Efficient Searching Top-k Semantic Similar Words

Zhenglu Yang and **Masaru Kitsuregawa**
Institute of Industrial Science
The University of Tokyo, Japan
{yangzl, kitsure}@tkl.iis.u-tokyo.ac.jp

Abstract

Measuring the semantic meaning between words is an important issue because it is the basis for many applications, such as word sense disambiguation, document summarization, and so forth. Although it has been explored for several decades, most of the studies focus on improving the effectiveness of the problem, i.e., precision and recall. In this paper, we propose to address the efficiency issue, that given a collection of words, how to efficiently discover the top-k most semantic similar words to the query. This issue is very important for real applications yet the existing state-of-the-art strategies cannot satisfy users with reasonable performance. Efficient strategies on searching top-k semantic similar words are proposed. We provide an extensive comparative experimental evaluation demonstrating the advantages of the introduced strategies over the state-of-the-art approaches.

1 Introduction

Searching semantic similar words is an important issue because it involves many applications, such as query suggestion, word disambiguation, machine translation, and so forth. From a given collection of words, such queries ask for those words that are most (i.e., top-k) semantically similar to a given one.

Intuitively, the problem can be solved by firstly measuring the semantic similarity score between the query and each word in the collection using the state-of-the-art techniques [Resnik, 1995; Turney, 2001; Li *et al.*, 2003; Budanitsky and Hirst, 2006; Bollegala *et al.*, 2007; Tsatsaronis *et al.*, 2010], and then sorting them with regard to the score, and finally extracting those top-k ones. However, the scale of the problem has dramatically increased and prevented existing strategies from conducting on large volume of data (i.e., the Web). Note that almost all the previous work focus on improving the effectiveness of the problem (i.e., precision and recall) yet this paper is the first study in the literature on addressing the efficiency issue, which is rather challenging especially when conducting on large datasets. Another issue is that most of the previous works are rooted in a threshold-based framework and the similarity threshold is difficult to predefine by common users because many answers may be returned if the value is too small and there may be no answers if the value is too large. Therefore, searching for the top-k most similar words is an interesting problem, as will be tackled in this paper.

The issue of finding similar words can be traced back to the 1940s [Shannon, 1948] in the theory field where n-gram was first introduced. Two words are considered similar to each other if they have enough common subsequences (i.e., n-grams). While there are quite a few works on studying and extending n-gram based strategies [Ukkonen, 1992] and they have been successfully applied in some applications such as spell checking, this line of work only takes into account the syntax of words and ignore the semantic meaning.

To remedy this problem, extensive studies have been explored and they can be classified into three main groups: (1) knowledge-based[1] strategies [Rada *et al.*, 1989; Resnik, 1995]; (2) corpus-based strategies [Turney, 2001]; and (3) hybrid methods [Bollegala *et al.*, 2007].

Due to the large datasets we may have, there are two main issues existing in the top-k semantic similar word extraction: (1) **progressiveness** [Hellerstein *et al.*, 1999]. The first results should be output to the user almost instantly, and the algorithm should produce more and more results the longer the execution time (i.e., first the top-1, then the top-2, and finally the top-k answer); and (2) **inefficiency on similarity measuring**. Evaluating the similarities of all candidate pairs is time consuming. This issue becomes even worse if more features are taken into account. In this work we aim to address these two challenges.

Materializing all the similarities of word pairs in the preprocessing may (partially) tackle the mentioned issues [Pantel *et al.*, 2009]. However, the difficulty is that the data (e.g., the Web) is frequently updated. Re-computing and storing all the similarities is space and time consuming. Moreover, it is possible that new similarity metric will be introduced. Therefore, computing the similarities on-the-fly seems to be a practical solution. Furthermore, it is impossible to materialize the string similarity beforehand. The reason is that we cannot predict a user's query, that it may be never seen before (i.e., misspelled word).

It should be noted that there exists some other optimization

[1]It is also called dictionary-based or lexicon-based.

strategies (e.g., caching, parallel computing). However, all of these methods require specific techniques (e.g., appropriate caching algorithm) and thus, the extension to these strategies is out of the scope of this paper.

The contributions of this paper are as follows:

- We propose to tackle the efficiency issue of searching top-k semantic similar words, which is different from most previous works that concern on the effectiveness aspect. The comprehensive similarity employed in this paper is composed of three representative kinds of measures. More similarities can be taken into account for the framework, as will be discussed in this paper.
- We propose efficient strategies to extract the top-k results. For each similarity measure, we introduce a corresponding best first search method to minimize the number of candidates to be evaluated. A rank aggregation approach is presented to optimally obtain the top-k results when assembling the features.
- We conduct comprehensive experiments on two real datasets, and evaluate the query efficiency of the proposed strategies in terms of several aspects. The results show that the proposed strategies exhibit a superior performance compared with the state-of-the-art approaches (i.e., more than two orders of magnitude improvement).

2 Problem Statement and Analysis

We consider the top-k semantic similar word query on a given collection of words W. Formally, for a query word Q, finding a set of k words R in W most similar to Q, that is, $\forall r \in R$ and $\forall s \in (W - R)$ will yield $sim(Q, r) \geq sim(Q, s)$.

To measure the similarity $sim(Q, P)$ between two words Q and P, we apply the state-of-the-art strategy by assembling multiple similarity metric features together [Li *et al.*, 2003; Bollegala *et al.*, 2007]. Because we focus on tackling the efficiency issue in this paper, for simplicity we select three representative features from the main categories in word similarity measuring literature.

- Knowledge-based Similarity: Word dictionaries, e.g., WordNet [Miller, 1995], have been acted as the knowledge base for text processing. The words are linked manually with labels and organized in a taxonomy manner (i.e., tree). An intuitive idea on measuring similarity is that words are similar to each other if the shortest path between them is small [Rada *et al.*, 1989]. This edge counting strategy has been extended later by taking into account the depth [Wu and Palmer, 1994; Leacock and Chodorow, 1998], semantic density [Jiang and Conrath, 1997], and information content [Resnik, 1995] of the words in the knowledge base. In this paper, we consider a representative metric, **Leacock & Chodorow** [Leacock and Chodorow, 1998], defined as:

$$Sim_{lch}(w_1, w_2) = -ln \frac{length(w_1, w_2)}{2 * D} \tag{1}$$

  where $length$ is the length of the shortest path between two concepts (that the two words w_1 and w_2 belong to) using node-counting, and D is the maximum depth of the taxonomy.

- Corpus-based Similarity: Corpus-based measures of word semantic similarity try to recognize the degree of similarity between words using large corpora. The intuitive idea is that two words are similar to each other if they co-occur frequently in the large corpus. There are several kinds of strategies: PMI-IR (Pointwise Mutual Information-Information Retrieval) [Turney, 2001] uses web search engine (i.e., AltaVista) to gather the existence information from the Web; LSA (Latent Semantic Analysis) ([Deerwester *et al.*, 1990]) applies Singular Value Decomposition (SVD) to analyze the statistical information between words and then further obtain their semantic relationship. Some other strategies include log-likelihood, chi-square, MI (Mutual Information), and so forth. In this paper, we choose PMI-IR as one of the representative similarity measures between two words, w_1 and w_2, defined as:

$$Sim_{pmi-ir}(w_1, w_2) = log_2 \frac{pagecount(w_1 \cap w_2) * WebSize}{pagecount(w_1) * pagecount(w_2)} \tag{2}$$

  where $pagecount(w_i)$ is the number of documents retrieved when w_i is submitted to the search engine (i.e., Google).

- String similarity: String similarity measures the difference of syntax between words. An intuitive idea is that two words are similar to each other if they have enough common subsequences (i.e., n-gram [Shannon, 1948]). We focus on a representative string similarity measure, i.e., edit distance, denoted as $Sim_{ed}(w_1, w_2)$.

2.1 A General Framework

A general framework of searching top-k similar words is illustrated in Fig. 1, with a concrete example presented to ground our discussion. The query word and each candidate in the collection are sent to the three modules (i.e. knowledge bases, web search engine, and string similarity evaluator), respectively, to obtain the corresponding similarity score. Then, the scores from different modules are normalized, assembled, and ranked, resulting in the final ranked lists [Li *et al.*, 2003; Bollegala *et al.*, 2007].

To normalize the scores, we have $\widetilde{Sim}_{lch}(Q, P) = \frac{Sim_{lch}(Q,P)}{ln(2*D_{max})}$, where D_{max}[2] denotes the maximum value of D for all the taxonomies of the collection. The PMI-IR similarity is normalized as $\widetilde{Sim}_{pmi-ir} = \frac{Sim_{pmi-ir}}{log_2 \frac{WebSize}{pagecount_{min}}}$, where $pagecount_{min}(P)$[3] denotes the minimum value of $pagecount$ for words in the collection. $WebSize$ is set to 10^{12} according to the report of Google[4]. The edit distance is normalized as $\widetilde{Sim}_{ed}(Q, P) = 1 - \frac{Sim_{ed}(Q,P)}{L_{max}}$, where L_{max}[5] is the maximum word length in the collection.

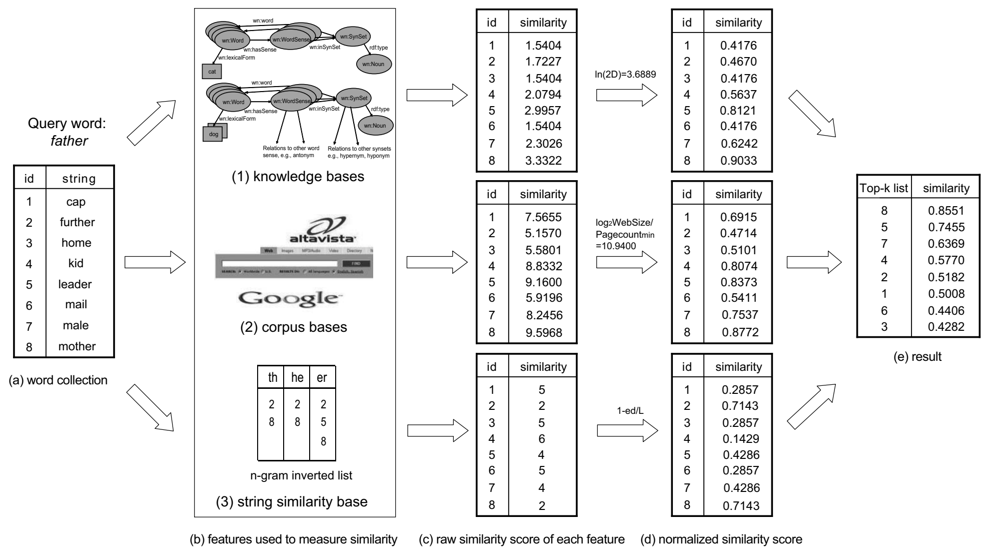
The assembling strategy follows the state-of-the-art approaches on measuring semantic similarity [Li *et al.*, 2003;

[2]For this concrete example, by querying WordNet D_{max}=20.

[3]We have $pagecount_{min}(P)$=$5.09 * 10^8$ by querying Google.

[4]http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html

[5]For this example, the value of L_{max} is 7.


Query word:
father
wn:word
wn:Word
wn:WordSense
wn:SynSet
wn:hasSense
wn:inSynSet
rdf:type
wn:Noun
wn:lexicalForm
cat
dog
Relations to other word sense, e.g., antonym
Relations to other synsets e.g., hypernym, hyponym
(1) knowledge bases
altavista
Google
(2) corpus bases
th
he
er
n-gram inverted list
(3) string similarity base
ln(2D)=3.6889
log2WebSize/Pagecountmin =10.9400
1-ed/L
(a) word collection
(b) features used to measure similarity
(c) raw similarity score of each feature
(d) normalized similarity score
(e) result

id	string
1	cap
2	further
3	home
4	kid
5	leader
6	mail
7	male
8	mother

th	he	er
2	2	2
8	8	5
		8

id	similarity
1	1.5404
2	1.7227
3	1.5404
4	2.0794
5	2.9957
6	1.5404
7	2.3026
8	3.3322

id	similarity
1	7.5655
2	5.1570
3	5.5801
4	8.8332
5	9.1600
6	5.9196
7	8.2456
8	9.5968

id	similarity
1	5
2	2
3	5
4	6
5	4
6	5
7	4
8	2

id	similarity
1	0.4176
2	0.4670
3	0.4176
4	0.5637
5	0.8121
6	0.4176
7	0.6242
8	0.9033

id	similarity
1	0.6915
2	0.4714
3	0.5101
4	0.8074
5	0.8373
6	0.5411
7	0.7537
8	0.8772

id	similarity
1	0.2857
2	0.7143
3	0.2857
4	0.1429
5	0.4286
6	0.2857
7	0.4286
8	0.7143

Top-k list	similarity
8	0.8551
5	0.7455
7	0.6369
4	0.5770
2	0.5182
1	0.5008
6	0.4406
3	0.4282

Figure 1: A general framework for searching top-k semantic strings

Bollegala *et al.*, 2007]:

$$\widetilde{Sim}(Q,P)=\sum_{i=1}^{n} w_i \widetilde{Sim_i}(Q,P) \tag{3}$$

where $\widetilde{Sim_i}$ represents a specific feature of similarity measure, w_i denotes the corresponding weight of the feature, and n is the total number of features taken into account.

Due to lack of large labeled data, the existing state-of-the-art assembling techniques on semantic similarity measure commonly set the weight values arbitrarily [Li *et al.*, 2003]. We apply the same strategy by emphasizing the effect of semantic similarity (i.e., w_{lch}=w_{pmi-ir}=$2w_{ed}$) and $\sum_{i=1}^{n} w_i$=1. While obtaining optimal values of these weights is certainly an interesting issue, it is out of the scope of this paper. We argue that this work is orthogonal to the existing effectiveness-oriented studies in a complementary manner.

For the example as illustrated in Fig. 1, all the eight candidates need to be evaluated with the query $father$ and the top-3 semantic similar words are $mother$, $leader$, and $male$.

3 Proposed Approaches

We first introduce best first search strategies for different similarity measures in Section 3.1-3.3, and then present a rank aggregation algorithm to assemble the features in Section 3.4.

3.1 Knowledge based Feature

WordNet is employed as the knowledge base in this paper. The traditional strategy is to measure the similarity of each candidate word and the query, by traversing the topology of WordNet. However, it is inefficient because every candidate needs to be tested. Fig. 2 illustrates the example aforementioned, where every pair of red word (query) and blue word (candidate) is evaluated.

We propose a best first search strategy to efficiently obtain the top-k similar words. We first tackle the issue when the query and the candidate exist in the same taxonomy (e.g.,

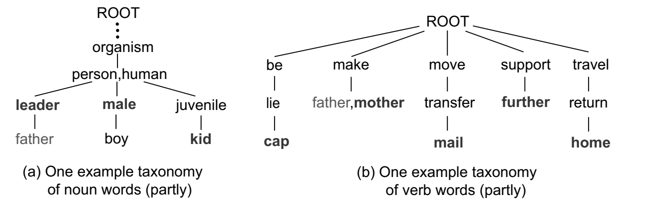


Figure 2: Top-k word searching on WordNet

Fig. 2 (a)). According to the definition of Sim_{lch}[6] (i.e., Eq. 1), we know that the ranking of a candidate only depends on its shortest path to the query. We can simply start from the node of the query, then expand to its neighbors in a recursive manner to find similar words.

The query word could be polysemous and we next address the issue when the query and the candidates exist in more than one taxonomy (i.e., Fig. 2 (a) and (b)). In this case, the ranking of a candidate not only depends on the shortest path in a taxonomy but also the maximum depth of the taxonomy. Derived from Eq. 1, we have the following lemma:

Lemma 1 (ordering in WordNet) *Let Q be the query. Let P and S be two candidates which exist in the same taxonomy of Q, i.e., T_P and T_S, respectively. The shortest path between P (or S) and Q is L_P (or L_S) in T_P (or T_S). The maximum depth of T_P (or T_S) is D_P (or D_S). Compared with S, P is more similar to Q if we have $\frac{D_P}{L_P} > \frac{D_S}{L_S}$.*

This lemma tells us that the similarity ordering between candidates in WordNet is dependent on the integration of the shortest path and the maximum depth of the taxonomy. Therefore, a best first search strategy can be constructed and the self-explanatory pseudo code for searching top-k similar words is shown in Algorithm 1.

We illustrate the main process of the algorithm (i.e., order of nodes accessed) in Fig. 3 by using the aforementioned example. The sequentially accessed nodes should

[6]Normalization does not affect the ranking here.

Algorithm 1: Top-k Similar Word Searching on WordNet

```
Input: Query $Q$, word collection $WC$, WordNet $WN$, $k$
Output: Top-$k$ similar words in $WC$
// preprocessing
1  $n$=number of taxonomies in $WN$;
2  $D[1,n]$=a list of maximal depth of taxonomies in $WN$;
3  $I$=an index that associates nodes in $WN$ to words in $WC$;
// query processing
4  $m$=number of polysemous meaning of $Q$;
5  $T[1,m]$=a list of taxonomy ID that $Q$ belongs to;
6  $L[1,m]$=a list of lengths (hops from $Q$), initialized as 1;
7  $N[1,m]$=a list of visited nodes in taxonomies, initialized as the nodes that Q
   belongs to;
8  while not $k$ words are output do
9      $\tau_{max}$=0; $ID_{max}$=0;
10     for $i$=1 to $m$ do // look for node to expand
11         $\tau_i=\frac{D[T[i]]}{L[i]}$;
12         if $\tau_i > \tau_{max}$ then $\tau_{max}$=$\tau_i$; $ID_{max}$=i;
13     forall $p \in N_{ID_{max}}$ do // expand nodes
14         $NN$=neighbor nodes of $p$ that are unvisited;
15         foreach node $e \in NN$ do
16             if $e \in WC$ based on $I$ then output $e$; $k$=$k$-1;
17     $N[ID_{max}]$=$N[ID_{max}] \cup NN$;
18     $L[ID_{max}]$++;
19 End
```

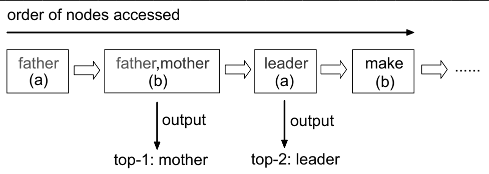


Figure 3: Progressive searching top-k similar words

be: (1) $father$ in the noun taxonomy (i.e., $\frac{D}{L}$=20)[7]; (2) $father, mother$ in the verb taxonomy (i.e., $\frac{D}{L}$=14); (3) $leader$ in the noun taxonomy (i.e., $\frac{D}{L}$=10); and so forth. As we can see, the top-k results are output in a progressive manner (i.e., $mother$, $leader$, etc).

3.2 Corpus based Feature

To obtain the top-k similar words based on large corpus (i.e., the Web), the traditional strategy is to submit the query and each candidate to the search engine and compute their similarity (i.e., PMI-IR [Turney, 2001]). Although this technique can benefit a lot from the power of search engine, evaluating every candidate pair words may cause large computation cost (i.e., network delay time used to transfer words to search engine and return answer, computation time at server, and parsing time on the answer stream). This issue becomes worse when testing on huge word collection.

In this paper, we propose to evaluate as few candidates as possible. We achieve this by constructing a simple index in the preprocessing, according to the following lemma.

Lemma 2 (upper bound of word $pagecount$) *Let Q be the query word and P be the top-k candidate similar word so far. Q and P have the PMI-IR similarity score as τ_{top-k}. We denote the set R of the remaining untested words. If $\forall r \in R, pagecount(r) > \frac{WebSize}{2^{\tau_{top-k}}}$, then the top-$k$ similar words w.r.t. PMI-IR score have been found.*

[7]The maximum depths of the two taxonomies are 20 for noun and 14 for verb by querying WordNet in the preprocessing.

Proof [Sketch of Proof] (Proof by Contradiction) Assume there is one candidate word r which has $pagecount(r) > \frac{WebSize}{2^{\tau_{top-k}}}$ in R and it is ranked in the top-k list. Then we have

$$\begin{aligned}\tau_{top-k} &\leq log_2 \frac{pagecount(Q \cap r) * WebSize}{pagecount(Q) * pagecount(r)} \\ &\leq log_2 \frac{pagecount(Q) * WebSize}{pagecount(Q) * pagecount(r)}, \quad hence \\ pagecount(r) &\leq \frac{WebSize}{2^{\tau_{top-k}}}\end{aligned}$$

which contradicts the assumption. □

This lemma tells us that we can sort all the candidates in ascending order of their $pagecount$ in the preprocessing, and measure the similarity while querying one by one. If we find that the $pagecount$ of the current candidate word is greater than $\frac{WebSize}{2^{\tau_{top-k}}}$, we can terminate the process and thus, avoid to evaluate the remaining candidates. The pseudo code is shown in Algorithm 2. To progressively extract the top-k similar words, we partition the whole process into k iterations. In each iteration only the best one word is output (lines [9-11]). Therefore, in each iteration we set the threshold as the top-1 score of the visited words (line [7,15]).

Algorithm 2: Top-k Similar Word Searching on Corpus

```
Input: Query $Q$, word collection $WC$, $k$
Output: Top-$k$ similar words in $WC$
// preprocessing
1  $n$=number of words in $WC$;
2  $H[1,n]$=a list of $pagecount$ in ascending order by querying Google;
3  $\tau_{top-1}$=0; $m$=1;
// query processing
4  $pagecount(Q)$=$pagecount$ of $Q$;
5  while not $k$ words are output do
6      if $|q| > 0$ then // q: max_queue stores visited words
7          $\tau_{top-1}$=the score value $\tau$ of $q$.top;
8      foreach $i$=$m$ to $n$ do
9          if $H[i] > \frac{WebSize}{2^{\tau_{top-1}}}$ then
10             pop and output $q$.top;
11             $m$=i; break;
12         $P$=the word corresponding to $H[i]$;
13         $\tau=log_2 \frac{pagecount(Q \cap P) \cdot WebSize}{pagecount(Q) \cdot pagecount(P)}$;
14         push $P$ into $q$ with value $\tau$;
15         $\tau_{top-1}$=the score value $\tau$ of $q$.top;
16 End
```

3.3 String Similarity Feature

There are not many studies on exploring how to efficiently search top-k similar words with respect to string similarity [Yang *et al.*, 2010]. In this paper, we apply the existing efficient strategies. Specifically, they are (1) count filtering; (2) length filtering; and (3) divide-merge-skip.

3.4 Assembling Similarity Features

Given the progressively extracted words in each feature, i.e., the result obtained from the above sections[8], we introduce an efficient approach to hasten the process of searching top-k similar words, based on the traditional rank aggregation algorithm [Fagin *et al.*, 2001].

[8]The transformation from the raw score to its normalized value is achieved by the approach presented in Section 2.1.

Lemma 3 (threshold filtering) *Let $\widetilde{Sim}_i^{top-k}$ be the normalized similarity score of the top-k word in feature i. Let t be the threshold where t=$\sum w_i \cdot \widetilde{Sim}_i^{top-k}$. A word P is in the top-k list if the following condition holds: $\sum w_i \cdot \widetilde{Sim}_i(P,Q) \geq t$.*

One interesting observation is that if we have a candidate word P whose total similarity score $\sum w_i \cdot \widetilde{Sim}_i(P,Q) \geq t$, there is at least for one feature that $\widetilde{Sim}_i(P,Q) \geq \widetilde{Sim}_i^{top-k}$. This intuition introduces a possible way to progressively output the top-k results. To illustrate the process, we use the aforementioned example to explain, as shown in Fig. 4.

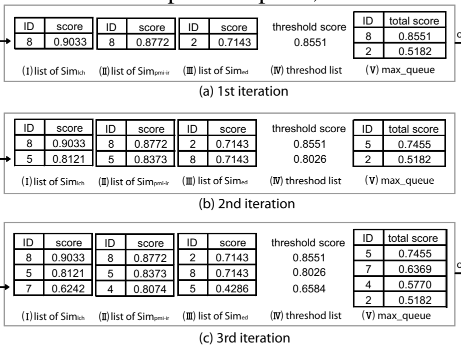


Figure 4: Integrated process for progressive searching

In the first iteration, we obtain the top-1 words with their scores in all the features, as shown in Fig. 4 (a) (I-III). Next we can calculate the threshold t=$\sum w_i \cdot \widetilde{Sim}_i^{top-1}$=0.8551, where the weights are set as described in Section 2.1. Moreover, we compute the comprehensive score of the accessed words (i.e., whose IDs are 8 and 2)[9] and put them into a maximum queue, as shown in Fig. 4 (a) (V). Because the score of word $mother$ (i.e., whose ID is 8) is not smaller than the threshold, $mother$ can be output immediately as the top-1 result (Lemma 3). We can see that for this example, the performance of obtaining the top-1 result is very efficiently because we avoid to evaluate many candidates. The remaining processes are executed in a similar way, as illustrated in Fig. 4.

4 Performance Analysis

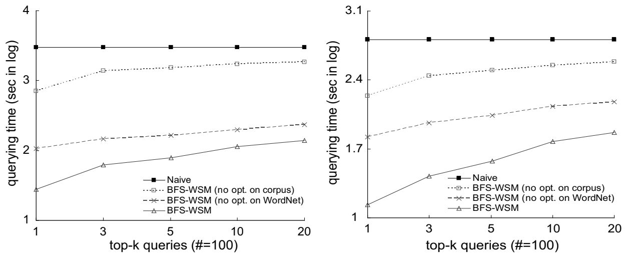
We performed the experiments using a Intel(R) Core(TM) 2 Dual CPU PC (3GHz) with a 2G memory, running Redhat linux. The $Naive$ algorithm was implemented based on the general top-k framework (Section 2.1). The BFS-WSM (Best First Search for Word Semantics Measure) algorithm was designed using the proposed strategies in this paper. We conducted experiments on two real life datasets: (1) $Dict$. It was downloaded from the Web site of GNU Aspell project[10]. It included 51K words; and (2) $Word$. It was downloaded from the Web site of WordNet in US[11], which includes 149K words. We randomly selected 10K words from this data as our word collection.

[9]Here we need to randomly access the value in each feature, by measuring it using traditional techniques.

[10]http://www.aspell.net/

[11]http://wordnet.princeton.edu/

4.1 Progressiveness



(a) **Dict** dataset (b) **Word** dataset

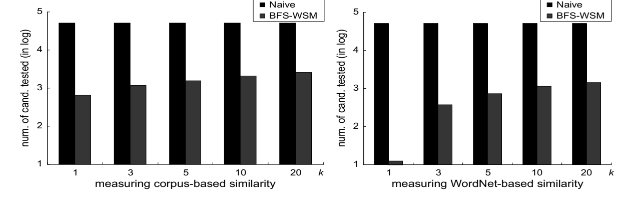
Figure 5: Progressiveness of query performance

The progressiveness performance of the proposed algorithm was evaluated by varying k. We randomly selected 100 queries from the datasets and reported the average value. To test the effect of the proposed strategies, we disable each one of them while keeping others unchanged[12].

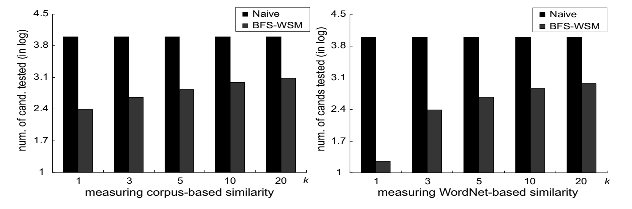
The result is shown in Fig. 5. We can see that $Naive$ outputs all the top-k results at the same time, and much slower than BSF-WSM (i.e., more than two orders of magnitude slower for top-1 query on $Dict$ data, as illustrated in Fig. 5 (a)). In contrast, our algorithm can extract the top-k (i.e., 1) answer efficiently, and progressively extract more results the longer the execution time. For the effect of different optimization strategies, optimization on corpus-based similarity measuring seems to dominate the whole query processing (i.e., red line in the figure). This is not surprising because the cost of requesting word $pagecount$ to search engine is high. Moreover, we can see that the proposed two optimizations (i.e., measuring on WordNet and corpus similarities) account for most of the performance improvement.

4.2 Number of Candidate Words Accessed

We evaluate the number of candidates tested in algorithms when varying k. As illustrated in Fig. 6, we can see that BSF-WSM accesses much smaller sets of words compared with $Naive$. This is the intrinsic reason why our algorithm performed better on query processing (as shown in Fig. 5).



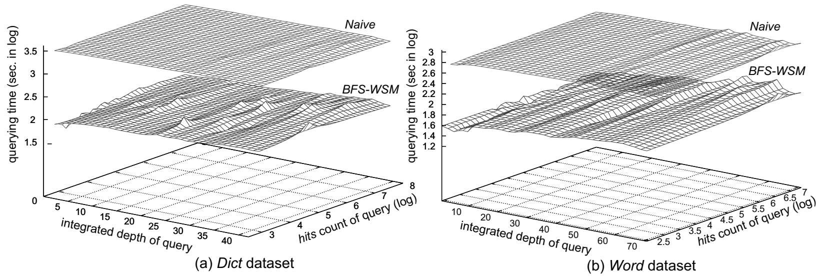
(a) #candidates vs k on **Dict** dataset



(b) #candidates vs k on on **Word** dataset

Figure 6: Number of words tested for different algorithms

[12]We focus on the effect of optimizing WordNet and corpus based measure, because the cost of measuring string similarity is much smaller and can be neglected compared with others.



(a) *Dict* dataset (b) *Word* dataset

Figure 7: Effect of $pagecount$ and $integrated\ depths$ of queries on performance (k=10)

4.3 Effect of $Pagecount$ and Depths

While conducting experiments, we mentioned that different queries may have distinct performance from each other. A "popular" word, i.e., that with large value of $pagecount$ and more meanings, has a high probability to be tested with more candidates. We also found that the depths of a word in the synsets of WordNet are critical to the performance because of the definition of the Sim_{lch} and the computation of the shortest path. Specifically, we define the $integrated\ depth$ of a word as the sum of all the depths of the word in the corresponding synsets of WordNet. Fig. 7 illustrates the performance of each randomly selected query. We can see that for $Naive$, the performance fluctuation is not obvious. The reason is that the cost of obtaining $pagecount$ is constant and it dominants the performance. In contrast, BFS-WSM largely reduces the number of candidates for acquiring $pagecount$. Therefore, the performance of queries may be very different from each other by taking into account the effect of optimization on WordNet similarity. Moreover, the result illustrates that many queries with high $pagecount$ and large value of $integrated\ depth$ spend more time on extracting the result, which is in accordance with the intuition mentioned above.

5 Discussion

Other similarity measures can be taken into account in a similar way to the proposed strategies. For example, for the information content based measures (e.g., [Resnik, 1995]), we can build an index similar to the n-gram list. Each word in the information content can be seen as a unigram and the corresponding list contains all the words whose information contents have the same gram. Then the candidate whose frequency is the highest in the lists of the query's grams is firstly evaluated, and so forth. For the other path-based measures (e.g., [Wu and Palmer, 1994]), we can optimally choose the nearest candidates to the query based on the specific measure.

6 Conclusion

We have studied the top-k similar word searching issue. This is the first work on tackling the efficiency issue that is very important for searching on large data. Several efficient strategies are proposed following the best first search manner. Our strategies are experimentally demonstrated to be efficient on answering the top-k semantic similar word queries.

References

[Bollegala *et al.*, 2007] D. Bollegala, Y. Matsuo, and M. Ishizuka. Measuring semantic similarity between words using web search engines. In *WWW*, 2007.

[Budanitsky and Hirst, 2006] A. Budanitsky and G. Hirst. Evaluating wordnet-based measures of lexical semantic relatedness. *Comput. Linguist.*, 2006.

[Deerwester *et al.*, 1990] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *JASIST*, 1990.

[Fagin *et al.*, 2001] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.

[Hellerstein *et al.*, 1999] J. M. Hellerstein, R. Avnur, A. Chou, C. Hidber, C. Olston, V. Raman, T. Roth, and P. J. Haas. Interactive data analysis: The control project. *Computer*, 1999.

[Jiang and Conrath, 1997] J. Jiang and D. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. In *ROCLING*, 1997.

[Leacock and Chodorow, 1998] C. Leacock and M. Chodorow. *Combining local context and WordNet similarity for word sense identification*. In C. Fellbaum (Ed.), MIT Press, 1998.

[Li *et al.*, 2003] Y. Li, Z. A. Bandar, and D. McLean. An approach for measuring semantic similarity between words using multiple information sources. *IEEE Trans. on Knowl. and Data Eng.*, 2003.

[Miller, 1995] G. A. Miller. Wordnet: a lexical database for english. *Commun. ACM*, 1995.

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

[Rada *et al.*, 1989] R. Rada, H. Mili, E. Bicknell, and M. Blettner. Development and application of a metric on semantic nets. *IEEE Trans. Sys., Man, and Cyber.*, 1989.

[Resnik, 1995] P. Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *IJCAI*, 1995.

[Shannon, 1948] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 1948.

[Tsatsaronis *et al.*, 2010] G. Tsatsaronis, I. Varlamis, and M. Vazirgiannis. Text relatedness based on a word thesaurus. *J. Artif. Int. Res.*, 2010.

[Turney, 2001] P. D. Turney. Mining the web for synonyms: Pmi-ir versus lsa on toefl. In *ECML*, 2001.

[Ukkonen, 1992] E. Ukkonen. Approximate string-matching with q-grams and maximal matches. *Theor. Comp. Sci.*, 1992.

[Wu and Palmer, 1994] Z. Wu and M. Palmer. Verbs semantics and lexical selection. In *ACL*, 1994.

[Yang *et al.*, 2010] Z. Yang, J. Yu, and M. Kitsuregawa. Fast algorithms for top-k approximate string matching. In *AAAI*, 2010.