Context Disambiguation in Web Search Results

Deepak P
Model Engineering College, Kochi, Kerala, India
deepak-p@eth.net

Jyothi John
Model Engineering College, Kochi, Kerala, India
jyothijohn@mec.ac.in

Sandeep Parameswaran
IBM Global Services India Pvt. Ltd, Bangalore, India
psandeep@in.ibm.com

Abstract

It is a common experience while web searching that one gets to see pages that are not of interest. Partly these are due to a word or words in the search query having different contexts, the user obviously expecting to find pages related to the context of interest. This paper proposes a method for disambiguating contexts in web search results.

1. Introduction

Among the most important intentions of web usage is information retrieval and the most common activity pursued to achieve that is web searching. As the web contains information on virtually all topics, precision is a very important yardstick to measure the quality of search engines. This paper reports (or describes) a method to make web search results more customized in certain cases where the query terms have multiple senses or even referents.

Section 2 defines the problem at hand and the nature of the proposed solution. Section 3 deals with the different issues that concern such a solution. Section 4 reviews some related works of interest in the context of the problem. Section 5 describes the approach used here. Section 5 presents the results of experiments using the approach in Section 6. Section 7 lists the conclusions followed by a listing of references in Section 8.

2. The problem and the target

Search results often are poorly customized. If a user is interested in an entity A (it may be anything ranging from an incident, a product, a person or a place) and there is more than one context for A, documents of all contexts are presented intermingled. The search engines currently use no method to distinguish documents from different contexts. From the results presented, the user has to eliminate the documents that that belong to contexts that he/she is not interested in. For this, a common user makes use of the automatically generated document extracts or descriptions that search engines provide with every page listed.

Common examples where there is more than one context for a query includes cases with multiple referents such as a search for a place, the name of which is so popular that places with that name occur in more than one area. A typical example is “Kochi”, there are two places named Kochi, one in Kerala, India and another in Japan. Similarly, there is a Hyderabad in India as well as Pakistan. Michael occurs as the first name of many sports stars. A search for Bachchan would yield pages relating to Amitabh Bachchan, a legendary Hindi actor as well as those on Abhishek Bachchan, the son of the former and a budding star in Hindi films.

Every user would have experienced this problem, but most of us take it for granted that it is our job to disambiguate the different contexts. This study aims at customizing web search results so that the pages relating to the same contexts (and same referents) may be presented together. This paper presents an approach whereby we can classify the pages and present to the user under different headings such as “pages on kochi in the context: ‘Kerala’, ‘India’”, “pages on Kochi in the context: ‘Japan’” etc., the former listing pages relating to the Kochi in Kerala, and the latter listing pages relating to the Kochi in Japan

3. Factors related to the problem

3.1. How it differs from the word sense disambiguation problem

Proceedings of the IEEE International Conference on Web Services (ICWS’04)
0-7695-2167-3/04 $ 20.00 IEEE

Word Sense Disambiguation is an active field of research in natural language processing. It addresses a similar issue, identifying the sense of a word (for a word having multiple senses) based on context. A typical example concerns differentiating the sense of 'letter' in the sentences, 'he wrote a letter' and 'e is a very frequently occurring letter'. The word sense disambiguation community works on largely language-based techniques (such as whether the word in question is used in the same grammatical form (noun, verb etc.) in both situations). Certain applications developed include usage of machine readable dictionaries. The problem being addressed here is very different. We are interested in the different contexts of the query (including multiple referents), rather than the sense of the query. A search for "Mumbai blast" would yield pages related to the blasts in 1993 as well as those related to the blast in 2003. Differentiating such contexts/referents would not be in the interests of the word sense disambiguation community. Moreover, our problem is not language specific and thus the solution should not rely on the language of the documents, provided that all documents are in the same language (we are not concerned with document-collections having documents from different languages). Thus the problem addressed here is inherently very different from the word sense disambiguation problem.

3.2. Differences from text categorization and neural-net based approaches

Text categorization, an active field in computational linguistics, is concerned with classifying documents of a set into disjoint sets. Current approaches rely on the similarity between documents and classify similar documents into the same set. Kohonen self-organizing maps also may be used to classify documents, such that similar ones are put into the same class. But the problem to be addressed here is to classify documents based on the context of the query used, and not just on the similarity. As the query is to be used as a parameter for classifying sets of documents, generalized neural networks and kohonen classifiers do not easily adapt to the problem due to their static structure. Furthermore, they do not provide the flexibility, (nor are there works to show that they are flexible enough) to incorporate the search query as a special parameter (such classifiers usually deal with all inputs uniformly).

3.3. Need for yet another clustering algorithm

There is enough reason to wonder why we need yet another clustering algorithm when we have so much of them in literature. The fundamental reason why many of the available clustering algorithms are unsuitable for this problem is the presence of an additional parameter, the search query. Most of the current clustering techniques take a set of documents and separate them into clusters, by methods which usually make use of the similarities between the documents. Here we have to cluster the web pages in the context of the search query used. The usage of similarity based clustering algorithms would surely be inappropriate (Such techniques would certainly put the government reports of "Mumbai blasts 1993" and "Mumbai blasts 2003" in the same context or cluster due to the inherent syntactical similarity between them as both are produced by the same government, probably using the same template) for the present problem.

Yet another reason is the need for speed. This service, when implemented as a meta-service has to work on the fly between the production and presentation of results. So speed is the major consideration in such cases and hence slow algorithms, (even if they are accurate and perfect) would not be acceptable. Literature that deals with this precise problem or present solutions which posses such desirable features as listed above could not be found.

3.4. Common context ambiguities

Cases where there is more than one context for a query are available in abundance. Common ones include:

- Multiple referents:
  - Place names: Two places having the same name
  - Names of people: There are many famous people with first names Michael
  - Different events in the same place: 'Mumbai blasts' has many referents, two of them being the blasts in 1993 and that in 2003
- Word Sense Ambiguities: These are much less important in the context of the web

3.5. Where to apply the solution

A context disambiguator designed to solve the said problem would invariably have to work on a corpus or a collection of pages. The amount of pages that the program gets to work on would provide more

insight into the nature of the solution to be developed and the optimizations that can be done on it. All depends on where the disambiguator has to work.

One possible implementation would be to have the disambiguator embedded into the search engine itself. Search engines that implement the popular HITS[1] algorithm [1] (hyperlink induced topic search) would then get a huge corpus of, say 1000 pages to work on. After splitting the whole set into different contexts containing few hundreds of pages each, the HITS algorithm can be applied separately to each one of those sets. The corpus that the HITS algorithm works on usually contains a lot of links among pages within it. Then, the solution would be able to make use of link-based information too. A possible disadvantage of this approach would be that of misinterpretation of unwanted pages present in the corpus as contexts, but more research has to go into investigating whether it is a problem in its own right, or whether its effects are too small to be taken seriously.

Another possible method would be to implement it as a meta-service, which gathers search results from a popular search engine and presents them to the user after disambiguation. The algorithm described in this paper is oriented towards such an implementation. Thus, the service would just present the same results, but differentiated into contexts/referents. The main disadvantage would be that this solution would get only 10-20 pages to work on, with very few links between them, thus almost ruling out the possibility of usage of link-based information. Unless we decide to consider links whose targets are not known to us, we would just be viewing the pages as text rather than the hypertext that they really are. But such solutions would be inherently fast, as they have to analyze only a handful of pages.

4. Related works of interest

Even though it has been postulated in section 3.3 that most of the current algorithms in literature do not suit the specific nature of the problem, a look into some other works whose results are of interest would definitely help.

A work [2] describes the methodology used by IBM's Textract to identify and extract relations in text. It extracts relations between concepts extracted from the documents using the documents themselves. A typical example mentioned in the work is that the text, "Gerstner, the CEO of IBM" can be used to extract a relation named "IBM" between the concepts, "Gerstner" and "CEO". It may well be argued that extraction of such relations from a collection of documents would provide a measure to classify documents into contexts. But the algorithms used by Textract are language-dependent, the first thing it does being the classification of each word as a name or member of a grammatical class. Further, the algorithms seem to be too inflexible to include the search query as a parameter. To add to it, such algorithms would be too computationally intensive (and thus slow) to work on the fly between generation and presentation of web search results.

Another work [3] describes an algorithm for clustering documents. It builds co-reference chains for each document, uses them to collect sentences from each document and eventually creates summaries for each document. The summaries are used to cluster documents into collections. The methodology is unsuitable for the problem at hand as it makes extensive use of language-based syntactic information like identifying the nouns, adjectives etc. Further, such techniques render the methodology inflexible to include the search query as a parameter. The last part of the approach uses the dot-product computation, a typical ingredient of clustering algorithms, as a similarity measure. The algorithm presented in this paper also uses the dot-product as a similarity measure.

A recent work [4] focuses on a very specific problem, that of distinguishing the real-world referent of a given name in context. A typical example would be distinguishing the different (real world) people from a collection of pages on different people having the same name. The approach presented by the work focuses on extracting biographical information such as year of birth, occupation etc. from the different documents using language dependent methods. Although distinguishing real-world referents of a given name would be of interest in our problem, such name ambiguities form just one of innumerous possible ambiguities in web search results. It can be readily recognized that devising such specific methods for every possible kind of ambiguity that a search engine user may come across would be impractical, if not impossible.

[1] HITS: This is an algorithm, which is used for web searching. It provides an ordered (ordered by relevance) list of authoritative pages and hubs (pages with lists of links to authoritative web pages) on the search query when supplied with a collection of about 1000 pages collected from a text based search for the query. It uses an iterative algorithm on the set of pages, aimed at boosting the scores of good hubs and authorities through the iterations.

5. A context/referent disambiguation algorithm

The context/referent disambiguator described below can be put to work on a collection of pages which contains pages from different contexts, such as multiple referents, for a search query. It builds a list of <word, score> tuples for each page and then, using the similarities between such lists, builds a graph with pages as nodes and undirected edges labeled with a measure of the similarity between pages. Every densely connected component in the graph is then taken to represent a different context/referent for the search query used. This algorithm is oriented towards implementation as a meta-service, and is expected to perform well even if it has just 10-20 documents to work with.

5.1. Part 1: Page-specific <word, score> list generation

This approach takes each page, analyses it and builds a list of <word, score> tuples for each page. Those keywords that are specific to the context/referent to which the page belongs should invariably be given high word scores. The more general inter-context keywords should end up with low scores. The proximity to the words in the search query would be an obvious parameter to the score computation function. The algorithm described here uses proximity to the search query and frequency as the parameters to the score computation function. It may be noted that the search query is a collection of inter-context (i.e., ambiguous) keywords. The algorithm makes no attempt to identify inter-context keywords (to suppress their scores) as such a procedure would be highly non-trivial and error-prone. The procedure is given below:

Table 1. Page-specific word-score list generation

```
Procedure ListGen(Page p)
{
  Score of every word in page p = 0;
  For every word, W in page p
  {
    For every occurrence w of W in page p
    {
      (freq_score of W) += 1;
      (prox_score of W) += (THRES - least
number of words intervening
       between w and a word in search query in
page p) + BOOST;
    }
    total_score of W = freq_score of W +
prox_score of W;
  }
  Normalize word scores such that the (sum of
total_score of every word in p = a limit);
  Make a list of <word, total_score> tuples
containing an entry for each word in p;
}
```

The freq_score holds the frequency of a word in page p. Prox_score holds the score of each word due to the proximity of its occurrences to the words in the search query. The scores are normalized such that they add up to an upper limit so that each page has the same influence in the next part of the stage of disambiguation. BOOST can be set to a value based on the relative weighting to be given to proximity and frequency based scores. The <word, score> tuples created here are used in subsequent stages.

5.2. Part 2: Document Clustering

This builds a huge list of unique words, with every word occurring in the list of atleast one page listed. Each page is represented as a vector with the i^{th} element holding the score for the i^{th} word in the list for that page. A graph is created with the pages as nodes and undirected edges[2] between two pages labeled with the dot product[3] of the vectors of the two pages. All edges having labels below a threshold are pruned so that densely connected subgraphs become isolated connected components. The pages in each such connected component are taken as belonging to a different context.

[2] Undirected Edges: Edges that do not have an orientation. They do not have a source or destination vertex, and they just connect the two vertices.

[3] Dot Product: It is the scalar product of two vectors and is defined as the sum of the products of corresponding components. It is denoted by a dot. E.g., <a,b,c>.<d,e,f> =ad+be+cf.

Table 2.Document Clustering

```
Procedure DocCluster( pages, each with an
associated list of <word, score> tuples )
{
   Create a list of n unique words, <word[1],
word[2]… word[n]> such that a word, w
    occurs in the list if it has an associated
score for atleast one page in the
    collection;
   Represent page p as a vector, <a[1], a[2],…
a[n]> where a[i] assumes the value,
       k where <word[i], k> is a tuple of
page p or
       0 if there is no tuple with word[i]
for page p
   Create an undirected graph[4] with pages as
nodes and the edges between any two
    pages labeled with the dot product of the
vectors of the two pages,
    scaled by an appropriate factor;
   Prune every edge in the graph where the
weight is below a THRESHOLD;
   Each densely connected subgraph of the
resulting graph represents
    a different context;
}
```

A value chosen for the THRESHOLD is crucial to the algorithm described here. Fixing the THRESHOLD at a low value may result in merger of contexts/referents, and thus poor ambiguity resolution. A high value would invariably break-up contexts/referents which seem logically coherent to the human into further smaller contexts, the worst case being interpreting each page as a different context/referent.

5.3. Part 3: Identification of context-keywords to describe the context/referent

After identifying the different contexts/referents, we have to describe each context/referent to the user by a set of keywords. The algorithm given below uses a rather naïve method, creating a context-wide list of words, each word associated with a score that is computed by taking the sum of scores for the word in each page of the context group. The words with the highest such scores are taken to be the context keywords. The procedure can be outlined as follows

Table 3.Identification of context-keywords to describe a context/referent

```
Procedure Context-Keywords (collection of
pages of a context)
{
  Create a list of n unique words, <word[1],
word[2]… word[n]> such that a word, w
    occurs in the list if it has an associated
score for atleast one page in the
    collection;
  Associate each word with a score initialized
to 0;
  For (each page, p)
  {
    Score of word[i] += k where there is a
tuple <word[i], k> for page p
  }
  Take the words with the highest scores as
the context-keywords for the context in
    question;
}
```

5.4. Related Issues

The score computation part assigns scores to words based on their relative frequency and proximity to the search query words. It is based on the assumption that context/referent-specific keywords occur close to occurrences of search query words. This should work very well in cases where the matching surname causes the ambiguity. No effort has been made to decrease the influence of inter-context words; as such efforts are likely to result in misinterpretations and decreased performance. No weighting has been given to link-based information as the target of the link would be unknown in most cases. Thus it treats hypertext just as text documents. Finding the densely connected disjoint subgraphs should be a complex and time-consuming algorithm, although usage of such an algorithm would certainly provide better results than pruning of edges which have weights below a certain threshold (the approach used here). If no isolated connected components are found by part 2 of the algorithm, we may conclude that there is only one context for the whole set of pages.

6. Test Results

The algorithm as given above was implemented and tested on two collections: a collection of pages on the search query, “Kochi” containing pages on the places with the said names in India and Japan and another collection which contained pages on Joe Jackson and Michael Jackson and collected using the search query, “Jackson”. After testing these two collections, where the two contexts/referents were explicitly known beforehand, testing was performed on various other collections as described in 6.3. For the tests, BOOST and THRES were given values of 4 and 10 respectively (Refer algorithm in section 5.2). In all

[4] Undirected Graph: It is a graph with only undirected edges connecting the vertices.

these tests, the graphs obtained were processed by first pruning edges having weights lesser than 20% of the weight of the strongest edge. Then the edges were pruned from the remaining graph with the threshold for pruning edges chosen as half the weight of the strongest edge or the value that preserves only 20% of edges, whichever was lower.

6.1. The "Kochi" Collection

This collection contained 10 pages, 5 on the Japanese city and 5 on the Indian city. The graph produced after part 2 of the algorithm is given in Figure 1.

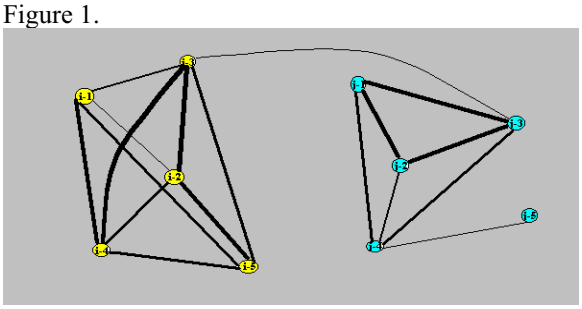

Fig. 1. The "kochi" collection Graph

In the above figure, Yellow nodes represent pages about kochi, India and the blue ones represent those about kochi, Japan. The thicknesses of the edges represent the magnitude of the labels of the edges. As can be seen, there is just one link, a very weak link that is inter-contextual. After analysis it was found that the word "page" occurred often in both and it was that inter-context keyword that manifested as an edge. The page "j-5" which is pendant with a weak edge associated with it, was found to contain a very small amount of information. The page mostly contained pictures and the text material occupied just about 926 bytes. The pages on Kochi, India seem to form a densely connected subgraph with strong edges and so do all the pages (except one) on Kochi, Japan. It can hence be concluded that the algorithm does work fine for this collection.

The keyword identification algorithm was applied to the collection of pages on Kochi, India. The highest ranked words were "Kerala" and "Cochin". Kerala is the state in India where Kochi is actually situated and Cochin is the former name of Kochi, India. The algorithm when applied on the set of pages on Kochi, Japan yielded "Japanese" as the word with the highest score followed by "Japan". Thus the keyword identification algorithm is also seen to work well.

6.2. The "Jackson" collection

This set contained 9 pages, 6 pages on "Joe Jackson" and the remaining three on "Michael Jackson". It was manually verified that all the pages on "Joe Jackson" referred to the same Joe Jackson and that all the pages on "Michael Jackson" referred to the same Michael Jackson. This test was done to test the performance of the algorithm when the relative number of pages in different contexts differs widely. The graph produced after part 2 of the algorithm is given in Figure 2.

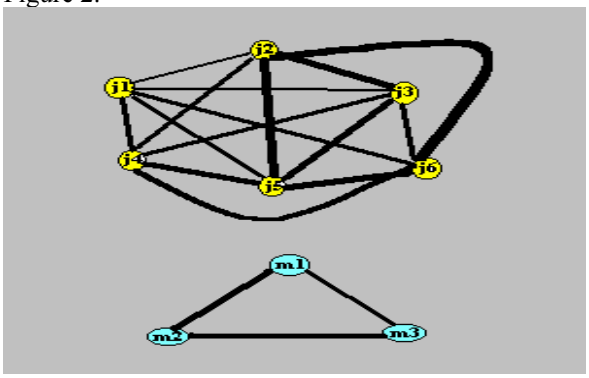

Fig. 2. The "Jackson" collection graph

Yellow nodes represent pages about Joe Jackson and the blue ones represent those about Michael Jackson. The thicknesses of the edges represent the magnitude of the labels of the edges. As can be seen, the pages from each context form a dense clique themselves. Thus the algorithm can be interpreted to have worked extremely well. The keyword identification algorithm identifies "Joe" as the top keyword for the "Joe Jackson" pages and "Michael" as the top keyword for the "Michael Jackson" pages. The algorithms seem to have disambiguated the contexts/referents exactly as humans would have done.

6.3. Tests on Miscellaneous Collections

Sections 6.1 and 6.2 were tests conducted on collections where the expected contexts were known beforehand. Further, tests were performed on other collections obtained from a search engine called Google (http://www.google.com) for various queries. Pages listed as the top 20 (in the results) were downloaded and subjected to the context disambiguation algorithm. Each of the contexts obtained were subjected to manual inspection and short descriptions of the pages listed in the different contexts were made. Further, disagreements between the contexts listed by the algorithm and the contexts that can be identified by manual inspection were also taken note of. The queries used were chosen at random (not like the tests of sections 6.1 and 6.2 where the desirable contexts were known beforehand). The

following subsections describe the results on a selected 6 collections, a random sample from the 21 collections used for testing. Each context is labeled by the search query used to gather the collection. From the results of the tests, it was seen that the algorithm does perform well even in cases were the different contexts were not as dissimilar as in the test collections of sections 6.1 and 6.2.

6.3.1. "Birthday" Collection. This collection contained 17 pages. The algorithm identified a context of 10 pages, all of which were seen to deal with online greeting pages or birthday parties and methods of celebrating them. But two pages on birthday parties among the other 7 pages were found worthy of inclusion in the context on manual inspection. The other pages were those dealing with the birthday celebrations of various political or cultural leaders, which bore no similarity among them.

6.3.2. "Compilers" Collection. This collection contained 11 pages. Two contexts were identified by the algorithm, one containing 3 pages giving information about 'what compilers are' and another of 2 pages providing lists of different compilers available. All other pages were homepages of different compilers which evidently cannot be clustered into a context given the nature of the algorithm. The algorithm was seen to have performed very well in this collection.

6.3.3. "Dijkstra" Collection. Dijkstra was a famous Dutch computer scientist. The collection contained 6 files. A context of 4 pages was identified by the algorithm, all of which contained biographical sketches on the life of the scientist. One among the excluded pages contained a description of Dijkstra's algorithm and the other was an extract from the scientist's lecture against the usage of 'goto' statements in programming. So the algorithm is seen to have performed very well here.

6.3.4. "Kerala" Collection. This collection contained 12 files. "Kerala" is the name of a state in India. The algorithm did not identify any worthy community. No edge survived the edge pruning phase. Upon analysis of the collection, it was found that the pages contained the homepages of kerala police, kerala high court, kerala tourism department, kerala government, a matrimony site etc. Thus there were no two pages which had some semantic similarity. Thus the algorithm did perform well even in this extremely adverse situation.

6.3.5. "Telgi" Collection. Telgi is the name of the prime-accused in a notorious fake stamp-paper case in India. The collection contained 17 files, of which the algorithm identified two contexts. One of them was a context of 6 pages, all of which were newspaper reports on a rumor as to whether Telgi was injected with HIV when in police custody. The other context also contained 6 pages, all of which were newspaper reports on the government stand on the Telgi issue. The other pages were reports on various minor details of the Telgi case and a page on the HIV rumor, which on manual inspection, clearly was found to be worthy of belonging to the first context. Thus the algorithm is seen to have performed well, although not "perfectly".

6.3.6. "Police" Collection. This collection contained 19 pages obtained by the query "Police" on pages relating to "Kochi". The algorithm identified a single context of 6 pages. 5 of those pages contained reports on crimes in and around the city of Kochi and the other one was a list of the names of different police officers in Kochi city. It was further seen that the list of officers bore strong edges to each of the other 5 pages, although it doesn't seem to be worthy of being included in a context consisting of crime reports. The links were found to be due a high frequency of investigating officers' names in crime reports. Such cases were a logically "different" page holds a community of cohesive pages together, may be worthy enough to be subjected to further investigation. The pages not listed in the identified community were seen to be very different from each other. Except for a 'surprising inclusion', the algorithm did not make a 'mistake'.

7. Conclusions and future work

It can be seen from the tests that the algorithms perform very well even when they have very little information to work with. Thus implementing context/referent disambiguation as a meta-service may be an interesting experiment. The obvious concern would be as to how the algorithm would scale with increase in the number of pages in input. Search engines usually provide results in pages of 10 (or more) results. Thus, a meta-service implementation of the disambiguator need process only 2-3 result lists, and thus 20-30 pages, at a given time and can defer the disambiguation of other pages to a time when they are actually required. Thus we are justified in ignoring the problem of scalability altogether if the algorithm is to be implemented as a meta-service. But in cases where the algorithm is to be embedded into the search

algorithm, such issues may assume greater significance.

Future work may be directed towards formulating possible optimizations when the disambiguator is implemented within a web search algorithm (in the search engine architecture itself). The optimizations may be formulated to exploit the larger corpus. Further link based information may also be used in such cases. The keyword identification algorithm can also be improved considerably making use of principles and results from computational linguistics. The performance of the algorithms in very broad topic queries such as "union" whose context ranges from "rugby unions" to "c unions" and in cases of word sense ambiguities and other issues, not dealt with in the current paper, have to be investigated. Such cases are clearly different from the cases discussed through the bulk of the paper, but investigations as to how the algorithms presented here work on such problems, might provide valuable clues on what the nature of solutions for such varied problems should be.

8. References

[1]. Jon M Klienberg, "Authoritative sources in a hyperlinked environment", Journal of the ACM, 1999, Vol:46, No: 5, p.604-632, http://citeseer.nj.nec.com/kleinberg99authoritative.html

[2]. Roy J Byrd, Yael Ravin, "Identifying and Extracting relations in text", In Proceedings of the 5th joint conference on Information Sciences, JCIS2000, 2000, pp. 207-210

[3]. Amit Bagga, Alan. W Biermann, "A methodology for cross-document coreference", in the proceedings of NLDB99, Austria

[4]. Giedeon S Mann, David Yarowsky, "Unsupervised personal name disambiguation", In Proceedings of the Seventh CoNLL Conference, Edmonton (May-June, 2003), pp. 33-40