An Efficient Method for Finding Paraphrases

Tagyoung Chung [*]

Kevin Knight [†]

University of Southern California

Collections of paraphrases have many uses in natural language applications. However, building such corpora may require many pairwise comparisons and becomes difficult at a large scale. This paper introduces a scalable method to extract paraphrases, greatly reducing such required comparisons.

1. Introduction

This paper explores a scalable method for extracting paraphrases from large corpora. There are several reasons to build paraphrase corpora. As such corpora become available, they can be effectively used to train paraphrase generators and paraphrase recognizers. These systems, in turn, can be used in applications like multi-document summarization (to help recognize and delete duplicate information), natural language generation (to diversify generated expressions), speech translation (to map spoken utterances onto synonymous canonical expressions whose translations are pre-stored), and others. There has been previous work on extracting paraphrases, e.g.(Barzilay and McKeown , 2001), but such work can only be used on a relatively small scale because of pairwise comparisons

* Information Sciences Institute, Marina del Rey, CA 90292

† Information Sciences Institute, Marina del Rey, CA 90292

that are required. In this work, we investigate a method that extracts paraphrases by making a single pass over a large text corpus.

2. Paraphrase Data

We look for paraphrases at the level of sentences and paragraphs. In order to test the precision and recall of our method, we constructed an English corpus for which we have an answer key of correct paraphrase pairings. Following previous work, we used two separate English translations of Jules Verne's *20,000 Leagues Under the Sea*. We aligned these manually at the paragraph level, then recorded the answer key of correct pairings.[1] In our experiments, we then scrambled the paragraphs, asked our paraphrase extractor to identify paragraph pairs, and compared those proposed pairs with the answer key.

In this section, we detail our data preparation. Translation A consists of 99,799 words, while translation B consists of 141,709 words. It turns out that translation A deletes much of the ponderous prose of the original, containing only 2,200 paragraphs versus translation B's 3,445 paragraphs. The alignment of A to B is not trivial, for example, the material in one 26-paragraph sequence from translation B is nowhere to be found in translation A.

To aid our manual alignment, we started with an automatic paragraph alignment, using the dynamic programming approach of (Gale and Church , 1991). Because of the noise in our data, the length-based matching method was not entirely satisfactory. Therefore, we added a word0based filter for deciding whether a paragraph in translation B was omitted in translation A. After checking all results by hand, we wound up with a corpus of 2,123 corresponding paragraphs.

Going over the corpus in this way gave us the chance to dig into the concept of

1 This aligned corpus is available at http://www.isi.edu/~knight.

paraphrasing. Many parallel paragraphs convey slightly different meanings. Other paraphrases involve different syntactic structures, word choices, style, and mood. Still others can be seen as paraphrases only in the particular context of use. For example:

> Translation 1: What are you saying?
>
> Translation 2: You think so?

These sentences are somewhat different, but legitimate paraphrases in context.

Here is another example of somewhat unexpected variation:

> Translation 1: ... the frigate lay abreast of Cabo Blanco, thirty miles to leeward of the coast of Patagonia.
>
> Translation 2: ... the frigate was abreast of Cape Blanc, thirty miles to leeward of the coast of Patagonia.

There are two very different paraphrases of the French 'Cap Blanc.' One translator uses the Spanish version, while the other converts the French halfway to English. Another example:

> Translation 1: You have right on your side and my arguments cannot stand up to yours.
>
> Translation 2: Your reasoning is against me.

The second translation is quite concise compared to the first, though the gist is the same. By contrast, consider this example:

> Translation 1: During the magnificent evening of June 25 ...
>
> Translation 2: One magnificent evening, the 30th of July ...

Clearly, some amount of true noise remains in the data. In general, matching (or mismatching) numbers are not reliable indicators of paraphrases, as one translation uses the metric system, while the other uses the English system.

3. Paraphrase Extraction Algorithm

As stated above, our input is a large collection of scrambled paragraphs, and we want to output proposed paraphrase pairs. The simplest way to look for paraphrases is to look for lexical similarities between pairs of paragraphs. If two paragraphs share enough words, then it is possible that they are paraphrases of each other. One measure of lexical similarity is the Jaccard coefficient. Let A be a paragraph containing the set of words S_A (duplicates removed), and let B be a paragraph containing a set of words S_B. Then:

$$Sim_J(A,B) = \frac{|S_A \cap S_B|}{|S_A \cup S_B|}$$

It is possible to compare every pair of paragraphs using this measure, and then sort the results according to the measure. However, for large corpora, the O(n^2) complexity is too high. Therefore, we employ a version of the randomized algorithm of (Broder , 2000), originally developed for finding near-duplicate web documents.

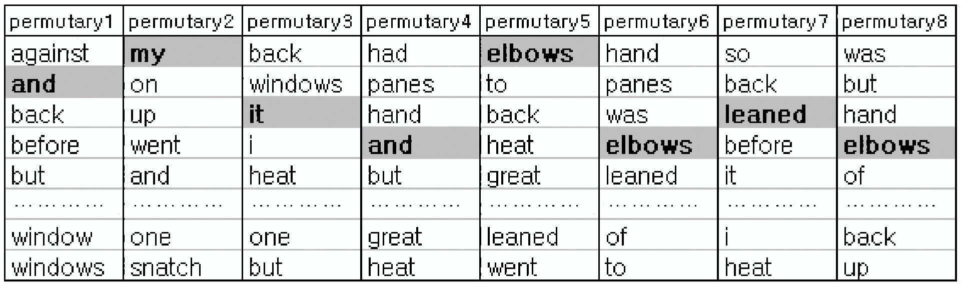
The simplest form of this algorithm is to reduce each paragraph to a hash key, such that hash keys of paraphrases collide. In this way, we can make a single pass over the corpus, recording collisions and outputting proposed matching pairs. While this basic idea works perfectly for duplicate detection, it is too simplistic for paraphrase detection, because collision is all or nothing. Even in a far-out scenario that employed a perfect semantic representation as a hash key, we have already seen that paraphrases are often not exact semantic matches. In fact, it is quite difficult in general to pin down what counts as a paraphrase, and the most desirable definition will vary depending on the larger application. For our purposes, we take the given answer key as a gold standard.

Inspired by (Ravichandran, Pantel, and Hovy, 2005), and following the work of (Broder , 2000), we instead create m hash keys for each paragraph, each capturing one aspect of the paragraph's contents. The paragraph is then hashed into m separate hash tables, according to those keys. Two paragraphs are considered similar if they consistently collide in many hash tables. The hash key creation method, which we described below, has the notable feature that as m gets very large, the percentage of collisions converges to the Jaccard coefficient. Therefore, this method can be considered a fast approximation for computing Jaccard coefficients. Here is the algorithm:

1. Create a dictionary of all words in the corpus.
2. Shuffle this dictionary m times to create m randomly permuted dictionaries (we call these *permutaries*). Create m corresponding hash tables.
3. For each paragraph p, and for each $k = 1$ to m, create a hash key consisting of that word in p which is *closest to the front of the kth permutary.*
4. Hash each paragraph ID into m separate hash tables using those keys. At this point, some of the hash buckets will contain multiple paragraph IDs, due to collisions.
5. For every hash bucket with multiple entries as described above, place each pair of entries into a new hash table H. The first time a pair goes into H, it is stored with a count of $1/m$. Each additional time, the count is incremented by $1/m$.
6. Now H contains a subset of paragraph pairs and their estimated Jaccard coefficients. Make a final pass through H, printing pairs with estimated Jaccard coefficients that exceed a given threshold.

Figures 1–3 show examples of hash keys of three sentences with eight permutaries. Notice that sentences 1 and 2 collide in five out of eight permutaries because they are paraphrases of each other, while sentences 1 and 3 only collide once.

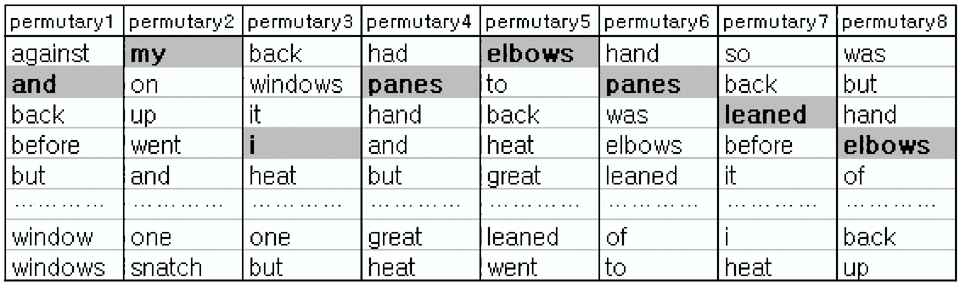
The Jaccard coefficients estimated by the above method will not be the same as the exact Jaccard coefficients one would get from comparing paragraphs pair by pair.



permutary1	permutary2	permutary3	permutary4	permutary5	permutary6	permutary7	permutary8
against	**my**	back	had	**elbows**	hand	so	was
and	on	windows	panes	to	panes	back	but
back	up	**it**	hand	back	was	**leaned**	hand
before	went	i	**and**	heat	**elbows**	before	**elbows**
but	and	heat	but	great	leaned	it	of
…………	…………	…………	…………	…………	…………	…………	…………
window	one	one	great	leaned	of	i	back
windows	snatch	but	heat	went	to	heat	up

Figure 1

Sentence 1 hash key: and my it and elbows elbows leaned elbows



permutary1	permutary2	permutary3	permutary4	permutary5	permutary6	permutary7	permutary8
against	**my**	back	had	**elbows**	hand	so	was
and	on	windows	**panes**	to	**panes**	back	but
back	up	it	hand	back	was	**leaned**	hand
before	went	**i**	and	heat	elbows	before	**elbows**
but	and	heat	but	great	leaned	it	of
…………	…………	…………	…………	…………	…………	…………	…………
window	one	one	great	leaned	of	i	back
windows	snatch	but	heat	went	to	heat	up

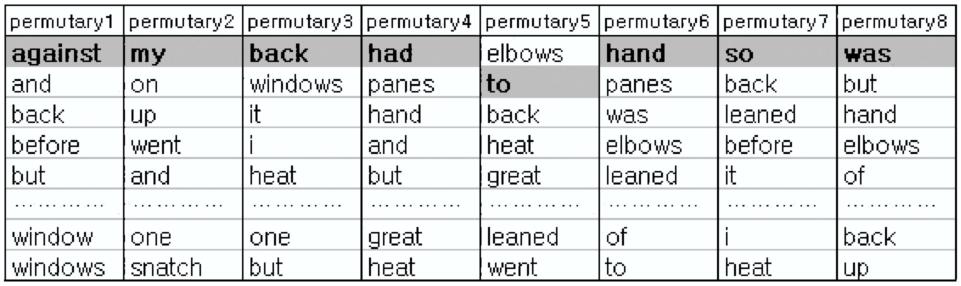
Figure 2

Sentence 2 hash key: and my i panes elbows panes leaned elbows

How well do the estimated Jaccard coefficients approximate the real thing, in practice? In Figure 4, paragraph pairs are plotted as points according to each pair's true Jaccard coefficient (y axis) versus its estimated Jaccard coefficient (using the method above, with 128 permutaries). Given a true Jaccard of 0.5, we see that the estimated Jaccard may be anywhere from 0.4 to 0.6. Figure 5 shows that the situation is improved when we employ 1024 permutaries, though at a cost of additional runtime.

4. Experimental Results

In our experiments, we use both true pairwise Jaccard and approximate Jaccard to locate proposed paraphrase pairs. We evaluate the proposed pairs against the answer key, recording how many of the proposed pairs are found in the answer key (precision), and how many of the answer key pairs are among those proposed (recall). We can trade off



permutary1	permutary2	permutary3	permutary4	permutary5	permutary6	permutary7	permutary8
against	**my**	**back**	**had**	elbows	**hand**	**so**	**was**
and	on	windows	panes	**to**	panes	back	but
back	up	it	hand	back	was	leaned	hand
before	went	i	and	heat	elbows	before	elbows
but	and	heat	but	great	leaned	it	of
…………	…………	…………	…………	…………	…………	…………	…………
window	one	one	great	leaned	of	i	back
windows	snatch	but	heat	went	to	heat	up

Figure 3

Sentence 3 hash key: against my back had to hand so was

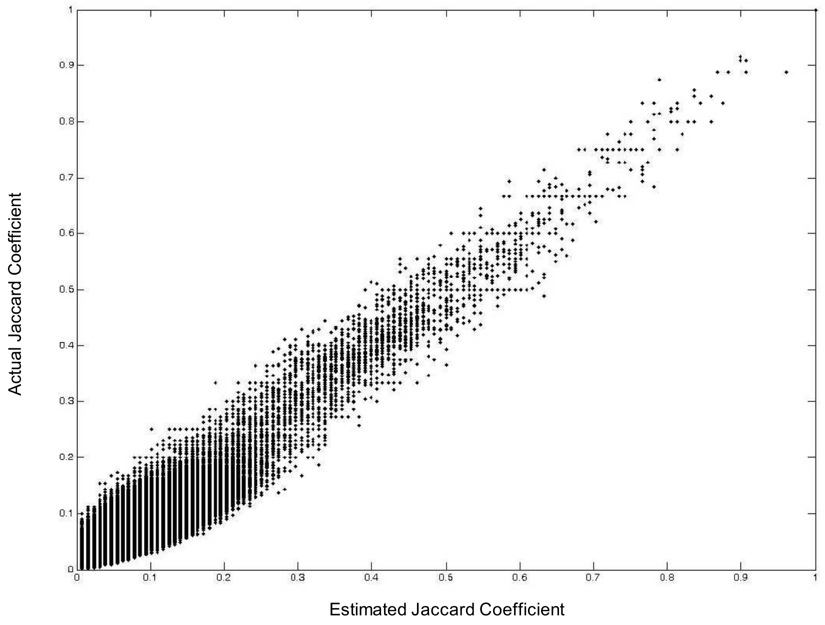


Figure 4

Jaccard coefficient vs. Estimated Jaccard coefficient with 128 permutaries.

Each point represents a pair of paragraphs.

precision against recall by setting the threshold mentioned in step 6 above—if the Jaccard threshold is very high, the algorithm will make only a few highly precise proposals, while if the threshold is low, many more proposals will be made (some right, some wrong).

Figure 6 shows the results for the O(n^2) method of computing true Jaccard coefficients for all of paragraph pairs. The x-axis shows various thresholds, and the y-axis

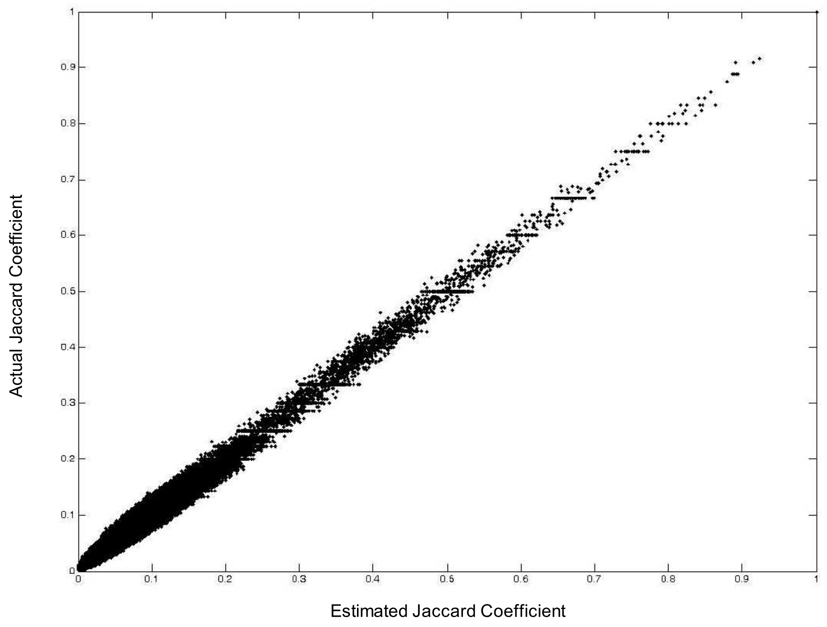


Figure 5

Jaccard coefficient vs. Estimated Jaccard coefficient with 1024 permutaries.

Each point represents a pair of paragraphs.

records precision, recall, and f-measure. If the threshold is set at 0.1, the algorithm identifies nearly all true paraphrase pairs, but its precision is very low. If the threshold is set to just above 0.5, then precision goes to 95%, while recall falls to 30%. A threshold of 0.33 optimizes f-measure at 75%.

Figure 7 and Figure 8 show results for the single pass approximate Jaccard method, using 16 and 64 permutaries respectively. In Figure 7, we see a peak f-measure of only 47%, which is far below 75%. Figure 8 shows that quadrupling the number of permutaries raises the peak f measure to 67%, at the cost of more CPU time. Another thing to notice is that with 64 permutaries and a Jaccard threshold of 1.0, all proposed pairs are true pairs (precision = 100%). However, with 16 permutaries, some false pairs sneak in with an overestimated Jaccard coefficient of 1.0.

What happens if we use more permutaries? Figure 9 shows results for 256 permutaries. We notice that the successive graphs ultimately converge to results similar to the $O(n^2)$ pairwise true Jaccard result of Figure 6.

The approximate, single pass method scales to larger corpora much more easily than the $O(n^2)$ method. Figure 10 compares running times of these two methods (y-axis), as we increase the number of paragraphs in the corpus (x-axis). To see how the methods scale beyond our *20,000 Leagues* corpus, we added several thousand dummy paragraphs to the collection. As we add more paragraphs, the gap between the two methods quickly renders the $O(n^2)$ method totally impractical.

In our experiments, we also studied the effect of excluding stop words from the permutaries. All of the above results were obtained by eliminating punctuation, determiners, and *'s*. This was not strictly necessary in the case of the $O(n^2)$ method, but was important for good results in the single pass method.

We also experimented with restricting the system to propose at most one paraphrase per paragraph. In this case, the proposed pairs are sorted by Jaccard, then enumerated in

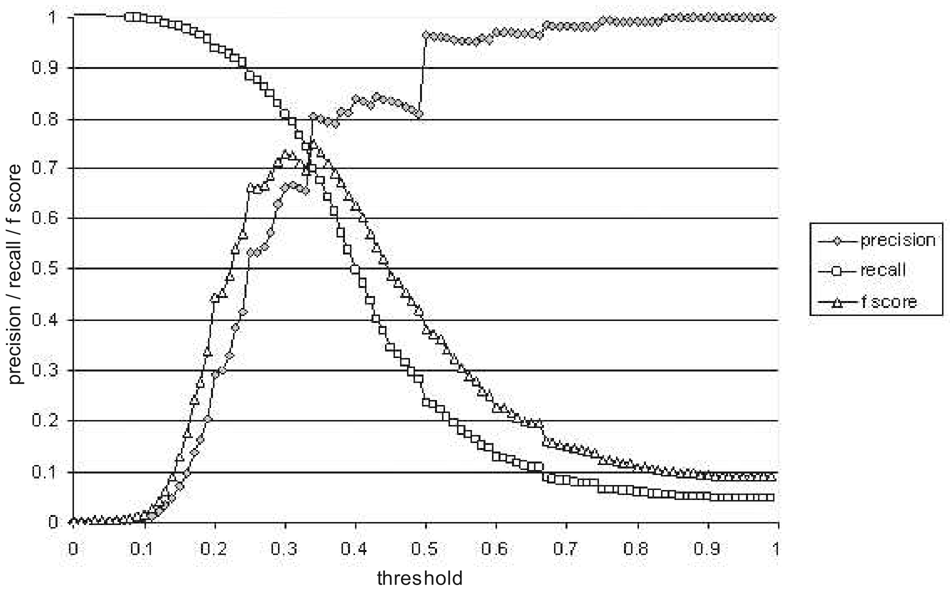


Figure 6

Real Jaccard coefficient - precision, recall and f score graph

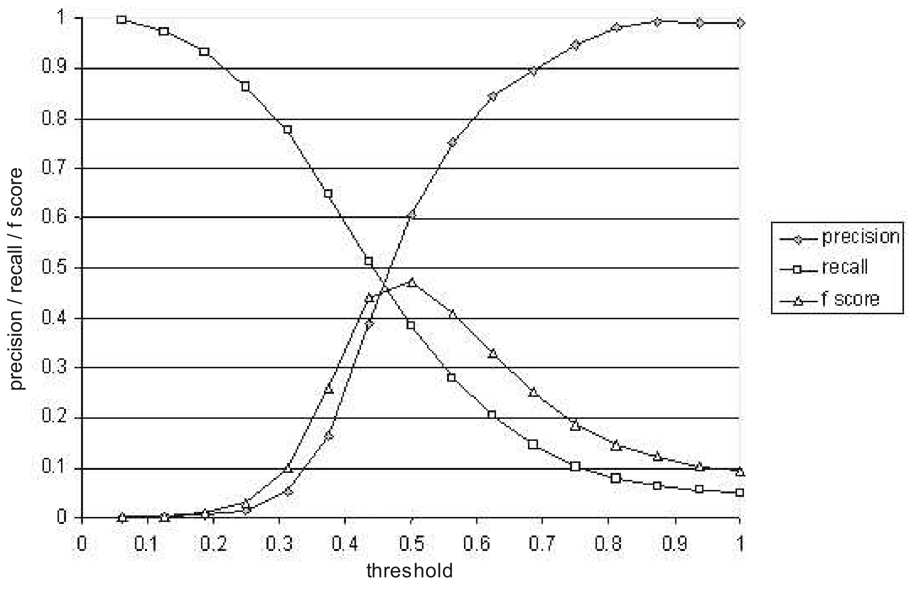


Figure 7

Estimated Jaccard coefficient with 16 permutaries

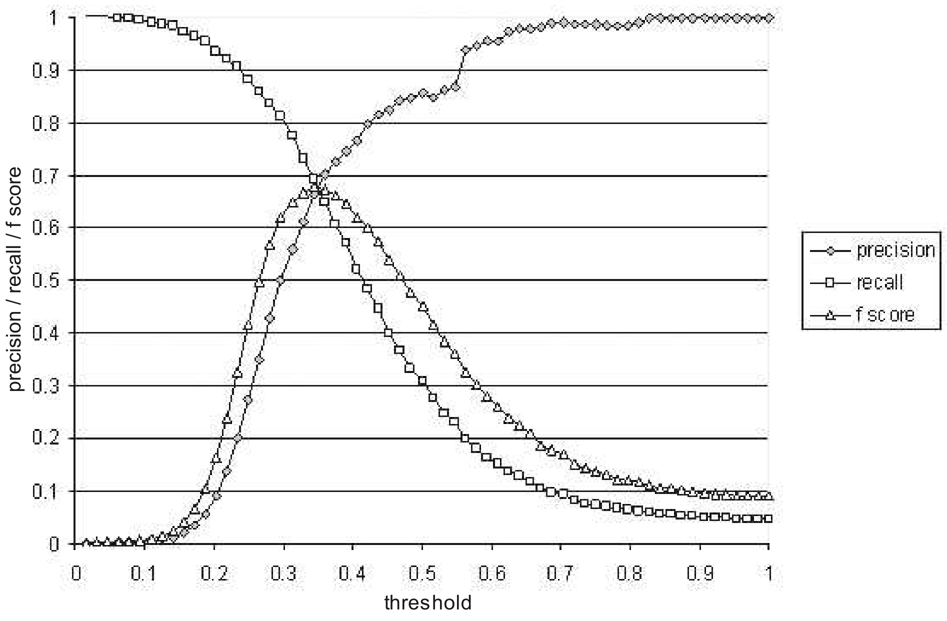


Figure 8

Estimated Jaccard coefficient with 64 permutaries

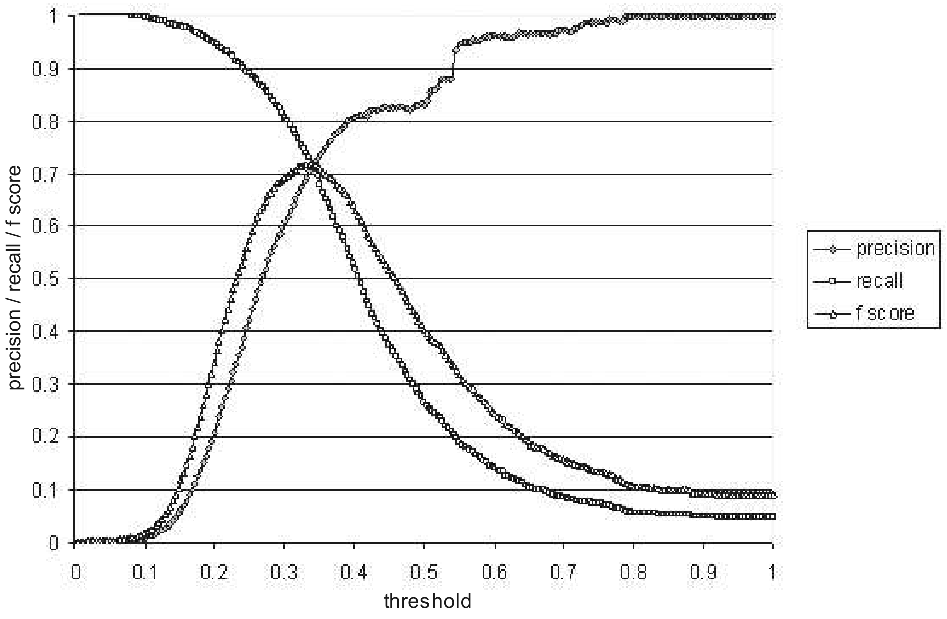


Figure 9

Estimated Jaccard coefficient with 256 permutaries

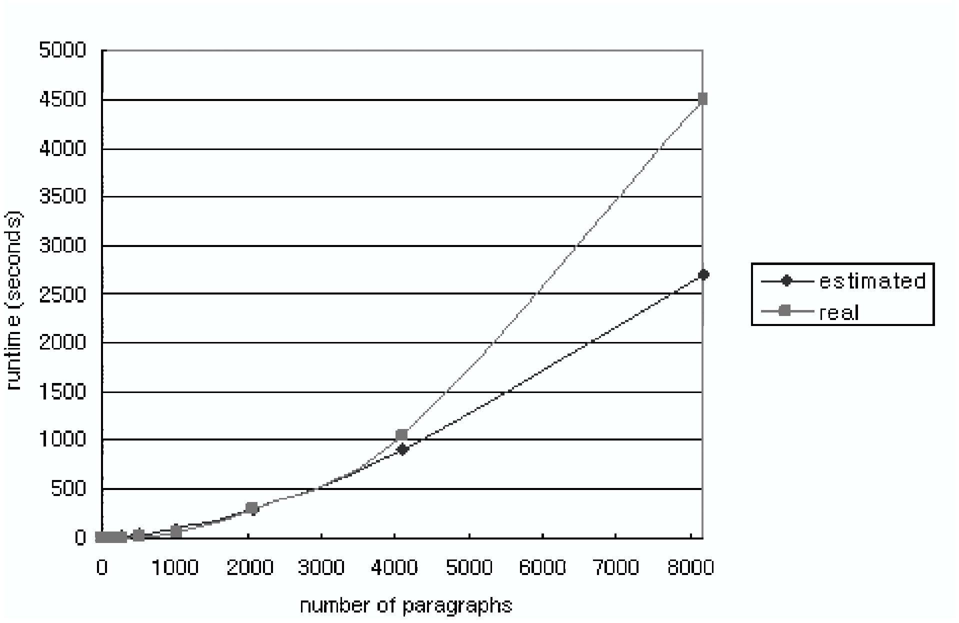


Figure 10

Runtime comparison between calculating the Jaccard coefficient exactly and estimating it.

order. When the system proposes a pair, it remembers the document IDs and skips over any subsequent pairs involving those IDs. This greedy assignment method is satisfactory, achieving a precision of 94% for the O(n^2) method and 92% for the single pass method. However, we expect that in many situations multiple paraphrases will exist, and that this greedy method will not be tenable.

5. Conclusion

In this paper, we have shown that the Jaccard coefficient is a useful and accurate tool for identifying paraphrases. We also showed that a method based on (Broder , 2000) can significantly speed up the identification of paraphrases in large corpora. In future work, we would like to apply the method to larger corpora, in a larger number of languages, and to develop evaluation techniques that can be applied in those scenarios.

References

Barzilay, R. and McKeown, R. K. 2001. Extracting Paraphrases from a Parallel Corpus. *In Proceedings of the Conference of the European Chapter of the Association for Computational Linguistics*, Toulouse.

Broder, A. Z. 2000. Identifying and Filtering Near-Duplicate Documents, *In Combinatorial Pattern Matching, 11th Annual Symposium*, pp. 1–10.

Gale, W. and Church, K. W. 1991. A Program for Aligning Sentences in Bilingual Corpora, *In Proceedings of the Annual Meeting of the Association for Computational Linguistics.*

Ravichandran, D., Pantel, P., and Hovy, E. 2005. Randomized Algorithms and NLP: Using Locality Sensitive Hash Function for High Speed Noun Clustering, Corpora, *In Proceedings of the Annual Meeting of the Association for Computational Linguistics.*