a component in a LAMP based web application or as a standalone database server. Much of MySQL's appeal originates in its relative simplicity and ease of use, which is enabled by an ecosystem of open source tools such as phpMyAdmin. In the medium range, MySQL can be scaled by deploying it on more powerful hardware, such as a multi-processor server with gigabytes of memory. The schemas for different websites we had parsed were designed to include most of the information available on the website. Some of the schemas were similar to that used by Facebook for storing its own data where different profile and connection tables were used.

2.3 Data Analysis

The other part of the software architecture was analysis. The MySQL format of the data was not suitable for doing any sort of analysis, because the size of the data was terabytes and even simple analysis would lead to a lot of join operations thereby, reducing the efficiency drastically. Once the amount of data scaled to terabytes and we had to carefully design and choose our filesystem framework. We wanted something which would minimize the number of disc seeks required for each access. Also we wanted information to be localized for a given name , hence when a user searches there is only one disc access theoretically and all the information related to it would be pre-fetched. Hence we wanted something like a huge file divided into sections belonging to different persons such that one section has all the information relating to a person. Also since the data was huge we would like to have it distributed. Keeping all these metrics in mind we found that big table provided this sort of functionality since it was a column based data store as well as kept all the meta data in memory for better performance. Also it was on top of a distributed filesystem HDFS, and was opensource which helped us modify it as well. HBase is the opensource version for Bigtable

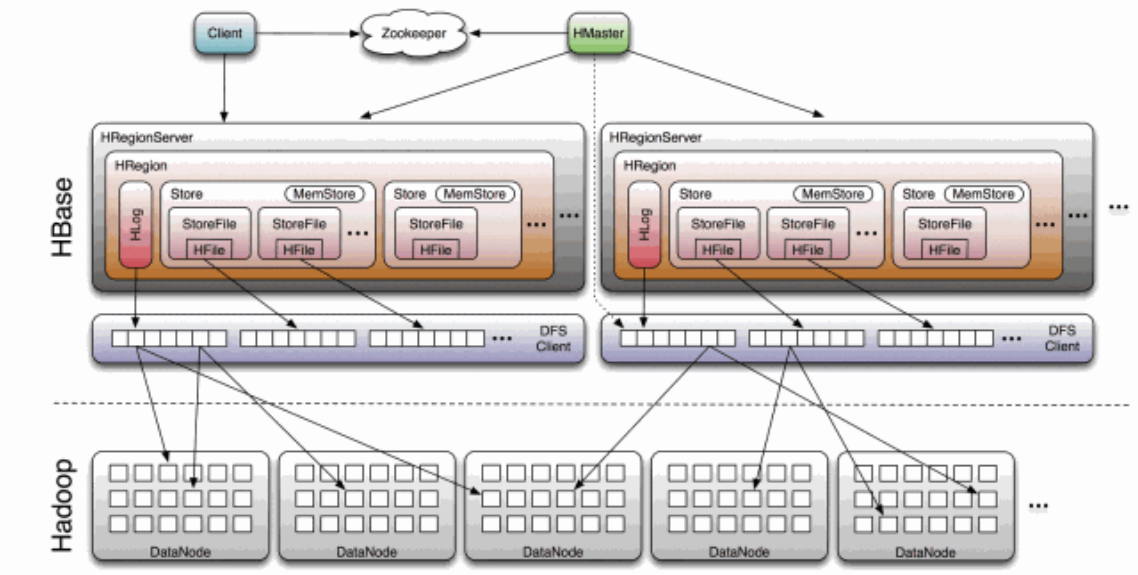

Figure 2.1: Hbase and Hadoop

2.3.1 Haddoop and HDFS

Hadoop Distributed File System (HDFS)[2] is the primary storage system used by Hadoop applications. HDFS creates multiple replicas of data blocks and distributes them on compute nodes throughout a cluster to enable reliable, extremely rapid computations. The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS was originally built as infrastructure for the Apache Nutch web search engine project. HDFS is part of the Apache Hadoop project, which is part of the Apache Lucene project.The figure describes its HDFS and HBase architecture.

2.3.2 HBase

HBase [1]is an Apache open source project whose goal is to provide Bigtable-like storage for the Hadoop Distributed Computing Environment. Just as Google's Bigtable leverages the distributed data storage provided by the Google Distributed

File System (GFS), HBase provides Bigtable-like capabilities on top of the Hadoop Distributed File System (HDFS).Data is logically organized into tables, rows and columns. An iterator-like interface is available for scanning through a row range and there is the ability to retrieve a column value for a specific row key. Any particular column may have multiple versions for the same row key.

Data Model

Applications store data rows in labeled tables. A data row has a sortable row key and an arbitrary number of columns. The table is stored sparsely, so that rows in the same table can have widely varying numbers of columns. A column name has the form <family> : <label> where <family> and <label> can be arbitrary byte arrays. A table enforces its set of <family>'s (called "column families"). HBase stores column families physically close on disk, so the items in a given column family should have roughly the same read/write characteristics and contain similar data. Only a single row at a time may be locked by default. Row writes are always atomic, but it is also possible to lock a single row and perform both read and write operations on that row atomically.

Conceptual View

Conceptually a table may be thought of a collection of rows that are located by a row key (and optional timestamp) and where any column may not have a value for a particular row key (sparse).

Table 2.1: Conceptual View

Row Key	Timestamp	Column attributes: source	Value	
Person Name	t9	attributes: dblp	value	{name= Shekar Gupta,loc= New York, pid=4444,org:Columbia}
	t8	attributes: linkedin	value	{name= John Smith,loc= San Diego, org:UCSD}
	t6	...	value	value Json string of attributes
	t5	...	value	value Json string of attributes
	t3	...	value	value Json string of attributes

Physical Storage View

Although at a conceptual level, tables may be viewed as a sparse set of rows, physically they are stored on a per-column family basis. Pictorially, the table shown in the conceptual view above would be stored as follows:

Table 2.2: Physical Storage View I

Row key	Timestamp	Column 'attributes'
John Smith	t6	ver_3
	t5	ver_2
	t3	ver_1

Table 2.3: Physical Storage View II

Row key	Timestamp	Column 'attributes'	
Shekar Gupta	t9	value	...
	t8	value	...

It is important to note in the diagram above that the empty cells shown in the conceptual view are not stored since they need not be in a column-oriented storage format. Thus a request for the value of the "attributes:" column at time stamp t8 would return no value. However, if no timestamp is supplied, the most recent value for a particular column would be returned and would also be the first one found since timestamps are stored in descending order.

Row Ranges: Regions

To an application, a table appears to be a list of tuples sorted by row key ascending, column name ascending and timestamp descending. Physically, tables are broken up into row ranges called regions .Each row range contains rows from start-key (inclusive) to end-key (exclusive). A set of regions, sorted appropriately, forms an entire table. HBase identifies a row range by the table name and start-key. Each column family in a region is managed by an HStore. Each HStore may

have one or more MapFiles (a Hadoop HDFS file type) . MapFiles are immutable once closed. MapFiles are stored in the Hadoop HDFS.

Architecture and Implementation

There are three major components of the HBase architecture:

1. The HBaseMaster
2. The HRegionServer
3. The HBase client, defined by org.apache.hadoop.hbase.client.HTable

Each will be discussed in the following sections.

HBaseMaster

The HBaseMaster is responsible for assigning regions to HRegionServers. The first region to be assigned is the ROOT region which locates all the META regions to be assigned. Each META region maps a number of user regions which comprise the multiple tables that a particular HBase instance serves. Once all the META regions have been assigned, the master will then assign user regions to the HRegionServers, attempting to balance the number of regions served by each HRegionServer. It also holds a pointer to the HRegionServer that is hosting the ROOT region. The HBaseMaster also monitors the health of each HRegionServer, and if it detects a HRegionServer is no longer reachable, it will split the HRegion-Server's write-ahead log so that there is now one write-ahead log for each region that the HRegionServer was serving. After it has accomplished this, it will reassign the regions that were being served by the unreachable HRegionServer. When the HBaseMaster dies, the cluster will shut down. HBase uses just a single central point for all HRegionServers to access: the HBaseMaster.

The META Table

The META table stores information about every user region in HBase which includes a HRegionInfo object containing information such as the start and end row keys, whether the region is on-line or off-line, etc. and the address of the

HRegionServer that is currently serving the region. The META table can grow as the number of user regions grows.

The ROOT Table

The ROOT table is confined to a single region and maps all the regions in the META table. Like the META table, it contains a HRegionInfo object for each META region and the location of the HRegionServer that is serving that META region. Each row in the ROOT and META tables is approximately 1KB in size. At the default region size of 256MB, this means that the ROOT region can map 2.6 x 105 META regions, which in turn map a total 6.9 x 1010 user regions, meaning that approximately 1.8 x 1019 (264) bytes of user data.

HRegionServer

The HRegionServer is responsible for handling client read and write requests. It communicates with the HBaseMaster to get a list of regions to serve and to tell the master that it is alive. Region assignments and other instructions from the master "piggy back" on the heart beat messages.

Read and Write Requests

Reads are handled by first checking the Memcache and if the requested data is not found, the MapFiles are searched for results.

When a write request is received, it is first written to a write-ahead log called a HLog. All write requests for every region the region server is serving are written to the same log. Once the request has been written to the HLog, it is stored in an in-memory cache called the Memcache. There is one Memcache for each HStore.

HBase Client

The HBase client is responsible for finding HRegionServers that are serving the particular row range of interest. On instantiation, the HBase client communicates with the HBaseMaster to find the location of the ROOT region. This is the

only communication between the client and the master. Once the ROOT region is located, the client contacts that region server and scans the ROOT region to find the META region that will contain the location of the user region that contains the desired row range. It then contacts the region server that is serving that META region and scans that META region to determine the location of the user region. After locating the user region, the client contacts the region server serving that region and issues the read or write request. This information is cached in the client so that subsequent requests need not go through this process. Should a region be reassigned either by the master for load balancing or because a region server has died, the client will rescan the META table to determine the new location of the user region. If the META region has been reassigned, the client will rescan the ROOT region to determine the new location of the META region. If the ROOT region has been reassigned, the client will contact the master to determine the new ROOT region location and will locate the user region by repeating the original process described above.

I would also like to thank my teammates of the project, Weiwei Xiong and Zhuoer Wang for their contribution.

Chapter 3

Implementation

3.1 Algorithm

We have developed an algorithm for analysis and clustering of same namespace people across different data sources. We divided the data fields with each of source of data into common entities and non-common entities. The common entities were name, city, state, employer, education and connections. Connections could be friends, co-authors , siblings or relatives , co-inventors etc. Non-common entities were industry, specialty, phone numbers etc. The division was necessary because for a name in the teacher database we had the name , edu, employer and the subject he/she taught. The last information was unique to this source, similarly some sources had complete addresses, phone numbers etc, which were unique and hence separated from analysis, but in the end result they would still be appended back to the original person. The algorithm would take the data from the source , will prune each record and convert it to a vector map of common entities. The vector map would be compared for two records and their linkage total weight would be determined. If the total linkage weight was above a threshold, than the two records would be merged. The weights for different entities was determined by the differentiation of entities into different classes. This was done primarily to account for the probability of two different John Smith existing in NewYork is higher than that of in Urbana-Champaign. These statistics were developed on our existing sources like Facebook and Linkedin.

3.2 Algorithm Example

Consider the given input data from the database. AB is the name of the person like John Smith and lets say we find four records of John Smith from our sources. These are numbered from 1 to 4. For simplicity of the example we have used only three entities city , organization and connections, coinventor in this case. The records are as follows:

Table 3.1: Original Records

index	city	organization	co-inventor
AB1	SD	ASU	Tim
AB2		ASU, Google	Tim
AB3	SD	UCSD, Google	John
AB4		Google	John

Step 1

Compare each items one time AB1 to AB2 and AB3 etc, AB2 to AB3, AB3 to AB4. Then build bit vector or integer array. Organization and co-PI(co-inventor) have weight 2 and location(city) has weight 1, and the threshold for merging is set to be 4.

Table 3.2: Working Set I

index	peer	city	organization	co-PI	total-score
AB1	AB2	0	1	1	4
	AB3	1	0	0	1
	AB4	0	0	0	0
AB2	AB3	0	1	0	2
	AB4	0	1	0	2
AB3	AB4	0	1	1	4

Step 2

We can now merge where the threshold is greater than or equal to 4. Let's assume we will merge AB1 and AB2 first. Put AB1-AB2 into "merge set". Now we will treat AB1 and AB2 are the same with the following algorithm.

```
algorithm (AB1, AB2 ){
        for (each line L){
                if (peer field == AB1 or AB2)
                        tag_of_the_L = index field
                        fetch the line L to update_set;
                else if (index field == AB1 or AB2)
                        tag_of_the_L = peer field
                        fetch the line L to update_set;
        }
        for (update_set){
                find a pair(L1, L2) by comparing their tag;
                 L1 = L2 = L1 or L2;
       }
 }
```

In above example, the followings are the result of the algorithm.

Merge set = {AB1-AB2 }

update_set =

Table 3.3: Update Set I

index	peer	city	organization	co-PI	total-score
AB1	AB2	0	1	1	4
AB1	**AB3**	1	0	0	1
AB1	**AB4**	0	0	0	0
AB2	**AB3**	0	1	0	2
AB2	**AB4**	0	1	0	2

Here, the bold font means a "tag" of each line.

Now we can find the pair (AB1-AB3 or AB2-AB3) (AB1-AB4 or AB2-AB4)

Let's update the table including total-score

Table 3.4: Update Set II

index	peer	city	organization	co-PI	total-score
AB1	AB2	0	1	1	4
AB1	**AB3**	1	1	0	3
AB1	**AB4**	0	1	0	2
AB2	**AB3**	1	1	0	3
AB2	**AB4**	0	1	0	2

So now the table is like this, and we will merge AB3-AB4 since it is also 4.

Table 3.5: Working Set II

index	peer	city	organization	co-PI	total-score
AB1	AB2	0	1	1	4
	AB3	1	1	0	3
	AB4	0	1	0	2
AB2	AB3	1	1	0	3
	AB4	0	1	0	2
AB3	AB4	0	1	1	4

Merge set = (AB1-AB2, AB3-AB4)

update_set =

Table 3.6: Update Set III

index	peer	city	organization	co-PI	total-score
AB1	AB3	1	1	0	3
AB1	AB4	0	1	0	2
AB2	AB3	1	1	0	3
AB2	AB4	0	1	0	2
AB3	AB4	0	1	1	4

update set = AB1-AB3 or AB1-AB4, AB2-AB3 or AB2-AB4

Table 3.7: Working Set III

index	peer	city	organization	co-PI	total-score
AB1	AB2	0	1	1	4
	AB3	1	1	0	3
	AB4	**1**	1	0	**3**
AB2	AB3	1	1	0	3
	AB4	**1**	1	0	**3**
AB3	AB4	0	1	1	4

Step 3

Just iterate this step2 until there is no line exceeding the threshold. This example stops here but it can go on if score is above threshold. With merge_set, let's do actual merge. Merge set = (AB1-AB2, AB3-AB4)
AB1-AB2 ——> AB5
AB3-AB4 ——> AB6
ORIGINAL

Table 3.8: Original Records

index	city	organization	co-inventor
AB1	SD	ASU	Tim
AB2		ASU, Google	Tim
AB3	SD	UCSD, Google	John
AB4		Google	John

MERGED

Table 3.9: Merged Records

index	city	organization	co-PI
AB5	SD	ASU,Google	Tim
AB6	SD	UCSD,Google	John

3.3 Inferences

The huge collection of our dataset gives us the opportunity to infer valuable data that is not visible. Each source has some sort of hidden information which can be used for updating the current results. In any given Patent record we have the organization , location and co-inventors mentioned in the record. The way any search engine would refer is from inventor to the Patent record. But we have also created a reverse index by which each co-inventor is also now attached to this company and hence more information can be extracted and used for any given person.

Similarly we have recorded people from same small companies as connections since its high probability that they would know each other. Such companies are found from Crunchbase. Similarly lawyers from the same organization, Doctors from the same clinic and researchers from the same group are modified as connections.

3.4 Data Cleaning

We required extensive amount of data cleaning operations. This was not the most research oriented work but to remove data and understand name formats was a very crucial task. A simple phone number can be expressed in so many ways.

- 800-555-1212
- 800 555 1212
- 800.555.1212
- (800) 555-1212
- 1-800-555-1212
- 800-555-1212-1234
- 800-555-1212x1234
- 800-555-1212 ext. 1234
- work 1-(800) 555.1212 #1234

For names it was even worse. Because phone numbers essentially considered only of digits , names could consist of a variety of different formats and Unicode characters. Some of them are as follows.

- Chun (Alex) Lau
- Jean-Francois (Jeff) JEGOU
- Yong Mei (Judy) Hill
- L. Marie (Floyd) Trotter
- J. E. (Joseph) Newman
- James E. (Jim) Johnson

- (Anthony) Dean Wilson
- Abraham P. (Bram) Valk
- mr.D.J.H. (Diederik) van Dijk
- Zhou Hai-tao
- Hsiu-ling Huang
- Doug Long–doug.long@ajilon.co
- Moises Herrera–OTIMA-PROF
- H.Douglas Nguyen–A Sustainability
- Shivani Bansal–¿Gupta
- C Fred Peterson–Locksmith
- A. Peter Hilger, AIA
- A. Reggie Mariner, Jr. P.E.
- Barb Bernstein, PHR • Barb Boyer, PMP
- Barb Bronson, SPHR, CIR
- Barb Graham, CHRP
- (CBC) Alfred Levy, Jr.
- (Dr. T.) William M. Thomas, Chirop
- (E-mail), Bert • (Skip) Williams, MD, Ed.D.
- (Uncle Bobby) Robert Ballard, Jr.

As seen from above there is no format for these kind of names, where people mention their affiliations with their names or their email address. Some of them have changed surnames, so they have different ways to displaying them. We also went by trial and error to figure out and for some of them we weren't able to figure out. But in our overall dataset such kinds of names were less than 1%, but we still cleaned most of them so that they could be used in our analysis.

3.5 Statistics

We have tried to collect a lot of statistical information from our enormous data. Mainly we picked facebook data to view how the data was distributed among

people. We got the information regarding state, college, city and employer. Also we calculated the mean and std. deviation for all to give us a better estimate. So while assigning different weights in the algorithm we would factor in if the state has more number of people , if yes than the probability of it having two individuals of the same name is also high. Similarly if the organization is large than probability of people having more names is higher than a smaller organization.

3.6 Name, Organization and Address Recognition

Address and name recognition has been done to help us figure out names, organizations and address information from a given webpage. The specifications are given a piece of text to identify names, organization and address information from the text. This would be mainly used in the general parsing of graduate student and professor webpages.

3.6.1 Name and Organization Recognition

This was done using two methods

1) A big database of given names and organizations was developed using our extensive collection from our sources. Any given text was tokenized into individual words and stored in a list. From this list we did a hash lookup into the database developed and returned whether the given word was an organization or not. The results were dependent on the size of the database.

2) The other approach was to use the standard nlp open-source techniques. Among the available ones we found the Stanford NLP entity to be good. The above two methods were used for identifying people names and organization names. Both of them had their advantages , first one was more accurate while the second one had less overhead. We planned on using the second approach as it required less overhead.

3.6.2 Address Recognition

The ner's (name entity recognition) can accurately find out the locations, i.e. a city name, state name, or abbreviations in any form. But the complete address cannot be identified ie our street address, block number etc. So if we plan to use only the location we can use the ner's . We initially tried a lot of different methods like just rules of finding cities first and then finding out nearby numbers but it didnt work very well. So now with some libraries and given the entire dataset from the tigerline database , as well as by using machine learning techniques we can identify an address from any given text .We have written an api which can be called by other programs and also written a sample program which takes the input as a url or the directory where the text files are stored and extract the address from it and write it to the output file. The program runs fast enough because the classifier can be easily stored in memory.We have also tested the accuracy of it and have found it to be pretty good. The algorithm used is as follows:

The text is tokenized and the system represents the context surrounding a token using the word n-gram model. Features that could potentially help to extract addresses are generated for each token in the n-gram. All the features of the n-gram are input to a decision tree inducer trained to determine the role of the features. The inducer will automatically choose the relevant features and learn the extraction rules. The trained classifier is then used to extract addresses. Every candidate token is presented to the classifier as a potential address token, and the classifier will label it as one of four classes: START, MIDDLE, END, and OTHER. At the final step, post-processing will be applied to the labeled tokens to extract and output addresses.

Chapter 4

Results and Evaluation

We performed a series of experiments to evaluate our framework. We wanted to evaluate the response time and the correctness of our solution.

4.1 Response Time

The response time was key in our evaluation because we could not take more than 0.5s to show the results. The number 0.5s comes from a study which was conducted for online users to experiment how much average latency is a user willing to tolerate, and it was found that over 0.5s the user doesn't have the best user experience. So we would like to have our framework respond within the limit. Initially as mentioned in the algorithm for comparing two rows , we used to do comparison on each entity. Each entity could be considered as a set and so the results could be evaluated by performing an all to all comparison. These comparisons were performed on regex (regular expressions) matches and we found that the entire operations to be quite slow for names which had a large number of records. This was quite intuitive as more the number of records more work had to be done and as a result it took more time. However the more time was not acceptable as it was of the order of 10s for small records and for person names such as John Smith , which had over 400 records, it was of the order of 100s. To find out the bottle neck we performed a series of benchmarks on our algorithm code.

Based on the results from the benchmark we made a series of optimizations. They are as follows:

1. We were using regex matching for each of data field in every entity, so to make it more efficient we used hashing. Since we were comparing two sets, we build a hash table for items of one set and did a look up on the items on the other set. If the item was found we would increase the count value. This significantly reduced our time to the order of 10s for even large records.
2. The other thing we found out was that when we merged, we started the algorithm all over again considering new information was available. This led to somewhat redundant calculations of the initial set vectors and recalculating them everytime we merged. So to remove this we stored the entire bit vector set and made further comparisons on the bit vectors itself , thus avoiding the overhead of calculations.
3. Also our use of hbase over mysql to store our data helped because it removed the step of running the query on mysql and waiting for the results because in hbase we have each person name as an index and as a result its a lookup and no longer a query. It also helped us improve our timings.
4. We also coded our algorithm in four different languages Perl,Python , C++ and Java. C++ and Java were more efficient than their scripting counterparts.

Finally we chose Java as the hbase api and hdfs were based on Java. We were able to achieve the timings to the order of 0.5s with the series of optimizations.

4.2 Completeness of our data

We also tried to understand the completeness of our data. By completeness we mean for each record how many attributes like edu,org,conn etc were present. We performed this statistics on our biggest source linkedin. As per the privacy setting of the individual, we would not be able to collect all of the records.To give a quantifiable measure , we decided to classify our data set into different classes

depending on the information they posses. We did sampling and the results are as follows.

4.2.1 LinkedIn Stat

LinkedIn profile records we have got can have education, employer, and location info (we do not have connection info).

Below is the statistics: We have profiled 12339412 records.

Table 4.1: LinkedIn Stat

Attributes	Percentage
3	41.9%
2	38.4%
1	19.6%
None	0.1%

4.3 Correctness and Accuracy

These were the relatively more difficult to evaluate since we could not automate the process. Hence we manually verified most of our data by cross checking the source on the internet and also verifying the results from our merge. To make our tasks a bit easier we evaluated professors from the cse/ece department at UCSD. Famous athletes and other personalties were evaluated. We also evaluated results on grad students.

The results table for Professors from the CSE dept are as follows:

P:Present

N:Not present in our source but present on web(Since we have not finished the complete crawling).

X:Not present on the web

Z:Present in our source but not merged

Table 4.2: Results

Name	Facebook	Linkedin	DBLP	Patent	Merged
Yuanyuan Zhou	P	P;Z	P	P	DBLP0 CRUNCHBASE0 FACEBOOK7 PATENT0
Stefan Savage	P	P	P	P	DBLP0 LINKEDIN0 FACEBOOK1 PATENT1 PATENT2 PATENT3 PATENT4 PATENT5
Amin Vahdat	P;Z	P;Z	P	N	** DBLP
George Varghese	X	X	P	P	DBLP0 PATENT0 PATENT2 PATENT4 PATENT5 PATENT6 .....
Geoffrey Voelker	X	P;Z	P	P	DBLP0 PATENT0 PATENT1
Alex Snoeren	X	P;Z	P	P	DBLP0 PATENT0 PATENT1
Vineet Bafna	X	N	P	X	** DBLP
Mihir Bellare	X	X	P	P	DBLP0 PATENT2 PATENT7 PATENT8
Serge J Belongie	N	N	P	P	DBLP0 PATENT2
Chung-Kuan Cheng	X	N	P	P	DBLP0 PATENT2 PATENT3 PATENT4 PATENT5

And for the grad students from our group:

Table 4.3: Grad Student Results

Name	Facebook	Linkedin	DBLP	Patent	Merged
Weiwei Xiong	P	P	P	X	DBLP0 LINKEDIN0 FACEBOOK1
Soyeon Park	P	P;Z	P	X	DBLP0 FACEBOOK70
Ding Yuan	P	P;Z	P	X	DBLP1 FACEBOOK11
Zuoning Yin	P	N	P	X	DBLP0 FACEBOOK0

We consider the above to be an ideal sample set. Before we analyze the results further, the above was carried for each attribute having weight 4 and the connection attribute having weight 12 , while the threshold was kept to be 8.

From the above results, we were able to merge records for 80% of the sample set. Now in each record we were sometimes not able to merge , and we found two possible reasons.

1. Our collected data lacked correlated information and hence no similarity between attributes, as a result of which we could not merge.
2. The weights assigned were less and hence it did not merge.

The next set of experiments justifies the weights and attributes which are most important. We tried to use different weights for each of our entities like

education,organization,location and connections . By testing the different weights across several domains we found that if one of the connections was common it was good enough for us to combine since source such as patents have few co-inventors, so if one of them is common to the co-author or a facebook friend , we found them to be the same person. Similarly for education and organization we found that having two similar entities was good enough for us to merge, If we relied on one common entity like in the connections we found a lot of false merging, since for a company like Microsoft, Google a lot of people work for them , hence we need a sort of second validation, with education and organization.

To measure which of the entity had the highest impact, we performed the above experiments by taking one entity at a time and comparing it with all the entities taken together. This experiment tries to point out which entity is useful in merging and to what extent. We manually checked around 200 records in our dataset. From these those that were merged using the thresholds were considered further and one entity was taken at a given time. So the experiment results give the performance if only organization field were considered or only if connection were considered. We did the analysis on connection, organization and education , we did not consider the location information as it led to a lot of false merging. The results are below:

Table 4.4: Merge Entity Evaluation

Entity	% Merged as compared to All entities
Education	8%
Organization	58%
Connection	47%

The above results percentage dont add up to 100% because some people would have enough information that they could be merged either with organization data alone or connection data alone. But if we considered only connection information we would have merged only 47% of the actual merges, whereas if we considered only organization we would have merged 58%. But if we considered

both of them together we could merge 92% of the data records. So on doing deeper analysis we found that if the education information is good indicator of same person, but we cannot rely on same school. for eg: there could be a lot of people from Bejing University having the same name. But if they posses two Universities that are same , then the probability of them being same is very high. At the same time if we reduce the threshold to be one university than there are a lot of false merges. Hence in the overall scheme education became less important in deciding whether to merge or not. For organization, we found that even one common organization was sufficient as threshold, and hence it played a sufficiently big role in merging. Organization is mainly used in merging records from Linkedin, facebook and patents. On the similar lines connections have been used to merge records from crucnhbase, facebook, patents and dblp.

Chapter 5

Related Work

In the past people have explored various techniques to name disambiguation and have also tried to find the truthfulness of the web. Name disambiguation is a well studied problem, which has commonly been applied to the publications domain. It can occur when one is seeking a list of publications of an author who has used different name variations and when there are multiple other authors with the same name. So name disambiguation helps in finding out about same authors in a list of authors, as well as correcting mistakes in which they have been erroneously combined.

Prior name disambiguation work[HEG06] mainly deals with the citation matching problem [MNU00][WMPH04][HZG05]. Some others have tried out Hybrid Naive Bayes and Support Vector Machine [HGZ+04] methods. These have been found out to be e inappropriate for large-scale data especially on the web scale, due to the cost of human annotation. Another method using K-spectral clustering was used in [HZG05] to find an approximation of the global optimal solution. However, the computation complexity O(N2) is intractable for huge dataset. Also, K is unknown a priori for an increasing database.

Name ambiguity can also be viewed as a special case of the general problem of identity uncertainty, where objects are not labeled with unique identifiers [PMM+02] [HGZ+04]. Much research has been done to address the identity uncertainty problem in different fields using different methods, such as record linkage [FS69], duplicate record detection and elimination , merge/purge , data association

, database hardening , citation matching , name matching , and name authority work in library cataloging practice

Name authority is the process through which librarians for the past century have intellectually provided disambiguation for personal and corporate names in the world's bibliographic output. However, much name authority work is conducted manually. DiLauro et. al. [WB01] propose a semi-automatic algorithm using Bayes probabilities to disambiguate composers and artists in the Levy music collection. However, their algorithm largely depends on the Library of Congress name authority file. A generative model can create other examples of the data, usually provides good insight into the nature of the data and facilitates easy incorporation of domain knowledge.

The approaches in [KM06], [KMC05], and [NTKM07] solve a disambiguation challenge known as Fuzzy Lookup. To address the web page clustering problem studied in this paper, one needs to address a different type of disambiguation known as Fuzzy Grouping. In Lookup, the algorithm is given a list of objects and the goal is for each reference in the data set to identify which object from that list it refers to [KM06], [KMC05]. For grouping, no such list is available, and the goal is to simply group all of the references that co-refer. Besides the differences in the types of problems, the solution in [KM06], [KMC05], and [NTKM07] is also completely different. It reduces the disambiguation challenge into a global optimization problem.

Most of the existing techniques are different from our methodology as they do not analyze the same type of data. Most algorithms exploit extended rich features such as NEs or URLs extracted from the web pages, while no relationships are analyzed as in our approach.

A lot of work has also been done to improve the web people search(WePS). The goal of WePS, is to output a set of clusters of webpages, one cluster per each distinct person, containing all of the webpages related to that person instead of returning webpages that are related to any people who happened to have the queried name.. The user then can locate the desired cluster and explore the webpages it contains. Even though this problem is not completely solved we find it highly un-

intuitive as the person performing the search query has to still find his answer from the cluster. Hence we proposed a system which, based on our algorithm combines the results and give all of it at one place instead of giving clusters.

There are some research efforts [AGV05] [AKE04] [BM05] [BMI06] [WGLD05] that have explored the problem of entity disambiguation in the Web setting. Web people search applications can be implemented in two different settings. One is a server-side setting, where the disambiguation mechanism is integrated into the searchengine directly. The other setting is a middleware approach, where we build people search capabilities on top of an existing search-engine such as Google by wrapping the original engine. The middleware would take a user query, use the search engine API to retrieve top K web pages most relevant to the user query, and then cluster those web pages based on their associations to real people. The middleware approach is more common, as it is difficult to conduct realistic testing of the server-side approach due to the lack of direct access to the search engine internal data.

There are a few publicly available Web search engines [KCMNT08] that offer related functionality in that Web search results are returned in clusters. Clusty (http://www.clusty.com) from Vivisimo Inc., Grokker (http://www.grokker.com), and Kartoo (http://www.kartoo.com) are search engines that return clustered results. However, the clusters are determined based on the intersection of broad topics (for instance, research related pages could form one cluster and family pages could form another cluster) or page source; also, the clustering does not take into account the fact that multiple persons can have the same name. For all of these engines, clustering is done based on the entire web page content or based on the title and abstract from a standard search-engine result. ZoomInfo (http://www.zoominfo.com) search engine is an example of person search on the Web. This search engine is similar to the one proposed in this thesis. It also extracts the named entities and after that applies some machine learning and data mining algorithms to identify different people on the Web. But, this system does not disambiguate/merge entities, also information from social networks is not considered . Among research efforts, such as [AGV05] [AKE04] [BM05] [BMI06]

[WGLD05], and the approach in [GG04] is somewhat similar to our approach in that there is an exploitation of relationships for disambiguation; however, the assembly of relationships and approach to exploiting such relationships are quite different as we now explain.

The approach starts with constructing a sketch of each web page (representative of a person with the name), which is essentially a set of attribute-value pairs for "common" distinguishing attributes of a person such as his affiliation, job title, etc. To construct the sketch, however, a variety of existing data sources (such as DBLP) and some preconstructed specialized knowledge bases (such as TAP) are used. This approach is thus restricted to person searches, where the persons are famous or prominent (famous enough for us to have compiled information about them in advance),whereas our approach does not rely on any such precompiled knowledge and thus will scale to person search for any person on the Web. Another approach is based on exploiting the link structure of pages on the Web, with the hypotheses that web pages belonging to the same real person are more likely to be linked together. This may not be true all the time.

Chapter 6

Conclusion and Future Work

In this thesis we have presented "Pittsburgh" that we have developed to improve people search over the Internet. Recently the problem of Web People Search has attracted significant attention from both the industry and academia. In the classic formulation problem the user issues a query to a web search engine that consists of a name of a person of interest. For such a query, a traditional search engine such as Yahoo or Google would return webpages that are related to any people who happened to have the queried name. We present an algorithm for aggregating and clustering persons from various domains into one single domain. We have presented an interface which can help people find information relating to people from aggregated information from different sources such as dblp, facebook, linkedin, pipl, spokeo, doctors, scientists, teachers etc .

The work we have done so far is a starting point for the future of web people search. The building of the system has given us key insight into many new technologies like hadoop, hbase, map reduce. There are several key improvements which can be brought to the current system and system could be enhanced

1. We would like to start to automatically extract biographic information from general webpages and thus improve our overall data as well as increase the efficiency on our data.
2. Currently we support search only on names but we would like to support

search on organizations, cities, states, universities etc.
3. In this thesis, we only evaluated our core technique. We plan to use more advanced extraction capabilities that would allow: a) a better interpretation of extracted entities by taking into account the roles they play with respect to each other (boss of somebody, student of somebody), b) extraction of relationships, as currently, the algorithm relies primarily on co-occurrence relationships only.
4. We plan to develop disambiguation algorithms for other people search problems that have different settings.
5. We also plan to allow people to add additional information to their profile, which can result in greater merging of current information. We also plan to let people add their contacts by importing their mail contact list. Thus improving on the connection analysis.

Overall, we have built a highly scalable framework for web people search, which uses an aggregation algorithm to provide clustered web results.

Bibliography

[1] Hbase (http://hbase.apache.org/).

[2] Hadoop and hdfs (http://hadoop.apache.org/hdfs/).

[3] Scrapy (http://scrapy.org/).

[AGV05] Javier Artiles, Julio Gonzalo, and Felisa Verdejo. A testbed for people searching strategies in the WWW. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Posters, pages 569–570, 2005.

[AKE04] Reema Al-Kamha and David W. Embley. Grouping search-engine returned citations for person-name queries. In Alberto H. F. Laender, Dongwon Lee, and Marc Ronthaler, editors, *WIDM*, pages 96–103. ACM, 2004.

[BM05] Ron Bekkerman and Andrew McCallum. Disambiguating web appearances of people in a social network. In Allan Ellis and Tatsuya Hagino, editors, *WWW*, pages 463–470. ACM, 2005.

[BMI06] Danushka Bollegala, Yutaka Matsuo, and Mitsuru Ishizuka. Extracting key phrases to disambiguate personal names on the web. In Alexander F. Gelbukh, editor, *CICLing*, volume 3878 of *Lecture Notes in Computer Science*, pages 223–234. Springer, 2006.

[FS69] I. Fellegi and A. Sunter. A theory for record linkage. *J. Amer. Stat. Assoc.*, 64(328):1183–1210, December 1969.

[GG04] R. Guha and A. Garg. Disambiguating people in search. In *World Wide Web Conference Series*, 2004.

[HEG06] Jian Huang, Seyda Ertekin, and C. Lee Giles. Efficient name disambiguation for large-scale databases. volume 4213 of *Lecture Notes in Computer Science*, pages 536–544, 2006.

[HGZ+04] Hui Han, Lee Giles, Hongyuan Zha, Cheng Li, and Kostas Tsioutsiouliklis. Two supervised learning approaches for name disambiguation in author citations. In *JCDL'04: Proceedings of the 4th ACM/IEEE-CS Joint Conference on Digital Libraries*, Mining and disambiguating names, pages 296–305, 2004.

[HZG05] Hui Han, Hongyuan Zha, and C. Lee Giles. Name disambiguation in author citations using a K-way spectral clustering method. In *Proceedings of the Fifth ACM/IEEE-CS Joint Conference on Digital Libraries*, 2005.

[KCMNT08] Dmitri V. Kalashnikov, Zhaoqi Chen, Sharad Mehrotra, and Rabia Nuray-Turan. Web people search via connection analysis. *IEEE Transactions on Knowledge and Data Engineering*, 20:1550–1565, 2008.

[KM06] Dmitri V. Kalashnikov and Sharad Mehrotra. Domain-independent data cleaning via analysis of entity-relationship graph. *ACM Trans. Database Syst*, 31(2):716–767, 2006.

[KMC05] Dmitri V. Kalashnikov, Sharad Mehrotra, and Zhaoqi Chen. Exploiting relationships for domain-independent data cleaning. In *SDM*, 2005.

[MNU00] Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD*, pages 169–178, 2000.

[NTKM07] Rabia Nuray-Turan, Dmitri V. Kalashnikov, and Sharad Mehrotra. Self-tuning in graph-based reference disambiguation. In Kotagiri Ramamohanarao, P. Radha Krishna, Mukesh K. Mohania, and Ekawit Nantajeewarawat, editors, *DASFAA*, volume 4443 of *Lecture Notes in Computer Science*, pages 325–336. Springer, 2007.

[PMM+02] Hanna Pasula, Bhaskara Marthi, Brian Milch, Stuart J. Russell, and Ilya Shpitser. Identity uncertainty and citation matching. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *NIPS*, pages 1401–1408. MIT Press, 2002.

[WB01] James W. Warner and Elizabeth W. Brown. Automated name authority control. In *JCDL*, pages 21–22. ACM, 2001.

[WGLD05] Xiaojun Wan, Jianfeng Gao, Mu Li, and Binggong Ding. Person resolution in person search results: Webhawk. In Otthein Herzog, Hans-Jörg Schek, Norbert Fuhr, Abdur Chowdhury, and Wilfried Teiken, editors, *CIKM*, pages 163–170. ACM, 2005.

[WMPH04] Ben Wellner, Andrew McCallum, Fuchun Peng, and Michael Hay. An integrated, conditional model of information extraction and coreference with application to citation matching, 2004.