Exploiting Web Querying for Web People Search

RABIA NURAY-TURAN, DMITRI V. KALASHNIKOV, and SHARAD MEHROTRA,
University of California, Irvine

Searching for people on the Web is one of the most common query types submitted to Web search engines today. However, when a person name is queried, the returned Webpages often contain documents related to several distinct namesakes who have the queried name. The task of disambiguating and finding the Webpages related to the specific person of interest is left to the user. Many Web People Search (WePS) approaches have been developed recently that attempt to automate this disambiguation process. Nevertheless, the disambiguation quality of these techniques leaves major room for improvement. In this article, we present a new WePS approach. It is based on issuing additional auxiliary queries to the Web to gain additional knowledge about the Webpages that need to be disambiguated. Thus, the approach uses the Web as an external data source by issuing queries to collect co-occurrence statistics. These statistics are used to assess the overlap of the *contextual entities* extracted from the Webpages. The article also proposes a methodology to make this Web querying technique efficient. Further, the article proposes an approach that is capable of combining various types of disambiguating information, including other common types of similarities, by applying a correlation clustering approach with after-clustering of singleton clusters. These properties allow the framework to get an advantage in terms of result quality over other state-of-the-art WePS techniques.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database Applications—*Web People Search*; H.2.8 [**Database Management**]: Database Applications—*Entity Resolution*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-based services*

General Terms: Algorithms, Design, Experimentation, Performance, Theory

Additional Key Words and Phrases: Web people search, WePS, entity resolution, automated Web querying, skyline-based classifier

ACM Reference Format:
Nuray-Turan, R., Kalashnikov, D. V., and Mehrotra, S. 2012. Exploiting web querying for web people search. ACM Trans. Datab. Syst. 37, 1, Article 7 (February 2012), 41 pages.
DOI = 10.1145/2109196.2109203 http://doi.acm.org/10.1145/2109196.2109203

1. INTRODUCTION

Search engines are among the most important Web technologies that empower users to locate and analyze information of interest from billions of Webpages. According to Search Engine Watch [2006], the Google search engine handles 91 million queries daily of which about 5 to 10% are Web people searches wherein users are looking for Webpages of a specific individual using the individual's name [Guha and Garg 2004]. Given the very large number of people search queries, it is surprising that support for such queries in modern search engines such as Google, Bing, and Yahoo! is still quite primitive. For instance, when a user query consists of a person name, search engine

This work was supported in part by NSF grant CNS-1118114 and a gift from Google.
Authors' addresses: R. Nuray-Turan, D. V. Kalashnikov (corresponding author), and S. Mehrotra, University of California, Irvine; email: dvk@ics.uci.edu.

DOI 10.1145/2109196.2109203 http://doi.acm.org/10.1145/2109196.2109203

return a ranked list of Webpages that correspond not only to the individual of interest to the user, but also to those of his namesakes. If a user is interested in Webpages of a particular person, he has to manually disambiguate the returned Webpages.

This problem of disambiguation arises almost every time a Web people search query is issued, since most individuals' namesakes also have Web presence.[1] For instance, given a query for "Tom Mitchell", the top 100 results returned by Google contain 37 different namesakes. Accordingly, if a user is interested in only a specific "Tom Mitchell" (e.g., the one who is a professor at CMU) he has to look through the top 100 results to figure out the Webpages of interest.[2]

One can argue that while having to manually filter out the returned results is a nuisance, it is not a very serious problem since users can browse through the ranked results and choose the Webpage corresponding to the individual of interest. There are at least two situations where the preceding argument does not hold. First, a user's interest might not be to identify a specific Webpage of an individual, but a set of Webpages to build a profile. For instance, a user might be looking for an author on Google Scholar to measure his/her impact using bibliometrics such as h-index [Hirsch 2005]. Since Google Scholar does not support the disambiguation of authors, the returned list of papers usually corresponds to the papers written by different namesakes, which makes the accurate determination of h-index difficult. Another situation where the ranked list returned by general search engines does not suffice is when the namesakes include someone who is very famous (referred to as the famous person problem [Kalashnikov et al. 2008a]). If the user's interest is in the Webpages related to someone other than the famous person, a user may have to scan through pages of search results most of which are irrelevant to the user's information need.

The limitations of general search engines to adequately support person name search have attracted significant research interest recently. Since the time the problem was identified [Artiles et al. 2005; Kalashnikov et al. 2007], many Web people search engines have been designed both in academia and in industry. Web People Search (WePS) systems differ in their support for queries based on individual's names. Instead of returning a ranked list of Webpages $D = \{d_1, d_2, \ldots, d_k\}$, these systems attempt to return clusters of Webpages in D such that each cluster corresponds to a namesake. For instance, for the previous "Tom Mitchell" query over Google, the WePS system will return one cluster for each "Tom Mitchell" enabling the user to quickly identify the specific individual of interest and perform the analysis. Likewise all the Webpages of a famous person will be clustered together, enabling other namesake's pages to be visible in the top few results.

The primary technology driving WePS is that of Entity Resolution (ER) and a variety of ER techniques have been explored in the context of WePS. A direct application of ER solutions to the people search problem is to extract features from Webpages such as n-grams, named entities, hyperlinks, etc., and then use an ER technique with these features [Yoshida et al. 2010; Mann and Yarowsky 2003; Niu et al. 2004; Iria et al. 2007; Gong and Oard 2009; Elmacioglu et al. 2007b]. Performance of such techniques that use features extracted directly from the Webpages (which we refer to as *direct features*) is limited due to a variety of reasons as we discuss in Section 9. To overcome the limitations, some approaches have considered using indirect features (also referred to as

[1] According to U.S. Census bureau about 100 million people have 90,000 names with an average of about 1000 people with one name. Therefore, the likelihood of multiple namesakes having Web presence is very high.

[2] Frequently the problem does not disappear even if the context keywords are provided along with the person name. For instance, the returned Webpages for the preceding query with the additional keyword "CMU": (a) will not include many of his Webpages that do not include keyword "CMU", making the recall lower, and (b) will actually still refer to more than one namesake, thus not leading to perfect precision either.

external features), which include publicly available knowledge-bases such as Wikipedia [Han and Zhao 2009, 2010] and Web directories [Vu et al. 2008], the connectivity of the Webpages [Bekkerman and McCallum 2005], and the Web search engine indexes [Chen et al. 2009a; Rao et al. 2007].

One of the most powerful external knowledge bases for the Web people search is the Web itself, as it contains a large body of information about people, the entities they are related to, the topics they are mentioned in, etc. The key question in exploiting such information for people search is how can an ER system systematically acquire and exploit such external knowledge from the Web. The primary observation this article is based on is that such information can be obtained from the Web search engines by rightly formulated search engine queries that can help identify interrelationships among entities mentioned on the Webpages returned as original results to the people search query. We will refer to such entities as *contextual entities*. Developing a people search mechanism that exploits interrelationships among the contextual entities for disambiguation raises multiple questions.

—What types of queries should be submitted to the Web search engine to gather information about interrelationships among the contextual entities returned as part of the answer to the person search query?
—How can such (external) interrelationships information be combined with the other (direct) features to get highly accurate clusters of search results?
—If a Webpage contains only limited amount of information about the person of interest, can some method be devised to increase confidence of whether the page should be in the result set by itself (i.e., a singleton cluster) or it should be merged with another cluster?
—Querying Web search engines increases computation time. What techniques can be devised to limit the number of queries issued to learn the interrelationships so that the execution time can be constrained?

In this article, we address the questions identified before to develop a principled approach that uses Web search engine queries for Web people search. In the proposed approach, interrelationships are defined in terms of the Web co-occurrence statistics of the contextual entities extracted from the original set of results returned to the people search query. To collect such information the proposed approach submits a set of queries to the search engine, which measures the strength of the interrelatedness of the contextual entities. Co-occurrence features are then combined using a specialized classifier based on *skylines* to derive the external feature-based similarities. The classifier is specialized for the purpose of entity resolution by taking into account the dominance that exists in co-occurrence data. Our experiments show that the skyline-based classifier gets quality improvement over other classifiers such as decision tree classifiers and support vector machines.

The output of the skyline-based classifier is aggregated with the direct feature similarities to compute the overall similarity of Webpage pairs. The aggregation is achieved using a *simulated-annealing*-based classification algorithm, which learns the importance of each feature for the overall similarity. Our experiments show that exploiting more direct features increases the quality of the overall clustering.

If there are Webpages with limited information about the namesake they mention, then the output of the clustering will contain singleton 1-page clusters even if this Webpage should be a part of another cluster. To overcome such wrong singleton clusters, the article develops a two-step clustering approach. In the first step direct and indirect features are used to create initial clustering. The second phase of the algorithm tries to utilize the limited information that exists in such a Webpage to decide whether it should be merged with another cluster. The cluster refinement step improves the clustering quality significantly, as will be shown in Section 8.

Finally, the article proposes an algorithm to make the information collection scalable, by minimizing the number of queries submitted to the search engine. It can get the same quality results as it could get if all of the queries were issued by submitting only a fraction of the queries to the search engine.

The rest of this article is organized as follows. Section 2 provides an overview of the proposed approach. Then Section 3 covers the steps of the Web querying part of the approach in more detail. The efficiency optimization for the Web querying algorithm is explained in Section 4. A learning algorithm for aggregating both direct and indirect feature sets is described in Section 5. The cluster refinement step, an approach to deal with singleton clusters, is explained in Section 6. Next, Section 7 compares different possible architectures for implementing a WePS system. The overall approach is then empirically evaluated in Section 8 and compared to state-of-the-art solutions. Section 9 summarizes the related work. Section 10 concludes the article by highlighting the impact of the proposed solution. Finally, Appendix A.1 discusses the filters in Web query selection and Appendix A.2 explains the selected direct feature set.

2. OVERVIEW OF THE APPROACH

The main task of a WePS system is to accurately cluster the Webpages in the result set D to a person search query, such that each resulting cluster corresponds to a namesake. The clustering problem can be represented graphically, where the set of nodes $V = \{v_1, v_2, \dots, v_k\}$ of the graph $G = (V, E)$ corresponds to the set of Webpages to be clustered. In the simplest case, an edge is created per each distinct pair of nodes $e_{ij} = (v_i, v_j)$. The edges are labeled later on by the algorithm with various types of similarities. The task of a clustering algorithm can be viewed as that of taking the labeled graph G as input and partitioning it into clusters according to the labeling of its edges, such that each cluster corresponds to a namesake. The WePS system we propose performs the following processing steps.

(1) *User input.* User submits a query Q to the WePS system.
(2) *Webpage retrieval.* The WePS system then queries a search engine such as Google with the query Q and retrieves top-K Webpages $D = \{d_1, d_2, \dots, d_k\}$.
(3) *Preprocessing.* The Webpages in D are preprocessed.
 —*Feature Extraction.* The preprocessing step first converts HTML files to plain text by filtering out HTML tags. It then performs tokenization and extracts features such as named entities, n-grams, hyperlinks, emails, etc., from the Webpage. Stemming is performed on the n-grams and tokens. Stop-words, punctuation, and popular Websites that are associated with the Webpage are removed.
 —*Indexing.* TF.IDF factors for terms and named entities are computed and indexed. Hyperlinks are normalized and stored, and n-gram counts are indexed as well.
(4) *Similarity computation.* As illustrated in Figure 1, this part computes various similarities among the Webpages in D to label the edges in graph G. The similarity can be in the form of direct similarity as well as dissimilarity, that is, the evidence that two Webpages are not the same. For example, the fact that the middle names of two names do not match can be converted into a measure of dissimilarity. The overall similarity computation is done in three steps.
 (a) *Direct Similarity Computation.* For each pair of Webpages d_i, d_j, various similarity metrics are computed using their extracted features. For instance, named-entity-based cosine similarity of the Webpages is computed using TF.IDF factors. The complete list of the direct features and corresponding similarity functions we use is explained in Appendix A.2.

```
Compute-Similarity($G, D$)
    $G \leftarrow$ Update-Named-Entity-Sim($G, D$)
    $G \leftarrow$ Update-N-Gram-Sim($G, D$)
    $G \leftarrow$ Update-Hyperlink-Sim($G, D$)
    $G \leftarrow$ Update-Email-Sim($G, D$)
    $G \leftarrow$ Update-Middle-Initial-Dissim($G, D$)
    $G \leftarrow$ Update-Social-Networking-Site-Dissim($G, D$)
    $G \leftarrow$ Update-Web-Query-Based-Sim($G, D$)
    return $G$
```

Fig. 1. Computing similarities.

(b) *Indirect Similarity Computation.* Indirect similarity of Webpage pairs is computed using Web co-occurrence statistics, which are collected using a set of queries to a Web search engine such as Yahoo!. The collected statistics are then converted into a similarity measure using a *skyline-based classifier* that learns how to convert the indirect feature-based similarities to a "merge" or "nonmerge" decision (Section 3).

(c) *Combining Similarities.* Direct and indirect similarities are then combined to form overall similarity that is used to label the graph *G*. An adaptive approach is used to combine the direct and indirect features to compute overall similarity (Section 5).

(5) *Clustering.* This step clusters the Webpages based on the collected similarities.

Phase 1: *Correlation Clustering.* The labeled graph *G* is partitioned into its clusters using the Correlation Clustering (CC) [Bansal et al. 2004]. The goal of CC is to find the partition of the graph that agrees the most with the assigned edge labels.

Phase 2: *Cluster Refinement.* After correlation clustering is applied, often the result consists of a few large clusters and several singleton 1-Webpage clusters. Naturally, there can be true and false singleton clusters. The cluster refinement step tries to resolve the issue of false singleton clusters (Section 6). It first computes the similarities of these singleton 1-page clusters to the remaining pages using a different strategy. The Webpages are merged with another cluster if their similarity exceeds a certain threshold. This threshold is estimated per queried name by exploiting the number of initial clusters.

(6) *Postprocessing.* Each resulting cluster is then further processed to generate cluster sketches to summarize the content of each cluster, ranks the clusters to decide the order in which they are presented to the user, and ranks Webpages inside each cluster.

(7) *Visualization of results.* The results are presented to the user in the form of clusters (and their sketches) corresponding to namesakes and that can be explored further.

We next explain the similarity computation steps.

3. WEB QUERY-BASED SIMILARITY COMPUTATIONS

Our goal in this section is to develop a mechanism that exploits Web search engine statistics to provide additional evidence of whether two Webpages correspond to the same namesake. Such additional evidence plays an important role in disambiguating among the namesakes, especially in difficult cases such as when direct features extracted from the Webpages do not provide enough evidence to make an appropriate decision.

Consider, for instance, two pages of Tom Mitchell, the CMU professor. Let us assume that the first page d_1 is his DBLP page, which is a good source to identify the people he is related to as well as his topic of interest, for example, “learning”, “Andrew McCallum”, and “Sebastian Thrun”. The second page d_2, on the other hand, is a talk announcement from USC, which has very brief information about him. The second page contains information only about the organizations he is related to, for example, “Carnegie Mellon University” and “Center for Automated Learning and Discovery,” which are not listed on his DBLP page. The content of these Webpages may not directly provide evidence to declare them as coreferring, so using direct features only might not group these pages correctly. However, if we look at the contextual entities (i.e.,“Andrew McCallum” and “Sebastian Thrun” for d_1, and “Carnegie Mellon University” and “Center for Automated Learning and Discovery” for d_2) and exploit their relationship by submitting queries to the search engine such as:

> Q: “Tom Mitchell” `AND` (“Sebastian Thrun” `OR` “Andrew McCallum”) `AND` (“Carnegie Mellon University” `OR` “Center for Automated Learning and Discovery”),

we may find enough evidence that these contextual entities are interrelated.[3] Let us contrast the preceding with a situation where Webpages of two different Tom Mitchell namesakes (and their corresponding contextual entities) are not expected to correlate as much as the Webpages of the same namesake. For instance, let us assume that we want to gather additional evidence through Web querying for the talk announcement page from USC and for the Rabbi Tom Mitchell’s homepage, which contains “Yahshua Messiah” and “JoAnne Elisabeth Kulow” as the contextual entities related to him. When we query the search engine about the interrelationships of these entities, we observe that there is no co-occurrence among them.[4]

In this section, we first explain the set of queries that can be executed on the search engine to learn such interrelationships and then we develop a skyline-based classifier to convert the collected co-occurrence statistics into a similarity score.

3.1. Co-Occurrence Queries and Features

Let d_i and d_j be the Webpages under consideration and $\mathcal{C}_i$ and $\mathcal{C}_j$ be the associated contextual entities of these Webpages, respectively. To learn the interrelationship of these entities, we use two different types of queries.

(1) N `AND` $\mathcal{C}_i$ `AND` $\mathcal{C}_j$, which estimates the degree of overlap of two contexts $\mathcal{C}_i$ and $\mathcal{C}_j$ for Webpages d_i and d_j for the queried name N.
(2) $\mathcal{C}_i$ `AND` $\mathcal{C}_j$, which on the other hand estimate the degree of overlap of two contexts $\mathcal{C}_i$ and $\mathcal{C}_j$ for Webpages d_i and d_j in general.

Here, $\mathcal{C}_i$ can be either the set of people NEs $\mathcal{P}_i$ or organization NEs $\mathcal{O}_i$. Context $\mathcal{C}_j$ is defined similarly for Webpage d_j. That is, each context $\mathcal{C}_i$ has two possible assignments, which lead to 4 context combinations for the $\mathcal{C}_i$ and $\mathcal{C}_j$ pair. For each context combination we create two types of queries, one with the queried person name and one without it. That is, we create 8 queries per Webpage pair d_i, d_j in total. The pseudocode in Figure 2 illustrates the procedure for formulating the queries.

For instance, for the “Tom Mitchell” example given in the Introduction, suppose that the algorithm extracts 2 named entities of each type per Webpage, that is, $m = 2$. Then assume that the person NEs extracted from d_i are “Sebastian Thrun” and “Andrew

[3] For instance, query Q currently returns 7,790 results when issued to the Google search engine, which might be enough evidence to declare them as coreferring.

[4] Web query for two webpages of Tom Mitchell who are not the same returns 0 answers.

```
GET-WEB-COUNTS($d_i, d_j$)
Let:
$N$ be the queried name
$\mathcal{P}_{i1}, \mathcal{P}_{i2}, \ldots, \mathcal{P}_{im}$ be the people NEs extracted from $d_i$
$\mathcal{P}_{j1}, \mathcal{P}_{j2}, \ldots, \mathcal{P}_{jm}$ be the people NEs extracted from $d_j$
$\mathcal{O}_{i1}, \mathcal{O}_{i2}, \ldots, \mathcal{O}_{im}$ be the org. NEs extracted from $d_i$
$\mathcal{O}_{j1}, \mathcal{O}_{j2}, \ldots, \mathcal{O}_{jm}$ be the org. NEs extracted from $d_j$

$\mathcal{P}_i \leftarrow (\mathcal{P}_{i1}$ OR $\mathcal{P}_{i2}$ OR $\cdots$ OR $\mathcal{P}_{im})$
$\mathcal{P}_j \leftarrow (\mathcal{P}_{j1}$ OR $\mathcal{P}_{j2}$ OR $\cdots$ OR $\mathcal{P}_{jm})$
$\mathcal{O}_i \leftarrow (\mathcal{O}_{i1}$ OR $\mathcal{O}_{i2}$ OR $\cdots$ OR $\mathcal{O}_{im})$
$\mathcal{O}_j \leftarrow (\mathcal{O}_{j1}$ OR $\mathcal{O}_{j2}$ OR $\cdots$ OR $\mathcal{O}_{jm})$

$c_{ij1} \leftarrow$ GetWebCount($N$ AND $\mathcal{P}_i$ AND $\mathcal{P}_j$)
$c_{ij2} \leftarrow$ GetWebCount($\mathcal{P}_i$ AND $\mathcal{P}_j$)
$c_{ij3} \leftarrow$ GetWebCount($N$ AND $\mathcal{P}_i$ AND $\mathcal{O}_j$)
$c_{ij4} \leftarrow$ GetWebCount($\mathcal{P}_i$ AND $\mathcal{O}_j$)
$c_{ij5} \leftarrow$ GetWebCount($N$ AND $\mathcal{O}_i$ AND $\mathcal{P}_j$)
$c_{ij6} \leftarrow$ GetWebCount($\mathcal{O}_i$ AND $\mathcal{P}_j$)
$c_{ij7} \leftarrow$ GetWebCount($N$ AND $\mathcal{O}_i$ AND $\mathcal{O}_j$)
$c_{ij8} \leftarrow$ GetWebCount($\mathcal{O}_i$ AND $\mathcal{O}_j$)
```

Fig. 2. Algorithm for querying the Web.

McCallum", and the person NEs extracted from Webpage d_j are "William Cohen" and "Andrew Ng". Then the person NE queries will be as follows.

> *Q*1: "Tom Mitchell" AND ("Sebastian Thrun" OR "Andrew McCallum") AND ("William Cohen" OR "Andrew Ng") .
> *Q*2: ("Sebastian Thrun" OR "Andrew McCallum") AND ("William Cohen" OR "Andrew Ng")

We can utilize the Web search engine APIs that return the estimated number of results for queries, without retrieving the results themselves. Once the queries are created and executed on the search engine, we need to convert the collected Web co-occurrence statistics, for example, $|N \cdot \mathcal{C}_i \cdot \mathcal{C}_j|$ which refers to the number of results for the N AND $\mathcal{C}_i$ AND $\mathcal{C}_j$ query, into the corresponding co-occurrence features that can be used by the algorithm. It might be difficult to interpret the absolute co-occurrence value without comparing it to certain other values. For instance, if this count value is high, does it mean the contexts overlap significantly and thus d_i and d_j should be merged? Or, is it simply because N is a common name and thus there are lots of Webpages that contain it under many contexts? Or, is it because contexts $\mathcal{C}_i$ and $\mathcal{C}_j$ are too unspecific, and thus too many Webpages contain them? To address these issues, we normalize co-occurrence counts using a set-based similarity measure such as Jaccard or Dice similarities. There have been studies showing that the differences in retrieval quality, when using different set-based measures, is insignificant and furthermore these measures are monotone with respect to each other [Lerman 1970]. Accordingly, the proposed algorithm uses the Dice similarity to get the normalized version of $|N \cdot \mathcal{C}_i \cdot \mathcal{C}_j|$.

The Dice similarity between two sets A and B computes the fraction of common elements in A and B among all the elements (including nondistinct) in A and B, and normalizes it to the [0, 1] interval.

$$Dice(A, B) = \frac{2|A \cap B|}{|A| + |B|} \tag{1}$$

Observe that there are two different ways to normalize the co-occurrence count using Dice similarity. We can view the $N \cdot \mathcal{C}_i \cdot \mathcal{C}_j$ set: (1) as the intersection of the sets $N \cdot \mathcal{C}_i$ and $N \cdot \mathcal{C}_j$ or (2) as the intersection of the sets N and $\mathcal{C}_i \cdot \mathcal{C}_j$. We have

$$Dice_1(N \cdot \mathcal{C}_i, N \cdot \mathcal{C}_j) = \frac{2|N \cdot \mathcal{C}_i \cdot \mathcal{C}_j|}{|N \cdot \mathcal{C}_i| + |N \cdot \mathcal{C}_j|},$$

and

$$Dice_2(N, \mathcal{C}_i \cdot \mathcal{C}_j) = \frac{2|N \cdot \mathcal{C}_i \cdot \mathcal{C}_j|}{|N| + |\mathcal{C}_i \cdot \mathcal{C}_j|}.$$

The algorithm employs the second formula, as it has proven to capture the ambiguity of context better. The following example provides just one type of scenario to illustrate the choice of the formula. Assume that the namesakes mentioned in two Webpages d_i and d_j are different. Suppose that the extractor makes a mistake and extracts "This" as the only person NE from d_i, and "That" as the only person NE from d_j. Assume that all (or, most of) the Webpages of the two namesakes contain both "this" and "that". Then, $Dice_1$ similarity will be 1 (or, very high), causing the wrong merge of the Webpages. However, the $Dice_2$ similarity will be low, since $|\mathcal{C}_i \cdot \mathcal{C}_j|$ will be large. That is, $Dice_2$ will automatically capture that "this" and "that" is not a good choice to be used as the contexts.

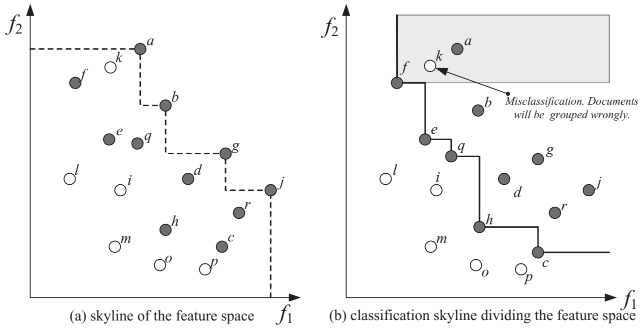
We note that we need to filter out certain types of NEs that are too ambiguous to be used in Web querying in order to make queries more effective in collecting merge evidence. For instance, the approach filters out an NE if it is also a common English word. Filters play an important role in the Web query generation, since without them the proposed technique might cause the algorithm to collect wrong evidence. This wrong evidence will lead to wrong merge decisions so that the cluster will not be pure. Interested readers can find the detailed explanation of filters we used in Appendix A.1.

At first glance it might seem the algorithm employs two different strategies to achieve the same goal. That is, it detects ambiguous context by using the filtering strategy. But, frequently the same can be achieved by simply using the $|\mathcal{C}_i \cdot \mathcal{C}_j|$ part of the Dice similarity. Notice, however, the Dice similarity can be low, for instance, because there is no evidence on the Web of the context co-occurrence. But it also can be low because some of the context terms were too ambiguous. In the latter case, perhaps if some of the ambiguous terms are filtered out, there actually will be sufficient evidence to merge d_i and d_j. This is what precisely the filtering strategy attempts to do, by filtering out potentially ambiguous context terms upfront.

For each of the 4 possible $N \cdot \mathcal{C}_i \cdot \mathcal{C}_j$ combinations (i.e., P-P, P-O, O-P, O-O), the algorithm generates two features. The first one is the raw $|N \cdot \mathcal{C}_i \cdot \mathcal{C}_j|$ count. The second is its normalized version computed using the $Dice_2$ formula. Therefore, the co-occurrence feature vector will contain 8 features in total.

3.2. Skyline-Based Classification

Now that we created the co-occurrence features, the next question is how to combine these features to compute the overall external similarity of the Webpage pairs. We use these features in a classifier to decide whether they correspond to a merge or do-not-merge decision. Observe that the feature vector is composed of co-occurrence features that satisfy the *dominance property*. That is when a point (i.e., the feature vector)



(a) skyline of the feature space

(b) classification skyline dividing the feature space

Fig. 3. Example of a skyline.

in the feature space is declared as merge decision, then any Webpage pair whose co-occurrence feature vector is dominated by this point should also be merged. Therefore we propose a new specialized classifier that uses the dominance property explicitly.

Let us first make a few auxiliary definitions. We will say that point $\mathbf{f} = (f_1, f_2, \ldots, f_8)$ *up-dominates* point $\mathbf{g} = (g_1, g_2, \ldots, g_8)$, if $f_1 \leq g_1$, $f_2 \leq g_2, \ldots, f_8 \leq g_8$, and will denote it as $\mathbf{f} \leq \mathbf{g}$. Similarly, we will say $\mathbf{f}$ *down-dominates* $\mathbf{g}$, if $f_1 \geq g_1$, $f_2 \geq g_2, \ldots, f_8 \geq g_8$, and will denote it as $\mathbf{f} \geq \mathbf{g}$. Similarly, there is a notion of *strict* dominance where "$\leq$" and "$\geq$" are substituted with "$<$" and "$>$". For instance, for 2-dimensional case, point $(2, 3)$ up-dominates point $(2, 4)$ since $2 \leq 2$ and $3 \leq 4$, but does not up-dominate $(1, 4)$ since $2 \not\leq 1$. Similarly, point $(2, 3)$ down-dominates $(2, 2)$ since $2 \geq 2$ and $3 \geq 2$, but does not down-dominate $(1, 4)$ since $3 \not\geq 4$.

Assume that d_i, d_j pair is characterized by the feature vector $\mathbf{f} = (f_1, f_2, \ldots, f_8)$. Suppose that based on $\mathbf{f}$ the algorithm decides that d_i and d_j should be merged. Assume that there is another pair of Webpages d_k and d_ℓ, which is characterized by vector $\mathbf{g} = (g_1, g_2, \ldots, g_8)$. Then, if $\mathbf{f} \leq \mathbf{g}$, then d_k and d_ℓ should also be merged, since their contextual entities contain even more evidence that the two namesakes are the same person. Thus, there is dominance in feature data in terms of merge decisions.

The proposed approach uses a well-studied notion of Skyline [Borzsonyi et al. 2001; Kossmann et al. 2002] for classifying points as "merge" or "do not merge". A skyline of a set of points, P, is the subset, S, of all points from P such that each point $p \in S$ is not strictly dominated by any other point from P. Skylines are originally proposed to filter out the interesting points from a potentially very large set of points in databases. Figure 3(a) illustrates an example of a down-dominating skyline consisting of points $\{a, b, g, j\}$.

Figure 3(b) plots the up-dominating skyline for all the merge ("+") points, plotted as filled circles. The do-not-merge ("$-$") points are plotted as empty circles in that figure. A *classification skyline* is an up-dominating skyline on all the points the classifier declares to be "merge" points. Given a classification skyline, the classifier will classify any point this skyline up-dominates (i.e., any point that is "on" or "above" the skyline) as a "merge" point, and any other point "–" as a do-not-merge point. Figure 3(b) illustrates one possible classification skyline for the plotted dataset. As any classifier, a skyline-based classifier can make mistakes. The figure shows that this choice of skyline will cause a do-not-merge point *k* to be wrongly classified as a merge point.

3.3. Training Classification Skyline

The clustering algorithm works by merging pairs of Webpages whose co-occurrence feature vectors are above the classification skyline. This section covers a greedy

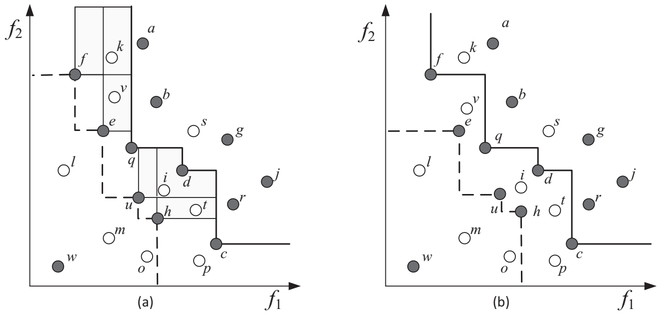


Fig. 4. Skyline learning steps.

```
TRAIN-CLASSIFICATION-SKYLINE($D$)
$S_{pool}$ // Skyline 1: a pool of points to choose from
$S_{clas}$ // Skyline 2: current classification skyline
$S_{best}$ // Skyline 3: best classification skyline observed
$S_{clas} \leftarrow \{(\infty,\infty,\infty,\infty,\infty,\infty,\infty,\infty)\}$
$S_{best} \leftarrow S_{clas}$
$S_{pool} \leftarrow$ COMPUTE-SKYLINE($P^{+}_{actv}$)
while $S_{pool} \neq \emptyset$ do
    $P_{lsterr} \leftarrow$ GET-BESTNEGQUAL-POINTSET($S_{pool}, S_{clas}, P_{actv}$)
    $p_{best} \leftarrow$ GET-BESTQUAL-POINT($P_{lsterr}, S_{clas}, P_{actv}$)
    $S_{clas} \leftarrow$ UPDATE-SKYLINE($S_{clas}, p_{best}$)
    if QUAL($S_{class}$) > QUAL($S_{best}$) then
        $S_{best} \leftarrow S_{class}$
    $P_{actv} \leftarrow$ UPDATE-ACTIVE-POINTSET($P_{actv}, p_{best}$)
    $S_{pool} \leftarrow$ UPDATE-POOL-POINTSET($S_{pool}, p_{best}$)
return $S_{best}$
```

Fig. 5. Algorithm for training.

algorithm for learning such a skyline. The example in Figure 4 illustrates one iteration of the algorithm.

The training algorithm is given a training dataset wherein each point that should be merged is labeled with “+”, and each point that should not is labeled with “−”. In Figure 5, the filled circles are the “+” points, and the empty circles are the “−” points. The algorithm is also given some quality metric, such as the pairwise F measure or B-cubed [Bagga and Baldwin 1998], so that at any point in time it can measure which quality it can get by making certain types of decisions.

The current classification skyline S_{clas} initially consists of one point $(\infty, \infty, \dots, \infty)$, which causes no points to be classified as “+”. At each iteration, the algorithm tries to greedily locate the best point to add to S_{clas}, to make it better. For that it examines all the points from a certain pool of “+” points S_{pool}, whose construction will be explained shortly . In Figure 4(a), the current classification skyline is $S_{clas} = \{q, d, c\}$ and $S_{pool} = \{f, e, u, h\}$.

Among points in S_{pool}, the algorithm first selects the subset of points $P_{lsterr} \subseteq S_{pool}$ such that adding a point $p \in P_{lsterr}$ to S_{clas} would cause the least error; see Figure 6. It does so by examining the quality of the clustering when using S_{clas} and using $p \in S_{pool}$ to classify only the “−” points. For the example in Figure 4(a), notice that among yet-unclassified “−” points (the ones below the classification skyline) f dominates 1 point: $\{k\}$, e dominates 2 points: $\{k, v\}$, u dominates 1 point: $\{i\}$, and h dominates 2 points: $\{i, t\}$.

```
Get-BestNegQual-PointSet($S_{pool}, S_{clas}, P_{actv}$)
$Q_{best} \leftarrow 0$ // best quality observed
$P_{best} \leftarrow \emptyset$ // set of best points
for each $p \in S_{pool}$ do
    $S \leftarrow$ Update-Skyline($S_{clas}, p$)
    $Q \leftarrow$ Qual($S, P^{-}_{actv}$)
    if $Q > Q_{best}$ then
        $P_{best} \leftarrow \{p\}$
        $Q_{best} \leftarrow Q$
    else if $Q = Q_{best}$ then
        $P_{best} \leftarrow P_{best} \cup \{p\}$
return $P_{best}$
```

Fig. 6. Get-BestNegQual-PointSet.

```
Get-BestQual-Point($P_{lsterr}, S_{clas}, P_{actv}$)
$Q_{best} \leftarrow 0$ // best quality observed
$p_{best}$ // best point observed
for each $p \in P_{lsterr}$ do
    $S \leftarrow$ Update-Skyline($S_{clas}, p$)
    $Q \leftarrow$ Qual($S, P_{actv}$)
    if $Q > Q_{best}$ then
        $p_{best} \leftarrow p$
        $Q_{best} \leftarrow Q$
return $p_{best}$
```

Fig. 7. Get-BestQual-Point.

If the goal of our quality measure is to minimize the number of false positives, then P_{lsterr} will be chosen as $P_{lsterr} = \{f, u\}$ since they introduce only 1 new error each.

Then, among the points in P_{lsterr} the algorithm selects a point p_{best} adding which to S_{clas} would lead to the best overall quality; see Figure 7. Point p_{best} is then added to the classification skyline S_{clas}. The algorithm keeps track of the best skyline observed so far by storing and updating it in S_{best}. For our running example in Figure 4(a), given a choice of adding f or u to the skyline S_{clas}, assume that adding f would lead to the best classification quality. Then the algorithm will choose $p_{best} = f$. Figure 4(b) demonstrates what happens after adding f to the skyline.

Adding point p_{best} to the classification skyline S_{clas} can make several (at least one) yet-unclassified points to be classified as "+" by the algorithm, where for the running example that was point $p_{best} = f$ itself. In general, this can happen for two reasons.

—*Direct Merges.* The points dominated by p_{best} will be classified as "+".

—*Transitive Merges.* Classifying p_{best} as "merge" can cause multiple clusters to merge, triggering the merges of the elements of those clusters due to the transitivity. This corresponds to classifying the corresponding points as "+".

The algorithm maintains in P_{actv} the set of yet-unclassified points. We will use notation P^{+}_{actv} and P^{-}_{actv} to denote the "+" and "−" points in this set. For instance, in Figure 4(a), $P_{actv} = \{f, k, v, e, l, u, i, h, t, w, m, o, p\}$ are all the points below the current classification skyline, $P^{+}_{actv} = \{f, e, u, h, w\}$, and $P^{-}_{actv} = \{k, v, i, t, l, m, o, p\}$.

The pool of points S_{pool} from which P_{lsterr} is constructed is the down-dominating skyline on P^{+}_{actv}, for example, in our example $S_{pool} = \{f, e, u, h\}$. The choice of S_{pool} is motivated by several factors. First, for efficiency reasons we want to avoid using large

```
UPDATE-SKYLINE($S_{clas}, p$)
    $S \leftarrow S_{clas}$
    remove from $S$ all points dominated by $p$ // use indexing
    $S \leftarrow S \cup \{p\}$
    return $S$
```

Fig. 8. UPDATE-SKYLINE.

```
UPDATE-ACTIVE-POINTSET($P_{actv}, p$)
    Use indexing to remove all points in $P_{actv}$ dominated by $p$.
    //Remove points in the region with diagonal $p - (1, 1, \dots, 1)$.
    return $P_{actv}$
```

Fig. 9. UPDATE-ACTIVE-POINTSET.

```
UPDATE-POOL-POINTSET($S_{pool}, p$)
    $S_{pool} \leftarrow S_{pool} \backslash \{p\}$
    Use indexing to find points $P$ from $P^{+}_{act}$ (if any) that now
    should be in $S_{pool}$ due to removal of $p$. This procedure
    incrementally maintains the skyline of $P^{+}_{act}$ in $S_{pool}$.
    $S_{pool} \leftarrow S_{pool} \cup P$
    return $S_{pool}$
```

Fig. 10. UPDATE-POOL-POINTSET .

sets, like for instance P_{actv}, and the chosen skyline tends to be much smaller than P_{actv}. For instance, in our example, the size of P_{actv} is 13 whereas the size of that skyline is 4. Second, since it is a set of +'s, this guarantees making at least one correct merge decision. Third, if we look to add a "+" point to S_{clas} that would minimize the number of misclassifications of −'s (false positives), then that skyline will contain all such points. Figures 6, 7, 8, 9, and 10 demonstrate the steps of the learning algorithm in more detail. The feature space is indexed for efficient processing and the skyline of P^{+}_{actv} is updated incrementally as the points are removed from it instead of being rebuilt from scratch.

Discussion. Observe that one of the solutions would be just to use any classifier, such as SVM, to classify the points as +'s and −'s. Unlike that solution, the skyline-based classier utilizes several additional factors to its advantage. First, it explicitly takes into account the dominance that exists in data. Second, it is aware that there is a clustering underneath the +'s and −'s and that classifying a point as a + can cause several other points be classified as + due to transitivity. Third, it greedily fine-tunes itself to a given quality metric; whereas the first approach is more geared toward specifically pairwise F-measure. As we shall see in the experimental section, these qualities allow the skyline-based solution to outperform SVM/DTC-based classifiers. We will now discuss how to make Web statistics collection step scalable.

4. EFFICIENCY OPTIMIZATIONS FOR WEB QUERYING

As discussed in Section 3, if implemented naively the Web-querying approach requires for each pair of Webpages in D to make 8 additional Web queries to collect Web co-occurrence information among contextual entities, resulting in a large number of Web queries which can affect the response time of the WePS system.

In this section we study a new method that considers the time and network limitations in choosing which queries to submit to the search engine. The proposed method aims to optimize the clustering quality by submitting smaller number of queries within

```
DISAMBIGUATE-WITH-SELECTIVE-QUERYING($G, D, \mathcal{T}, \mathcal{B}$)
$T_{start} \leftarrow T_{current}$
$E_{promise} \leftarrow \emptyset$
$E_{processed} \leftarrow \emptyset$
$C \leftarrow$ CLUSTER($G$) //$G$ is labeled with the direct features
repeat
    if $C$ changed and $E_{promise} = \emptyset$ then
        $E_{promise} \leftarrow$ FIND-PROMISING-EDGES($C, G, E_{processed}$)
    if $E_{promise} = \emptyset$ then
        break
    if $C$ changed then
        $E_{qry} \leftarrow$ SELECT-EDGES-TO-QUERY($G, E_{promise}$)
    $E_{select} \leftarrow$ TOP-$\mathcal{B}$-EDGES($E_{qry}$)
    $G \leftarrow$ WEB-QUERY($E_{select}, G$)//updates the graph using web evidence
    $E_{processed} \leftarrow E_{processed} \cup E_{select}$
    $C \leftarrow$ CLUSTER($G$)
until $(T_{current} - T_{start}) \leq \mathcal{T}$
return $C$
```

Fig. 11. Algorithm for efficient web statistics collection.

the given time budget. In addition, it utilizes the option of batch querying, using either multithreading or Yahoo! Query Language (YQL)-like Web search engine query languages.

Formally, let us assume that the algorithm is given a time budget $\mathcal{T}$, a batch size $\mathcal{B}$, and an initial clustering C for graph $G = (V, E)$ where the nodes correspond to the Webpages and the edges to the similarities between these pages. Initially, these similarities are computed by using the direct features only. Let $Q = \{q_1, q_2, \ldots, q_{|E_0|}\}$ be the complete set of queries that will be submitted to the search engine by the default Web querying approach. The objective of the proposed approach is to select n different batches (subsets) from Q (i.e., total of $n\mathcal{B}$ queries) so that the collected Web co-occurrence-based (external) similarity *esim* feature maximizes the expected quality within the given time budget $\mathcal{T}$. Here n is essentially the number of iterations of the algorithm which depends on the search engine's response time and traffic on the network. The problem of finding an optimal set of queries to put in $n\mathcal{B}$ batches is $\mathcal{NP}$-hard; please refer to Appendix A.3 for the proof. Hence in this section we explore an efficient approximate solution to this problem. In the following subsections we first explain a high-level algorithm which iteratively queries the Web in a given time budget in Section 4.1. After that in Section 4.2 we discuss strategies to select the most promising queries to put in the next batch.

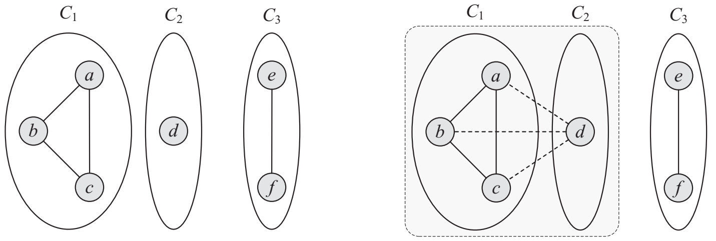
4.1. Disambiguation with Selective Querying

Figure 11 highlights the steps of the proposed clustering algorithm. Its inputs are graph G, dataset D, time budget $\mathcal{T}$, and batch size $\mathcal{B}$. The algorithm starts by clustering the Webpages using their direct feature similarities *dsim*. These *initial clusters* are then exploited in deciding which edges are more important for querying. After that, the algorithm iteratively identifies the set of promising edges $E_{promise}$ that can potentially affect the clustering. At each iteration a subset E_{select} of $E_{promise}$ edges is selected and queried. The query results are then used to update the similarities on G and the resulting clusters.

The algorithm for identifying the $E_{promise}$ set is illustrated in Figure 12. Initially all the edges in the graph are unprocessed. An *unprocessed edge* is an edge that has not

```
Find-Promising-Edges($C, G, E_{processed}$)
    $E_{yes} \leftarrow$ Find-Intra-CL-Edges($C$)
    $G_{temp} \leftarrow$ Create-Temp-Copy($G$)
    $G_{temp} \leftarrow$ Update-All-Unprocessed-Edges($G_{temp}, \alpha_{web}, E_{processed}$)
    // the similarity of the edges in ($E \setminus E_{processed}$) is updated with the weight of
    // the web querying feature $\alpha_{web}$ which is learned from the past data as will be explained in
    // Section 5.
    $C_{temp} \leftarrow$ Cluster($G_{temp}$)
    $E_{no} \leftarrow$ Find-Inter-CL-Edges($C_{temp}$)
    $E_{maybe} \leftarrow (E \setminus E_{processed}) \setminus (E_{yes} \cup E_{no})$
    if $E_{maybe} \neq \emptyset$ then
        return $E_{maybe}$
    else if $E_{no} \neq \emptyset$ then
        return $E_{no}$
    else
        return $E_{yes}$
```

Fig. 12. Algorithm for finding the edges that can change the clustering.



(a) “yes” edges. initial clusters.

(b) “yes” (solid), “maybe” (broken), “no” (absent).

Fig. 13. Example graph to explain the concepts which are used in the efficiency section. (a) The initial clusters and E_{yes} edge set is shown. (b) The pessimistic clusters under the assumption that all queries returned a merge decision. All possible edges that can be in the final clusters are shown.

been queried yet. The algorithm classifies the unprocessed edges into three distinct categories.

(1) *“Yes” edges E_{yes}*. The inter-cluster edges of initial clusters are called yes-edges. Recall that initial clusters are created using direct features only, so these are the edges which are already identified as the “merge” edges by using only direct similarities. The Web evidence gathered by collecting co-occurrence statistics mostly contributes more evidence in favor of the “merge” decision. Thus, such edges are likely to still remain “merge” edges even after co-occurrence queries are issued.[5] Acquiring more evidence for such queries might not be beneficial for the clustering. Figure 13(a) shows an example of three clusters $C_1 = \{a, b, c\}, C_2 = \{d\}, C_3 = \{e, f\}$ as the output of the initial clustering step and the corresponding yes-edges.

(2) *“No” edges E_{no}*. These are the edges which are likely to remain “nonmerge” decisions despite the Web querying. To identify the “no” edges, the approach assumes that Web querying is done and all of the issued Web queries return “merge” decision. After applying clustering on both the direct features and these decisions, some of the edges will be classified as “merge” edges, and some as “nonmerge” edges.

[5] They might change into “nonmerge” edges due to using correlation clustering, but this is rare.

The nonmerge edges are called *no-edges* to reflect the fact that even if all the Web evidence is positive, these edges still remain nonmerge edges.[6] Lines 2–5 in Figure 12 illustrate the procedure of finding E_{no} edges. Figure 13(b) illustrates an example where the assumptions that all Web queries return "merge" decisions leads to merging of initial clusters C_1 and C_2, whereas C_3 still remains a separate cluster. Similar to the E_{yes} edges, the queries corresponding to the edges in E_{no} are also not expected to change clustering significantly. The edges (not shown) among the elements of the resulting clusters $\{a, b, c, d\}$ and $\{e, f\}$ are the no-edges.

(3) *"Maybe" edges E_{maybe}*. Note that the merge decisions for edges in E_{yes} and E_{no} are unlikely to change after Web querying. The most beneficial Web queries to issue are for those *unprocessed edges* that do not belong to E_{yes} and E_{no} sets. These edges are called maybe-edges E_{maybe}: they are the "nonmerge" edges in the initial clustering that can become merge edges through Web querying. In Figure 13(b), edges (d, a), (d, b), and (d, c) are maybe-edges.

Now that the edges are classified into three different categories, the next step is to identify the $E_{promise}$ set. Since the edges in the E_{maybe} set are expected to be the most beneficial they are first inserted into the $E_{promise}$ set. The approach processes all such queries and if time budget permits, edges in the other sets E_{no} and E_{yes} may also be queried. In the next section, we discuss multiple strategies for choosing the best edges from $E_{promise}$ to put in the next batch, such that the clustering quality is optimized under the time constraint.

4.2. Selecting Edges to Query

Given a set of promising edges $E_{promise}$, our next task is to select from it $\mathcal{B}$ queries to submit to the Web search engine. Our goal is to identify edges that will maximize the quality of the clustering with minimum number of queries under the time and network limitations. To do that, the approach needs to select the queries that are more beneficial for the clustering quality. The benefit of Web querying for an edge depends on the benefit of Web querying for the cluster pair that this edge belongs to. The proposed solution computes the expected benefit of each query and submits more useful ones to the Web search engine. We will refer to this solution as `ERWeb-QB`. The expected benefit of each query is calculated in three main steps. It first estimates the expected quality improvement ΔQ_{ij} if the cluster pair C_i, C_j is merged. After that it calculates the number of queries required to merge the cluster pair C_i, C_j. Finally, the benefit of each query is computed using these values. Now, we discuss each of these steps in detail. For each cluster pair C_i, C_j, the proposed solution follows these steps.

(1) *Measuring the expected quality improvement.* Since the ground-truth clustering is unknown to the approach, it computes the expected quality improvement ΔQ_{ij} as follows. It assumes that the ground truth $\mathcal{C}_G$ is equal to the current clustering $\mathcal{C}$ except C_i and C_j which are merged in $\mathcal{C}_G$. ΔQ_{ij} is measured as the error from the expected true clustering

$$\Delta Q_{ij} = 1 - Q_{ij}(\mathcal{C}, \mathcal{C}_G), \tag{2}$$

where $Q(\mathcal{C}, \mathcal{C}_G)$ is the quality of the current clustering C for the expected true clustering $\mathcal{C}_G$. Any quality metric such as F_B and pairwise F_P can be used to compute the quality $Q(\mathcal{C}, \mathcal{C}_G)$. In our experiments we utilize F_B.

(2) *Estimating the number of queries.* The number of queries m_{ij} to merge the cluster pair C_i, C_j depends on the number of merge decisions N_{ij} required to merge these

[6] Due to the use of correlation clustering, these edges could become "merge" edges, but it is rare.

```
SELECT-EDGES-TO-QUERY($G, C, E_{promise}$)
for each cluster pair $C_i$ and $C_j$ do
    $\Delta Q_{ij} \leftarrow$ EXPECTED-QUALITY-IMPROVEMENT($G, C_i, C_j$)
    $N_{ij} \leftarrow$ REQUIRED-NUMBER-OF-MERGE-DECISIONS($G, C_i, C_j$)
    $m_{ij} \leftarrow$ REQUIRED-NUMBER-OF-QUERIES($G, C_i, C_j, N_{ij}$)
    $s_{factor} \leftarrow \min(\frac{\sum_k^{m_{ij}} p_k}{N_{ij}}, 1)$
// $p_k$ is the probability of the $k$-th unprocessed edge between $C_i$ and $C_j$
    $\Delta Q_{ij} \leftarrow \Delta Q_{ij} \cdot \min(1, s_{factor})$
    $b_{ij} \leftarrow \frac{\Delta Q_{ij}}{m_{ij}}$ //benefit for edges between $C_i$ and $C_j$.
$E_{qry} \leftarrow$ sort edges in $E_{promise}$ on $b_{ij}$
return $E_{qry}$
```

Fig. 14. Algorithm for selecting edges to query.

two clusters and the probability of the queries to return “merge decisions”. Therefore the approach needs to find out how many “merge decisions” N_{ij} are needed to group C_i, C_j together. It should also be able to estimate the probabilities for each query to return “merge decisions”. After identifying the number “merge decisions” and the probabilities, the proposed solution can estimate the number of queries m_{ij} to send using both N_{ij} and the probabilities. Next we explain how these values are computed.

—*Estimating the number of “merge decisions”.* The proposed solution uses a binary search algorithm to find the minimum number of “merge decisions” N_{ij} between C_i and C_j that will cause the current clustering to change, for example, to merge C_i and C_j. It utilizes the intuition that the edges that are more similar based on their direct feature similarity are more likely to return a “merge decision” through Web querying. First, the unprocessed intercluster edges of C_i, C_j pair that are in $E_{promise}$ are sorted using their direct feature similarity. On each iteration, the approach creates new clusters assuming that top N of these intercluster edges (or queries) return “merge decisions”. Since the clustering is time consuming, to speed up the computations, the approach works on a subgraph of G which contains only the Webpages in C_i and C_j and their corresponding edges. Then the output of the clustering step is compared to the input clustering. If these two clusterings are found to be different, top N of these edges are identified as beneficial for the C_i and C_j pair. The proposed solution utilizes binary search on N to find N_{ij} that will change the input clustering. The pseudocode for this algorithm is illustrated in Figure 15.

—*Estimating probabilities.* Probability of a query to return a merge decision is estimated using a logistic regression classifier that is trained on the past data. The input to the classifier is the direct similarity of the corresponding edge and the output of the classifier is the probability of that edge returning “merge decision.”

—*Estimating the number of queries.* The number of “merge decisions” N_{ij} that is needed to change the current clustering is a lower bound on the actual number of queries that the approach must submit to the Web search engine to determine if two clusters merge, since some of the queries might return “no-merge” decisions. Further, the approach needs to submit queries for the most probable edges first. Thus to estimate the required number of queries m_{ij}, the approach sorts the queries using their probabilities to return a “merge decision”. Then starting from the largest probability, the approach calculates m_{ij} by adding up the probabilities $\sum_{k=1}^{m_{ij}} p_k < N_{ij}$ of the queries in this sorted list until $\sum_{k=1}^{m_{ij}} p_k < N_{ij}$ is greater than

```
Number-Of-Queries-For-Merge($G, C_i, C_j$)
$N \leftarrow |C_i| \cdot |C_j|$, $min \leftarrow 1$, $max \leftarrow N$
do
    $med \leftarrow \lfloor \frac{min+max}{2} \rfloor$
    $G' \leftarrow$ Update-Top-N-Unprocessed-Edges($G, med$)
    $C \leftarrow$ Cluster($G'$)
    if $C_i$ and $C_j$ grouped together in C then
        $max \leftarrow med$
    else
        if $min = med$ then
            return $max$
        $min \leftarrow med$
while $min \leq max$
return $max$
```

Fig. 15. Algorithm for finding the minimum number of "merge decisions" to group clusters C_i and C_j together.

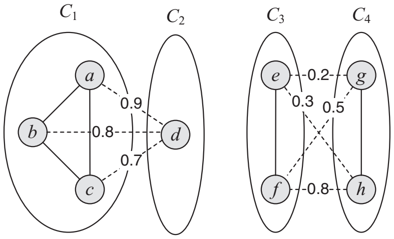


Fig. 16. Example of maybe-edges with associated probabilities.

or equal to N_{ij}. The value of m_{ij} represents the expected number of edges that if probed would result in merging C_i, C_j, since N_{ij} out of m_{ij} queries are expected to return "merge decision."

Example. Let us assume that the approach identifies $N_{12} = 1$ and $N_{34} = 2$ for the cluster pairs C_1, C_2 and C_3, C_4 shown in Figure 16. Suppose that the probability for each edge is as it is shown in the figure. Then the approach computes $m_{12} = 2$, since the sum of the probabilities of the edges (a, d) and (b, d) is greater than 1. Similarly for the C_3, C_4 pair, the approach calculates $m_{34} = 4$.

(3) *Calculating the expected edge benefit.* Finally the expected edge benefit of the cluster pair C_i, C_j is computed by dividing the expected quality ΔQ_{ij} by the required number of queries m_{ij}. When $\sum_{k=1}^{m_{ij}} p_k < N_{ij}$, the cluster pair C_i, C_j is more likely to stay split despite the Web querying. Therefore, the expected benefit of C_i, C_j is scaled when the total probabilities of the edges to be submitted is less than the number of required "merge decisions". Accordingly, we propose to scale the expected benefit of the cluster pair by multiplying it with the ratio of the sum of the total probabilities of the to-be-submitted edges to N_{ij}.

Let us consider the cluster pairs C_1, C_2 and C_3, C_4 illustrated in Figure 17. Suppose that for both cluster pairs the required number of "merge decisions" is 1, that is, $N_{12} = 1$ and $N_{34} = 1$. The approach then computes the number of queries to submit for both cluster pairs as $m_{12} = 2$ and $m_{34} = 2$. Further, the expected quality improvement of both cluster pairs is the same, since for the sake of simplicity we select the cluster pairs to be identical except the probabilities on the edges. Let us assume that it is ΔQ. Then the expected benefit of both cluster pairs is equal,

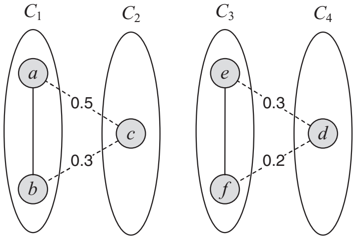


Fig. 17. Motivation figure for scaling.

that is, $\frac{\Delta Q}{2}$. However, when we look at the probabilities of the edges we observe that C_1, C_2 pair is more likely to return "merge decision" than the C_3, C_4 pair. Therefore, we will scale the expected benefit of the cluster pairs. As a result, the expected benefit of C_1, C_2 cluster pair is no longer ΔQ, but $\frac{\Delta Q}{2}(0.5 + 0.3) = 0.4\Delta Q$. Similarly, the expected benefit of C_3, C_4 is $\frac{\Delta Q}{2}(0.3 + 0.2) = 0.25\Delta Q$. The approach therefore submits queries for C_1, C_2 before C_3, C_4. Note that if $\sum_k^{m_{ij}} p_k > N_{ij}$, then the scaling factor is 1; see Figure 14 step 4.

Finally, the cluster pairs are ranked according to their benefits. Starting from the most beneficial cluster pair the approach submits $\mathcal{B}$ queries to the search engine, which includes m_{ij} queries for the cluster pair C_i, C_j.

Next, we discuss how the direct features-based similarities are combined with the Web-based merge decisions.

5. LEARNING TO COMBINE DIRECT AND INDIRECT FEATURES

Intuitively, the more information we have about the people mentioned in the Webpage, the more likely it is that we will be able to correctly distinguish among them. Hence, in addition to using Web features, we explore a rich and diverse set of direct features as well. The specific direct feature similarities that we use include: named entity similarity, word N-gram similarity, URL and email similarities, middle-name dissimilarity, social networking sites dissimilarity. A detailed explanation of these features is provided in Appendix A.2. In this section we discuss how to combine such features for the purpose of clustering.

A simple and intuitive way to combine these features in order to compute the overall similarity $s(d_u, d_v)$ between Webpages d_u and d_v is to use the weighted sum of these features

$$s(d_u, d_v) = \sum_{f_i \in F} w_i \, f_i(d_u, d_v), \tag{3}$$

where F is the feature set and $w_i \in [0, 1]$. The intuition behind the selection of the weighted sum approach is that each feature f_i has some influence on the similarity computations and this influence can be captured by associating the weight w_i with the feature value f_i.

Now, let us discuss an algorithm that learns the weights $\overrightarrow{w} = (w_1, w_2, \ldots, w_{|F|})$ of the features in the feature set that maximize the clustering quality. The problem can be viewed as a multidimensional search problem. Such problems have been studied extensively in the past, including hill climbing, tabu search, gradient descent, and simulated annealing algorithms. We use a modification of the Simulated Annealing (SA) algorithm, since it can find a good approximation to the global optima and is capable of escaping local optima.

```
TRAIN-WITH-SA($C, k_{max}, \tau$)  //$\tau$ is the expected highest quality
    $\overrightarrow{w} \leftarrow$ GET-RANDOM-STARTING-POINT()
    $q \leftarrow$ COMPUTE-QUALITY($C, \overrightarrow{w}$)
    $k \leftarrow 0$, $T \leftarrow 1$, $q_{best} \leftarrow q$, $\overrightarrow{w}_{best} \leftarrow \overrightarrow{w}$
    $W \leftarrow \{\overrightarrow{w}_{best}\}$ //set of all the best points
    while $k \leq k_{max}$ and $q \leq \tau$ do
        $\overrightarrow{w}_{new} \leftarrow$ GET-NEIGHBOR($\overrightarrow{w}$)
        $q_{new} \leftarrow$ COMPUTE-QUALITY($C, \overrightarrow{w}_{new}$)
        if $q_{new} \geq q_{best}$ then
            $\overrightarrow{w}_{best} \leftarrow \overrightarrow{w}_{new}$, $q_{best} \leftarrow q_{new}$
            $W \leftarrow \{\overrightarrow{w}_{best}\}$
        else if $q_{new} = q_{best}$ then
            $W \leftarrow W \cup \{\overrightarrow{w}_{new}\}$
        $r \leftarrow \text{rand}(0, 1)$
        if $(q_{new} - q) > 0$ or $r < c$ then
            $q \leftarrow q_{new}$, $\overrightarrow{w} \leftarrow \overrightarrow{w}_{new}$
        else if $(q_{best} - q) > 0.1$ then //restart
            $\overrightarrow{w} \leftarrow \overrightarrow{w}_{best}$, $T \leftarrow 0.8$
        $k \leftarrow k + 1$
        if $k \mod 100 = 0$ then
            $T \leftarrow T \cdot 0.8$
    return $W$
```

Fig. 18. Simulated annealing-based training to learn feature importance.

We adapt the original SA as a supervised learning algorithm by replacing the cost function with the quality function as illustrated in Figure 18. The goal of the SA-based training algorithm is to find a combination of weights $\overrightarrow{w}_{best} = (w_1, w_2, \ldots, w_{|F|})$ which maximizes the average clustering quality on the training data. The quality of a weight vector $\overrightarrow{w}$ is computed as alpha-beta sum of F_B (the harmonic mean of B-cubed precision and recall) and the absolute difference of precision (Pr) and recall (R)

$$Q(\overrightarrow{w}) = \alpha F_B(\overrightarrow{w}) + \beta |Pr(\overrightarrow{w}) - R(\overrightarrow{w})|, \tag{4}$$

where we set $\alpha = 1$ and $\beta = -0.5$. Formally, the *goal* of the SA-based training is to find the weight vector $\overrightarrow{w}_{best}$ which maximizes the value of Eq. (4).

Different weight vectors $\overrightarrow{w}$ with different relative values of precision and recall could lead to same optimal or near-optimal F_B. Therefore, SA with the quality function in Eq. (4) will find a good weight vector on the training data, which maximizes the F_B value while minimizing the difference between the precision and recall. For instance, the algorithm might consider $F_B = 0.89$ for both $Pr = R = 0.89$ and $Pr = 0.99$, $R = 0.60$. In this case, the algorithm will prefer the first one. This formulation makes the learning algorithm more robust, such that it can get good results on any datasets with different characteristics.

The algorithm follows a simple cooling schedule, where the initial temperature T is set to 1 and cooled by multiplying it by 0.8 after each k iterations. The simulated annealing algorithms are designed in such a way that occasionally they move to a state whose quality is worse than that of the current state so that they will be able to escape local maxima. Thus, the proposed approach follows the strategy of moving to a worse state if the guessed random number is less than the current temperature. We adapted the simulated annealing with restarts, such that the approach restarts if the current quality does not change for N times. At the time of restart the initial weight vector is set to best weight vector found so far and T is initialized to 0.8. To stop the search we use the limit on the number of iterations.

Another important thing to note is that it is possible that more than one feature vector with slightly different feature values can lead to the same quality. In such cases, those feature vectors should all be considered as the best feature vectors. Thus, the SA algorithm will output all such vectors. These vectors are then used to create a new weight vector, where each element of the vector is the average of the corresponding elements in these vectors.

6. CLUSTER REFINEMENT OF SINGLETON CLUSTERS

After correlation clustering is applied, often the result consists of a few large clusters and several singleton 1-Webpage clusters. Naturally, there can be true and false singleton clusters. Typically, the Webpage in a false singleton cluster belongs to one of the larger clusters in the result set. A primary cause of a false singleton cluster is the lack of sufficient content in the Webpage of the cluster that is related to the namesake it mentions. Therefore, the direct features such as named entity and n-grams-based similarities fail to group them with their true clusters. Further, such pages often do not contain any contextual entities (i.e., person and organization names) that are related to the mentioned namesake. Thus, the Web co-occurrence queries are not able to capture the relationships between the Webpages of the same namesake as well. Hence, it is hard to find enough evidence to create a positive connection between such pages and the other Webpages about the same namesake using the direct features as well as the Web-based cooccurrence statistics. In this section, we discuss different strategies to deal with such pages. We first explain the basic strategies that we can follow to recompute the similarities between the singleton Webpages and the remaining Webpages in Section 6.1. These new similarity values are used to merge the singleton cluster with another cluster, if the similarity exceeds a certain threshold. The threshold used in the merging step is learned from past data by applying an interval-based regression model as discussed in Section 6.2.

6.1. Basic Approaches

In this section, we discuss two basic strategies that one can use to revise the clusters. The first strategy is to compute the similarity of the Webpage in the singleton cluster to the remaining clusters by using the tokens extracted from the Webpages. Similarities can be calculated using TF.IDF-based cosine similarity. The Webpage in the singleton cluster is merged with another cluster only if the similarity of the Webpage to any Webpage in that other cluster exceeds a predefined threshold.

However, keyword-level similarities might not be enough to resolve such false singleton cluster problems. In such situations, we can use approaches that are more complex. One such solution is to use the Web to gather additional information about the Webpage in each singleton cluster. This can be done by creating a special type of query by following the strategy in Chen et al. [2009a] and Rao et al. [2007]. The suggested solution iterates over all the 1-page clusters in the resulting clustering of the previous clustering step. First, a Web query is formulated for each such Webpage by combining the queried name with the surrounding bigrams. For instance, if the queried name is “Michael Jordan” and the terms neighboring the name are “legendary career”, the query will be Q = `legendary career ‘‘Michael Jordan’’`. Then, these queries are issued to the search engine and the page profile is generated using the collected snippets and titles. These profiles are then used in similarity computation. Similarities are calculated using TF.IDF-based cosine similarity. The profiles are compared to each document in each cluster to find out the maximally similar Webpage. If the maximum

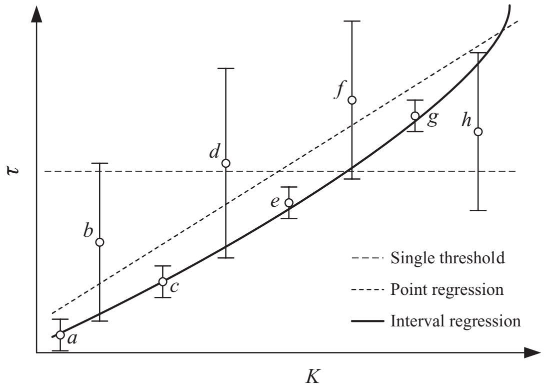


Fig. 19. Motivating example for interval regression.

similarity value of the page is above a threshold, then that singleton cluster is merged with the maximally similar cluster.

We have experimented with both of these strategies. Our experiments reveal that the Web querying approach is only marginally better than the approach that uses token-based similarities. Therefore, we suggest using the non-Web querying approach, since it is faster. Both of the preceding strategies rely heavily on the accuracy of the selected threshold. In the next section, we explain a strategy that uses an interval-based regression model to compute such threshold per person name.

6.2. Learning the Threshold

The quality of the cluster refinement step depends on being able to choose the right threshold τ for dataset D being processed. The threshold determines when a given singleton cluster should be merged with a different cluster. Different strategies to determine such thresholds are possible. One such strategy is to set the threshold to a single value that is learned to be optimal for all possible people names. It turns out that in practice choosing such a single fixed threshold leads to little improvement over the initial correlation clustering results. For instance, let us assume that there are eight different names in our dataset. Each person name has a different threshold range that can merge the singleton clusters with their true clusters, as illustrated in Figure 19. A single threshold value might not be able to capture all the intervals, therefore it might lead to suboptimal clusters. Figure 19 shows that the single threshold misses 4 out of 8 intervals. Thus, a more advanced algorithm that chooses a different threshold for each queried person name might work better.

To address this problem we will use the fact that in a merging strategy knowing the number of clusters K in D can help in choosing a better value of threshold τ. Specifically, if the number of clusters K is large then choosing larger value of the threshold will tend to increase the number of resulting clusters, which often leads to better clustering results. Similarly, if K is small, lower values of the threshold often produce better results.

However, one of the challenges in using this observation is that the number of clusters K is not known beforehand to the algorithm. We can handle this issue by observing that the value of K could be estimated from the number of clusters $|R|$ that are outputted as the result of the initial correlation clustering step. That is, in the cluster refinement phase, the number of clusters generated by the initial clustering step is utilized to learn the merging threshold. Observe that this is done per person name (per dataset D).

Approach 1. Let K_i by the true number of clusters for i-th person name in the training data and let τ_i be the optimal threshold for that person for merging singleton clusters. One approach to learn the threshold would be to try to use these τ_i and K_i values for all values of i to train a model $\tau = f(K)$ on past data, for example, using a regression method. For instance, for linear regression the task might be formulated as

$$\begin{aligned} &\text{Minimize } \textstyle\sum_{i=1} \varepsilon_i, \\ &\text{subject to } \tau_i - \varepsilon_i \leq \beta_0 + \beta_1 K_i + \beta_2 K_i^2 + \cdots + \beta_n K_i^n \leq \tau_i + \varepsilon_i. \end{aligned} \tag{5}$$

Here, $f(K)$ is chosen as a function $f(K) = \beta_0 + \beta_1 K_i + \beta_2 K_i^2 + \cdots + \beta_n K_i^n$. The task is to learn weights $\beta_0, \beta_1, \ldots, \beta_n$ such that $f(K)$ approximates as closely as possible values τ_i given K_i. Since the system is unlikely to approximate τ_i's exactly, the slack variables ε_i's are used in the constraints and the goal of the optimization problem is to minimize the overall slack.

Approach 2. A better approach for choosing the threshold exists that reaches even higher disambiguation quality. It leverages the observation that for each case i there is typically a *range* of values $[\tau_i^\ell, \tau_i^h]$ that leads to optimal or near-optimal results, instead of a single optimal threshold value τ_i. This range of values needs to be taken into account during the training phase. If point regression is used it might learn a curve as shown in Figure 19, which could get only 3 out of 8 intervals. However, if the intervals are taken into consideration with the interval regression, it might find a curve that can capture all the intervals. Consequently, we suggest employing an interval fitting linear regression model.

$$\begin{aligned} &\text{Minimize } \textstyle\sum_{i=1} \varepsilon_i, \\ &\text{subject to } \tau_i^\ell - \varepsilon_i \leq \beta_0 + \beta_1 K_i + \beta_2 K_i^2 + \cdots + \beta_n K_i^n \leq \tau_i^h + \varepsilon_i. \end{aligned} \tag{6}$$

The previous optimization problem is a linear programming problem, and linear programming is known to have efficient solutions. The goal is to learn a combination of weights $\beta_0, \beta_1, \ldots, \beta_n$ such that for each i the predicted value of $\tau = f(K_i)$ fits the best the interval $[\tau_i^\ell, \tau_i^h]$. Since not all of the predicted values might strictly fall inside $[\tau_i^\ell, \tau_i^h]$, nonnegative slack of ε_i is added to the corresponding constraints.

With the interval-based regression model, we learn the β_i values from past data which are then used for the threshold computations $\tau = f(k)$ on the test data.

7. POSSIBLE ARCHITECTURES AND THEIR EFFICIENCY

There are several possible ways of implementing a WePS engine, such as client-side, third-party proxy, or server-side approaches. The solution proposed in this article can work with any of these architectures.

In a third-party proxy approach, the user query is issued first to a proxy, which in turn queries a Web search engine and then clusters the returned results. A client-side solution is similar, except for the software installed on the client acts as the proxy. The advantage of proxy (and client-side) solutions is that such a strategy can be built independently of the primary search engine without requiring existing engines to be modified. However, such flexibility comes at a (potentially high) cost of having to query search engines to collect the search engine statistics over the Internet during the query execution.

In a server-side approach the WePS component sits on top of the search engine and has direct access to its indices as well as to the most recent Web snapshot of the

Table I. Comparing Efficiency Costs of Proxy-Based and Server-Side Approaches

Step	Proxy-based	Server-side
(1) User input	Runtime	
(2) Web page retrieval	Through Internet	Direct, without Internet
(3) Preprocessing	Runtime	Beforehand
(4) Similarity Comp.		
(a) Direct Sim. Comp.	Runtime	
(b) Indir. Sim. Comp.	Applying skyline: runtime. Training skyline: beforehand	
− Web co-occurrence	Through Internet	Direct, without Internet
(c) Combining Sim.	Applying sim. anneal.: runtime. Training it: beforehand	
(5) Clustering		
(a) Correlation Clust.	Runtime	
(b) Cluster Refinement	Runtime. Training interval regression: beforehand	
(6) Postprocessing	Runtime	
(7) Visualization	Runtime	

entire Web, thus avoiding the need to connect to a search engine via the Internet. The component does the clustering and returns the result to the user. The advantage of such an approach is that it is likely to be more efficient, as many Webpage preprocessing steps can be done at the server, before any user query is issued. In addition, Web querying is done internally instead of querying over the Internet. The disadvantage is that only a Web search company could realize a server-side solution, or alternatively a third party should maintain a snapshot of the entire Web, for example, by using technology similar to that of WebBase developed at Stanford [Cho et al. 2006].

The differences between the proxy-based and server-side solutions with respect to their efficiency are summarized in Table I. It lists the processing steps of the WePS system discussed in Section 2 and the corresponding efficiency issues, such as whether a step is performed during query runtime, or whether it can be done beforehand, that is, before any query processing starts.

First, it should be noted that in both of these approaches, all of the tuning and *training* of the approach, including training: (1) skyline classifier, (2) regression model, and (3) simulated annealing model is done offline/beforehand, before any query processing. Therefore, while the training part can be slow, it does not affect the query response time. Unlike training, *applying* these three mechanisms is naturally very fast. This is since the found classification skyline typically consists of less than 100 points, so even scanning these points without using any auxiliary indexes to classify feature vectors is very efficient. For the simulated annealing algorithm, once it learns the weights, computing the overall similarity $s(d_u, d_v)$ by using these weights in Eq. (3) is also fast. Similarly, to compute the threshold for the cluster refinement, the algorithm simply needs to input the number of clusters K (which it gets after correlation clustering) into function $\tau = f(K) = \sum_{i=0}^{n} \beta_i K^n$, where n is small, for example, 2, and this computation is done only once per user query.

Second, unlike the proxy-based approach, the server-side solution avoids sending certain information over the Internet, and thus can be made much faster. For instance, it does not need to retrieve the top-K Webpages or collect Web co-occurrence statistics over the Internet; instead it can process them locally.

Third, the server-side approach can perform the entire preprocessing step beforehand. This includes various extractions of direct features such as named entity extraction and building various indexes such as for computing TF. IDF, giving it yet another advantage in terms of the efficiency over the proxy-based approach.

While the server-side solution has many advantages, a proxy-based solution is also a viable option. Our current proxy-based implementation has two interfaces: default and advanced. The default interface is faster but less accurate than the advanced one. It runs the approach using the direct features only, without the Web features, to produce the initial clusters. If the user, however, is not satisfied with the quality of the initial clusters, (s)he can use the advanced interface to run the full algorithm, which is slower but more accurate. As demonstrated empirically in Section 8.2.1, the difference in clustering quality is especially noticeable for the cases where the quality of the initial clustering is low. This essentially creates the quality-versus-efficiency trade-off, where the user runs more expensive queries only for lower-quality cases, for which the gain due to (slower) Web co-occurrence queries is significant. Our implementation employs caching of results. Hence, if the results for a user query are already in the cache, they are simply displayed from the cache without any query recomputation. Caching also allows incremental processing of the "advanced" query after a "default" query, as the top-K Webpages are already retrieved and cached after the default query and all the direct feature information is also precomputed and cached.

We also envision that in addition to having an advanced interface, a proxy-based solution can have a *background-processing mode* where the user will be initially presented with the clusters generated by the default interface, but the system will continue to refine and update clusters in the background (using Web queries) while the user explores the initial clusters. That way the cost of Web querying will be partially hidden from the user.

8. EXPERIMENTAL DESIGN AND RESULTS

This section empirically studies the proposed approach in terms of both quality and efficiency and compares it to the state-of-the-art techniques on the `WePS-2` dataset.

8.1. Experimental Design

8.1.1. Datasets. Since the proposed algorithm is a supervised learning approach, we use training datasets to learn the parameters for the classifiers and test the effectiveness of the classifiers with those parameters on the test dataset.

Training datasets. We merge two different datasets to create the training dataset for the experiments. The first one, `WWW05`, which is used in Bekkerman and McCallum [2005] contains 12 different people each with 100 pages. The second one is the trial and training datasets published for the WePS-1 task [Artiles et al. 2007]. The trial dataset contains 9 person names with 100 pages, while the training dataset contains 49 different names. The number of pages for the training dataset, however, varies from 2 pages to 400 pages.

Test datasets. We evaluate our approach and compare it to the state-of-the-art techniques on the `WePS-2` test dataset which is the current de facto standard for testing WePS solutions [Artiles et al. 2009]. `WePS-2` contains Webpages for 30 person names where there are 150 Webpages for each person used in the disambiguation.

8.1.2. Tools. Stanford's Named Entity Recognizer [Stanford NER 2012] is utilized to extract named entities from Webpages. The co-occurrence statistics are collected using Yahoo! and BOSS! APIs. We extracted text and hyperlinks from the Webpages using Java's HTMLParser. Emails and hyperlinks in the text are extracted using a simple rule-based extractor.

To test significance of the improvements achieved by the proposed approach we use a standard statistical significance test, namely paired two-tailed t-test. We use the t-test to measure if the means of two different evaluations are different. If the computed p-value is below some threshold (typically 0.10, 0.05, or 0.01) the difference is declared to be statistically significant for that threshold value.

8.1.3. Versions of the Proposed Method. We compare different versions of the proposed method with other techniques. These versions are as follows.

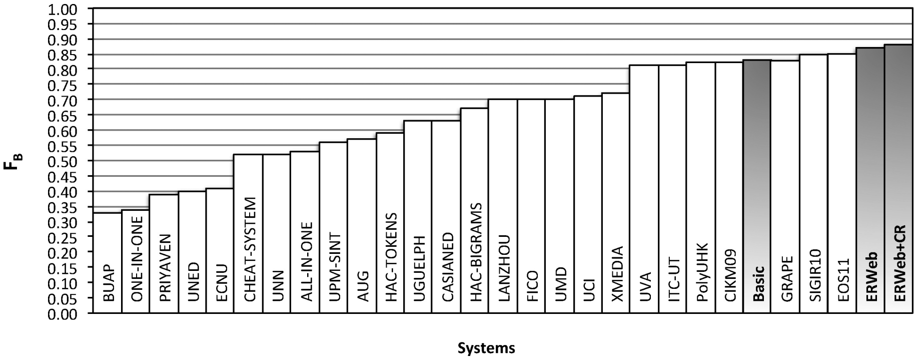


Fig. 20. Comparing different systems. Different versions of the proposed approach are shown as shaded bars.

(1) Basic version which only utilizes the direct features.
(2) ERWeb version which uses all the direct features of Basic and the Web querying (Web), so it is Basic+Web.
(3) ERWeb+CR version is the ERWeb which in addition applies the Cluster Refinement(CR) to improve on the singleton clusters.

8.1.4. Quality Measures. We use F_B measure which is currently known to be the best metric to use for the WePS problem [Artiles et al. 2009]. F_B is the harmonic mean of B-cubed Precision and B-cubed Recall [Bagga and Baldwin 1998]. B-cubed measures are based on computing the precision and recall of each item in the collection. The item precision measures how many items in the same cluster belong to the category of that item. Symmetrically, item recall measures how many items from its category are in the same cluster with this item. We use the WePS-2 version of the B-Cubed F-measure as explained in Artiles et al. [2009].

8.2. Experimental Results

8.2.1. Effectiveness of the Overall Approach. We first compare the quality of the proposed approach with the other systems. The best system participated in WePS-2 competition was PolyUHK (82%). EOS11 [Liu et al. 2011], CIKM09, GRAPE, and SIGIR10 are the techniques that have been published after the competitions and that used this benchmark dataset to report the effectiveness of their methods. As Figure 20 shows, the ERWeb+CR (88%) and ERWeb (87%) approaches outperform all of the previous systems. In addition the Basic (83%) approach also gets better-quality results than the systems participating in the WePS-2 competition. The improvement of ERWeb+CR over ERWeb and Basic is statistically significant at p-values of 0.09 and 0.0007, respectively.[7] The statistical significance over the other methods cannot be measured using the t-test, since for most of these methods only average F_B has been reported, whereas the t-test requires knowing F_B values per each of the 30 person names in the dataset.

Table II presents the detailed F_B values for ERWeb without the cluster refinement. Table III is similar, but shows the result on another dataset that has been used in GRAPE'09. The results are sorted in ascending order on F_B values of Basic. The table also demonstrates the improvement over Basic achieved due to using the Web co-occurrence features. Notice that the largest improvements, shown in bold, tend to occur for the cases where Basic does not reach very good quality. Similarly, the smallest improvements are found for the cases where the quality of Basic is already high. Based

[7] These p-values mean with 91% and 99.93% chance ERWeb+CR is better than ERWeb and Basic.

Table II. Detailed F_B Values for ERWeb Without Cluster Refinement, Sorted on Basic Values

Person	ERWeb (no CR)	Improvement	Basic	Skyline	DTC
Franz Masereel	0.64	**0.18**	0.46	0.05	0.47
Bertram Brooker	0.81	**0.19**	0.62	0.21	0.75
Nicholas Maw	0.84	**0.21**	0.63	0.69	0.88
Amanda Lentz	0.7	0.01	0.69	0.40	0.61
Judith Schwartz	0.8	**0.11**	0.69	0.51	0.41
Herb Ritts	0.84	**0.11**	0.73	0.55	0.88
Jonathan Shaw	0.79	0.02	0.77	0.55	0.40
Hao Zhang	0.79	0.01	0.78	0.55	0.45
David Tua	0.92	**0.12**	0.8	0.52	0.98
Louis Lowe	0.85	0.05	0.8	0.48	0.43
David Weir	0.8	−0.01	0.81	0.62	0.49
James Patterson	0.92	0.09	0.83	0.86	0.91
Sharon Cummings	0.86	0.03	0.83	0.53	0.63
Tamer Elsayed	0.87	0.04	0.83	0.30	0.54
Jason Hart	0.88	0.04	0.84	0.80	0.74
Tom Linton	0.89	0.04	0.85	0.73	0.79
Benjamin Snyder	0.94	0.07	0.87	0.71	0.81
Emily Bender	0.92	0.05	0.87	0.47	0.50
Hui Fang	0.86	−0.02	0.88	0.63	0.66
Rita Sher	0.9	0.02	0.88	0.46	0.66
Gideon Mann	0.96	0.06	0.9	0.31	0.70
Mike Robertson	0.88	−0.02	0.9	0.73	0.42
Ivan Titov	0.94	0.02	0.92	0.37	0.71
Otis Lee	0.95	0.03	0.92	0.63	0.54
Susan Jones	0.88	−0.04	0.92	0.81	0.40
Cheng Niu	0.92	−0.02	0.94	0.46	0.86
Janelle Lee	0.95	0.01	0.94	0.76	0.92
Mirella Lapata	0.99	0.05	0.94	0.74	0.98
Theodore Smith	0.95	−0.01	0.96	0.89	0.82
Helen Thomas	0.99	0.02	0.97	0.92	0.97
Average	0.874	0.048	0.826	0.57	0.68

Table III. Results on the GRAPE'09 Dataset, Sorted on Basic Values

Person	ERWeb (no CR)	Improvement	Basic	GRAPE'09
Lynn Voss	0.75	**0.20**	0.55	0.60
Steve Hardt	0.64	0.05	0.59	0.45
Samuel Baker	0.89	**0.22**	0.67	0.70
Fernando Pereira	0.71	0.00	0.71	0.79
David Mulford	0.80	**0.08**	0.72	0.73
William Cohen	0.85	**0.07**	0.78	0.85
Andrew Ng	0.86	**0.07**	0.79	0.84
Bill Mark	0.83	0.04	0.79	0.74
Mary Johnson	0.86	**0.07**	0.79	0.81
Lisa Harris	0.83	0.03	0.80	0.75
Nancy Thompson	0.86	0.03	0.83	0.96
Sarah Wilson	0.78	−0.05	0.83	0.77
Tom Mitchell	0.82	−0.02	0.84	0.86
Ann Hill	0.85	0.00	0.85	0.90
Andrew Mccallum	0.94	**0.08**	0.86	0.87
David Israel	0.81	−0.05	0.86	0.68
Adam Cheyer	0.86	−0.05	0.91	0.81
Helen Miller	0.90	−0.03	0.93	0.92
Christine King	0.89	−0.05	0.94	0.91
Brenda Clark	0.95	−0.01	0.96	0.94
Leslie Kaelbling	0.98	0.00	0.98	0.95
Average	0.84	0.03	0.81	0.80

Table IV. Web Feature Classification Using Different Classifiers

Approach	F_B measure
ERWeb with DTC	0.85
ERWeb with SVM	0.85
ERWeb with Skyline	0.87

Table V. Effect of Different Direct Features and Web Co-Occurrence Statistics

Model	Direct Features Only(F_B)	+Web (F_B)
NE	0.76	0.82 (+6%)
NE+NG	0.82	0.86 (+4%)
NE+NG+MI	0.82	0.86 (+4%)
NE+NG+MI+HY	0.83	0.87 (+4%)
NE+NG+MI+HY+EM	0.83	0.87 (+4%)

on these observations, our prototype implementation has two interfaces: default and advanced. The default interface employs a Basic algorithm, which does not use Web queries and thus is more efficient. If the user, however, is not satisfied with the clusters returned by the default interface (which is likely to happen when the quality results of Basic are low), the user has the option of using the advanced interface, which will issue (slower) Web queries to improve the results' quality (and if the results of Basic are low, the user might see a major quality improvement).

Table III also compares ERWeb to GRAPE'09. The improvement of ERWeb over GRAPE'09 is statistically significant for p-value of 0.05.

8.2.2. ERWeb vs. Using a Classifier. The last two columns of Table II study what will happen if a classifier is employed to analyze all the features that are used by ERWeb, instead of using the proposed ERWeb itself for that analysis. The table plots results for the Skyline classifier and Decision Tree Classifier (DTC).[8] The table shows that both of these classifiers are unable to reach reasonable quality on their own, and a better approach such as ERWeb should be used instead.

8.2.3. Skyline-Based Classifier vs. Standard Classifiers in ERWeb. In this experiment, we compare the effectiveness of ERWeb with the proposed skyline-based classifier to ERWeb with two different widely used classifiers: Support Vector Machines (SVM) and Decision Tree Classifier (DTC). Both of these classifiers are used to measure the similarity of a given Webpage pair and their Web co-occurrence vectors. The decisions obtained with the classifiers are then incorporated into the simulated-annealing-based classifier as Web-based evidence.

As shown in Table IV, the proposed skyline-based classifier outperforms both of the classifiers, as it is a specialized classifier that takes into account the dominance in data and tunes itself to the given quality metric, F_B in this case. The improvement over DTC and SVM is statistically significant at p-value of 0.08. It should be noted that in the initial version of this work [Kalashnikov et al. 2008b] the improvement was 8% over SVM and 7% over DTC, but the overall quality was lower due to not using many features.

8.2.4. Effectiveness of Features. Table V studies the effect of various direct features on the quality of clustering. The table studies two cases: (1) direct features only and (2) the same direct features plus indirect (Web co-occurrence) feature. The direct features are

[8] Experiments with some other classifiers, such as SVM, have been conducted as well and results were similar to those of DTC.

Table VI. Acronyms for Features

Notation	Meaning
NE	*Named Entities based similarity*
NG	*N-gram based similarity*
MI	*Middle Name Initial based dis-similarity*
HY	*Hyperlinks based similarity*
EM	*Emails based similarity*

Table VII. Effect of Cluster Refinement on the Algorithm With and Without Web Features

Model	No Cluster Refinement (F_B)	With Cluster Refinement (F_B)
Basic	0.83	0.86 (+3%)
ERWeb	0.87	0.88 (+1%)

Table VIII. Comparing Results With and Without Cluster Refinement

Query	ERWeb ($P/R/F_B$)	ERWeb+CR ($P/R/F_B$)	False Singletons (Before/After)
Amanda Lentz	0.77/0.65/0.70	0.77/0.65/0.70	6/6
Bertram Brooker	1.00/0.69/0.81	**1.00/0.83/0.90**	11/2
David Tua	1.00/0.84/0.92	**1.00/0.98/0.99**	7/0
Franz Masereel	0.98/0.47/0.64	**0.98/0.55/0.71**	16/7
Herb Ritts	1.00/0.73/0.84	**1.00/0.78/0.88**	7/4
Nicholas Maw	1.00/0.81/0.90	**0.96/0.88/0.92**	8/1
James Patterson	0.96/0.88/0.92	**0.94/0.95/0.94**	5/0
Theodore Smith	0.95/0.95/0.95	0.95/0.95/0.95	3/3
Tom Linton	0.97/0.83/0.89	0.91/0.87/0.89	6/2
Overall	**0.93/0.83/0.87**	**0.92/0.85/0.88**	170/121

Significant improvement is shown for the names with many wrong singleton clusters.

referred by their aliases presented in Table VI. As expected, higher quality is reached as more direct features are used.

A simulated annealing-based classifier is trained to learn the best weights for each combination of the features on the training data as explained in Section 5. As expected, Table V illustrates that the Web querying always improves the quality over the direct features. As more direct features are utilized the quality of the clustering also increases or stays the same and the improvement with the Web querying is still significant. However, the effectiveness of the Web querying decreases from 6% to 4% as more direct features are utilized.

8.2.5. Cluster Refinement Experiments. In this section, we study cluster refinement approach in detail.

Table VII illustrates the overall quality of the proposed approach with and without the cluster refinement phase. The improvement achieved by using the cluster refinement is more visible over the Basic approach (3%) compared to ERWeb that utilizes Web co-occurrence statistics (1%).

Table VIII illustrates the detailed quality results and the number of false singleton clusters before and after the cluster refinement approach is applied for a set of person names. On average (for 30 people) the recall value increased by 2%, while the precision decreased only by 1%, leading to 1% improvement (statistically significant for $p = 0.05$). “Bertram Brooker”, “David Tua”, “Franz Masereel”, “Herb Ritts”, and “Nicholas Maw” datasets contain mainly one person, therefore there were many wrong singleton-1-page clusters in the output of the ERWeb and the improvement for these cases is very large (i.e., 4.8% on average). On the other hand, for the remaining cases, the approach does not change the overall clustering much since singleton-1-page clusters are rare for them.

Table IX. Single Query End-to-End Running Time (in seconds) in a Proxy-Based Approach

Name	Downl.	NE Extr.	Sim.Cmp.	WebQry	Clust.	Total
Amanda Lentz	4.82	3.89	9.91	12.72	2.23	33.57
Benjamin Snyder	5.65	7.35	11.78	14.47	2.32	41.57
Bertram Brooker	7.87	4.09	10.91	12.01	2.15	37.02
Cheng Niu	8.25	8.29	15.60	29.59	1.59	63.33
David Tua	6.55	8.48	11.34	55.85	1.61	83.83
David Weir	8.76	1.33	7.43	33.60	2.61	53.74
Emily Bender	8.06	0.97	2.21	28.40	2.12	41.76
Franz Masereel	6.74	2.22	5.33	45.03	2.49	61.82
Gideon Mann	6.44	1.70	6.49	13.86	2.30	30.79
Hao Zhang	5.38	6.62	6.53	27.24	2.45	48.21
Helen Thomas	11.99	3.24	5.46	54.19	2.52	77.42
Herb Ritts	7.59	7.37	9.84	38.90	2.56	66.26
Hui Fang	5.58	3.31	5.40	31.21	2.14	47.64
Ivan Titov	5.40	0.46	4.97	25.21	2.34	38.37
James Patterson	6.60	4.02	8.67	28.43	3.49	51.21
Janelle Lee	5.98	5.38	6.49	28.50	2.35	48.70
Jason Hart	8.31	7.09	5.17	31.08	1.31	52.97
Jonathan Shaw	6.12	6.18	6.16	46.89	2.48	67.83
Judith Schwartz	6.29	7.77	8.56	64.22	1.30	88.15
Louis Lowe	5.64	3.36	9.32	55.19	1.80	75.32
Mike Robertson	7.17	8.08	8.86	25.90	1.89	51.89
Mirella Lapata	5.32	1.47	10.80	40.11	2.12	59.83
Nicolas Maw	6.89	3.61	9.96	59.76	2.86	83.07
Otis Lee	5.59	8.01	7.61	27.95	2.49	51.66
Rita Fisher	7.09	1.52	8.32	23.20	1.25	41.39
Sharon Cummings	5.99	2.27	6.73	26.03	1.50	42.51
Susan Jones	11.24	0.75	6.59	24.80	2.98	46.35
Tamer Elsayed	5.91	2.12	6.45	21.94	2.49	38.89
Theodore Smith	5.01	3.81	9.62	29.42	3.09	50.96
Tom Linton	5.44	2.32	8.03	58.23	2.80	76.82
Median	6.37	3.71	7.82	28.96	2.33	51.43
Average	6.78	4.22	8.01	33.64	2.26	54.98

8.2.6. End-to-End Query Running Time. As discussed in Section 7, we have developed a research prototype of a fully functioning WePS system that implements the proposed solution using the proxy-based architecture. It provides a Web interface for the user to enter a WePS query, performs disambiguation, and outputs the results back to the user. In this experiment we test our system by taking the 30 names from the WePS dataset and use our system to disambiguate among the top-100 Webpages returned by Yahoo! for these names. Table IX reports the overall end-to-end time (in seconds) it takes to process each query, as well as the median and average times. For instance, it shows that it takes 4.82 sec to query Yahoo! with name “Amanda Lentz” and download the top-100 returned Webpages. Then it takes 3.89 sec to extract named entities off these Webpages, 9.91 sec to compute the direct similarities, 12.72 sec to collect Web co-occurrence statistics, and 2.23 sec for clustering including cluster refinement, or 33.57 sec in total.

It should be noted that, in our system, the downloading and NE extraction phases are pipelined. Webpages are downloaded in parallel, some faster some slower, depending on the Web server where they are located. As soon as a Webpage is downloaded, it is put into the NE extraction queue to be processed. The download time in the table shows the time to download the last page. The NE extraction time is the time to download all the pages and extract entities from them, minus the time to download the last page. We use a third-party extractor SNER, developed by Stanford. This tool is implemented in Java (and not, say, C++), meaning potentially even the same NE extractor can be made faster in practice by using C++.

Table X. Estimated Single Query End-to-End Running Time (in seconds) in a Server-Side Approach

Name	Downl.	NE Extr.	Sim.Cmp.	WebQry	Clust.	Total
Amanda Lentz	0.20	0.00	9.91	3.18	2.23	15.52
Benjamin Snyder	0.20	0.00	11.78	3.62	2.32	17.93
Bertram Brooker	0.20	0.00	10.91	3.00	2.15	16.26
⋮	⋮	⋮	⋮	⋮	⋮	⋮
Tom Linton	0.20	0.00	8.03	14.56	2.80	25.59
Median	0.20	0.00	7.82	7.24	2.33	17.68
Average	0.20	0.00	8.01	8.41	2.26	18.88

We can see that the average similarity computation time is 8 sec. Should that be required, we have a few reasons to believe this result can be easily improved in a real nonprototype WePS system. First, at present WePS researchers have focused on achieving high quality of their techniques, treating efficiency as a secondary issue. In our case, more efficient versions of some of the algorithms can be designed as well. Second, we have used Java, which is known not very efficient, to implement our prototype system since many third-party tools we employ use the Java API, including Google/Yahoo! Search API, Yahoo! BOSS API, SNER. Trivially recoding the same algorithms in C++, while making them interface with the Java APIs, should improve the efficiency. Third, that part of the code is currently running as a single thread and can be improved by developing parallel versions of the algorithm.

The average clustering time is 2.3 sec. It can also be improved by the aforementioned techniques. Most of these 2.3 sec is due to the cluster refinement phase and the time to perform correlation clustering is negligible compared to 2.3 sec.

The Web co-occurrence part of the algorithm takes 33.64 sec on average. This is the time needed for the third-party Web service (Yahoo! BOSS) to process these queries. This part will be difficult to optimize further in a proxy-based solution, unless more aggressive query pruning techniques are implemented (or unless the proxy-based one maintains a WebBase-like Internet index [Cho et al. 2006]), which is beyond the scope of this article.

The Web querying part, however, can obviously be faster if a server-side approach is used. Table X tries to conservatively estimate what will happen to the execution time in a server-side solution. It makes the assumption that the time needed by the Web co-occurrence part can be made to be 25% of what is reported in Table IX. In a server-side approach, there will not be a need to download the top-K Webpages, but there will be a need to locate them. Hence, we set the download time to 0.2 sec based on the fact that currently it takes Google $\simeq$ 0.2 sec on average to return results to these queries. The similarity computation part will be faster than is reported in this table, as extraction of direct features can be decoupled from similarity computations and be made offline/beforehand as well.

8.2.7. Effectiveness of the Web-Querying Optimization. In this section, we study the quality of the proposed efficiency optimization. We compare the ERWeb-QB solution from Section 4 with four baseline approaches.

(1) RND (Random). This strategy selects a random edge to query, ignoring the $E_{promise}$ edge set. Since it does not spend computing resources to carefully choose edges, it quickly floods the system with many queries.

(2) MW (Max-Weight). This strategy selects the Webpage pair based on their initial direct similarity, also ignoring $E_{promise}$. The intuition behind using this model is that querying more similar pages (according to their direct features) is more likely to return "merge decisions".

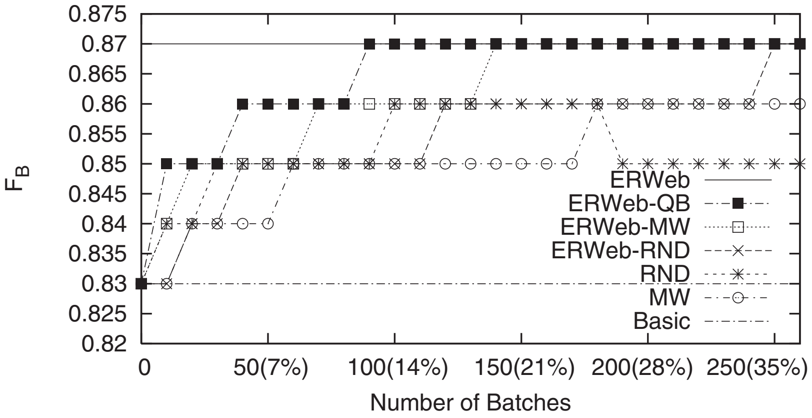


Fig. 21. Number of queries vs. quality on WePS-2 dataset.

(3) ERWeb-RND. This strategy also selects a random edge, but now from the $E_{promise}$ set.
(4) ERWeb-MW. This is like WM, but selects edges from the $E_{promise}$ set.

We first study how effective our strategy is in terms of just separating edges into "maybe" edges, that have higher potential for improving the results when queried, from the remaining edges that have lower potential. For that we compute the $\frac{|E|-|E_{maybe}|}{|E|}$ ratio that corresponds to the fraction of saved queries just due to this strategy, where E is the set of all edges in the graph. We observe that if there are only a few initial clusters then this ratio is typically 30–40%.

Figure 21 compares the proposed ERWeb-QB strategy with the baseline approaches based on the number of queries submitted versus the quality improvement. The figure demonstrates that the approaches that utilize the maybe-edges can reach higher quality with a smaller number of queries (only 12%–19% of the original queries) compared to the MW and RND methods that do not distinguish maybe-edges. ERWeb-QB reaches the highest F_B quality of 0.87 slightly ahead of the ERWeb-MW which is followed by ERWeb-RND. At 12% point where ERWeb-QB first reaches the highest quality, the difference with the other techniques is statistically significant at p-values of 0.01.

Figure 22 illustrates the change in the quality with respect to the time for the ERWeb-QB and ERWeb-QB+CR approaches. In these experiments, we submitted 15 batches (a batch consists of eight queries corresponding to a Webpage pair) of queries at each iteration, since Yahoo! Boss engine recently has imposed stricter limits on the concurrent query submission. Previously we were able to submit 100 queries at a time, and the algorithm was 6 times faster than what is shown in this figure. We observe that the proposed approach is more efficient than its original, since it can get high-quality clusters in less than two minutes by submitting only around 12% of the queries. The results demonstrate that the proposed approach reaches higher-quality results with less number of queries at the price of longer processing time.

8.3. Discussion on the Experiments

The experimental evaluations showed that the proposed approach outperform the state-of-the art solutions in terms of quality but at the price of increased response time due to Web querying. Further, Web querying has significant impact on the quality of the approach. However, this response time overhead is expected to be negligible if the approach is implemented as a server-side solution.

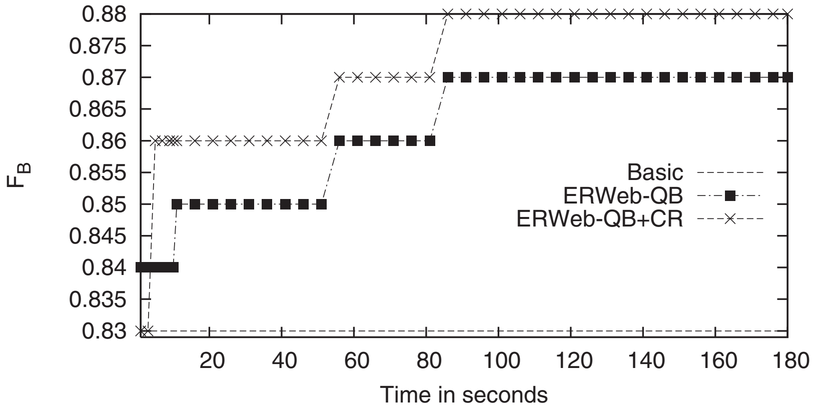


Fig. 22. Quality change as time increases for the proposed strategy.

9. RELATED WORK

The Web people search task is related to the problems of entity resolution and Word Sense Disambiguation (WSD). Entity resolution aims to resolve the references to the real-world objects in structured datasets, whereas WePS's objective is to associate the Webpages to the namesake they mention, hence it works on free and semistructured text. On the other hand, WSD aims to resolve the senses of terms in the documents, so the solutions usually work on free text as in WePS. To resolve ambiguity of terms, WSD approaches usually use an available list of senses presented as dictionaries, ontologies, etc. Hence, for a given term, the number of alternatives and their characteristics is known. For WePS task, however, the number of real-world entities and their characteristics are not known beforehand. That is, WePS is a clustering while WSD is a classification task.

In this section we first review the related work in Web people search (Section 9.1). After that we discuss the approaches that use external knowledge-bases for related tasks such as entity resolution (Section 9.2). We then highlight the new contributions of this article over its conference version (Section 9.3).

9.1. Web People Search

WePS has recently received significant attention from academia. Over the last few years, there have been three different WePS competitions. The first one was held in Sem-Eval 2007 [Artiles et al. 2007]. Sixteen different teams from different universities have participated in the task. The participating systems utilized named entities, tokens, URLs, etc., that exist in the documents for disambiguation. Named Entities (NE)-based single-link clustering was the one of the top three systems [Elmacioglu et al. 2007b], so NEs play an important role for this collection.

The second competition had eighteen different participants [Artiles et al. 2009]. In contrast to the WePS-1 competition, named entities did not play an important role in this dataset, as algorithms that do not depend on NEs performed better. The last WePS competition was held in 2010. This competition contained 300 person names and 200 Web documents for each name. However, ground-truth data for this dataset is still not publicly available, making it impossible to use for testing to nonparticipants.

WePS algorithms can be classified along multiple dimensions including type of features they exploit, as well as the clustering algorithms they use. As discussed previously, some algorithms rely only on the direct features, while others are capable of utilizing more data (i.e., external features) collected through a knowledge-base such as the Web or ontologies.

There are many different approaches that exploit direct features for clustering of Webpages for the purpose of WePS such as Wan et al. [2005], Artiles et al. [2005, 2007, 2009], Elmacioglu et al. [2007b], Jiang et al. [2009], Ono et al. [2008], Balog et al. [2009], Chen and Martin [2007], Iria et al. [2007], Li et al. [2005], and Yoshida et al. [2010]. Balog et al. [2009] analyzes the effect of direct textual features with different clustering algorithms such as k-means, agglomerative hierarchical cluster, probabilistic latent semantic indexing, while Elmacioglu et al. [2007b] analyzes the effect of different features if the single link clustering is used. Kalashnikov et al. [2008a], Jiang et al. [2009], and Iria et al. [2007], on the other hand, utilize the direct features with graph-based disambiguation algorithms. The work in Li et al. [2005] clusters documents based on the entity (person, organization, and location) names and can be applied to the disambiguation of semistructured documents, such as Webpages. The primary new contribution of that paper is the development of a document generation model that explains, for a given document, how entities of various types (other person names, locations, and organizations) are "sprinkled" onto the document. The work in Yoshida et al. [2010] uses a two-stage clustering to improve the quality of WePS. In the first stage some strong features such as named entities, compound keywords, and URLs are utilized for disambiguation. The first stage creates high-precision clusters, and in order to improve the recall of the clusters in the second stage they apply bootstrapping with single keywords. The main difference with the approach proposed in that paper and ours is that we utilize external knowledge-bases in both stages of the clustering, while they do not consider the external knowledge at all.

Bagga and Baldwin [1998] used a vector space model with the keywords in the text for the person name disambiguation in documents, while Niu et al. [2004] used information extraction along with the words in the document to do the disambiguation. Mann and Yarowsky [2003] used extracted biographic data for disambiguation. If one can extract biographic data, then it can help disambiguation, however, most of the Webpages do not contain biographic information about people at all. Thus relying exclusively on the bibliographic information will not work on the Web domain. Similarly, the same people might be present on the Web in documents with different topics. Accordingly an approach that relies only on the topic information will also fail to group all of the Webpages of the same namesake.

Techniques that only use only direct features are limited for a variety of reasons. First, usually in a Webpage, the amount of information referring to the person of interest varies from a few sentences to the whole content of the Webpage. Second, the Webpages are usually very noisy and cover a wide range of topics. Lastly, different Webpages often contain different portions of the related context (e.g., topics, entities) to the individual of interest. Such limitation of using direct features-based similarity in WePS is recognized and several solutions to address the problem have been proposed recently. One such solution is to exploit external resources for the purpose of disambiguation.

Approaches that exploit the external resources use either some knowledge-base such as Wikipedia [Han and Zhao 2009, 2010], Web directories [Kalashnikov et al. 2008a; Vu et al. 2008], or the Web content [Bekkerman and McCallum 2005; Chen et al. 2009a; Rao et al. 2007]. The approaches in Chen et al. [2009a; Rao et al. [2007] utilize the Web search engine results by collecting snippets, titles, and URLs. In addition, the work in Chen et al. [2009a] uses Google's 1TB n-grams dataset to compute the importance of keywords and bigrams. The approach of Bekkerman and McCallum [2005] is based on exploiting the link structure of pages on the Web, with the hypotheses that Webpages belonging to the same real person are more likely to be linked together. Further, the approach uses Google's index, with Web queries to compute IDF of each term (referred as Google IDF), which indeed might also be very expensive given the size of vocabulary.

9.2. Techniques that Exploit External Sources for Other Related Tasks

Recently, researchers have started to use external databases, such as ontology and Web search engine indexes in order to improve the classification and clustering qualities in different domains [Bollegala et al. 2007; Kanani et al. 2007; Elmacioglu et al. 2007a; Gabrilovich and Markovitch 2007]. For example, querying the Web and utilizing the search results are used for Word Sense Disambiguation (WSD) [Bollegala et al. 2007] and record linkage in publications domain [Elmacioglu et al. 2007a; Kanani et al. 2007]. However, the number of queries to a search engine is a bottleneck in all of these approaches. Hence, the study in Kanani et al. [2007] tries to solve the problem in the case of limited resources. The suggested algorithm increased the accuracy of data cleaning while keeping the number of queries to a search engine minimal. Similarly the approach in Elmacioglu et al. [2007a] uses the Web as a knowledge-base for data cleaning. The study proposed a way to formulate queries and used some standard measures like TF.IDF similarity to compute the similarity of two different references to an entity. The work in Elmacioglu et al. [2007a] is a complementary work to the one in Kanani et al. [2007].

In Bollegala et al. [2007] the authors proposed to use the co-occurrence counts to compute the semantic relatedness of words and cluster the words accordingly. The authors used Web-based similarity measures like WebJaccard, WebDice, and so on. These measures are utilized as features for the SVM-based trainer along with a set of-token-based features, where the trainer learns the probability of two terms being the same.

9.3. Contributions over the Conference Version of the Article

In this article, we study an approach that utilizes the external data sources with the aim of increasing the quality of the Web people search. Our study differs from the others in the way that it formulates the queries and utilizes the Web search results. This article is an extended version of our initial study [Kalashnikov et al. 2008a; Nuray-Turan et al. 2009] and the new contributions are as follows.

(1) In this article, we discuss an efficient version of the proposed approach which decreases the number of queries submitted to the search engine as well as the time to answer the query.
(2) In Kalashnikov et al. [2008a], only the named entity-based features were utilized as direct features; however, in this article we use significantly richer set of direct features and propose a simulated annealing-based mechanism to combine these features effectively for the purpose of WePS.
(3) The preliminary version of this article uses the single-link clustering algorithm; however, in this article we use correlation clustering, resulting in a higher clustering quality.
(4) We further proposed a new cluster refinement strategy for resolving the singleton cluster problem.

Some of our past entity resolution work is also related, but not directly applicable and uses different methodologies [Kalashnikov and Mehrotra 2006; Chen et al. 2007, 2005, 2008b; Kalashnikov et al. 2007, 2005; Nuray-Turan et al. 2007; Chen et al. 2009; Nuray-Turan et al. 2011].

```
Named-Entity-Filtering($D, m$)
for each webpage $d \in D$
    $\mathcal{P} \leftarrow$ Clean-PeopleNE-Set ($\mathcal{P}$)
    $\mathcal{P}_d \leftarrow$ Pick-M-PeopleNEs ($\mathcal{P}, d, m$)
    $\mathcal{O} \leftarrow$ Clean-OrgNE-Set ($\mathcal{O}$)
    $\mathcal{O}_d \leftarrow$ Pick-M-OrgNEs($\mathcal{O}, d, m$)
return $\{\mathcal{P}_d, \mathcal{O}_d\}$
```

Fig. 23. Named entity filtering.

10. CONCLUSIONS AND FUTURE WORK

This article proposes a novel Web people search approach that uses the Web as an external data source to achieve higher-quality clustering results. The proposed solution is based on collecting co-occurrence statistics from the Web. A new skyline-based classifier has been developed to classify these Web-based co-occurrence features. This new ad hoc classifier takes into account the dominance that exits in the feature space and greedily fine-tunes itself to any given quality measure. The article employs a simulation-annealing-based method to learn the importance of the direct and indirect features in order to combine them into the overall similarity value. A novel cluster refinement algorithm has been developed to handle singleton 1-page clusters. An algorithmic solution has been designed that improves the efficiency of the algorithm by reducing the overall number of queries sent to collect Web co-occurrence statistics. All these qualities combined allow the proposed approach to outperform other state-of-the-art techniques.

The proposed solution has been implemented as a research prototype system using the middleware architecture. Building the system has made us realize several new interesting problems. Some of these problems include: the summarization of clusters for generating proper cluster sketches, incrementally retrieving additional relevant pages for the selected namesake beyond top-K, identifying the automatically generated Webpages such as CiteseerX pages, which inherently contain ambiguous data. For instance, identifying auto-generated pages and treating them separately from the other pages decreases the mistakes made in the clustering process. In the future, we plan to tackle some of these problems.

APPENDIXES

A.1. Filtering For Web Queries

The pseudocode in Figure 23 demonstrates the filtering step applied on the named entities extracted off the top-K Webpages, which is done by using a set of filters. The idea is that NEs will be utilized by the Web querying component. However, certain types of NEs are too ambiguous for that purpose, in which case they are filtered out from further consideration in Web query generation. Given that the goal is to minimize the number of queries to the Web search engine, the filters are specifically designed not to use any extra queries.

Filter 1. We have found locations to be too ambiguous to be used as context queries for a few reasons. First of all the locations mentioned in the Webpages might be too general such that many namesakes might be mentioned in the context of this location. Second, even though we can identify the generality of the locations and use the ones that are more specific in queries, it is hard associate each namesake with just one location. For instance, although the CMU professor "Tom Mitchell" lives in Pittsburgh, he travels around the U.S. and give talks. Therefore it is easy to associate him say for instance with "Irvine, CA" assuming that he visited UC Irvine to give a talk, where there is another “Tom Mitchell” who is “Vice Chancellor of University Advancement and President of the University of California, Irvine Foundation.” For instance, when the query *“Tom Mitchell” AND “Irvine” AND “Pittsburgh”* is submitted to Google, it returns 97,700 Webpages. Therefore, the algorithm does not use location information as one of the contextual entities in Web query generation; however, the location information is utilized in other similarity features including TF.IDF-based similarity of the Webpages. Moreover, NE extractors sometimes wrongly extract the location names as organization names. This also creates the same problem mentioned before. Thus, to eliminate the ambiguity, the preprocessing step filters out the locations extracted as organizations. It does so by performing a lookup in a locally stored gazetteer with the organization name as the query. If there is a matching location in the gazetteer, then the organization name is simply filtered out and not considered in the Web query generation step.

Filter 2. The second filter deals with the NEs that consist of one-word person names, such as “John”. Such NEs are highly ambiguous since they can appear in the context of many namesakes on the Web. Consequently, the algorithm prunes away the NEs consisting of one-word names. This filter works by performing a lookup into the dataset that stores first names.

Filter 3. Similarly, the third filter handles NEs that are common English words. For example, word “defense” might be extracted from a Webpage as an organization by the extraction software. However, it is a commonly used word which can appear in the context of many namesakes. To detect common words the algorithm selects the most frequent 5000 terms from Wikipedia[9] as common English words. If an NE is a common English word, it is filtered out.

Filter 4. Suppose that we are disambiguating Webpages for “Jack Smith”. It is not rare to find out that two or more distinct “Jack Smith” namesakes are related to two distinct namesakes “John Smith”. These two Jacks might, for example, be relatives of the two Johns. Thus, “John Smith” cannot serve as a good context to identify a particular “Jack Smith” namesake. To capture this intuition, the algorithm filters out people NEs whose last name is the same as the last name specified in the original query.

A.2. Direct Features

A.2.1. Named-Entity-Based Similarity. Named entities include person, organization, and location names extracted using a named entity recognizer. For each Webpage d_i in D, initially all the named entities in them are extracted, then these named entities are tokenized and stop-words are removed. These tokens (terms) are then used to create document vectors for each d_i. Each term is weighted using a standard TF.IDF weighing scheme, where the importance (weight) of a term is computed by multiplying its frequency in the document with its Inverse Document Frequency (IDF). IDF measures how common a word is in the document collection. These feature vectors are used to calculate pairwise similarities of the documents, where the similarity function is selected as the cosine similarity.

For example, Webpage d_i might mention one publication of Tom Mitchel with his colleagues. Another Webpage d_j might mention another publication of the same Tom Mitchel and almost the same list of coauthors, but on a completely different topic. Hence, TF/IDF similarity computed on all of the keywords might not find sufficient evidence that the two Webpages are the same. However, the similarity of the extracted named entities can capture the relatedness of the Webpages.

[9] http://www.wikipedia.org.

A.2.2. Middle Name Initial Dissimilarity. It is used as evidence to identify likely different namesakes. First, for each Webpage in the collection full names are extracted using a set of rules. If any of the names (extracted from a Webpage) contain a middle name, then that name is assigned as the "full name" related to that Webpage. Then, the middle names of all pages are compared. For any two Webpages, if they both contain a middle name and it is different[10], this information is used as strong dissimilarity evidence and the similarity graph is updated accordingly.

For example, one Webpage might contain multiple references to "Tom A. Mitchell", while another Webpage would contain mentions of "Tom B. Mitchell". Since the middle initials do not match, this can serve as evidence that the two Webpages are about two different Tom Mitchells.

A.2.3. Hyperlinks-Based Similarity. Hyperlinks-based similarity is used to capture the fact that Webpages related to one of the namesakes often link to each other. First, the hyperlinks in each page are extracted. Then, for each pair of Webpages d_i and d_j, the approach checks if there is a link from one page to the other. Two pages are more likely to corefer if there is a direct link from one to the other. For example, a USC Webpage describing a talk of Tom Mitchel at USC might contain a link to his homepage at CMU, suggesting that the two Webpages are talking about the same Tom Mitchel. Similarly, Tom Mitchel from his CMU's homepage might provide a link to his talk at USC.

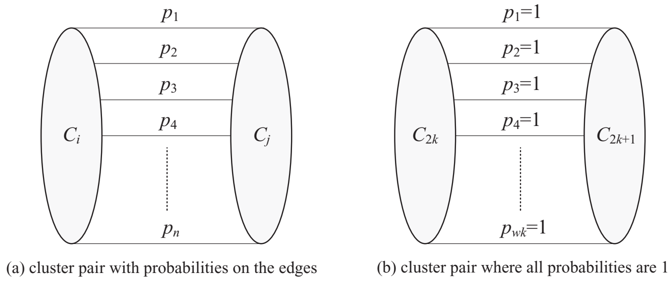
On the other hand, if there are no links between two Webpages, but there is a link from d_i (or d_j) to another page in the same URL path as d_j(or d_i), then it is more likely that both d_i and d_j corefer. For instance, suppose for a query "Tom Mitchel" the top-K results contains homepages of Tom Mitchel of CMU (`www.cs.cmu.edu/∼tom`) and Andrew McCallum (`www.cs.umass.edu/∼mccallum`). While these two homepages might not contain links to each other, the homepage of Tom Mitchel might refer to datasets available on McCallum's homepage (`www.cs.umass.edu/∼mccallum/data.html`). Notice how this URL contains in it the `www.cs.umass.edu/∼mccallum` part, linking the two homepages. Hence, such evidence is also used as positive evidence for merging.

A.2.4. Email-Based Similarity. Based on the intuition that any two Webpages that contain the same email address are probably about the same namesake, the approach derives another similarity feature for each Webpage pair. However, some emails might be too general to be used as evidence, such as `webmaster@xyz.com`. Thus, the approach uses a set of rules to identify such emails, and filters them from the Webpages. Consequently, from the remaining emails, for each pair of Webpages, the method checks if they share the same email address. If such an email is found then email-based similarity is used in similarity computations and graph is updated accordingly.

A.2.5. Social-Networking-Sites-Based Dissimilarity. When computing the similarity of two Webpages, the algorithm checks the similarity of their domains. If both Webpages are from the same domain and the domain is known as a social networking Website, then it is more likely that the two individuals mentioned in these Webpages are two different individuals. Thus, using this observation the algorithm uses the social networking site similarity as negative evidence. The list of social networking sites is collected from Wikipedia.

A.2.6. Word-Level N-Grams Similarity. As its name implies, word-level N-grams similarity computes the similarity of the two pages d_u and d_v using common n-grams in those pages. This feature is selected due to the observation that the longer the n-gram the more important it is in the similarity computations. N-grams capture the topical

[10] We have implemented an ad hoc function for comparing middle names. It, for instance, knows that "Andrew" and "A.", could be the same even though the spelling is different.



(a) cluster pair with probabilities on the edges (b) cluster pair where all probabilities are 1

Fig. 24. Query selection problem.

similarity of d_u and d_v, which is measured by the number of longest common n-grams in d_u and d_v. In our implementations, we extracted n-grams of size 2 to 6. Each word in an n-gram is stemmed and if there is a stop-word in the n-gram it is removed from it. In SA training explained in Section 5, five different weights are learned for the n-gram feature, one for each n, where $n = 2, 3, 4, 5, 6$.

To illustrate the usefulness of n-grams, suppose that a USC Webpage contains a long list of all the recent talks of its visitors in the form of the name of each visitor and the title of his/her talk. Assume that one of the many entries is a talk by Tom Mitchel with the 6-word title that matches the title of one of his recent publications. Assume that the homepage of Tom Mitchel, among many other things, mentions this recent publication as well. In this scenario, TF/IDF on keywords, or even named entities, is unlikely to help, because these small matching entries (for the title of the paper and talk) will be lost in the abundance of other information in these Webpages. But even in the absence of any other linking information, the matching 6-gram might be positive evidence that the two Webpages corefer to the same Tom Mitchel, especially if the URLs of the two Webpages do not overlap. This is since the likelihood that 6 random words would appear in the same 6-gram sequence in two different Webpages for two different namesakes is low.

A.3. Complexity of the Query Selection Problem

Under simplified assumptions, we can view the query selection problem as follows; see Figure 24(a). Let C_i, C_j be a cluster pair under consideration and $\triangle Q_{ij}$ be the expected quality improvement if these clusters are grouped together. Let E_{ij} be the inter-cluster edges between C_i, C_j and G_{ij} be the to-be-chosen subset of E_{ij} that corresponds to the queries that will be issued to the search engine. Given the probabilities on the edges of G_{ij} we can compute probability $\mathbb{P}(C_i, C_j \text{ merged}|G_{ij})$ that C_i, C_j getting merged given the results of G_{ij}. We can compute the expected benefit for C_i, C_j given G_{ij} as $\triangle Q_{ij} \cdot \mathbb{P}(C_i, C_j \text{ merged}|G_{ij})$. Now our objective is to select a subset of queries such that the total expected benefit of the selected queries is maximized and the total number of queries does not exceed the limit $\mathcal{L}$, where $\mathcal{L}$ is the total number of queries we can submit in n iterations (i.e., $\mathcal{L} = n \cdot \mathcal{B}$). Therefore, our goal can be formulated as a solution to the following optimization problem.

$$\begin{aligned} &\text{Maximize } Z = \textstyle\sum_{G_{ij}} \triangle Q_{ij}\mathbb{P}(C_i, C_j \text{ merged}|G_{ij}), \\ &\text{subject to } \textstyle\sum_{G_{ij}} |G_{ij}| \leq \mathcal{L}. \end{aligned} \tag{7}$$

We can prove the following.

THEOREM 1. *Finding a subset of no more than $\mathcal{L}$ queries that maximize the expected benefit is $\mathcal{NP}$-hard.*

PROOF. By reduction from the 0-1 knapsack problem. Recall, in 0-1 knapsack problem there are n objects $o_1, o_2, \ldots, o_n$. Each object o_i has integer benefit v_i and integer weight w_i. The task is to pack the knapsack such that the benefit is maximized while the overall weight does not exceed the maximum weight of W [Kellerer et al. 2004]. That is

$$\begin{aligned} &\text{Maximize } Z = \textstyle\sum_{k=1}^{n} v_k x_k, \\ &\text{subject to } \textstyle\sum_{k=1}^{n} w_k x_k \leq W, \\ &\qquad x_k \in \{0, 1\}, k = 1, 2, \ldots, n. \end{aligned} \tag{8}$$

The 0-1 knapsack problem can be mapped, in polynomial time, into an instance of our problem, as illustrated in Figure 24(b). For each object o_k a pair of clusters C_{2k}, C_{2k+1} is created. They are then connected via exactly w_k edges, where each edge has probability 1. The expected quality improvement $\Delta Q_{2k,2k+1}$ by merging these two clusters is set to $\Delta Q_{2k,2k+1} = v_k$. The parameter $N_{2k,2k+1}$ that corresponds to the minimum number of merge decisions needed to merge C_{2k}, C_{2k+1} is set to $N_{2k,2k+1} = w_k$. That way, $\mathbb{P}(C_{2k}, C_{2k+1} \text{ merged}|G_{2k,2k+1}) = 1$ only if all edges in $E_{2k,2k+1}$ are chosen for querying, that is, when $|G_{2k,2k+1}| = w_k$. Otherwise, when $|G_{2k,2k+1}| < w_k$ it follows that $\mathbb{P}(C_{2k}, C_{2k+1} \text{ merged}|G_{2k,2k+1}) = 0$. Parameter $\mathcal{L}$ is set to $\mathcal{L} = W$. Clearly, given the one-to-one mapping, if this instance of the problem has a polynomial solution then 0-1 knapsack problem has a polynomial solution and vice versa. Thus the problem is NP-hard. □

ACKNOWLEDGMENTS

We would like to thank Raghu Ramakrishnan, Rahul Hampole, and Nagesh Pobbathi of Yahoo! for making Yahoo! BOSS system[11] available for large-scale experimentation. Their help has resulted in a significant improvement in the quality of our publication.

REFERENCES

ARTILES, J., GONZALO, J., AND SEKINE, S. 2007. The SemEval-2007 WePS evaluation: Establishing a benchmark for the web people search task. In *Proceedings of SemEval-2007 Workshop*.

ARTILES, J., GONZALO, J., AND SEKINE, S. 2009. Weps 2 evaluation campaign: Overview of the web people search clustering task. In *Proceedings of the Web People Search Evaluation Workshop (WePS'09)*.

ARTILES, J., GONZALO, J., AND VERDEJO, F. 2005. A testbed for people searching strategies in the WWW. In *Proceedings of the ACM SIGIR Conference*.

BAGGA, A. AND BALDWIN, B. 1998. Entity-Based cross-document coreferencing using the vector space model. In *Proceedings of the International Conference on Computational Linguistics*.

BALOG, K., AZZOPARDI, L., AND RIJKE, M. 2009. Resolving person names in web people search. In *Weaving Services and People on the World Wide Web*, 301–323.

BANSAL, N., BLUM, A., AND CHAWLA, S. 2004. Correlation clustering. *Mach. Learn. 56,* 1.

BEKKERMAN, R. AND MCCALLUM, A. 2005. Disambiguating web appearances of people in a social network. In *Proceedings of the International Conference on World Wide Web (WWW)*.

BOLLEGALA, D., MATSUO, Y., AND ISHIZUKA, M. 2007. Measuring semantic similarity between words using web search engines. In *Proceedings of the International Conference on World Wide Web (WWW)*.

BORZSONYI, S., KOSSMANN, D., STOCKER, K., AND PASSAU, U. 2001. The skyline operator. In *Proceedings of the International Conference on Data Engineering (ICDE)*. 421–430.

[11] Yahoo! Search BOSS is a commercial service that provides developers access to various information sources such as Web, image and news search for a low per-query fee. It supports advanced capabilities to run complex queries on such data sources. More information regarding BOSS is available at http://developer.yahoo.com/search/boss/.

Chen, S., Kalashnikov, D. V., and Mehrotra, S. 2007. Adaptive graphical approach to entity resolution. In *Proceedings of the ACM IEEE Joint Conference on Digital Libraries (JCDL)*.

Chen, Y., Lee, S. Y. M., and Huang, C.-R. 2009a. Polyuhk: A robust information extraction system for web personal names. In *Proceedings of the Web People Search Evaluation Workshop (WePS 2009)*.

Chen, Y. and Martin, J. H. 2007. Cu-Comsem: Exploring rich features for unsupervised web personal name disambiguation. In *Proceedings of the SemEval-2007 Workshop*.

Chen, Z., Kalashnikov, D. V., and Mehrotra, S. 2005. Exploiting relationships for object consolidation. In *Proceedings of the International ACM SIGMOD Workshop on Information Quality in Information Systems (IQIS)*.

Chen, Z. S., Kalashnikov, D. V., and Mehrotra, S. 2009b. Exploiting context analysis for combining multiple entity resolution systems. In *Proceedings of the ACM SIGMOD Conference*.

Cho, J., Garcia-Molina, H., Haveliwala, T., Lam, W., Paepcke, A., Raghavan, S., and Wesley, G. 2006. Stanford webbase components and applications. In *ACM Trans. Internet Technol.*

Elmacioglu, E., Kan, M.-Y., Lee, D., and Zhang, Y. 2007a. Web based linkage. In *Proceedings of the WIDM07 Conference*.

Elmacioglu, E., Tan, Y. F., Yan, S., Kan, M.-Y., and Lee, D. 2007b. Psnus: Web people name disambiguation by simple clustering with rich features. In *Proceedings of the SemEval Conference*.

Gabrilovich, E. and Markovitch, S. 2007. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *Proceedings of the IJCAI Conference*.

Gong, J. and Oard, D. 2009. Determine the entity number in hierarchical clustering for web personal name disambiguation. In *Proceedings of the Web People Search Evaluation Workshop (WePS)*.

Guha, R. and Garg, A. 2004. Disambiguating people in search. Tech. rep., Stanford University.

Han, X. and Zhao, J. 2009. Named entity disambiguation by leveraging wikipedia semantic knowledge. In *Proceedings of the CIKM Conference*.

Han, X. and Zhao, J. 2010. Structural semantic relatedness: A knowledge-based method to named entity disambiguation. In *Proceedings of the Annual Meeting of the ACL*.

Hirsch, J. 2005. An index to quantify an individual's scientific research output. *Proc. Nat. Acad. Sci. 102,* 46, 16569.

Iria, J., Xia, L., and Zhang, Z. 2007. Wit: Web people search disambiguation using random walks. In *Proceedings of the SemEval-2007 Workshop*.

Jiang, L., Wang, J., An, N., Wang, S., Zhan, J., and Li, L. 2009. Grape: A graph-based framework for disambiguating people appearances in web search. In *Proceedings of the ICDM Conference*.

Kalashnikov, D. V., Chen, Z., Mehrotra, S., and Nuray, R. 2008a. Web people search via connection analysis. *IEEE Trans. Knowl. Data Engin. 20,* 11.

Kalashnikov, D. V. and Mehrotra, S. 2006. Domain-independent data cleaning via analysis of entity-relationship graph. *ACM Trans. Datab. Syst. 31,* 2, 716–767.

Kalashnikov, D. V., Mehrotra, S., Chen, S., Nuray, R., and Ashish, N. 2007. Disambiguation algorithm for people search on the web. In *Proceedings of the IEEE 23rd International Conference on Data Engineering (ICDE)*.

Kalashnikov, D. V., Mehrotra, S., and Chen, Z. 2005. Exploiting relationships for domain-independent data cleaning. In *Proceedings of the SIAM International Conference on Data Mining (Data Mining)*.

Kalashnikov, D. V., Nuray-Turan, R., and Mehrotra, S. 2008b. Towards breaking the quality curse. A web-querying approach to Web People Search. In *Proceedings of the SIGIR Conference*.

Kanani, P., McCallum, A., and Pal, C. 2007. Improving author coreference by resource-bounded information gathering from the web. In *Proceedings of the IJCAI Conference*.

Kellerer, H., Pferschy, U., and Pisinger, D. 2004. *Knapsack Problems*. Springer.

Kossmann, D., Ramsak, F., and Rost, S. 2002. Shooting stars in the sky: An online algorithm for skyline queries. In *Proceedings of the VLDB Conference*.

Lerman, I. C. 1970. *Les Bases de la Classification Automatique*. Gauthier-Villars.

Li, X., Morie, P., and Roth, D. 2005. Semantic integration in text: From ambiguous names to identifiable entities. *AI Mag.* 45–68.

Liu, Z., Lu, Q., and Xu, J. 2011. High performance clustering for web person name disambiguation using topic capturing. In *Proceedings of the International Workshop on Entity-Oriented Search (EOS)*.

Mann, G. S. and Yarowsky, D. 2003. Unsupervised personal name disambiguation. In *Proceedings of the ACL Conference*.

Niu, C., Li, W., and Srihari, R. K. 2004. Weakly supervised learning for cross-document person name disambiguation supported by information extraction. In *Proceedings of the Conference ACL*.

NURAY-TURAN, R., CHEN, Z., KALASHNIKOV, D. V., AND MEHROTRA, S. 2009. Exploiting Web querying for Web People Search in WePS2. In *Proceedings of the 2nd Web People Search Evaluation Workshop (WePS)*.

NURAY-TURAN, R., KALASHNIKOV, D. V., AND MEHROTRA, S. 2007. Self-tuning in graph-based reference disambiguation. In *Proceedings of the DASFAA Conference*.

NURAY-TURAN, R., KALASHNIKOV, D. V., MEHROTRA, S., AND YU, Y. 2011. Attribute and object selection queries on objects with probabilistic attributes. *ACM Trans. Datab. Syst. 36,* 4.

ONO, S., SATO, I., YOSHIDA, M., AND NAKAGAWA, H. 2008. Person name disambiguation in web pages using social network, compound words and latent topics. In *Proceedings of the PAKDD Conference*. 260–271.

RAO, D., GARERA, N., AND YAROWSKY, D. 2007. Jhu1 : An unsupervised approach to person name disambiguation using web snippets. In *Proceedings of the SemEval-2007 Workshop*.

SEARCH ENGINE WATCH. 2006. Searches per day. `http://searchenginewatch.com/2156461`.

STANFORD NER. Stanford ner. `http://nlp.stanford.edu/ner/index.shtml`.

VU, Q., TAKASU, A., AND ADACHI, J. 2008. Improving the performance of personal name disambiguation using web directories. *Inf. Process. Manag. 44,* 4, 1546–1561.

WAN, X., GAO, J., LI, M., AND DING, B. 2005. Person resolution in person search results: Webhawk. In *Proceedings of the CIKM Conference*.

YOSHIDA, M., IKEDA, M., ONO, S., SATO, I., AND NAKAGAWA, H. 2010. Person name disambiguation by bootstrapping. In *Proceedings of the SIGIR Conference*.

Received February 2011; revised July 2011; accepted October 2011