Google based Name Search: Resolving Mixed Entities on the Web

Byung-Won On*
School of Information Systems
Singapore Management University
bwon@smu.edu.sg

Ingyu Lee
Sorrell College of Business
Troy University
inlee@troy.edu

Abstract

When non-unique values are used as the identifier of entities, due to their homonym, confusion can occur. In particular, when part of "names" of entities are used as their identifiers, the problem is often referred to as a mixed entity resolution *problem, where goal is to sort out the erroneous entities due to name homonyms (e.g., if only last name is used as an identifier, one cannot distinguish "Vannevar Bush" from "George Bush"). Especially, a mixed entity resolution problem is common on the Web data. For instance, to search for a product name (e.g., Oracle) in Google, there exist a mixture of web pages due to the name homonyms (e.g., Oracle Database, Oracle Audio, Oracle Academy, etc.). In this paper, we present a practical system for resolving such mixed entities on the Web. For development of such a system, we propose a web service based interface, an unsupervised clustering scheme, and cluster ranking algorithms. In particular, since the correct number of clusters is often unknown, we study a state-of-the-art unsupervised clustering solution based on propagation of pairwise similarities of entities. Our claim is empirically validated via experimentation, showing that our approach outperforms main competing solution.*

1 Introduction

According to recent U.S. Census Bureau reports, about 30% queries include person names and 100 million persons share only about 90,000 person names. This statistical data says that a search result is a mixture of web pages of different people with the same name spellings. In general, this problem is known as *Mixed Entity Resolution Problem* for named entity search tasks (e.g., product or person names) on the Web [3]. To demonstrate the need for a solution to mixed entities, let us present a real case drawn from Google, shown in Figure 1. In the search result, there exists a mixture of web pages of a professor at CMU, an actor, a hockey player, a historian, a Jazz guitarist, etc., who have the same name spellings of "Tom Mitchell". Actually, there are 37 different *Tom Mitchell*s among 100 top ranked web pages as illustrated in Table 2. Furthermore, mixed entities commonly occur on the Web when we are searching for a product by name. For instance, if a user searches for a product name such as *Oracle*, she also finds different web pages of Oracle Database, Oracle Audio, Oracle Academy, and so forth.

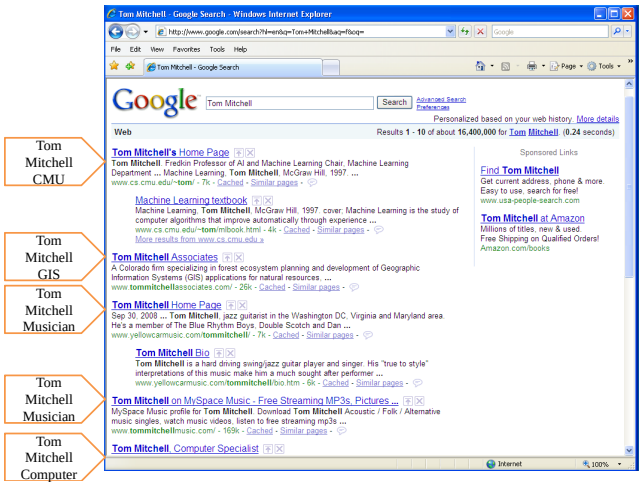

Figure 1. A snippet of Google which shows the ranked links of web pages associated with the keyword of "Tom Mitchell".

In this case, unlike traditional search engines, we focus on developing an effective system that identifies mixed entities such as person or product names (as a query) on the Web, and then displays its query result containing ranked groups, each of which contains URL links and corresponds to each different entity with the same description. However, it is non-trivial to resolve mixed entities due to the following four challenges. First, since the number of clusters within top-k ranked web pages is not given a priori, we are unable to take advantage of supervised clustering schemes such as K-means and K-spectral clustering. Second, skewed clus-

*Work was done while the author was at the University of British Columbia.

ter sizes make it hard to group web pages correctly. Next, the running time of clustering web pages should be so efficient that users do not feel bored, waiting search results for a long time. Finally, a set of clusters is required to be re-ranked. For instance, take a look at the name data set of Tom Mitchell in Table 2, where we observed that 92 top ranked web pages are grouped to 37 clusters of a CMU professor, a hockey player, a historian, and so on. In the next step, the 37 clusters should be ranked in a certain order. In other words, the CMU professor cluster is first ranked, and next the historian cluster is ranked, and in turn. To cope with these challenges, we develop an effective framework for resolving mixed entities on the Web. In our work, we propose a web service based interface, an unsupervised clustering, and cluster ranking schemes, shown in Figure 2. In particular, we devise an unsupervised clustering technique using *similarity propagation*.

The rest of this paper is organized as follows. In Section 2, we formally define our problem. In Section 3, we introduce the overview of our framework, followed by discussion of our main ideas. In Section 4, we report preliminary experimental results. In Section 5, we discuss the background and related work. Finally, some discussion and conclusion follow in Section 6.

2 Problem Definition

Formally, the mixed entity resolution problem in our setting is defined as follows [1]:

> Given a set of mixed entities E=$\{e_1, ..., e_p, ..., e_q, ..., e_N\}$ with the same name description d, group E into K disjoint clusters C=$\{c_1, ..., c_K\}$ such that entities $\{e_p^i, ..., e_q^i\}$ within each cluster c_i belongs to the same real-world group.

Clustering is the key part for this task. More specifically, we view this problem as an unsupervised hard clustering problem. In this problem, we observed that the majority of input pages map to a single individual, although there are a few that are assigned to multiple individuals sharing the same name. Hence, we view the problem as *hard* clustering, assigning input pages to exactly one individual, so that the produced clusters do not overlap. Hard clustering algorithms can be classified as either partitioning or hierarchical clustering. Hierarchical agglomerative clustering approach generates a series of nested clusters by merging simple clusters into larger ones, while partitive methods try to find a pre-specified number of clusters that best capture the data. For instance, a prior knowledge of the probable number of clusters must be required in K-means and K-way spectral clustering algorithms. In particular, note that our problem is an "unsupervised" clustering problem. In other words, since the correct number of clusters is not given *a priori* in our problem, *Hierarchical Agglomerative Clustering* (HAC) algorithms [11], rather than partitive clustering methods, are employed as a solution to our problem. However, hierarchical clustering methods are not able to reallocate entities. Therefore, it is plausible to be poorly classified in the early stages of text analysis. In this paper, we study an unsupervised clustering method that is more accurate than HAC algorithms. We will discuss the details in Section 3.4.

[1] In our work, E is a collection of web pages and e_p stands for the p-th web page. d denotes person or product name.

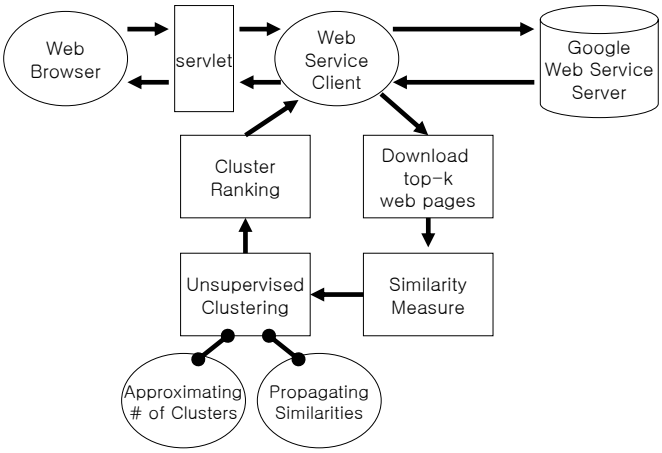

Figure 2. System overview.

3 Main Proposal

Figure 2 illustrates the overall architecture of our system. To search for a product name (e.g., Oracle), we use Google web service framework in which Google web service server provides us with a list of top-k ranked web pages which are associated with the product name.

Then, we tokenize top-k ranked web pages and each web page is represented as a vector using TF/IDF weighting scheme and the similarity of each pair of vectors are computed by TF/IDF cosine similarity measure. Subsequently, the vectors are grouped into a set of clusters in terms of our *similarity-propagation*-based unsupervised clustering method. Finally, K clusters are ranked by one of our cluster ranking algorithms followed by the search result.

3.1 Web Service based Interface

As shown in Figure 2, our web service client to Google [6] uses a keyword search in the Google search engine. Since the Google web service supports Simple Object Access Protocol (SOAP), which is a technology to allow for Remote Procedure Call (RPC) over the web, the client program creates a SOAP request message that contains a person name entered by a user, and then sends it to the Google's web service server. After the client receives a SOAP response message from the server, it parses the SOAP response, and then extracts the top-k links. According to the

Input: Graph G
Output: # of clusters
For $\{\mu = 0.01; \mu \leq 1; \mu = \mu + 0.01\}$
Generate a subgraph G_i by removing
links s.t. their link weights $\leq \mu$
$numClus_i$ = # of disconnected graph segments in G_i
E=(μ,$numClus_i$)
End For
Based on the E sequences, classifies as
Type I (concave), Type II (convex), Type III (linear)
Find the maximum and minimum difference between
$numClus_i$ and $numClus_{i+1}$ in E
If graph segments $\{G_i\}$ are in Type I
Return maximum difference $numClus_i$ as cluster #
ElseIf graph segments $\{G_i\}$ are in Type II
Return minimum difference $numClus_{i+1}$ as cluster #
Else graph segments $\{G_i\}$ are in Type III
Return average of maximum and minimum difference

Figure 3. Algorithm: APPROXIMATION OF CLUSTER NUMBERS.

recent study [5], the majority of people only tend to look into the first returned page (i.e., 10 links) of Google. Thus, we focus on at most $k = 100$ links which would be the URLs of web pages that contain the keyword.

3.2 Similarity Measure

Web pages corresponding to top-k links are downloaded in our system. Let e_p, t_i, and T be the p-th web page, the i-th term in e_p, and the total number of distinct terms in the top-k web pages, respectively. w_{t_i} is the weight value which is associated with the pair (t_i, e_p), and further document vector $\vec{v}_{e_p} = (w_{t_1}, w_{t_2}, ..., w_{t_T})$. First, to weight each term t_i in e_p, $w_{t_i} = \frac{\text{Frequency}(t_i)}{max_{t_l}\text{Frequency}(t_l)} \times log(\frac{N}{N_{t_i}})$, where Frequency$(t_i)$ is the raw frequency of term t_i in e_p; N is the total number of documents in the system; and N_{t_i} is the number of documents in which term t_i appears [7]. Then, the similarity between $\vec{v}_{e_p}$ and $\vec{v}_{e_q}$ is estimated as

$$sim(\vec{v}_{e_p}, \vec{v}_{e_q}) = \frac{\vec{v}_{e_p} \cdot \vec{v}_{e_q}}{|\vec{v}_{e_p}| \times |\vec{v}_{e_q}|} = \frac{\sum_{i=1}^{T} w_{t_i,e_p} \times w_{t_i,e_q}}{\sqrt{\sum_{i=1}^{T} w_{t_i,e_p}^2 \times \sum_{i=1}^{T} w_{t_i,e_q}^2}}. \quad (1)$$

3.3 Approximating Number of Clusters

Based on our similarity scheme, top-k web pages can be represented as Graph G, where the p-th web page denotes Node e_p, and the edge weight between nodes e_p and e_q is the TF/IDF cosine similarity between e_p and e_q as discussed in Section 3.2. In the next step, we need an unsupervised clustering method to gather similar entities together. Since the unsupervised clustering generally shows poor performance, we devise an algorithm of approximately estimating the number of clusters shown in Figure 3. The main idea of the algorithm is to gradually disconnect several edges based on the connectivity between nodes and to analyze the patterns of segmented subgraph sequences. Assume N_{μ_i} and $N_{\mu_{i+1}}$ are number of subgraphs with μ_i and μ_{i+1}, respectively. Then, we assume that the number of clusters are assumed to be near when the difference between N_{μ_i} and $N_{\mu_{i+1}}$ is drastically changed. We observe three different types of graph segment sequences with an incremental threshold value μ: gradually increasing, gradually decreasing, and staying approximately as a constant. In the gradually increasing sequence (convex), the sequence reaches the maximum difference when the difference starts to decrease. For the gradually decreasing sequence (concave), the sequence reaches the minimum difference when the difference starts to increase. Based on this property, our algorithm assumes that the number of cluster will be close to the maximum difference in the number of subgraph sequences when it shows concave function, the minimum difference when it shows convex function, and the average of these two when it stays approximately as a constant (linear function).

3.4 Similarity Propagation

The similarity $sim(\vec{v}_{e_p}, \vec{v}_{e_q})$ stands for how well $\vec{v}_{e_q}$ is suited to be the centroid vector for $\vec{v}_{e_p}$, defined as $sim(\vec{v}_{e_p}, \vec{v}_{e_q}) = -\left\|\vec{v}_{e_p} - \vec{v}_{e_q}\right\|_2$. To determine which vectors are centroids and which centroid each vector belongs to, two kinds of messages are exchanged between vectors. One is **Responsibility** $R(\vec{v}_{e_p}, \vec{v}_{e_q})$ and the other one is **Availability** $A(\vec{v}_{e_p}, \vec{v}_{e_q})$. More specifically, $R(\vec{v}_{e_p}, \vec{v}_{e_q})$ indicates how well suited $\vec{v}_{e_q}$ is to work as centroid for $\vec{v}_{e_p}$, considering other centroids for $\vec{v}_{e_p}$. $R(\vec{v}_{e_p}, \vec{v}_{e_q})$ is sent from $\vec{v}_{e_p}$ to $\vec{v}_{e_q}$. In contrast, $A(\vec{v}_{e_p}, \vec{v}_{e_q})$ reflects how appropriate it would be for $\vec{v}_{e_p}$ to choose $\vec{v}_{e_q}$ as its centroid vector, sending from centroid $\vec{v}_{e_q}$ to $\vec{v}_{e_p}$. Initially, $A(\vec{v}_{e_p}, \vec{v}_{e_q}) = 0$. The responsibilities are computed by

$$R(\vec{v}_{e_p}, \vec{v}_{e_q}) = sim(\vec{v}_{e_p}, \vec{v}_{e_q}) - \max_{\vec{v}'_{e_q} \neq \vec{v}_{e_q}} \{A(\vec{v}_{e_p}, \vec{v}'_{e_q}) + sim(\vec{v}_{e_p}, \vec{v}'_{e_q})\}. \quad (2)$$

The availability updates to see if each candidate centroid vector would make a good centroid defined as

$$A(\vec{v}_{e_p}, \vec{v}_{e_q}) = \min\{0, R(\vec{v}_{e_q}, \vec{v}_{e_q}) + \sum_{\vec{v}'_{e_p} \in \{\vec{v}_{e_p}, \vec{v}_{e_q}\}} \max\{0, R(\vec{v}'_{e_p}, \vec{v}_{e_q})\}\}. \quad (3)$$

The self-availability $A(\vec{v}_{e_q}, \vec{v}_{e_q})$ is updated differently in terms of $A(\vec{v}_{e_q}, \vec{v}_{e_q}) = \sum_{\vec{v}'_{e_p} \neq \vec{v}_{e_q}} max\{0, R(\vec{v}'_{e_p}, \vec{v}_{e_q})\}$. This self availability indicates that $\vec{v}_{e_q}$ is a centroid, based on the positive responsibilities sent to candidate centroid $\vec{v}_{e_q}$ from other vectors. Using the above functions, messages are exchanged between vectors until a high quality

set of centroids and corresponding clusters gradually converges. In other words, for vector $\vec{v}_{e_p}$, the value of $\vec{v}_{e_q}$ maximizing $A(\vec{v}_{e_p}, \vec{v}_{e_q}) + R(\vec{v}_{e_p}, \vec{v}_{e_q})$ identifies the vector that is the centroid for $\vec{v}_{e_p}$ or the vector itself as the centroid. The time complexity of the similarity propagation is $O(i \times K \times N)$, where i, K, and N are the number of iterations, centroid vectors, and entity vectors, respectively. Details of the method are described in [1].

3.5 Cluster Ranking Algorithms

As the result of our clustering method using similarity propagation, relevant web pages are clustered to the same group. For instance, suppose there are four different individual persons with the same name spellings, shown in Table 1. In the example, the total number of web pages related to "Tom Mitchell" is 13, where four web pages (i.e., e_1, e_2, e_3, and e_4), three ones (i.e., e_5, e_6, and e_7), five ones (i.e., e_8, e_9, e_{10}, e_{11}, and e_{12}), and one web page (e_{13}) are associated with the professor at CMU, the reporter at CNN, the musician at Shady record company, and the minister at Kansas city, respectively. Let us assume that web pages are ranked by PageRank scores like the third column in Table 1. The ranking order of web pages in terms of Google PageRank is e_1, e_5, e_{10}, e_6, e_7, e_9, e_8, e_4, e_{12}, e_2, e_{11}, e_3, and e_{13}. These 13 web pages are clustered to four groups in terms of our clustering scheme in Section 3.4, and furthermore we need to re-arrange the four groups in a certain order (like PageRank). Let us denote this process as *ranking* clusters.

This is a considerably challenging issue in this paper. For instance, see the four web pages of "Tom Mitchell" at CMU. e_1 is firstly ranked by Google PageRank algorithm, while the rest pages are mostly located in bottom – e_2 (10-th), e_3 (12-th), and e_4 (8-th). In this case, it is hard to determine which position the cluster (labeled as CMU/Prof) should be ranked in among the four clusters (labeled as CMU/Prof, CNN/Rep, Shaddy/Mus, and Kansas/Min). To address this problem, we propose an approach based on the hypothesis that *re-using ranking information generated by Google is sufficiently effective when we attempt to rank a set of clusters*. For this, we consider three different methods as follows:

- Consider the *relative ranking positions* of web pages per cluster which is defined as

$$ClusterRank_j = \frac{\sum_{i \in cluster_j} PageRank_i / N}{N_{cluster_j}} \quad (4)$$

  where $Cluster_j$ is the number of pages in cluster j and N is the total number of pages.

- Consider the *highest ranking position* of web pages per cluster which is defined as

$$ClusterRank_j = \min_{i \in Cluster_j} PageRank_i. \quad (5)$$

Cluster Label	Web Page ID	Google PageRank
Tom Mitchell Professor CMU	e_1	1
	e_2	10
	e_3	12
	e_4	8
Tom Mitchell Reporter CNN	e_5	2
	e_6	4
	e_7	5
Tom Mitchell Musician Shady Records	e_8	7
	e_9	6
	e_{10}	3
	e_{11}	11
	e_{12}	9
Tom Mitchell Minister Kansas City	e_{13}	13

Table 1. An example for ranking clusters.

- Consider the *median ranking position* of web pages per cluster which is defined as

$$ClusterRank_j = median\{Cluster_j\} \quad (6)$$

Considering that our ranking algorithms compute the ranks of clusters based on document ranks by Google PageRank, the result lists of our ranking algorithms would be analogous to Google's standard output. However, we expect that the result lists of our ranking algorithms would provide users with better presentation. For instance, please see Tom Mitchell at CMU in Table 1, in which e_1, e_2, e_3, and e_4 pertaining to the CMU professor are ranked in 1-st, 10-th, 12th, and 8-th. In general, Google standard output is a set of pages containing top-10 documents. Thus users are able to see three documents in the first returned page, and the other page (i.e., e_3) in the second page. On the other hand, in terms of our relative ranking algorithm, e_1, e_2, e_3, and e_4 are ranked in 5-th, 7-th, 8th, and 6-th, and all the documents appear in the first returned page containing top-10 documents. In the end, users are able to search for all documents (related to Tom Mitchell at CMU) once.

4 Experimental Validation

For implementation of Google web service client and server programs, we used Google APIs [6]. We also measure similarities of documents, using TF/IDF cosine similarity of SecondString [12]. For the similarity-propagation-based clustering method, we used the Matlab code of [1] in public. All experimentation were done on 4×2.6 Ghz Opteron processors with 32GB of RAM.

4.1 Set-up

Data sets. For validation, we have used two data sets from real examples on the Web. The `person name` is

Keyword Type	Name	N	K
Person	Adam Cheyer	97	2
	William Cohen	88	10
	Steve Hardt	81	6
	David Israel	92	19
	Leslie Pack Kaelbling	89	2
	Bill Mark	94	8
	Andrew McCallum	94	16
	Tom Mitchell	92	37
	David Mulford	94	13
	Andrew Ng	87	29
	Fernando Pereira	88	19
	Lynn Voss	89	26
Product	Oracle	25	8
	Sun	25	14
	Trojan	25	9

Table 2. Characteristics of name data set [4] (N: # of top-k pages and K: # of clusters as solution).

a test case using the 1,085 web pages that [4] used. In 2004, [4] extracted 12 personal names from Melinda Gervasio's email directory. Then, 100 top-ranked web pages of each name were retrieved from Google, and cleaned and manually labeled by authors. The resulting data set consists of 1,085 web pages, 187 different persons, and 420 relevant pages. Table 2 shows the statistics of the data set. For instance, when "Tom Mitchell" is issued as a query to Google, 92 web pages are retrieved. Among these 92, there are 37 namesakes to "Tom Mitchell". For example, among 92 web pages, "Tom Mitchell" appears as musicians, executive managers, an astrologist, a hacker, and a rabbi – 32 different kinds. That is, a set of 32 individual persons are mixed since they all have the same name description of "Tom Mitchell". Similarly, the `product name` is the other test case as illustrated in Table 2.

Evaluation Metrics. To evaluate competitive clustering methods, we use *rand index* R_I [14] [15]. Given a set of N entities and two clusters X and Y, $R_I = \frac{(a+b)}{(a+b+c+d)}$, where a : # of pairs of elements that are in the same set in X and Y; b : # of pairs of elements that are in different set in X and Y; c : # of pairs of elements that are in the same set in X and in different set in Y; and d : # of pairs of elements that are in different set in X and in the same set in Y. If R_I is closed to 1, the two partitions X and Y are the same such that $0 \leq R \leq 1$. Another metric we are using to measure the performance is relative error which is defined as the difference between the actual and predicted number of clusters divided by the actual number of clusters. For example, if the algorithm generates 4 clusters but the true solution is 5, then the relative error is $(5-4)/5 = 0.2$. The relative error shows how close the clustering result is to the true solution.

4.2 Results

Table 3 shows the experimental results of two algorithms on 15 name data sets. For the hierarchical clustering algorithm, we manually try several different cut off values and choose the best results. However, hierarchical clustering shows wide margin of errors in predicting the number of clusters with our name data set. Similarity propagation shows almost **ten times** better result in terms of relative error. In addition, hierarchical clustering requires pairwise similarities to build up the linkage tree which requires $O(N^3)$ number of computations and also needs $O(N^2/2)$ amount of memory space to store the linkage data where N is the number of documents. On the other hand, similarity propagation based on sparse matrix requires only $O(NNZ^2)$ computations and $O(NNZ)$ memory space where NNZ is number of related document pairs.

As a conventional method to measure the quality of clustering results, we use rand index as described in the previous section. The experimental results show that our similarity propagation clustering shows 4% better performance than hierarchical clustering algorithm. The 4% performance difference between two algorithms is significant, considering the rand index becomes much smaller if the number of clusters are increasing. Due to the space constraint, we do not show the clustering ranking results in this paper.

5 Related Work

Bekkerman et. al. proposed two algorithms to disambiguate web appearances of people in a social network in their paper [4]. One is based on link structure of web pages and the other is using multi-way distributional clustering method. Their algorithms show more than *20%* improvement in the aspect of accuracy. Minkov et. al. used lazy graph walk algorithm to disambiguate names in email documents in their paper [8]. They provided a framework for email data, where content, social networks and a timeline are integrated in a structured graph. Banerjee et. al. proposed multi-way clustering on relation graphs in [9]. Different types of entities are simultaneously clustered based not only on their intrinsic attribute values, but also on multiple relations between entities. On et al. introduced multilevel graph partitioning scheme to address the scalable issue of name disambiguation problem on both bibliographic and information retrieval domains [10].

6 Conclusion and Future Work

In a nutshell, we formalized the mixed entity problem which commonly appear on the Web. Then, we developed a practical system for resolving mixed entities such as person or product names for name search tasks. For the development of such a system, we introduced a web service based

	Solution	Similarity Propagation			Hierarchical Clustering		
	Cluster #	Cluster #	Relative Error	Rand Index	Cluster #	Relative Error	Rand Index
Adam Cheyer	2	9	3.50	0.17	39	18.5	0.11
William Cohen	10	6	0.40	0.46	59	4.90	0.42
Steve Hardt	6	9	0.50	0.43	45	6.50	0.41
David Israel	19	12	0.37	0.70	50	1.63	0.76
Leslie Pack Kaebling	2	2	0.00	0.50	59	28.5	0.03
Bill Mark	8	7	0.13	0.64	58	6.25	0.60
Andrew McCallum	16	12	0.25	0.67	67	3.19	0.65
Tom Mitchell	37	27	0.27	0.88	66	0.78	0.94
David Mulford	13	31	1.38	0.65	52	3.00	0.65
Andrew Ng	31	37	0.19	0.82	24	0.23	0.67
Fernando Pereira	19	15	0.21	0.74	33	0.74	0.67
Lynn Voss	26	26	0.00	0.82	40	0.54	0.86
Sun	14	9	0.36	0.90	12	0.14	0.92
Oracle	8	5	0.38	0.61	8	0.00	0.66
Trojan	9	4	0.56	0.80	6	0.33	0.76
Average			**0.57**	**0.65**		**5.02**	**0.61**

Table 3. Experimental results. Relative error shows huge difference between two algorithms.

interface. In addition, since prior knowledge of the probable number of clusters is unknown, we presented an unsupervised clustering scheme based on similarity propagation that outperforms the existing well-known hierarchical agglomerative clustering algorithm. Finally, we proposed three ranking algorithms for arranging the resulted clusters in an appropriate order. In practice, our proposal can be used as *name search* in Google [2]

For our future direction, the scalability of the algorithm is an ongoing problem. Note that we focus on top-k web pages retrieved from Google. In this paper, our system correctly group only top-k web pages to a set of clusters. As the k value is increased, our clustering scheme suffers from scalable problem. To address this challenging problem, we are working on an unsupervised clustering method based on multi-level graph partitioning approach. In addition, it is infeasible for every clustering methods to correctly (perfectly) cluster web pages at all. To cope with this practical issue, we will apply the concept of feedback and investigate semi-clustering problem (induced by users' feedback) in our future framework.

References

[1] B. Frey and D. Dueck. "Clustering by Passing Messages Between Data Points". J. Science, vol. 315, February, 2007.

[2] I. Lee, B. On, and S. Yoon. "Algebraic Algorithms to Solve Name Disambiguation Problem". Int'l Conf. on Data Mining, Las Vegas, USA, July, 2009.

[3] D. Lee, B. On, J. Kang, and S. Park. "Effective and Scalable Solutions for Mixed and Split Citation Problems in Digital Libraries". ACM SIGMOD Workshop on Information Quality in Information Systems (IQIS), Baltimore, MD, USA, June, 2005.

[4] R. Bekkerman, and A. McCallum. "Disambiguating Web Appearances of People in a Social Network". Int'l Conf. on World Wide Web (WWW), 2005.

[5] B. Jansen, and A. Spink. "An Analysis of Web Documents Retrieved and Viewed". Int'l Conf. on Internet Computing, Las Vegas, Nevada, USA, 2003.

[6] Google Web APIs. "$http://www.google.com/apis$".

[7] R. Baeza-Yates, and B. Ribeiro-Neto. "Modern Information Retrieval". Addison Weseley, 1999.

[8] E. Monkov, W. Cohen, and A. Ng. "Contextual Search and Name Disambiguation in Email using Graphs". SIGIR'06.

[9] A. Banerjee, S. Basu, and S. Merugu. "Multi-way Clustering on Relation Graphs". SIAM Data Mining'07.

[10] B. On, and D. Lee. "Scalable Name Disambiguation using Multi-level Graph Partition". SIAM Data Mining'07.

[11] E. Elmacioglu, Y. Tan, S. Yan, M. Kan, and D. Lee. "PSNUS: Web People Name Disambiguation by Simple Clustering with Rich Features". Int'l Workshop on Semantic Evaluation (SemEval), page 268-271, Prague, Czech Republic, June 2007.

[12] SecondString: Open source Java-based package of Approximate String-Matching "$http://secondstring.sourceforge.net/$".

[13] N. Slonim, N. Friedman, and N. Tishby. "Unsupervised Document Classification using Sequential Information Maximization". SIGIR'02.

[14] W. Rand. "Objective Criteria for the Evaluation of Clustering Methods". J. American Statistical Association, 66:846-850, 1971.

[15] K. Yeung, and W. Ruzzo. "Principal Component Analysis for Clustering Gene Expression Data". Bioinformatics, 17(9):763-774, 2001.

[16] M. Berry, and M Browne. "Understanding Search Engines". SIAM Press, 2005.

[2] Google currently provides us with a variety of keyword searching services – normal search, advanced search, image search, and so on.