No Free Lunch: Brute Force vs. Locality-Sensitive Hashing for Cross-lingual Pairwise Similarity

Ferhan Ture[1], Tamer Elsayed[2], Jimmy Lin[1,3]

[1]Dept. of Computer Science, [3]The iSchool
University of Maryland

[2]Mathematical and Computer Sciences and Engineering Division
King Abdullah University of Science and Technology (KAUST)

fture@cs.umd.edu, tamer.elsayedaly@kaust.edu.sa, jimmylin@umd.edu

ABSTRACT

This work explores the problem of cross-lingual pairwise similarity, where the task is to extract similar pairs of documents *across* two different languages. Solutions to this problem are of general interest for text mining in the multilingual context and have specific applications in statistical machine translation. Our approach takes advantage of cross-language information retrieval (CLIR) techniques to project feature vectors from one language into another, and then uses locality-sensitive hashing (LSH) to extract similar pairs. We show that effective cross-lingual pairwise similarity requires working with similarity thresholds that are much lower than in typical monolingual applications, making the problem quite challenging. We present a parallel, scalable MapReduce implementation of the sort-based sliding window algorithm, which is compared to a brute-force approach on German and English Wikipedia collections. Our central finding can be summarized as "no free lunch": there is no single optimal solution. Instead, we characterize effectiveness-efficiency tradeoffs in the solution space, which can guide the developer to locate a desirable operating point based on application- and resource-specific constraints.

Categories and Subject Descriptors: H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval

General Terms: Algorithms, Performance

Keywords: LSH, machine translation, Wikipedia

1. INTRODUCTION

In its most general form, pairwise similarity computation deals with finding pairs of objects in a large dataset that are similar according to some measure. This problem is frequently encountered in text processing applications, for example, clustering for unsupervised learning [28, 26], generation of similarity lists for "more-like-this" queries [18], and near-duplicate detection in the web context. Instances of this problem are encountered in other domains such as bioinformatics as well [22].

This work focuses on pairwise similarity in text collections, but introduces a new twist—we wish to mine similar pairs of documents that are in *different* languages. At a high level, this problem is driven by an evolution toward more multi-lingual societies, which makes the ability to communicate across language barriers increasingly important. Machine translation (MT) is one technological solution, and modern systems are heavily driven by statistical methods dependent on large amounts of training data [16]. Traditionally, algorithms have required texts that are mutual translations of each other—called *parallel* corpora—but there has also been work on exploiting *comparable* corpora—or texts in different languages that are similar, but not necessarily mutual translations [21, 29, 24, 31]. In our view, parallel and comparable corpora lie on a continuum of similarity, and solutions to the cross-lingual pairwise similarity problem can unlock new sources of training data for MT systems.

The specific application we focus on is automatically generating links between Wikipedia articles in different languages (so-called "interwiki" language links). In addition to contributing to the overall goal of data mining for machine translation, we believe the task is independently interesting. For example, there presently exists a link between the article "Friedensnobelpreis" in German and "Nobel Peace Prize" in English. However, as far as we know, these links are manually created, and therefore suffers from many shortcomings: link creation requires volunteers who know *both* languages and is a labor-intensive, error-prone endeavor. As a result, link coverage is relatively sparse. A cross-lingual pairwise similarity algorithm may help: its output can feed into a manual verification process that improves both speed and accuracy (e.g., perhaps aided by crowdsourcing).

Our approach uses locality-sensitive hashing (LSH) [4, 5, 2], which represents documents as bit-signatures, such that two similar documents are likely to have similar signatures. A sort-based sliding window algorithm on permutations of bit signatures is used to extract similar pairs. LSH provides a tradeoff between efficiency and effectiveness by user-controlled parameters, and can be straightforwardly parallelized since multiple randomizations run independently.

Although LSH is a well-explored solution to the pairwise similarity problem, this work has several contributions.

SIGIR'11, July 24–28, 2011, Beijing, China.
Copyright 2011 ACM 978-1-4503-0757-4/11/07 ...$10.00.

First, we introduce and motivate the cross-lingual variant of the pairwise similarity problem, which has received little attention in the literature. Second, we demonstrate that the cross-lingual variant of the problem is considerably more challenging since it requires mining document pairs that have low similarities, far lower than thresholds typically used in monolingual solutions. Third, we describe a scalable MapReduce [10] implementation of the sort-based sliding window LSH algorithm for solving the problem. The solution exhibits linear scalability and a high degree of parallelism that makes it suitable for large collections. Finally, we present an analytical and empirical analysis of our approach across a wide range of parameter settings, using the brute-force N^2 approach as a baseline. Our central finding can be summarized as "no free lunch": there is no single optimal solution to the cross-lingual pairwise similarity problem. Instead, we characterize effectiveness-efficiency tradeoffs within the solution space, which can guide the developer to locate a desirable operating point based on application- and resource-specific constraints. This characterization addresses a weakness of LSH approaches in general, in that they present a bewildering number of parameters that need to be set, and provide little guidance for an application developer approaching new problems. To this end, we show that the efficiency and the effectiveness of our algorithm can be analytically estimated, thus allowing the developer to characterize the tradeoff space for a particular problem without actually needing to run any experiments.

The remainder of the paper is organized as follows. We begin with related work in Section 2. A formal definition of the cross-lingual pairwise similarity problem is provided in Section 3. Our LSH-based approach is detailed in Section 4, followed by an analytical model in Section 5. Experimental results are presented in Section 6 before concluding.

2. RELATED WORK

The problem of efficiently computing pairwise similarity has been extensively studied by text processing and data mining researchers; in the database community, this is better known as the "set similarity join" problem [12, 33]. Another variant that has received attention is computing pairwise "functional" computations (not necessarily similarities) between elements in large datasets [23, 15].

In general, two approaches to the problem exist: the *index-based* approach focuses on building an inverted index and pruning it to achieve efficient similarity computations [3, 12, 32, 33]; and the *signature-based* approach that transforms the data into a more compact representation for performing similarity comparisons. The LSH [4, 5, 2] techniques we adopt exemplifies the second approach.

There is also extensive work on near-duplicate detection of web pages, which can be viewed as a special case that aims to detect highly-similar pairs, sometimes without a predefined similarity threshold or even a similarity measure. Both index-based [32, 35] and signature-based [6, 20, 17, 13, 14] approaches have been proposed to address this problem.

All of the above focus on monolingual or homogeneous similarity, either similarity within one language or similarity within a homogeneous collection of objects. We are aware of some work that attempts to solve the cross-lingual version of the problem, but in a slightly different setting. Anderka et al. [1] concluded that both signature-based and index-based approaches require at least a linear scan of the collection due to the low similarity thresholds. However, the authors do not describe an algorithm or show any experimental results. In another recent paper, Platt et al. [27] described techniques for projecting multi-lingual documents into a single vector space: training "encourages" comparable document pairs to have similar vector representations. This and other work on extracting parallel sentences [24, 31] focus primarily on representational issues and do not explicitly address questions of scale. Thus, these techniques can be viewed as complementary to our own.

Our work takes advantage of Hadoop, the open-source implementation of MapReduce [10], which has recently emerged as a popular model for large-scale data processing. The problem of pairwise similarity computation has been studied in MapReduce previously [18, 11, 33]. However, these algorithms adopt an index-based approach, which stands in contrast to our signature-based approach.

3. CROSS-LINGUAL SIMILARITY

Assuming a "bag of words" model, where a document d is represented as a vector of term weights $w_{t,d}$, one for each term t in the vocabulary space V, similarity between two document vectors u and v is computed via cosine similarity: $\text{Sim}(u, v) = \cos(\theta(u, v))$. There are, of course, many possible weighting schemes for $w_{t,d}$; here, we adopt BM25 [30], which has been shown to be effective for many retrieval tasks. In this scheme, each weight is a function of document frequency (df) and term frequency (tf) values.

The pairwise similarity problem is that of finding all pairs of documents (u, v) with cosine similarities above a certain threshold τ. In *cross-lingual* pairwise similarity, we additionally stipulate that the document vectors represent documents in different languages, thus introducing additional complexity. Due to different vocabulary spaces, document vectors in different languages are not directly comparable.

3.1 Matching Vocabulary Spaces

We experimented with two different solutions to overcome vocabulary mismatch: translation and vector projection.

The first approach is to *translate* documents from one language into the other using an off-the-shelf machine translation system. If we are working with German and English as the language pair, we could translate all German documents into English, and then perform pairwise similarity in the English vocabulary space. The advantage of this approach is that modern machine translation systems produce quite reasonable results, especially for European languages. The downside, however, is that machine translation is typically time-consuming and resource-intensive.

The second approach is to *project* document vectors from one language into another using cross-language information retrieval (CLIR) techniques. Adopting the approach proposed by Darwish and Oard [8], a document vector v in some foreign language F can be translated into a document vector v^* in the target language E by computing df^* and tf^* values for every term $e \in E$ as follows:

$$df^*(e) = \sum_{f \in F} p(f|e) \cdot df(f)$$

$$tf^*(e) = \sum_{f \in F} p(f|e) \cdot tf(f)$$

where we assume df and tf values are represented in v. Since

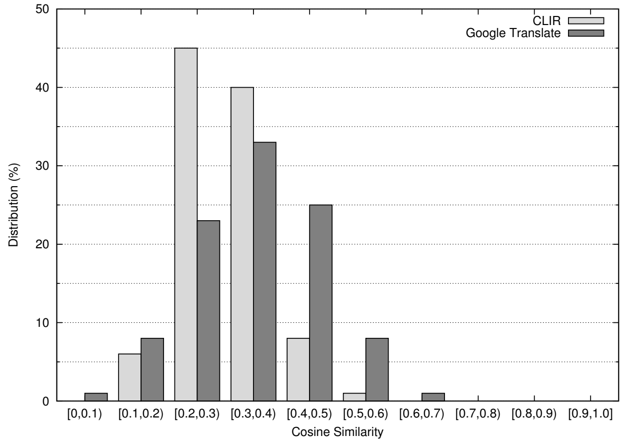

Figure 1: Histogram of cosine similarities from 505 semantically similar documents in German and English, comparing Google Translate with CLIR.

term weight is a function of df^* and tf^* values, this information is sufficient to construct the vector v^*. The quantity $p(f|e)$ is the conditional probability of a foreign language word f being the translation of a target language word e; this distribution captures lexical translation ambiguities—the fact that words in one language can translate to multiple words in another language. This distribution can be estimated from parallel corpora using unsupervised techniques, as the output of the word alignment problem [34, 25]. Solutions to this problem are well understood, and there exist multiple off-the-shelf implementations.

3.2 Implications

Typical instances of the monolingual pairwise similarity problem involve extracting pairs with high cosine similarities (e.g., 0.9 or higher). For the cross-lingual case, we expect similar documents in different languages to have lower similarities, since the translation process is noisy.

To empirically determine appropriate similarity thresholds for the cross-lingual case, we experimented with the German and English portions of the Europarl corpus, which contains proceedings from the European Parliament (1996—2009). We constructed artificial "documents" by concatenating every 10 consecutive sentences into a single document. In this manner, we sampled 505 document pairs that are mutual translations of each other (and therefore semantically similar by construction). This provides ground truth to evaluate the effectiveness of the two translation approaches discussed above: machine translation (in this case, we used Google Translate[1]) and direct vector projection using the CLIR approach. In the first case, we translated each German document into English and then processed the resulting translated English document into vector form. In the second case, we first processed the German document into vector form, and then projected the vector over into English. We then computed the cosine similarities of the document pairs, under both conditions.

The distribution of the cosine similarities is shown in Figure 1, binned in 0.1 intervals. We make two important observations. First, the cosine similarities are surprisingly low, and we expect even lower values in practice, since these are artificially created perfect document translations. This implies that in order to mine semantically similar document pairs in different languages, we need to extract documents that have low cosine similarities. Second, cosine similarity scores are higher for Google Translate, but vector projection seems reasonably competitive, especially considering the computational tradeoff: running an MT system on millions of documents is computationally expensive. For this reason, the remainder of this paper uses the CLIR vector projection technique.

4. SIMILARITY COMPUTATIONS

Our approach, which is based on LSH [4, 5, 2], consists of three steps. First, all documents are preprocessed into document vectors (Section 4.1). Second, document signatures are generated from document vectors (Section 4.2). Finally, a sliding window algorithm applied to the signatures finds all cross-lingual pairs above a similarity threshold (Section 4.3).

4.1 Preprocessing Documents

In the preprocessing step, documents from both languages are parsed, tokenized, and represented as weighted document vectors. All terms that occur only once in the collection are removed and each remaining term is converted into an integer for efficiency reasons. Document vectors of the foreign language (i.e., German) are projected into the target language (English) by the CLIR approach explained in Section 3. Thus, the collections in two languages are converted into a single collection of document vectors in the target language. Since we are only interested in cross-lingual pairs, we use document identifiers (docids) to determine the original language of a document vector.

4.2 Generating Document Signatures

At the core of any pairwise similarity algorithm is the similarity calculation between pairs of documents. The major drawback of cosine similarity is that the calculation requires a number of floating point operations linear to the number of terms in u and v, which is computationally expensive. Although it is possible to accurately approximate floating point values with integers using quantization methods, the similarity calculation still remains a bottleneck when dealing with many documents and large vocabularies.

Signatures are an efficient and effective way of representing large feature spaces. We explore three well-known methods to generate signatures from document vectors: random projection [5], Simhash [20], and Minhash [7].

Random projection (RP) signatures [5] use a series of random hyperplanes as hash functions to encode document vectors as fixed-size bit vectors. Let us assume a collection with vocabulary V, so that each document vector is from the domain $\mathbb{R}^{|V|}$. To obtain a signature of D bits using this approach, D randomly generated real-valued vectors of length $|V|$ are used to map each document vector u onto a RP signature $s_u \in [0,1]^D$. The i^{th} bit of s_u is determined by an inner product of u and the i^{th} random vector.

Given D random vectors $r_1, ..., r_D$, the signature s_u is computed as follows:

$$s_u[i] = h_{r_i}(u) = \begin{cases} 1 & \text{if } r_i \cdot u \geq 0 \\ 0 & \text{if } r_i \cdot u < 0 \end{cases} \quad (1)$$

The cosine similarity between two documents can be com-

[1] http://translate.google.com

puted via hamming distance between their signatures, according to the following relationship [5]:

$$\text{Sim}(u, v) = \cos\left[\left(\frac{\pi}{D}\right)\text{hamming}(s_u, s_v)\right] \tag{2}$$

Simhash signatures are essentially a "hash" of the document vector. This approach relies on a hash function that generates hash values for every term in the document vector. Given a document vector u and a hash function h that maps string terms to D-bit hash values, we generate a D-bit signature s as follows:

$$\begin{array}{rl} s[i] = 1 & \text{if } \sum_{(t,w)\in u}(2h(t)[i] - 1)w > 0 \\ 0 & \text{otherwise} \end{array}$$

where every term-weight pair (t, w) in u contributes $+w$ to $s[i]$ if $h(t)[i] = 1$ and $-w$ otherwise. If the sum of contributions to $s[i]$ is greater than 0, then $s[i]$ is set to 1, otherwise, it is set to 0.

For this approach, signature quality depends on the quality of the hash function, which also dictates the signature length (which cannot be set arbitrarily). In this work, we use the hash function in Manku et al. [20], where it was applied to detect near-duplicate web pages. We leave exploration of more recent methods along similar lines, such as self-taught hashing [36] for future work.

Minhash signature for a document vector u requires K random orderings of terms in u. Given a hash function, terms can be ordered by their hash value, or alternatively, terms can be ordered by a random permutation on the vocabulary. For each of the K orderings, the term in a document that has lowest order is picked as the "minimum hash" of that document. The probability that two documents u and v have the same "minimum hash" term for a given ordering is $\frac{|u \cap v|}{|u \cup v|}$ (i.e., Jaccard similarity). This procedure is repeated for K different randomly selected orderings to reduce the risk of false positives; thus, the *Minhash* signature of a document vector consists of all K "minimum hash" terms. The cosine similarity between u and v is approximated by the proportion of terms they share. Chum et al. [7] showed its effectiveness on near-duplicate image detection.

In order to compare the precision of these three signature generation techniques, we selected a sample of 1064 document vectors from German Wikipedia. For every pair of document vectors, we computed the true cosine similarity and estimated values using each of the three signature methods and varying signature lengths. In each case, we computed the average absolute difference between the true and estimated cosine similarity value, shown in Table 1. For comparison, the last column shows the average time (in milliseconds) taken to generate one signature on a laptop with an Intel Core 2 Duo 2.26 GHz processor.

Simhash signatures generated by the approach in Manku et al. [20] are 64 bits. Since Minhash signatures are represented as a list of integers (corresponding to term ids), they take up 32 times more space than a Simhash or RP signature of the same length. We included 2-term (64 bits) and 32-term (992 bits) Minhash signatures in the comparison.

We performed three sets of experiments: First, we computed error for all possible pairs, then filtered out pairs with cosine similarity less than 0.1, and finally less than 0.2. Precision at very low similarities is not very important because relevant documents usually have values higher than 0.2 (based on Figure 1). We noticed that Minhash signatures perform worse as we filter out low-similarity pairs because it simply predicts 0.0 for many pairs. In contrast, RP signatures are more accurate when the cosine similarity is higher, an indicator of the robustness of the method.

Method	**Bits**	**Avg. Absolute Error**			**Time (ms)**
		> 0.0	> 0.1	> 0.2	
Minhash	64	0.061	0.168	0.215	0.28
Simhash	64	0.236	0.299	0.273	0.25
RP	64	0.154	0.152	0.123	1.17
RP	100	0.124	0.122	0.101	2.03
RP	200	0.088	0.086	0.069	4.52
RP	500	0.056	0.055	0.045	10.93
RP	1000	0.040	0.039	0.033	20.91
Minhash	992	0.050	0.056	0.075	1.82

Table 1: Average cosine similarity error per signature with different methods

In all cases, 64-bit RP signatures are more precise than Simhash signatures, and this becomes more apparent once the low-similarity pairs are filtered out. The only drawback of RP signatures is the amount of time necessary to generate them: an inner product is needed to determine every bit. The average time to create a 64-bit RP signature is about 5 times slower than other methods with the same number of bits and generating 1000-bit RP signatures is more than 10 times slower than 32-term Minhash signatures. However, the running time of the signature construction step is relatively brief compared to the actual pairwise similarity algorithm. Based on this analysis, we selected 1000-bit RP signatures for our experiments.

4.3 Sliding Window Algorithm

The sliding window algorithm [5] is a probabilistic method which uses randomization and approximation heuristics to provide a tradeoff between efficiency and effectiveness. In this section, we first describe a straightforward parallel implementation of the original version of this algorithm in MapReduce (Figure 2). Then, we discuss modifications to exploit the framework and take advantage of greater parallelism (Figures 3 and 4). For space considerations, we assume that the reader has at least passing familiarity with the sliding window algorithm, as it is a well-known LSH technique. We further assume that the reader is already familiar with the MapReduce programming model.

4.3.1 Basic Algorithm

In the sliding window algorithm, Q random permutation functions $p_1, ..., p_Q$ are created as a preprocessing step.[2] Input is the sequence of (docid, signature) pairs for the entire collection. Each mapper takes a docid n and signature s as input, and emits an intermediate key-value pair $(\langle i, s^i\rangle, n)$ for each permutation function $p_i, i = 1...Q$.[3]

Each key sent to a reducer is a pair of the permutation group number p and the signature s (denoted as $\langle p, s\rangle$) and the values are docids sharing that pair. The reduce step is designed so that all keys with the same permutation group number are sent to the same reducer (by appropriately partitioning the key space), and they are sorted according to the

[2] A permutation function is basically a permuted list of the integers $[1 \dots D]$, where D is the signature size.

[3] We denote s^i as the permutation of s with respect to the permutation function p_i.

```
Mapper
 1: method Map(docid $n$, signature $s$)
 2:     for all permute func $p_i \in [p_1, p_2, \ldots, p_Q]$ do
 3:         Emit($\langle i, s.\textsc{Permute}(p_i) \rangle, n$)
Reducer
 4: method Initialize
 5:     $docids \leftarrow$ new Queue($B$)
 6:     $sigs \leftarrow$ new Queue($B$)
 7: method Reduce($\langle$permno $p$, sig $s\rangle$, docids $[n_1, n_2, \ldots]$)
 8:     for all docid $n \in [n_1, n_2, \ldots]$ do
 9:         for $i = 1$ to $sigs$.Size() do
10:             $distance \leftarrow s$.Distance($sigs$.get($i$))
11:             if $distance \leq T$ then
12:                 Emit($\langle docids.\text{get}(i), n \rangle, distance$)
13:         $sigs$.Enqueue($s$)
14:         $docids$.Enqueue($n$)
15:         if $sigs$.Size() $> B$ then
16:             $sigs$.Dequeue()
17:             $docids$.Dequeue()
```

Figure 2: Pseudo-code of the initial version of the sliding window algorithm.

signature bits (comparison of two signatures is from most significant to least significant bit). Therefore, each of the Q reducers receive a sorted list of permuted signatures, paired with respective docids, which we call a "table".

Based on LSH, more similar documents have higher probabilities of being closer in a table. Given a large enough Q and B, it can be shown that signatures of similar documents will be at most B positions apart in at least one table with high probability. Since sorted order depends on the positions of bits, having multiple permutations of the same signature increases the chance of retrieving a similar pair.

A queue of size B is used to implement the idea above (Reduce method in Figure 2): in each table, we compare all signatures that are at most B positions away from each other. The reducer iterates over the signatures and keeps the last B in the queue (lines 13–17). At each iteration, the current signature is compared to all signatures in the queue, and hamming distances are emitted as output (lines 8–12).

4.3.2 Improved Algorithm

A drawback of the basic algorithm is the inability to increase the number of reducers, because each reduce group processes a single permuted list of all the signatures in the collection. In other words, in the reduce phase, only Q reducers can operate in parallel, which may be an underutilization in very large clusters that are commonly used for MapReduce. In this case, being able to increase the number of reducers arbitrarily can help the algorithm better scale.

We achieved this by modifying the previous algorithm in the following way. The map function is exactly the same, but the reduce step divides each table into an arbitrary number of consecutive blocks (Figure 3). Each of the Q reducers iterates over sorted signatures, stores them in a list (lines 9–11), and outputs the list when its size reaches a pre-specified size, M (lines 12–14). We denote each of these M-sized lists as a *chunk*, and refer to this as the *chunk generation* phase. The actual comparison and similarity computation is performed in an additional map-only MapReduce task (Figure 4), called the *detection* phase, where the input is the chunks and each mapper finds all similar pairs in one chunk.

```
Mapper
 1: method Map(docid $n$, signature $s$)
 2:     for all permute func $p_i \in [p_1, p_2, \ldots, p_Q]$ do
 3:         Emit($\langle i, s.\textsc{Permute}(p_i) \rangle, n$)
Reducer
 4: method Initialize
 5:     $docids \leftarrow$ new List
 6:     $sigs \leftarrow$ new List
 7:     $chunk \leftarrow$ new SignatureChunk
 8: method Reduce($\langle$permno $p$, sig $s\rangle$, docids $[n_1, n_2, \ldots]$)
 9:     for all docno $n \in [n_1, n_2, \ldots]$ do
10:         $sigs$.Add($s$)
11:         $docids$.Add($n$)
12:         if $sigs$.Size() $= M$ then
13:             $chunk$.Set($sigs, docids$)
14:             Emit($p, chunk$)
15:             $sigs \leftarrow sigs$.SubList($M - B + 1, M$)
16:             $docids \leftarrow docids$.SubList($M - B + 1, M$)
17: method Close
18:     $chunk$.Set($sigs, docids$)
19:     Emit($p, chunk$)
```

Figure 3: Pseudo-code of the *chunk generation* phase of the sliding window algorithm.

Only pairs that have a distance less than the threshold T are emitted by the algorithm. We use the docids to make sure only cross-lingual pairs are included in the output. In this phase, there can be as many mappers as there are chunks, which allows full utilization of computational resources.

Note that some comparisons will be missed at the boundary of chunks (e.g., the last element of a chunk is not compared to the first element of the next chunk). We solve this problem by appending the last B signatures of the previous chunk at the beginning of the current chunk (Figure 3, lines 15–16). This results in redundancy in the chunks emitted to disk (the last B signatures of a chunk are the same as the first B signatures of the next one), but the redundancy is negligible especially since $B \ll M$.

5. ANALYTICAL MODEL

In this section, we provide a theoretical analysis of our LSH algorithm, which provides a model for estimating effectiveness analytically. The starting point is Equation 1, which shows how random vectors are used to generate bit signatures. For any two vectors u and v, the probability that a single random projection h_r collides is

$$\Pr[h_r(u) = h_r(v)] = 1 - \frac{\theta(u, v)}{\pi} \tag{3}$$

The proof can be found in [5], which we omit here. The intuition is that the hyperplane defined by the random vector divides our collection of vectors into two disjoint sets, and the probability that any two vectors will land in the same set is determined by the angle θ between them, according to the relation above.

Let $S_0 = \{u | h_r(u) = 0\}$ and $S_1 = \{u | h_r(u) = 1\}$, corresponding to the subsets of our collection of vectors that share the same hash value. Given a vector u, we can compute $h_r(u)$ and perform a linear scan in the corresponding set to find similar vectors. Given that vectors u and v have

```
Mapper
method $\textsc{Map}$(permno $p$, signatureChunk $chunk$)
    for $i = 1$ to $(chunk.\textsc{Size}() - B)$ do        ▷ $B$ is the size of the sliding window
        for $j = (i + 1)$ to $(i + B)$ do        ▷ check that $i$ and $j$ are from different languages
            $distance \leftarrow chunk.\textsc{Signature}(i).\textsc{Distance}(chunk.\textsc{Signature}(j))$
            if $distance \leq T$ then        ▷ detects similar pairs, where $T$ is the distance threshold
                $\textsc{Emit}(\langle chunk.\textsc{Docid}(j), chunk.\textsc{Docid}(i)\rangle, distance)$
```

Figure 4: Pseudo-code of the *detection* phase of the sliding window algorithm.

cosine similarity t, the probability that they are in the same subset is given by $1 - \cos^{-1}(t)/\pi$.

We can extend this basic approach by selecting n random vectors $\{r_1, r_2, ...r_n\}$ and construct corresponding hash functions $\{h_{r_1}(u), h_{r_2}(u), ...h_{r_n}(u)\}$. Using these hash functions we can divide up the collection into 2^n disjoint sets:

$$\begin{aligned}
&S_{000...0} = \{u | h_{r_1}(u) = 0 \wedge h_{r_2}(u) = 0 \wedge ...h_{r_n}(u) = 0\}\\
&S_{000...1} = \{u | h_{r_1}(u) = 0 \wedge h_{r_2}(u) = 0 \wedge ...h_{r_n}(u) = 1\}\\
&...\\
&S_{111...1} = \{u | h_{r_1}(u) = 1 \wedge h_{r_2}(u) = 1 \wedge ...h_{r_n}(u) = 1\}
\end{aligned}$$

Suppose we wish to find document vectors that are similar to u: we can apply the hash functions to determine the correct candidate set of vectors, and then perform a linear scan through all those candidates to compute the actual dot product. With n random projections, the probability that we'll find similar vectors becomes $(1 - \cos^{-1}(t)/\pi)^n$. More random projections reduce the number of comparisons we must make, but at the cost of compounding the error introduced by each random projection.

To alleviate this issue, we can repeat the entire process m times using m sets of n different random projections. The probability that we'll identify a valid similar pair becomes:

$$\begin{aligned}
&\Pr[u, v \text{ in same set in at least one trial} \mid \cos(u, v) = t]\\
&= 1 - \left[1 - \left[1 - \frac{\cos^{-1}(t)}{\pi}\right]^n\right]^m \qquad (4)
\end{aligned}$$

The above equation quantifies the error when searching for similar vectors using LSH. The second source of error is the hamming distance computation. As shown in Section 4.2, cosine similarity can be estimated from the hamming distance with Equation 2.

We use this similarity estimate to efficiently decide which documents to return during the linear scan of the candidate set. For a given vector u and similarity threshold τ, we calculate the corresponding hamming distance threshold T using Equation 2 and return all candidates with hamming distance less than or equal to T. The precision of this decision is given by the following:

$$\begin{aligned}
&\Pr[\text{hamming}(u, v) \leq T | \cos(u, v) = t \wedge u, v \text{ share } n\text{-bit prefix}]\\
&= \sum_{i=0}^{T} \binom{D-n}{i} \rho^i (1 - \rho)^{D-n-i}, \text{ where } \rho = \frac{\cos^{-1}(t)}{\pi} \qquad (5)
\end{aligned}$$

How does this relate to our sliding window algorithm? Let us first consider a variation of our algorithm. Suppose instead of applying a sliding window B over the sorted bit signatures, we performed a pairwise N^2 comparison of all bit signatures that share n prefix bits. Another way to think about this is to dynamically adjust B so that it encompasses only those signatures that share the prefix. In this case, the expected probability of extracting a pair of vectors with similarity t (hamming distance $\leq T$) is quantified by the product of Equations 4 and 5 above. The number of tables (permutations) is equal to the number of trials m and the prefix length is equal to the number of random projections n in the above analysis. The probability of successfully extracting a similar pair has contributions from two sources: LSH (the two documents sharing the same signature prefix) and accuracy of hamming distance.

In actuality, we scan a fixed window B. However, we approximate n to be between $\lfloor \log_2(C/B) \rfloor$ and $\lceil \log_2(C/B) \rceil$, where C is the total number of vectors in the collection. We use both estimates to obtain a range of the estimated recall values. Although this is merely a rough model, it provides us with a basis for analytically estimating effectiveness, which we demonstrate later.

6. EXPERIMENTAL EVALUATION

We evaluated our cross-lingual pairwise similarity algorithm on English and German Wikipedia, selected because they are the largest Wikipedia collections available and because significant amounts of parallel corpora exist for the language pair. We used the German Wikipedia dump from 1/31/2011, which contains 2.41m articles totaling 8.5 GB. We used the English Wikipedia dump from 1/15/2011, which contains 10.86m articles totaling 30.6 GB. For both collections we discarded redirect pages and stub articles.

Our system is implemented on top of Ivory, an open-source Hadoop toolkit for web-scale information retrieval [19]. German articles were projected into English document vectors, as described in Section 3. For translation probabilities, we trained a word alignment using the Berkeley Aligner[4] on the Europarl German-English corpus, containing 1.08m sentence pairs from European parliament speeches. For tokenization, we used the Java-based OpenNLP toolkit.[5] After projection, document vectors containing fewer than five terms were discarded. We arrived at 1.47m German articles and 3.44m English articles. In the next step, RP signatures were generated from document vectors. Finally, we ran the improved sliding window algorithm (Section 4.3.2) to extract all document pairs with cosine similarity above a specified threshold. We only output pairs in which one document is from German Wikipedia and the other is from English Wikipedia.

All experiments were conducted on a Hadoop cluster (running Cloudera's distribution, version 0.20.2+320) with 16 nodes, each having 2 quad-core 2.2 GHz Intel Nehalem Processors, 24 GB RAM, and three 2 TB drives.

6.1 Effectiveness vs. Efficiency

The parameters in our algorithm are: the length in bits of each signature (D), the number of tables (Q), the chunk size (M), the window size (B), and the hamming distance threshold (T). We fixed D to 1000 based on the discussion

[4] http://code.google.com/p/berkeleyaligner

[5] http://opennlp.sourceforge.net

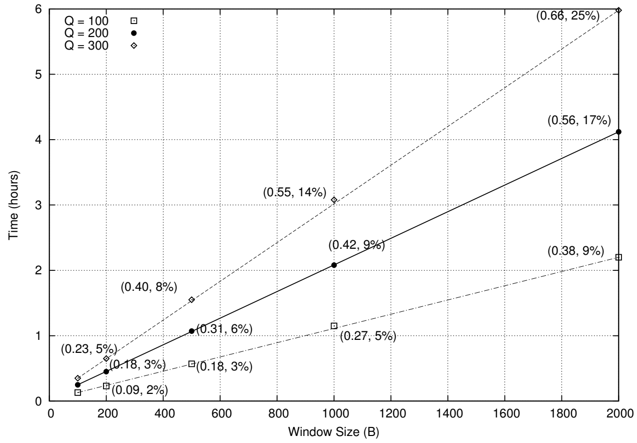

Figure 5: Running times of the detection phase of the sliding window LSH algorithm for $\tau = 0.3$, using 1000-bit signatures. Data points are annotated with recall and relative cost.

in Section 4.2, and fixed M to 0.49m so that each table is split into 10 chunks. For the rest of the parameters, we explored a number of choices and observed the changes in efficiency and effectiveness. Note that we need to run the preprocessing step and signature generation step only once for these experiments (this takes 1.67 hours). The *chunk generation* phase of the sliding window algorithm must be executed once for every value of Q (this takes 0.28, 0.53, and 0.75 hours, respectively, for $Q = 100, 200, 300$). The *detection* phase needs to be run for every different set of the remaining parameters, for which running times on the entire German-English Wikipedia collection are plotted in Figure 5 (varying window sizes for each value of Q). The regression lines clearly show that the algorithm scales linearly as Q and B increase ($R^2 > 0.999$ in all three cases). Note that the hamming distance threshold T does not affect the running time, but only determines which pairs are extracted; therefore, it is not included in this figure.

To assess effectiveness, we selected a sample of 1064 German Wikipedia articles. For every article, we calculated its true cosine similarity with all other documents in English Wikipedia and retained all pairs above threshold $\tau = 0.3$ to serve as the ground truth, informed by the similarity distribution in Figure 1. This corresponds to a hamming distance threshold of $T = 400$ for 1000-bit signatures, since $\cos(400\pi/1000) \sim 0.3$. Note that a hamming distance of 500 corresponds to no similarity at all, so we are looking for documents that may be very dissimilar. From this, we are able to measure the quality of the pairs extracted by our algorithm. We argue that recall is the most salient evaluation metric, i.e., the fraction of pairs in the ground truth set that are actually extracted, since the task is recall-oriented by definition. If precision is desired, one could always filter the extracted pairs and discard results that fall below the similarity threshold—the time necessary for this is negligible compared to the time for extracting the pairs to begin with. For instance, with parameters $Q = 300, B = 2000$, the number of extracted pairs is 64 million, 0.0013% of all possible cross-lingual pairs in the collection. This gives an idea of how much of the search space is filtered by the algorithm. Moreover, from an application point of view, it may not even be necessary to filter, since our τ threshold of 0.3 is somewhat arbitrary, and pairs with lower similarity scores may nevertheless be useful (see Figure 1).

The data points in Figure 5 are annotated with recall and a measure we call *relative cost*, defined as follows: for each condition, we can analytically compute the total number of similarity comparisons (i.e., in terms of hamming distance) necessary over the entire run. We can express this as a percentage of the total number of comparisons that a brute-force approach would require, which is the product of the size of the two collections: with 3.44m English articles and 1.47m German articles, this equals 5.05 trillion comparisons. The cost of the chunk generation process is also taken into account in these calculations (by translating that time into an equivalent in terms of number of comparisons). A relative cost lower than 100% means we're "saving work" by not considering pairs unlikely to be similar—the entire point of LSH. In essence, each pair in the graph delineates a point in the tradeoff space: the level of recall and efficiency that can be expected for a particular parameter setting.

To better understand these results, it is necessary to quantify the upper bound on effectiveness. Note that our algorithm has two sources of error: those introduced by the bit signatures (cf. Table 1) and those introduced by the sliding window algorithm (i.e., failure to find similar pairs within B). We can isolate the error introduced by the sliding window and compute an effectiveness upper bound by repeating the procedure that generated the ground truth, but using signatures instead. For every sample document's signature, we computed the hamming distance with all other signatures in English Wikipedia and extracted all pairs that have a distance less than $T = 400$. We then compared these pairs to the ground truth set and obtained 0.76 recall.[6] We see that the 0.04 absolute error in cosine similarity (cf. Table 1) introduced by 1000-bit random projections, which is negligible for other tasks that adopt high similarity thresholds (e.g., [28]), has a large impact for our task. Another way to understand this is that absolute error has an increasing impact as the similarity threshold is lowered (i.e., 0.04 absolute error is equal to 13% relative error when we are dealing with cosine similarity values of 0.3). The conclusion to draw from this analysis is that the upper bound on recall for our sliding window algorithm is not 1.0, but a more modest 0.76.

The adage "no free lunch" best characterizes our experimental results. There does not appear to be a single optimal solution to the cross-lingual pairwise similarity problem using LSH. Gains in efficiency inevitably come at a cost in effectiveness, and the extent to which it is worthwhile to trade off one for the other depends on application- and resource-specific constraints: i.e., how much recall is necessary, how much computational resources are available, etc. We provide a guide that helps the application developer locate a desirable operating point in the solution space.

We additionally explored the impact on upper bound effectiveness of different representations. Increasing the number of random projections produces longer and therefore more precise signatures, at the cost of linearly increasing running times. This is shown in Table 2, where we repeated the same experiment for 2000- and 3000-bit signatures and show the average time (in ns) to process a single pair. Increasing signature length, however, does not appear

[6] Based on micro-averaging, where the denominator is the sum of ground truth pairs across *all* topics.

Representation	Time (ns)	Precision	Recall
1000-bit	28	0.59	0.76
2000-bit	52	0.74	0.81
3000-bit	84	0.86	0.78
Doc Vector	450	-	-

Table 2: Comparing different representations: average time for a similarity comparison and the level of effectiveness achieved.

to have much of an impact on recall. As a reference, we also show results with the original document vector. Even with 3000 bits, hamming distance calculations are still more than 5 times faster than computing similarity directly on document vectors (which require floating point operations). Once again, there's no free lunch: representations that support faster similarity comparisons sacrifice fidelity.

6.2 Parameter Exploration

It would be desirable to more exhaustively explore the parameter space, beyond the experiments in Figure 5. The major impediment, however, is the long running times of the experiments. Nevertheless, it is possible to separately evaluate effectiveness and efficiency in a meaningful way. The amount of work required by our sliding window algorithm can be quantified by the number of comparisons required. This measure is convenient since it can be computed analytically without actually running experiments, and in some sense is better than (wall clock) running time since it abstracts over hardware differences and other natural variations. Since the results in Figure 5 confirm that our algorithm scales linearly, we can derive a straightforward mapping from number of comparisons to actual running time. To compute effectiveness, we can greatly speed up the experiments by considering only documents in the test set.

The results of separately evaluating effectiveness and efficiency are shown in Figure 6, where we vary the number of tables Q for $B = 200, 500, 1000$. Once again, this characterizes the tradeoff space by showing the recall and relative cost (labeled on the points) that is realized by a particular parameter setting.

Figure 6 also shows recall estimates based on our analytical model from Section 5. As previously discussed, we bound n by $\lfloor \log_2(C/B) \rfloor$ and $\lceil \log_2(C/B) \rceil$, where C is the total number of vectors in the collection: these are shown in the figure as dotted lines (but note overlaps). Despite the simplifications made in that model, we see that the estimates are fairly good. This rough model allows the application developer to tackle arbitrary collections, and *without performing any actual experiments*, estimate the effectiveness that can be achieved with different parameter settings. Combined with the ability to compute efficiency analytically, as we have shown above, the developer can characterize the tradeoff space with minimal effort. Such an analysis addresses a general weakness of LSH approaches: they present a bewildering number of parameters that need to be set, and provide little guidance for a developer approaching new problems. The ability to quantify efficiency and analytically estimate effectiveness provides a powerful tool for tackling the pairwise similarity problem.

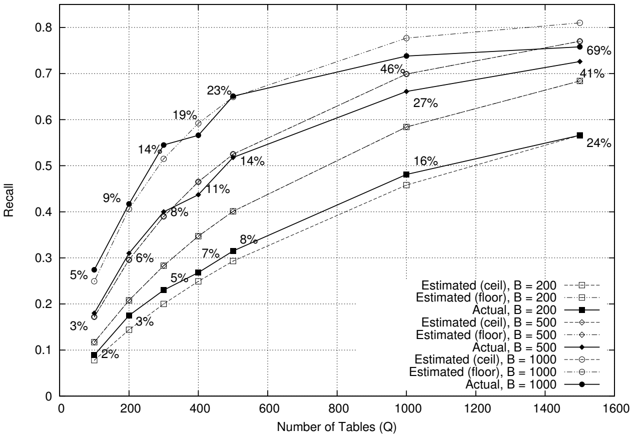

Figure 6: Effectiveness-efficiency tradeoffs of the sliding window LSH algorithm for $\tau = 0.3$, using 1000-bit signatures.

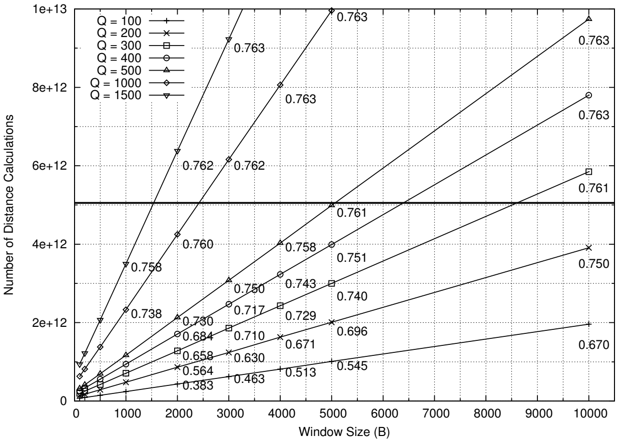

Figure 7: Number of distance calculations and corresponding recall values.

In Figure 7, we push our analysis even further with a wider setting of parameters, enabled by separating effectiveness and efficiency measures as above. Each data point is annotated with actual recall achieved. The thick horizontal line indicates the number of comparisons required for the brute-force approach. This particular analysis holds important implications for the type of underlying hardware that can be used to run our algorithm. Since the number of tables Q dictates the amount of parallelism that can be extracted, this suggests that larger clusters with more processing capacity may benefit from larger values of Q to increase cluster utilization. On the other hand, a smaller cluster with faster processors might benefit from larger values of B, since the size of the sliding window determines how fast an individual chunk is processed. Once again, there is no one-size-fits-all solution: the most optimal operating point in the tradeoff space will depend on the specifics of a particular application and the computational resources at hand.

One interesting observation is that the brute-force solution should not be so readily dismissed. In fact, to achieve very high levels of recall, the brute-force approach actually requires less work. This is shown in Figure 8: the x-axis plots relative cost, while on the y-axis we normalize effectiveness with respect to the recall upper bound to obtain a relative effectiveness measure. Each of the lines corresponds to a different setting of Q under different settings of B. This

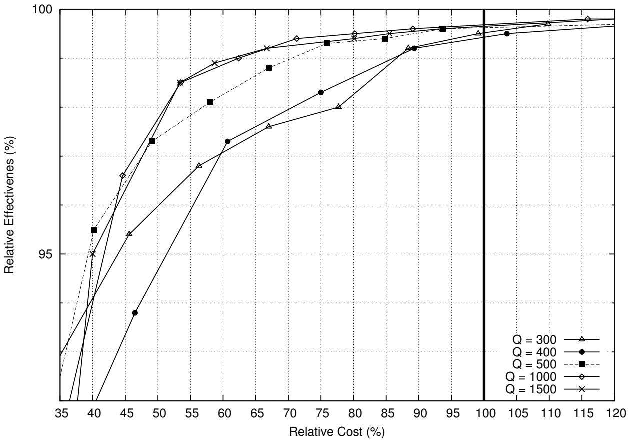

Figure 8: Relative effectiveness vs. relative cost of the sliding window algorithm

shows that if a relative effectiveness of greater than 99.5% is desired, it is more efficient to simply use the brute-force approach. On the other hand, the cost can be reduced significantly by trading off effectiveness. For instance, only 35% of the cost is required to achieve close to 93% relative effectiveness. The algorithm obtains 95% and 98% relative effectiveness with a cost about 60% and 50% less than the brute force approach, respectively. This analysis shows that $Q = 1000$ appears to be a good setting—but once again, there are many practical concerns that need to be taken into account. But inevitably, there is no free lunch, as efficiency can only be gained at the cost of effectiveness.

6.3 Error Analysis

Finally, we analyze the benefits and drawbacks of our end-to-end system, on the task of matching relevant cross-lingual documents in Wikipedia. This can be done quantitatively by comparing the algorithm's output to "interwiki" language links. These links are created manually by users, and are supposed to connect pages about the same entity across languages. However, there has been some work showing that these links are inaccurate [9]. Therefore, we find these links unsuitable for use as ground truth, and present the following results with this strong caveat.

To establish a reference on output quality, for each sample German article, we extracted the English article with the highest cosine similarity (using document vectors). The actual "interwiki" links are available from the source of each Wikipedia page, but only 401 of the sample articles had one listed in our downloaded version of Wikipedia. Among those links, our algorithm identified 130 of them (33%) as the most similar article.

A deeper, qualitative analysis shows that our proposed links are generally helpful, and in many cases they are more comprehensive than the existing Wikipedia links. Table 3 shows examples of article pairs that were assigned a cosine score above 0.3. For the German article titled "Metadaten", the algorithm correctly puts "Metadata" at the top of the list and includes two other related articles: "Semantic Web" and "File format". Although "Pierre Curie", "Marie Curie" and "Hélène Langevin-Joliot" are all physicists from the same family, and therefore similar, the algorithm places Marie above Pierre in terms of similarity for the German version. This may be due to the richer content of the article about Marie Curie. For the German article "Kirgisistan", 10 articles about recent events in Kyrgyzstan, its leaders and neighbors are in the output (we show the top five matching articles in Table 3). In short, linking cross-lingual articles via similarity has the potential to discover related articles, not just articles on exactly the same entity.

German article	Top similar articles in English
Metadaten	1. Metadata
	2. Semantic Web
	3. File format
Pierre Curie	1. Marie Curie
	2. Pierre Curie
	3. Hélène Langevin-Joliot
Kirgisistan	1. Kyrgyzstan
	2. Tulip Revolution
	3. 2010 Kyrgyzstani uprising
	4. 2010 South Kyrgyzstan riots
	5. Uzbekistan

Table 3: Examples of relevant German–English article pairs found by our algorithm.

It is interesting that our system was not able to return any similar article for 571 of the 1064 sample German articles. For the remainder, it may simply be the case that a similar article does not exist. This is true especially for articles about specific locations in Germany (e.g., "Dortmund-Aplerbeck"). After analyzing some of these cases by hand, we discovered that there are two common situations where the algorithm struggles. The first is when there is a large gap between the document lengths and contents of the articles in different languages. For instance, the German version of the article about the music show on German TV channel ZDF ("ZDF-Hitparade") is far more comprehensive than the English version. Since cosine similarity measures how similar documents are, it falls below 0.3 in cases like this. Secondly, our system has difficulty matching highly technical articles (e.g., "Farbeindringprüfung", English "Dye penetrant testing"), due to out-of-domain vocabulary: the vector projection has difficulty matching technical terms across languages. This could be solved with more data or special processing of named entities. We will address these issues as part of future work.

7. CONCLUSION AND FUTURE WORK

The cross-lingual pairwise similarity problem requires extracting pairs that have low cosine similarity values—a requirement imposed by the nature of the task. In this work, we explored an LSH-based approach to the problem and analyzed two different components of the solution: the signature generation process and the sliding window algorithm for extracting similar pairs. We showed that 1000-bit signature vectors with random projections are not sufficient to achieve anywhere close to perfect recall but this is the cost of a representation that is significantly faster to process. Our solution is implemented as a parallel, scalable MapReduce algorithm and demonstrated to scale linearly on the two largest Wikipedia collections. We experimentally and analytically quantified the effectiveness-efficiency tradeoff of the sliding window approach with respect to a multitude of parameters. This characterization provides a guide to the application developer in selecting the best operating point for a specific situation. A somewhat surprisingly finding is that a brute-force approach should not be readily dismissed as a viable solution, especially when high recall is desired.

Part of our future work is to improve the representation and translation of documents by including named entity recognition and using broader and larger vocabularies, as well as techniques from natural language processing. German word translations may be very noisy due to compounding and other morphological phenomena, so we plan on experimenting with other language pairs. Using learning methods to find better random projections has been shown to work well [27] and we are interested in adapting that approach to our system. As a logical next step, we would like to go beyond cross-lingual pairwise similarity at the document level to directly extract bilingual sentence pairs, and then use these as an additional source for training machine translation systems. Finally, we hope to show the scalability of our algorithm on even larger datasets.

8. ACKNOWLEDGMENTS

This work has been supported in part by DARPA contract HR0011-06-2-0001 (GALE); NSF under awards IIS-0836560, IIS-0916043, and CCF-1018625. Any opinions, findings, conclusions, or recommendations expressed are the authors' and do not necessarily reflect those of the sponsors. The last author is grateful to Esther and Kiri for their loving support and dedicates this work to Joshua and Jacob.

9. REFERENCES

[1] M. Anderka, B. Stein, and M. Potthast. Cross-language high similarity search. *ECIR*, 2010.

[2] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *CACM*, 51(1):117–122, 2008.

[3] R. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. *WWW*, 2007.

[4] A. Broder. On the resemblance and containment of documents. *SEQUENCES*, 1997.

[5] M. Charikar. Similarity estimation techniques from rounding algorithms. *STOC*, 2002.

[6] A. Chowdhury, O. Frieder, D. Grossman, and M. McCabe. Collection statistics for fast duplicate document detection. *ACM TOIS*, 20(2):171–191, 2002.

[7] O. Chum, J. Philbin, and A. Zisserman. Near Duplicate Image Detection: min-Hash and tf-idf Weighting. *British Machine Vision Conference*, 2008.

[8] K. Darwish and D. Oard. Probabilistic structured query methods. *SIGIR*, 2003.

[9] G. de Melo and G. Weikum. Untangling the cross-lingual link structure of Wikipedia. *ACL*, 2010.

[10] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *OSDI*, 2004.

[11] T. Elsayed, J. Lin, and D. Oard. Pairwise document similarity in large collections with MapReduce. *HLT*, 2008.

[12] M. Hadjieleftheriou, A. Chandel, N. Koudas, and D. Srivastava. Fast indexes and algorithms for set similarity selection queries. *ICDE*, 2008.

[13] M. Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. *SIGIR*, 2006.

[14] L. Huang, L. Wang, and X. Li. Achieving both high precision and high recall in near-duplicate detection. *CIKM*, 2008.

[15] T. Kiefer, P. Volk, and W. Lehner. Pairwise element computation with MapReduce. *HPDC*, 2010.

[16] P. Koehn. *Statistical Machine Translation*. Cambridge University Press, 2010.

[17] A. Kolcz, A. Chowdhury, and J. Alspector. Improved robustness of signature-based near-replica detection via lexicon randomization. *KDD*, 2004.

[18] J. Lin. Brute force and indexed approaches to pairwise document similarity comparisons with MapReduce. *SIGIR*, 2009.

[19] J. Lin, D. Metzler, T. Elsayed, and L. Wang. Of Ivory and Smurfs: Loxodontan MapReduce experiments for web search. *TREC*, 2009.

[20] G. Manku, A. Jain, and A. Das Sarma. Detecting near-duplicates for web crawling. *WWW*, 2007.

[21] H. Masuichi, R. Flournoy, S. Kaufmann, and S. Peters. A bootstrapping method for extracting bilingual text pairs. *COLING*, 2000.

[22] S. Matthews and T. Williams. MrsRF: An efficient MapReduce algorithm for analyzing large collections of evolutionary trees. *BMC Bioinformatics*, 11(Suppl 1):S15, 2010.

[23] C. Moretti, J. Bulosan, D. Thain, and P. Flynn. All-Pairs: An abstraction for data-intensive cloud computing. *IPDPS*, 2008.

[24] D. Munteanu and D. Marcu. Improving machine translation performance by exploiting non-parallel corpora. *Comp. Ling.*, 31(4):477–504, 2005.

[25] F. Och and H. Ney. A systematic comparison of various statistical alignment models. *Comp. Ling.*, 29(1):19–51, 2003.

[26] P. Pantel, E. Crestan, A. Borkovsky, A.-M. Popescu, and V. Vyas. Web-scale distributional similarity and entity set expansion. *EMNLP*, 2009.

[27] J. Platt, K. Toutanova, and W.-t. Yih. Translingual document representations from discriminative projections. *EMNLP*, 2010.

[28] D. Ravichandran, P. Pantel, and E. Hovy. Randomized algorithms and NLP: Using locality sensitive hash functions for high speed noun clustering. *ACL*, 2005.

[29] P. Resnik and N. Smith. The web as a parallel corpus. *Comp. Ling.*, 29(3):349–380, 2003.

[30] S. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. *TREC-3*, 1994.

[31] J. Smith, C. Quirk, and K. Toutanova. Extracting parallel sentences from comparable corpora using document level alignment. *HLT*, 2010.

[32] M. Theobald, J. Siddharth, and A. Paepcke. SpotSigs: robust and efficient near duplicate detection in large web collections. *SIGIR*, 2008.

[33] R. Vernica, M. Carey, and C. Li. Efficient parallel set-similarity joins using MapReduce. *SIGMOD*, 2010.

[34] S. Vogel, H. Ney, and C. Tillmann. HMM-based word alignment in statistical translation. *COLING*, 1996.

[35] H. Yang and J. Callan. Near-duplicate detection by instance-level constrained clustering. *SIGIR*, 2006.

[36] D. Zhang, J. Wang, D. Cai, and J. Lu. Self-taught hashing for fast similarity search. *SIGIR*, 2010.