World Wide Web (2014) 17:595–626
DOI 10.1007/s11280-012-0195-z

Exploration on efficient similar sentences extraction

Yanhui Gu · Zhenglu Yang · Guandong Xu · Miyuki Nakano · Masashi Toyoda · Masaru Kitsuregawa

Received: 18 July 2012 / Revised: 17 November 2012 /
Accepted: 21 November 2012 / Published online: 6 January 2013

Abstract Measuring the semantic similarity between sentences is an essential issue for many applications, such as text summarization, Web page retrieval, question-answer model, image extraction, and so forth. A few studies have explored on this issue by several techniques, e.g., knowledge-based strategies, corpus-based strategies, hybrid strategies, etc. Most of these studies focus on how to improve the effectiveness of the problem. In this paper, we address the efficiency issue, i.e., for a given sentence collection, how to efficiently discover the top-*k* semantic similar sentences to a query. The previous methods cannot handle the big data efficiently, i.e., applying such strategies directly is time consuming because every candidate sentence needs to be tested. In this paper, we propose efficient strategies to tackle such problem based on a general framework. The basic idea is that for each similarity, we build a corresponding index in the preprocessing. Traversing these indices in the querying process can avoid to test many candidates, so as to improve

Y. Gu · Z. Yang (✉) · M. Nakano · M. Toyoda · M. Kitsuregawa
Institute of Industrial Science, The University of Tokyo, Tokyo, Japan
e-mail: yangzl@tkl.iis.u-tokyo.ac.jp

Y. Gu
e-mail: guyanhui@tkl.iis.u-tokyo.ac.jp

M. Nakano
e-mail: miyuki@tkl.iis.u-tokyo.ac.jp

M. Toyoda
e-mail: toyoda@tkl.iis.u-tokyo.ac.jp

M. Kitsuregawa
e-mail: kitsure@tkl.iis.u-tokyo.ac.jp

G. Xu (✉)
Advanced Analytics Institute, University of Technology Sydney,
Sydney, New South Wales, Australia
e-mail: guandong.xu@uts.edu.au

the efficiency. Moreover, an optimal aggregation algorithm is introduced to assemble these similarities. Our framework is general enough that many similarity metrics can be incorporated, as will be discussed in the paper. We conduct extensive experimental evaluation on three real datasets to evaluate the efficiency of our proposal. In addition, we illustrate the trade-off between the effectiveness and efficiency. The experimental results demonstrate that the performance of our proposal outperforms the state-of-the-art techniques on efficiency while keeping the same high precision as them.

Keywords semantic similarity · query aggregation · top-*k*

1 Introduction

Measuring semantic similarity between sentences is an essential issue because it is the basis of many applications, such as snippet extraction, image retrieval, question-answer model, document retrieval, and so forth [5, 8, 39, 41, 47, 62, 63].

Traditional techniques, for measuring similarity between documents (long texts), e.g., TF-IDF [27], have been introduced based on an intuitive assumption that many common words exit in similar documents [52–54]. However, these methods are inappropriate for measuring similarities between sentences because for short texts common words are few or even null [35, 40, 51, 57]. To tackle this challenge, many strategies have been proposed, which can be classified into four kinds of techniques to measure the similarity between sentences: (1) knowledge-based strategy [13, 40, 57]; (2) string similarity based strategy [7, 33, 42, 43, 59]; (3) corpus-based strategy [24, 30, 31, 58]; and (4) hybrid strategy [24, 35, 58]. **Knowledge-based strategy** (e.g., WordNet [32]) utilizes the labeled (or semi-labeled) data for text related research tasks. In [57], the authors firstly create semantic networks from word thesauri and then measure the relatedness between words. Finally, they compute the similarities between sentences based on the semantic network. The hierarchy property of WordNet has been explored in [35]. The similarity is estimated from the hierarchy based on a node counting strategy. **String similarity based strategy** measures the difference of the character position between two words [7, 33]. For instance, edit distance (a.k.a, the Levenshtein distance) is a classical string metric for measuring the syntax difference between two sequences. There are many algorithms which have used q-gram based strategies [42, 43, 59] to measure the edit distance. **Corpus based strategy** estimates the statistical information of large corpus to calculate the similarity between two texts, i.e., LSA (Latent Semantic Analysis) [12, 31] and HAL (Hyperspace Analogues to Language) [4]. In the LSA approach, a few of representative words will first need to be identified from a large number of contexts, i.e., corpus. Then the SVD (Singular Value Decomposition) technique will be applied to reduce the dimensionality. A vector for each sentence is formed in the reduced dimension space and finally similarity is measured by computing the similarity between these two vectors [12]. HAL builds a word-by-word matrix based on word co-occurrences within a moving window of a predefined width. The similarity can be measured based on the matrix. In [13], the author represents the meaning of texts in a high-dimensional space of concepts derived from Wikipedia and apply machine learning techniques to explicitly represent the meaning of any text as a weighted vector of Wikipeida-based concept to measure the similarities between texts. **Hybrid strategy** integrates several (or all) techniques mentioned above [24, 35, 58].

Currently, searching similar sentences from big data is an important issue [2, 37, 70]. From a given sentence collection, this kind of queries asks for those sentences which are most semantically similar to a given one. The problem can be solved as follows: we firstly measure the semantic similarity score between the query and each sentence in the data collection using the state-of-the-art techniques [24, 35, 40, 51, 57], then sort them with regard to the score and finally return the top-*k* ones. Almost all the previous studies focus on improving the effectiveness (i.e., precision) of the problem and the datasets conducted are small. However, when the size of the data collection increases, the scale of the problem will dramatically increase and the state-of-the-art techniques will be impractical [14, 15, 21, 22, 28, 44, 48, 67]. As far as we know, this paper is the first study that aims to address the efficiency issue in the literature. Moreover, most of the previous strategies applied the threshold-based strategy [24, 35, 57], that a threshold is predefined to filter out those dissimilar sequences. However, this threshold is difficult for users to determine. For real applications (e.g., Google), users may prefer the top-*k* results.

Naively testing every candidate sentence is time consuming, especially when the size of the sentence collection is huge. To tackle this issue, we introduce efficient strategies to evaluate as few candidates as possible. Moreover, we aim to progressively output the top-*k* results, i.e., the top-1 result should be output almost instantly, then the top-2 and more results will be obtained as the execution time becomes longer. This satisfies the requirement of the real applications [19]. As such, these issues are the challenges of the paper, which have not been studied before.

It should be noted that other optimization strategies such as caching [50], parallel processing [61] may improve the efficiency. However, all of these approaches need additional techniques and such extension is out of the scope of this paper. Our contributions of this paper are listed as follows:

- We propose to tackle the efficiency issue for searching top-*k* semantic similar sentences, which is different from the previous works that focus on the effectiveness aspect. This is a very important issue because in many real applications the data becomes too huge to be practically dealt with. Based on the most comprehensive work [24], we maintain the same effectiveness (i.e., precision) while aiming at improving the efficiency.
- We propose efficient strategies to extract the top-*k* results. For each similarity measurement, we introduce a corresponding strategy to minimize the number of candidates to be evaluated. A rank aggregation method is introduced to progressively obtain the top-*k* results when assembling the features. Detail analysis on time and space complexity is presented. Moreover, we illustrate the workflow of our proposal by using an concrete example.
- We conduct comprehensive experiments and evaluate the performance of the proposed strategies. The results show that the proposed strategies outperform the state-of-the-art method. We also illustrate the trade-off between effectiveness and efficiency on different datasets.
- Our introduced framework is general enough that more similarity metrics can be incorporated and will be discussed in this paper.

2 Related work

There are many studies [3, 13, 34, 43, 46, 49, 56, 59] introduced to measure the similarities between words, i.e., string similarity or semantic similarity. To estimate the string similarity between two words, quite a few metrics are introduced, e.g., edit-distance [1], hamming distance [16], Damerau Levenshtein distance [9], Jaro distance [25], etc. On the other hand, there are many strategies proposed for measuring the semantic similarity between two words. Based on the different strategies applied, the existing work can be classified into external resources based [32, 34, 56] and statistics information based [3, 12, 13, 31]. String similarity only measures the difference of positions between words, while semantic similarity can measure the latent meaning between two words. Therefore, taking into account the string similarity and semantic similarity comprehensively seems to be an optimal solution, as demonstrated in [24, 34, 35, 46, 58].

Text is composed of words. Measuring similarity between long texts has been extensively studied [18, 30, 36, 38] while just a few of them can be directly applied to the sentence similarity measurement because of the short length property [24, 35, 40]. Based on the different strategies applied, the existing work on measuring similarity between sentences can be classified into several categories:

Word co-occurrence or vector-based model strategy. This strategy is commonly used in information retrieval systems [38]. The basic idea is that, the text is first represented as a bag of words in a vector-based model. Thereafter, the traditional similarity measurements, i.e., cosine, Jaccard, etc., can be applied. However, this kind of term-wise similarity strategy which directly applies the classic document similarity approaches [6, 54] often obtains inadequate results for sentences because of the length of short text (i.e., sentences) and the limited common terms between sentences [24, 35, 40, 57]. Therefore, measuring the similarity between sentences should capture more semantic context.

String similarity based strategy. Many strategies estimate the string similarity between two texts [7, 33]. For instance, edit distance (a.k.a, the Levenshtein distance) is a classical string metric for measuring the syntax difference between two sequences. There are many algorithms which have used q-gram based strategies [42, 43, 59] to compute the edit distance between words. In [66] the authors proposed several strategies, e.g., adaptive q-gram selection, to efficiently retrieval the top-k results. In [55], the authors introduced deliberated techniques, e.g., divide-skip-merge, to extract similar strings.

Knowledge-based strategy. Knowledge base (sometimes called word thesauri), e.g., WordNet [32], contains the labeled (or semi-labeled) data for text related research tasks. In [57], they firstly create semantic networks from word thesauri and then measure the relatedness between words based on these semantic networks. The text-to-text semantic relatedness is composed of word-to-word similarity, which is computed by measuring the semantic relatedness between words in the networks. The hierarchy property of WordNet has been explored in [35]. The word pair similarity is estimated from the hierarchy based on a node counting strategy, i.e., calculating the number of nodes between the target words.

Corpus-based strategy. Large corpus is taken into account as a third-party resource which can fairly reflect the distribution of texts. This statistical information is thus, can be used to calculate the similarity between two words or texts. Some well known methods in corpus-based similarity are LSA (Latent Semantic Analysis) [12, 31] and HAL (Hyperspace Analogues to Language) [4]. In the LSA approach, a few of representative words will first need to be identified from a large number of contexts, i.e., corpus. Then the SVD (Singular Value Decomposition) technique will be applied to reduce the dimensionality. A vector for each sentence is formed in the reduced dimension space and finally similarity is measured by computing the similarity between these two vectors [12]. Because of the computation limitation of SVD, the dimension size of the context matrix is limited to several hundred. Since LSA ignores the syntactic information, it is more appropriate for large texts instead of the sentences [31]. HAL is closely related to LSA and it captures the meaning of a word or text by using lexical co-occurrence. Unlike LSA, which builds an information matrix of words based on text units of paragraphs or documents, HAL builds a word-by-word matrix based on word co-occurrences within a moving window of a predefined width. Therefore its drawback may be due to the building of the memory matrix and its approach to forming sentence vectors, i.e., the word-by-word matrix does not capture sentence meaning well and the sentence vector becomes diluted since a large number of words are added to it [4]. ESA (Explicit Semantic Analysis) [13] represents the meaning of texts in a high-dimensional space of concepts derived from Wikipedia. It applies machine learning techniques to explicitly represent the meaning of any text as a weighted vector of Wikipeida-based concept.

Hybrid strategy. To tackle the drawback of single strategy, the hybrid strategy was proposed [24, 35, 58]. The combination of knowledge based strategy and word order based strategy was proposed in [35]. In this paper, the authors present a method for measuring the semantic similarity between sentences which is based on semantic and word order information. They apply the knowledge base as the outer semantic resource. In [24], the authors apply the string based strategy, the word order based strategy, and the corpus strategy. The three strategies are linearly aggregated to measure the similarity between sentences. The string similarity measures the gram level relatedness while the corpus based strategy takes into account the semantic relatedness between two words. The word order based strategy measures the syntax similarity between sentences based on the order of the common words in sentences.

However, all the existing approaches have not taken into account the efficiency issue, which is an important issue for large data [14, 15, 21, 22, 28, 44, 48, 67]. When searching the top-k similar sentences, these techniques need to evaluate all the candidate sentences in the data collection, which are very time consuming.

Materializing all the similarities of sentence pairs in the preprocessing may be a possible solution. However, the data is frequently updated, e.g., the Web data. Storing and computing frequently all the similarities is time and space consuming.

Moreover, due to the unpredictable property of user input, e.g., misspelled sentences, we can not preserve all the similarities, e.g., string similarity, beforehand. As a consequence, computing the similarities on-the-fly seems to be a practical solution to the similar sentences extraction.

Some studies [17, 46] were introduced from another perspective. They regard that the previous approaches only consider the resources as a static collection of texts or concepts. There is additional resource of rich information about the semantic similarity of words which can be revealed by studying the patterns of word occurrence over time. Therefore, they compute the word similarity using temporal semantic analysis.

The research introduced in this paper (searching semantic similar sentences) is related to the linguistic aspect of the faceted search and interactive semantics. Faceted search (i.e., faceted browsing) is a search paradigm which is developed originally in the field of information retrieval [29, 70–72]. The idea of the scheme is to analyze and index search items along multiple orthogonal taxonomies that are called subject facets. This basic idea has been developed into semantic faceted search later, where the facets are based on ontological structures, e.g., subclass and part-of hierarchies. Therefore, searching in a faceted semantic space seeks to improve the search accuracy by understanding the intention of users and the contextual meaning of terms either on the Web [75] or within a closed system. The semantics can be represented in a space [68, 73] or by links [69]. To discover more accurate results, the interactive semantic techniques were introduced in [70]. Different from the traditional dataspace, the model based on the probabilistic resource space was proposed in [74]. In the context of faceted search and semantic interaction, semantic similarity and query efficiency are two important factors needed to be fully addressed. Semantic similarity is the foundation of all semantic computation, no matter which types the data objects are, e.g., words, sentences and documents. Query efficiency is another key issue involved in search especially for large data. The study proposed in this paper aims to address the above two issues, especially in applications of semantic similar sentence extraction. Our proposed techniques in this paper can be seamlessly coupled with the traditional strategies to tackle the issues of faceted search and interactive semantics.

3 Problem statement

The issue we aim to tackle is to extract the top-k similar sentences to a query. Formally, for a query sentence Q, finding a set of k sentences P in a given sentence collection S which are most similar to Q, i.e., $\forall p \in P$ and $\forall r \in (S - P)$ will yield $sim(Q, p) \geq sim(Q, r)$.

To measure the similarity $sim(Q, P)$ between two sentences, we apply the state-of-the-art strategies by assembling multiple similarity metric features together [24, 35]. Because we focus on tackling the efficiency issue in this paper, we select several representative features based on the framework which was introduced in [24]. Note that a sentence is composed of a set of words and therefore, the similarity score between two sentences is the overall scores of all the word pairs whose components belong to each sentence, respectively. See [24] for detail on computing sentence similarity based on word similarity. We will introduce the representative word similarity in the next section.

3.1 Similarity measurement strategies

3.1.1 String-based similarity

String similarity measures the difference of syntax between strings. An intuitive idea is that two strings are similar to each other if they have enough common subsequences (i.e., LCS [20]). We focus on three representative string similarity measurement strategies, i.e., NLCS, NMCLCS_1 and NMCLCSn[1] which are denoted as Sim_{NLCS}, Sim_{NMCLCS_1} and Sim_{NMCLCS_n}.

NLCS LCS is a common string similarity measurement strategy and it measures the longest common subsequence of two strings. NLCS is the normalized LCS of two words w_i and w_j, defined as follows:

$$Sim_{NLCS}(w_i, w_j) = \frac{length^2(LCS)}{length(w_i) \cdot length(w_j)} \tag{1}$$

$NMCLCS_1$ NMCLCS_1 measures the similarity between two strings where they have the maximal consecutive LCS from the first character. Different from NLCS, NMCLCS_1 has two properties: (1) the longest common subsequence in NMCLCS_1 should be consecutive; and (2) the common subsequence should start from the first character of each string. The NMCCLS_1similarity is calculated based on the following formula:

$$Sim_{NMCLCS_1}(w_i, w_j) = \frac{length^2(MCLCS_1)}{length(w_i) \cdot length(w_j)} \tag{2}$$

NMCLCSn Similar to NMCLCS_1, NMCLCSn measures the similarity between two strings where they have the maximal consecutive common subsequence. The only difference here is that it can start at any position, defined as follows:

$$Sim_{NMCLCSn}(w_i, w_j) = \frac{length^2(MCLCS_{n)}}{length(w_i) \cdot length(w_j)} \tag{3}$$

For detail information about these string similarities, refer [24].

3.1.2 Corpus-based similarity

Corpus-based similarity measurement strategy is to recognize the degree of similarity between words using large corpora [31]. There are several kinds of strategies: PMI-IR [58], LSA [30], HAL [4], chi-square, log-likelihood, and so forth. In this paper,

[1]NLCS: Normalized Longest Common Substring, NMCLCS_1: Normalized Maximal Consecutive LCS starting at character 1, NMCLCSn: Normalized Maximal Consecutive LCS starting at any character *n* [24].

we use the SOC-PMI (Second Order Co-occurrence PMI) [23, 24] which employs PMI-IR to take into account the important neighbor words in a context window of the two target words from a large corpus. The PMI-IR between two words, word w_1 and w_2, is defined as follows [58]:

$$f^{pmi}(w_1, w_2) = log_2 \frac{p(w_1, w_2)}{p(w_1) \cdot p(w_2)} \tag{4}$$

where, $p(w_i)$ is the probability that w_i occurs in corpus and $p(w_1, w_2)$ is the probability that w_1 and w_2 co-occur. Since probability is a measure of the expectation that an event will occur or a statement is true, e.g., in (4), we have $p(w_i) = \frac{f(w_i)}{m}$, where $f(w_i)$ is the frequency that w_i occurs in the whole corpus, m is the size of corpus. So, (4) can be represented as follows:

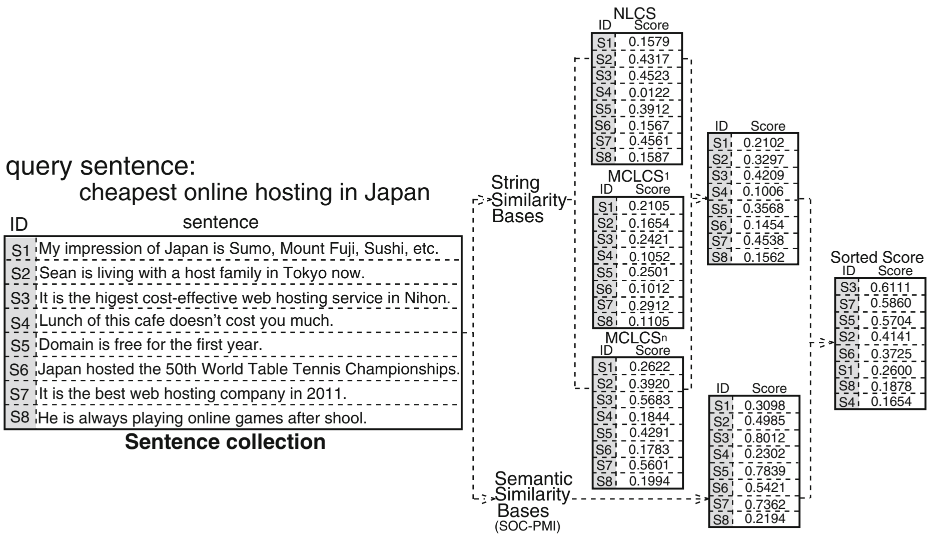
$$f^{pmi}(w_1, w_2) = log_2 \frac{\frac{f(w_1,w_2)}{m}}{\frac{f(w_1)}{m} \cdot \frac{f(w_2)}{m}} = log_2 \frac{f(w_1, w_2) \cdot m}{f(w_1) \cdot f(w_2)} \tag{5}$$

The main characteristic of the SOC-PMI method is that we can determine the semantic similarity of two words even though they do not co-occur within the window size in the corpus at all. In fact, we are considering the second order co-occurrences, as we are judging by the co-occurrences of the neighbor words, but not only the co-occurrence of the two target words. PMI-IR and many other corpus-based semantic similarity measures do not have such characteristic. However, SOC-PMI applies the basic PMI-IR in measuring each pair of words and the intuitive idea is that the neighbors of the two target words have abundant semantic context and should be considered. Equation (5) is used to calculate the similarities between word pairs (i.e., including neighbor words), and then high PMI scores are aggregated to obtain the final SOC-PMI score. Refer [23] for detail.

3.2 A general framework

To measure the overall similarity between two sentences, a general framework is presented by incorporating all the similarity measure strategies. As far as we know, [24] is the most comprehensive approach which incorporates several representative similarity metrics (i.e., string similarity and semantic similarity[2]). Due to lack of large labeled data, the existing state-of-the-art assembling techniques commonly set the weight values arbitrarily [24, 35, 40, 51]. We apply the same strategy (i.e., the same weigh value α, $\alpha_{\text{string}} = \alpha_{\text{semantic}} = \frac{1}{2}$). 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. As the experiments shows, our approach can obtain the same high precision as the state-of-the-art ones.

[2]The common word order similarity is neglected here because [24] has demonstrated that it has no influence on the overall score.



ID	sentence
S1	My impression of Japan is Sumo, Mount Fuji, Sushi, etc.
S2	Sean is living with a host family in Tokyo now.
S3	It is the higest cost-effective web hosting service in Nihon.
S4	Lunch of this cafe doesn't cost you much.
S5	Domain is free for the first year.
S6	Japan hosted the 50th World Table Tennis Championships.
S7	It is the best web hosting company in 2011.
S8	He is always playing online games after shool.

NLCS

ID	Score
S1	0.1579
S2	0.4317
S3	0.4523
S4	0.0122
S5	0.3912
S6	0.1567
S7	0.4561
S8	0.1587

MCLCS1

ID	Score
S1	0.2105
S2	0.1654
S3	0.2421
S4	0.1052
S5	0.2501
S6	0.1012
S7	0.2912
S8	0.1105

MCLCSn

ID	Score
S1	0.2622
S2	0.3920
S3	0.5683
S4	0.1844
S5	0.4291
S6	0.1783
S7	0.5601
S8	0.1994

ID	Score
S1	0.2102
S2	0.3297
S3	0.4209
S4	0.1006
S5	0.3568
S6	0.1454
S7	0.4538
S8	0.1562

ID	Score
S1	0.3098
S2	0.4985
S3	0.8012
S4	0.2302
S5	0.7839
S6	0.5421
S7	0.7362
S8	0.2194

Sorted Score

ID	Score
S3	0.6111
S7	0.5860
S5	0.5704
S2	0.4141
S6	0.3725
S1	0.2600
S8	0.1878
S4	0.1654

Figure 1 The implementation of the top-k similar sentences searching framework.

The string similarity is further composed of three different measurements, i.e., NLCS, NMCLCS$_1$, and NMCLCS$_n$. Figure 1 illustrates the general framework for searching the top-k similar sentences. Suppose we have 8 sentences in the sentence collection, for a given query sentence "*cheapest online hosting in Japan*", the existing strategies compute the similarity scores of all the sentence pairs between the query and each sentence in the collection. Then linearly aggregate the similarity scores based on sentence IDs. Finally, the top-k value can be retrieved after sorting these similarity scores. However, it is very time consuming especially when the data collection is large. Therefore, efficient strategies on searching top-k semantic similar sentences are necessary.

4 Proposed approaches

In this section, we propose efficient strategies for extracting the top-k similar sentences. The key idea is that by building appropriate index in the preprocessing, we only need to test a small part candidates of the whole data collection. The optimizations on the two similarity features are fulfilled on the word level (Sections 4.1 and 4.2). Efficient computation on sentence similarity is presented in Section 4.3. The optimization on assembling similarity features is introduced in Section 4.4.

4.1 Optimization on string-based similarity

We employ NLCS, NMCLCS$_1$ and NMCLCS$_n$ as our string-based similarity features [24]. Discussion on other similarities will be presented in Section 6.

• *NLCS* The basic idea for improving the efficiency of similar word extraction, is to test as few candidates as possible. To address this issue, in the preprocessing we need to build an effective index which facilitates the candidate test process. A well known technique is n-gram (or q-gram [59]) model, which is utilized in our framework. Moreover, because our purpose is to search the top-k similar words, we can further improve the efficiency based on the property of the similarity.

From (1) we know that NLCS is based on two factors: (1) length of LCS; and (2) length of the candidate word w_i (length of the query is not taken into account because we cannot know it in the preprocessing). The similarity increases when $length(LCS)$ increases or $length(w_i)$ decreases. Based on this property, we have the following lemma.

Lemma 1 (Lower bound of gram length) *Let P be the top-k similar word to the query Q so far. Q and P have the NLCS score as $\tau_{\text{top}-k}$. We denote the set R be the untested words and the gram set G be the untested grams. If $\forall g \in G$ and $\forall r \in R$, $|g_r| < |Q| \cdot \tau_{\text{top}-k}$, then the top-$k$ similar words w.r.t. string similarity score have been found.*

Proof (Sketch of Proof) (Proof by Contradiction) Assume there is one candidate word r with gram $|g_r| < |Q| \cdot \tau_{\text{top}-k}$ in R and it is ranked in the top-k list. Then we have $Sim(Q, r) \geq \tau_{\text{top}-k}$, $\frac{|g_r|^2}{|Q||r|} \geq \tau_{\text{top}-k}$, $|g_r|^2 \geq |Q| \cdot |r| \cdot \tau_{\text{top}-k}$, since $|r| \geq |g_r|$ in any case, so $|Q| \cdot |r| \cdot \tau_{\text{top}-k} \geq |Q| \cdot |g_r| \cdot \tau_{\text{top}-k}$, accordingly $|g_r|^2 \geq |Q| \cdot |g_r| \cdot \tau_{\text{top}-k}$, we can get $|g_r| \geq |Q| \cdot \tau_{\text{top}-k}$, which contradicts the assumption. □

This lemma tells us that we can sort all the grams in descending order of their lengths in the index. Moreover, for each *gram*, we sort all the corresponding words (which include this gram) in ascending order of their lengths. To search the top-k similar words, we start from the word list which has the longest gram. In the list, we first test shorter words. A concrete example is presented in Figure 2 to illustrate the process. Suppose we want to obtain the top-1 similar word to the query *host*. By extracting the corresponding gram lists (i.e., decompose the query into grams) from the index, we first test the word *hoster*, where $Sim_{NLCS}(host, hoster) = \frac{16}{24}$. The lower bound is therefore, equal to $\frac{16}{6} = 2.67$. So the grams whose lengths are less than 2.67 can be omitted and those related words (e.g., *ho*) do not need to be tested. The pseudo code is shown in Algorithm 1. Line 1 is to index for the whole data collection and lines [2–5] is query preprocessing. The query includes parsing query into grams and extracting the corresponding gram lists G_Q from the whole gram G. We compute the threshold τ in lines [6–8] and put it into a *max_queue* which is illustrated in lines [10–14]. The top value will be outputted if the inequality $|g| < |Q| \cdot \tau_{\text{top}-1}$ is satisfied. If not, we test the other words and compute the new threshold τ which is presented in lines [17–21]. The algorithm will terminated when all results are retrieved.

• *NMCLCS*$_1$ This similarity measures the maximal common consecutive prefix substrings of two strings. We give an analysis similar to that of NLCS. From (2), we can see that NMCLCS$_1$ is determined by two factors: length of MCLCS$_1$ and length of candidate word in the collection. The NMCLCS$_1$ similarity increases when the length of MCLCS$_1$ increases or the length of the candidate decreases. Therefore, we can build a gram index that all the grams are in descending order of their lengths. For each *gram*, the related words are sorted in ascending order of their lengths.

Algorithm 1: Top-k Similar Word Searching for NLCS

```
Input: Query $Q$, word collection $WC$, $k$
Output: Top-$k$ similar words in $WC$
// preprocessing
construct gram index $G$ on $WC$ (// NLCS gram index);
// query processing
parse $Q$ into grams and extract the corresponding gram lists $G_Q$ from $G$;
$n$=$|G_Q|$; i=1;
g=$G_Q$[i];
$w_g[1,m]$=the list of words in $G_Q$ which includes $g$;
foreach $j$=1 to $m$ do
    $\tau=\frac{|g|^2}{|Q|\cdot|w_g[j]|}$ (// g: NLCS gram);
    push $w_g[j]$ into $q$ with value $\tau$;
while not k words are output do
    if $|q|>0$ then // q: max_queue stores visited words
        $\tau_{top-1}$=the score value $\tau$ of $q$.top;
        if $|g|<|Q|\cdot\tau_{top-1}$ then
            pop and output $q$.top;
            continue;
    i=i+1;
    if $i$==$n$ then break;
    g=$G_Q$[i];
    $w_g[1,m]$=the list of words in $G_Q$ which includes $g$;
    foreach $j$=1 to $m$ do
        $\tau=\frac{|g|^2}{|Q|\cdot|w_g[j]|}$ (// g: NLCS gram);
        push $w_g[j]$ into $q$ with value $\tau$;
End
```

The difference for indices between NLCS and NMCLCS$_1$ is that the grams for the latter are only a part of the former (i.e., those grams should start at the beginning of each string). The lower bound used to terminate the process is $|Q|\cdot\tau_{\text{top}-k}$. The computation of $\tau_{\text{top}-k}$ is based on (2). The algorithm of searching top-k similar words for NMCLCS$_1$ is very similar to that for NLCS. We can modify the code on lines [1, 7, 20] in Algorithm 1 accordingly and the pseudo code is shown in Algorithm 2. The only difference between Algorithms 1 and 2 is how to parse words into grams.

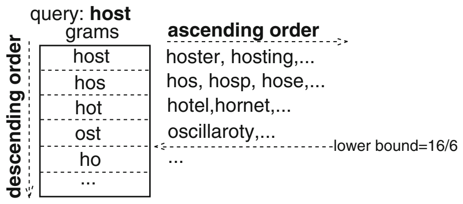


Figure 2 Illustration on boundary of NLCS.

Algorithm 2: Top-k Searching for NMCLCS$_1$(modification based on Algorithm 1)

```
Input: Query $Q$, word collection $WC$, $k$
Output: Top-$k$ similar words in $WC$
1  construct gram index $G$ on $WC$   (// NMCLCS$_1$ gram index);
   // query processing
       ......
7      $\tau = \frac{|g|^2}{|Q| \cdot |w_g[j]|}$   (// g: NMCLCS$_1$ gram);
       ......
20      $\tau = \frac{|g|^2}{|Q| \cdot |w_g[j]|}$   (// g: NMCLCS$_1$ gram);
       ......
   End
```

• *NMCLCSn* NMCLCSn measures the maximal common consecutive substring which can starts at any position n. The index for NMCLCSn is similar to that for NMCLCS$_1$ (i.e., the order for grams and the order for related words). The only difference is the strategy for decomposing words into grams. NMCLCSn parses words into consecutive grams which can start at any positions. The lower bound used to terminate the process is also $|Q| \cdot \tau_{\text{top}-k}$, where $\tau_{\text{top}-k}$ is calculated by (3). The algorithm for NMCLCSn is similar to that for the above mentioned string similarities, with modification according to the definition of NMCLCSn.

4.2 Optimization on corpus-based similarity

Traditional corpus-based similarity measurement is to submit the query and each candidate into the large corpus (e.g., BNC[3]) and then compute the similarity. However, computing all the query and candidate pairs may cause large computation cost. We apply SOC-PMI [24] as the corpus-based similarity evaluator. SOC-PMI linearly aggregates the top large PMI values (i.e., (5)) of each pair words with their neighbors. We introduce an efficient technique which was first presented in [65].

Lemma 2 (Upper bound of word frequency) *Let Q be the query word and P be the top-k similar word so far. Q and P have the similarity score τ_{top-k}. If $\forall r \in R$, $f(r) > \frac{m}{2^{\tau_{top-k}}}$, the top-k similar words have been found. Here R is the remaining untested words and m is the size of the corpus.*

Proof (Sketch of Proof) (Proof by Contradiction) Assume there is one candidate word r which has $f(r) > \frac{m}{2^{\tau_{\text{top}-k}}}$ in untested set R and it is in the ranked top-k list. Then we have $\tau_{\text{top}-k} \le log_2 \frac{f(Q,r) \cdot m}{f(Q) \cdot f(r)}$, since $f(Q) \ge f(Q,r)$ in any case, we have $\tau_{\text{top}-k} \le log_2 \frac{f(Q) \cdot m}{f(Q) \cdot f(r)}$, hence $f(r) \le \frac{m}{2^{\tau_{\text{top}-k}}}$, which contradicts the assumption. □

Based on this lemma, we sort all the candidates in ascending order of their frequencies in the preprocessing and measure the similarity while querying one by one. When we find the frequency of current candidate word is greater than

[3]British National Corpus, http://www.natcorp.ox.ac.uk/

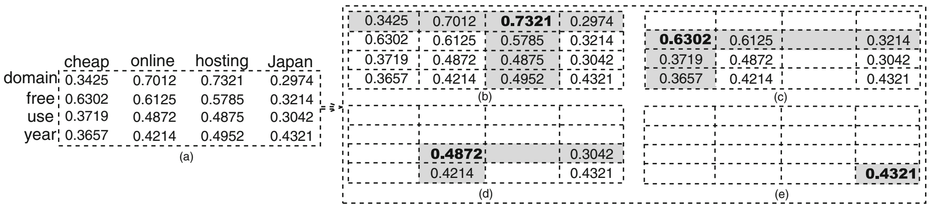
Algorithm 3: Top-k Similar Word Searching for Corpus-based Similarity

```
Input: Query $Q$, Word collection $C$, top value $k$;
Output: top-$k$ similar words in $C$
// preprocessing
1  G[1..n]: a list of $words$ in ascending order by their frequency;
2  $n$: number of words in $C$; $m$: corpus size;
3  $\tau_{top-k} = 0$, $p = 1$;
4  $f(Q)$=frequency of $Q$;
// processing
5  while  $output$ $output < k$ do
6      if $| q | > 0$ then // q: max_queue stores visited words
7          $\tau_{top-1}$=q.top; // max_queue(q) ← accessed words
8      foreach $i$=$p$ to $n$ do
9          if $G[i] > \frac{m}{2^{top-1}}$ then
10             output q.top; p=i; break;
11         P=$W_{G[i]}$; // word corresponding to G[i]
12         $\tau = log_2 \frac{f(Q,P) \cdot m}{f(Q)f(p)}$;
13         push $P$ into $q$ with $\tau$;
14         $\tau_{top-1} = q.top$;
15 End
```

$\frac{m}{2^{\text{top}-k}}$, we can terminate the process and avoid to test the remaining candidates. The pseudo code is presented in Algorithm 3. Based on the analysis of this algorithm, we first index all the words in a ascending order listed by their frequency in corpus in preprocessing which is illustrated in lines [1–2]. We also set the initial value of $\tau_{\text{top}-k}$. Lines [6–7] presents that the *max_queue* pops *top* − 1 out if $|q| > 0$. If not, we compare the frequency of the word in $G[i]$ and compute the new boundary. Moreover, the old $\tau_{\text{top}-1}$ is replaced by new τ which is illustrated in lines [8–15]. The algorithm terminates when all the top-k values are outputted. For each word pair, we can apply such algorithm to evaluate the pairs between neighbors. The final results are obtained by linearly aggregated the top large PMI values.

4.3 Sentence similarity computation

We have introduced how to measure the similarity score between two words. Here we take an example to illustrate how to get the similarity score between two


(a)

	cheap	online	hosting	Japan
domain	0.3425	0.7012	0.7321	0.2974
free	0.6302	0.6125	0.5785	0.3214
use	0.3719	0.4872	0.4875	0.3042
year	0.3657	0.4214	0.4952	0.4321

(b)

0.3425	0.7012	**0.7321**	0.2974
0.6302	0.6125	0.5785	0.3214
0.3719	0.4872	0.4875	0.3042
0.3657	0.4214	0.4952	0.4321

(c)

0.6302	0.6125		0.3214
0.3719	0.4872		0.3042
0.3657	0.4214		0.4321

(d)

	0.4872		0.3042
	0.4214		0.4321

(e)

			0.4321

Figure 3 Similarity measurement between two sentences.

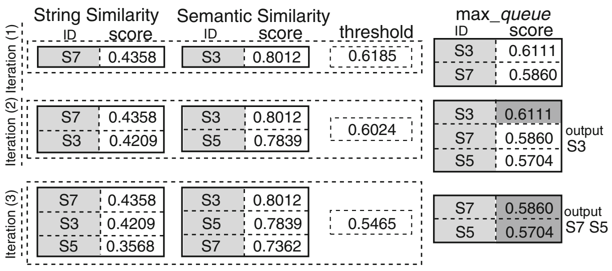


Figure 4 Efficient searching top-k semantic sentences framework.

sentences [24]. Let $P =$ "*Cheapest online hosting in Japan.*", $Q =$ "*Domain is free for the first year*". After removing all stop words and lemmatizing, we obtain $P = \{$*cheap online hosting Japan*$\}$ and $Q = \{$*domain free use year*$\}$. Through similarity measuring on each combination of word pair for the sentences, we construct a similarity matrix which is illustrated in Figure 3. Here, each element in the matrix denotes the final similarity score of the word pair, i.e., the linear aggregation of string-based similarity and corpus-based similarity. To obtain the similarity between two sentences, we first find the maximal-valued element which is regarded as the representative word. Then the related row and column which includes this representative word are removed. We recursively execute this process and finally all the similarities of the representative words have been extracted. The similarity between these two sentences is computed as follows:

$$S(P, Q) = \frac{\Sigma_{i=1}^{|\min(|P|,|Q|)|} \rho_i(|P| + |Q|)}{2|P| * |Q|} \tag{6}$$

where ρ_i is the value of the representative word of round i, $|P|$ and $|Q|$ are the length of P and Q respectively. Therefore we have

$$S(P, Q) = \frac{(0.7321 + 0.6302 + 0.4872 + 0.4321)(4 + 4)}{2 * 4 * 4} = 0.5704.$$

4.4 Assembling similarity features

We introduce an efficient assembling approach to hasten the process of searching top-k similar sentences based on the rank aggregation algorithm [11]. The key idea is as follows. Suppose there are two ordered lists (in descending order) A and B. We denote A_i as the element which ranked i in the A list. Let t_i be the threshold that $t_i = w_A \cdot Sim_{A_i} + w_B \cdot Sim_{B_i}$, where w is the weight value[4]. If an element P whose total score is not smaller than t_i, there is at least in one list that the rank of P is higher than or equal to i. This intuition introduces a possible way to progressively output the top-k results. To illustrate the process, we use the concrete example (i.e., Figure 1) to explain the process, as shown in Figure 4.

We use a *max_queue* to store the intermediate candidates. In the first iteration, we obtain the top-1 sentences with their scores in the features, i.e. S7: 0.4358 and S3: 0.8012. Next the threshold is computed, i.e., 0.6185. Because the overall scores

[4]The weights are set to 1/2, respectively, as explained in Section 3.

of the two accessed sentences are smaller than the threshold, no result is output in this round. In the second iteration (i.e., testing on top-2 sentences in both lists), the threshold is computed as 0.6024. Because the score of S3 is 0.6111 which is larger than the threshold, S3 can be output immediately. We can see that for this example, the performance of obtaining the top-1 result is very efficient because we do not need to evaluate many candidates. The remaining processes are executed in a similar way, as illustrated in Figure 4.

Because of the general property of the rank aggregation algorithm [11], the element in the lists can be word. As such, we apply the threshold-based strategy in assembling different string similarities (i.e., NLCS, NMCLCS$_1$, and NMCLCS$_n$), and assembling words into sentence to obtain the top elements. We take an concrete example to illustrate our proposal which is presented in Figure 5. We use the same data collection which has been introduced in Figure 1. For example, the query sentence is: “Cheapest online hosting in Japan”. After removing the stopwords and stemming, the query is: “cheap online host Japan”. From Figure 5 we can see the hierachical structure of our framework. If we want to retrieval top-1 result from the data collection, we firstly should know the ranked list of string similarity base and semantic similarity base respectively. The top-1 result comes from these two ranked list based on the property of threshold algorithm. However, we cannot know such ranking directly because string similarity base is composed of three different string

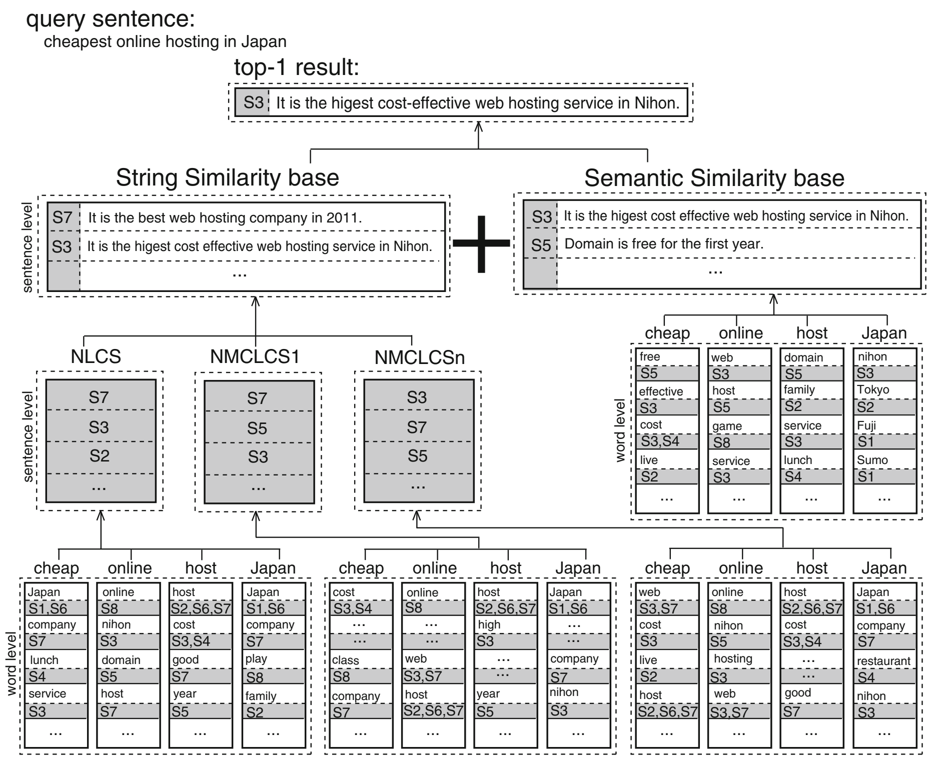


Figure 5 The framework of the top-*k* similar sentences extraction.

measurement strategies, i.e., NLCS, NMCLCS$_1$ and NMCLCS$_n$. Unfortunately, we cannot know the ranked list of NLCS directly because NLCS has the lower layer, i.e., word level. Here, the words are query words and each query word corresponds a list. We take an example to illustrate how to obtain top-1 result of NLCS. Be different with the traditional threshold algorithm, our method here should take the sentence ID into consideration. For list **cheap**, white rectangle denotes the word and grey one indicates its' corresponding sentence ID. We can see that, "Japan" appears in both list **cheap** and list **Japan**. However, such top value only one word of sentence S1 and S6 but no other words of these two sentences can achieve top value in the other lists. Although word "company" for S7 stands the second position in list **cheap**, "host" for S7 in list **online** and "good" for S7 in list **host** make sentence S7 dominate the other sentences and it becomes the top-1 value of NLCS. Accordingly, we can obtain the other two top-1 value of NMCLCS$_1$ and NMCLCS$_n$. However, sometimes we may go deeply to get the corresponding words for a related sentence, e.g., list **cheap** of NMCLCS$_1$ (We use ellipsis to illustrate tested words because of the figure length limitation.), we still can obtain top-k value by accessing a small part of such list. For semantic similarity base, the top-k value can be retrieved from 4 words lists directly. The final top-1 value comes from two similarity bases and it is S3 in this example.

4.5 Space and time complexity

In this section, we present the complexity analysis on space and time for the introduced strategies.

4.5.1 Space complexity

For the space complexity, let n be the size of data collection, m be the number of features considered, p be the average length of sentences and l be the average length of words.

NLCS Based on the definition of NLCS in (1), we should know all the possible cases of grams, i.e., we should take all cases into consideration. For example, word *host*, the indices of NLCS should be:

$$h, o, s, t, ho, hs, ht, os, ot, st, hos, hot, ost, host.$$

So, we should index all these grams for NLCS. However, the number of the index grams is related with the length of gram. Therefore, it is a combinations problem for each gram. We take *host* as an example. When length is "1", the number of indices is "4", i.e., C_4^1. Accordingly, the number of indices are C_4^2, C_4^3, C_4^4 when gram length are "2", "3" and "4" respectively. Therefore, the total number of indices are: $C_4^1 + C_4^2 + C_4^3 + C_4^4$. It can be regarded as a summation of binomial coefficients problem. In general, the total number of indices are: $C_l^1 + C_l^2 + ... + C_l^l$. So, the space cost of NLCS is: $S_{NLCS} = n * (2^l - 1)$, the space complexity is $O(n * 2^l)$.

NMCLCS$_1$ Because NMCLCS$_1$ only need the grams which have the same first character. So, for word *host*, the indices of NMCLCS$_1$ should be: h, ho, hos, $host$.

From this example we can easily see that the number of indices is the same as the word length. In general, the space cost of NMCLCS$_1$ is: $S_{NMCLCS_1} = n * l$, the space complexity is $O(n * l)$.

NMCLCS$_n$ NMCLCS$_n$ contains all the possible consecutive grams. So the index of NMCLCS$_n$ for word *host* are:

$$h, o, s, t, ho, os, st, hos, ost, host.$$

Be different with NLCS, we should only index the consecutive grams. We also take *host* as an example. When length is "1", the number of indices is "4". Accordingly, when length is "2", "3", "4", the number of indices is "3", "2" and "1" respectively. The total number of indices can be presented as: $4 + (4 - 1) + (4 - 2) + (4 - 3)$. Generally, the total number of indices is: $n + (n - 1) + ... + (n - (n - 1))$, i.e., $n + (n - 1) + ... + 1$. So, it is a summation of arithmetic sequence problem. Therefore, the space cost is: $S_{NMCLCSn} = n * \frac{l(l+1)}{2}$, the space complexity is: $O(n * l^2)$.

Semantic From the analysis of [65], we can see that, for a given word, we should index all the frequency of the words in the data collection. The space complexity for semantic is the size of data collection, i.e., $O(n)$.

Therefore, the whole space cost is to integrate all the similarity cost above, which is: $O(n * (2^l + l^2))$.

4.5.2 Time complexity

Online query processing In our proposed algorithm, let n be the size of data collection, m be the number of features considered, p be the average length of sentences and l be the average length of words, the performance cost of our proposed algorithm is list as follows: we retrieval top-k values based on threshold algorithm [11]. Based on the property of the threshold algorithm, our measure of cost corresponds intuitively to the cost of access ranked lists, i.e., *sorted access cost* and *random access cost*. The *sorted access cost* is the total number of items obtained under sorted access. For example, in our proposed strategy, we have two lists, and the algorithm requests altogether the top-n_{l_1} items from the first list and the top-n_{l_2} items from the second list, then the sorted access cost is $n_{l_1} + n_{l_2}$. Similarily, the *random access cost* is the total number of items obtained under random access. Based on the property of the threshold algorithm [11], the time complexity is $O(n^{\frac{m-1}{m}} k^{\frac{1}{m}})$.

Offline Preprocessing We should index all the words of the data collection based on order of each strategy. Since there are 4 types of indices, i.e., NLCS, NMCLCS$_1$, NMCLCS$_n$ and semantic, we introduce the analysis one by one. For NLCS, the indices are related with the length of word (we set l in this paper). For example, we have a word *host*. Based on the property of NLCS, we parse *host* into *h,o,s,t,ho,hs,ht,os,ot,st,hos,hot,ost,host*. Based on the space cost analysis, the time cost of preprocessing on NLCS is: $O(2^l - 1)$ and time complexity is: $O(2^l)$. Be similar to NLCS, NMCLCS$_1$ and NMCLCS$_n$ are $O(l)$ and $O(l^2)$ respectively. The time cost for semantic is only related with the size of data collection. For each word, the time complexity is: $O(1)$. So, $O(n)$ is the time cost for semantic preprocessing. Therefore, for total n words, the time complexity of preprocessing is: $O(n * (2^l + l^2))$.

5 Experimental evaluation

To evaluate our approach, we conducted extensive experiments by using 16-core Intel(R) Xeon(R) E5530 server running Debian 2.6.26-2. All the algorithms were written in C language and compiled by GNU gcc. The baseline algorithm is implemented according to the state-of-the-art work [24]. Be the same parameters setting as [24], the total similarity is defined as: $Sim = 0.5 * Sim_{\text{string}} + 0.5 * Sim_{\text{semantic}}$. For $Sim_{\text{string}} = \frac{1}{3} * Sim_{NLCS} + \frac{1}{3} * Sim_{NMCLCS_1} + \frac{1}{3} * Sim_{NMCLCS_n}$, $Sim = \frac{1}{6} * Sim_{NLCS} + \frac{1}{6} * Sim_{NMCLCS_1} + \frac{1}{6} * Sim_{NMCLCS_n} + \frac{1}{3} * Sim_{\text{semantic}}$.

In the whole experimental evaluation, we use three different datasets, i.e., the *benchmark* dataset which was used in [24, 35], BNC dataset (extracted from British National Corpus) and MSC dataset [10] (extracted from Microsoft Research Paraphrase Corpus). Table 1 shows the statistics of these three datasets. In this table, type A indicates the original dataset and B is after preprocessing(i.e., removing stopwords).

5.1 Evaluation on efficiency

5.1.1 Effect of size of data collection

To evaluate the efficiency of our proposal, we conducted experiments on the datasets come from BNC and MSC. We randomly extracted 1k, 5k, 10k, 20k sentences from BNC and divided MSC into different size parts, i.e.,10 %, 20 %, 50 %, 100 % as our datasets. Figure 6 shows the top-5 results under 10 randomly selected queries. We can see that our proposal is much faster than the baseline strategy for both datasets because the proposal largely reduces the number of candidates tested. With the size of data collection increasing, the query time of our proposal is increasing linearly and scaling well.

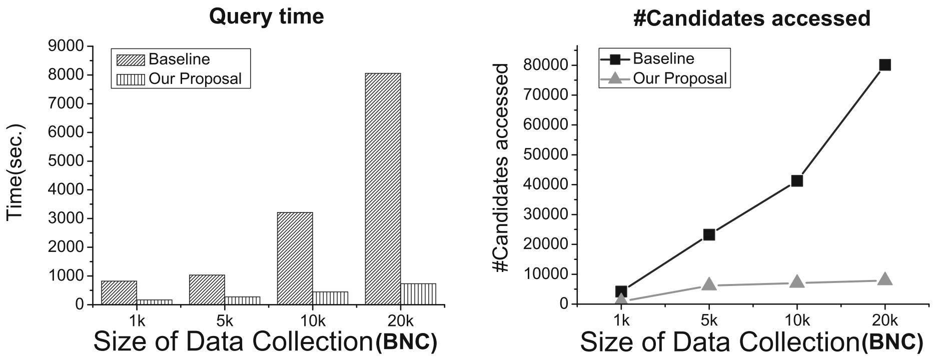
5.1.2 Effect of k value

The effect of k is evaluated in this section. We randomly chose 10 queries from the data collections. The size of BNC was fixed to 5k and the whole MSC was used. From Figure 7 we can see that the baseline needs to access all the candidate sentences and

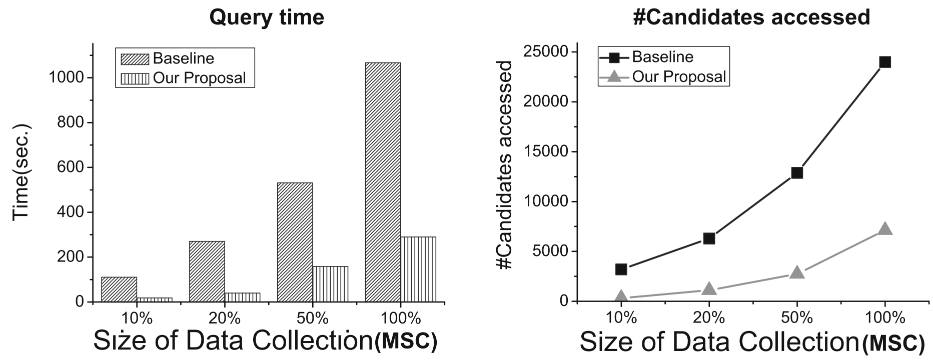
Table 1 Dataset statistics.

	MSC		Benchmark		BNC	
Labeled?	YES (“0” or “1”)		YES (Concrete value)		NO	
Size (num. of sentences)	11.2k*		49*		20k	
Type	(A)	(B)	(A)	(B)	(A)	(B)
Avg. sentence length	15.23	9.31	14.12	6.2	11.72	7.19
Min. sentence length	5	3	5	2	3	2
Max. sentence length	41	25	33	12	107	38
Max word length	13	13	14	14	17	17

*The number is the distinct sentence number in both datasets. The original data of MSC is 5,801 sentence pairs. We have removed the duplicated sentences and 11,242 sentences are remained. The original data of benchmark dataset includes 65 sentence pairs.



(a) BNC dataset



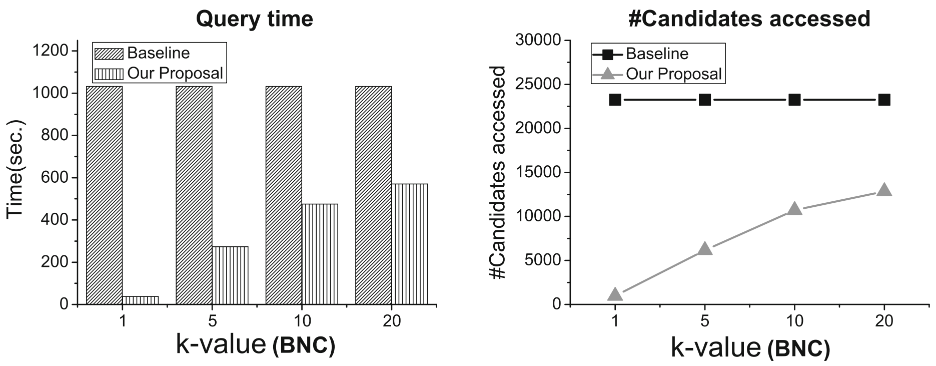
(b) MSC dataset

Figure 6 Effect on size of data collection.

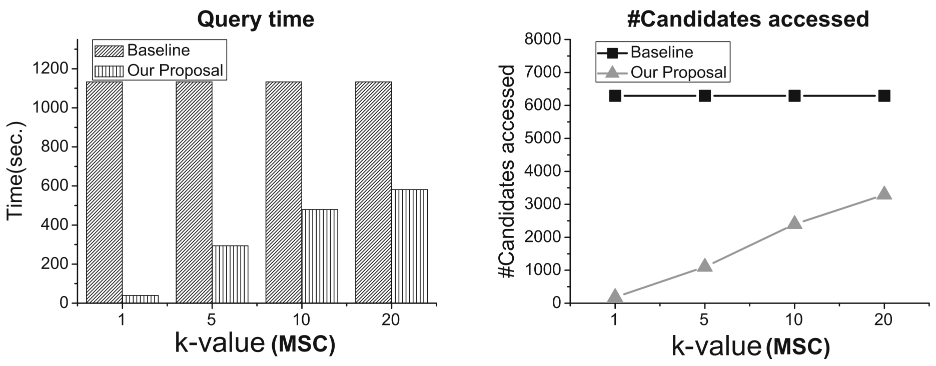
thus, the query time is the same for all the situations. For our proposal, the top-1 result can be returned almost instantly. When *k* increases, the query time increases because more candidates need to be evaluated.

5.1.3 Effect of sentence length

We evaluate the effect of the sentence length. There are two types of lengths, i.e., query sentence length and average sentence length in the data collection. Figure 8a illustrates the effect on the query sentence length for the BNC and MSC datasets. We can see that the query time increases when the query length increases. The reason is that more candidates are tested. We randomly chose sentences with the average length be 2, 4, 6, 8 from BNC and MSC. The query length was kept as 4. From the results we can see that, the query time does not increase when the average sentence length increases. The reason is that the similarity between two sentences mainly depends on the length of the shorter one. Figure 8b shows the results w.r.t. different lengths of sentences in the data collection. We can see that when the average sentence length is larger than the query length, the query time does not change.



(a) BNC dataset



(b) MSC dataset

Figure 7 Effect of k-value.

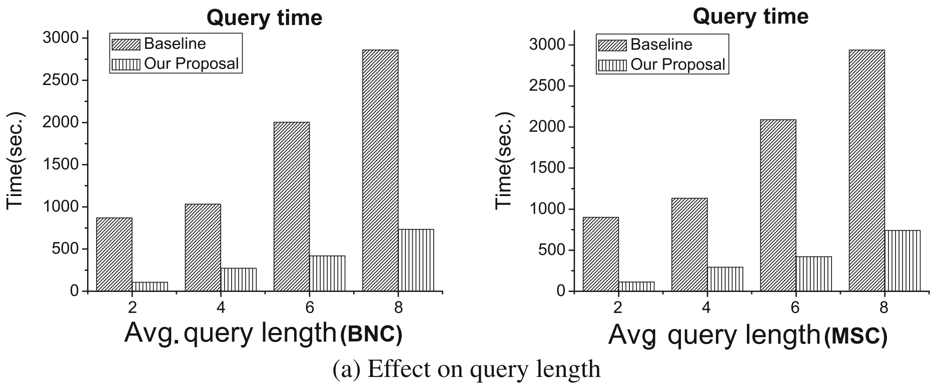
5.2 Evaluation on effectiveness

In the former experiments, we have demonstrated that our proposal outperforms the state-of-the-art technique with regard to the efficiency issue. In this section, we evaluate the effectiveness of our proposal. We conduct experiments on two labeled datasets, i.e., the *benchmark* dataset and MCS dataset.

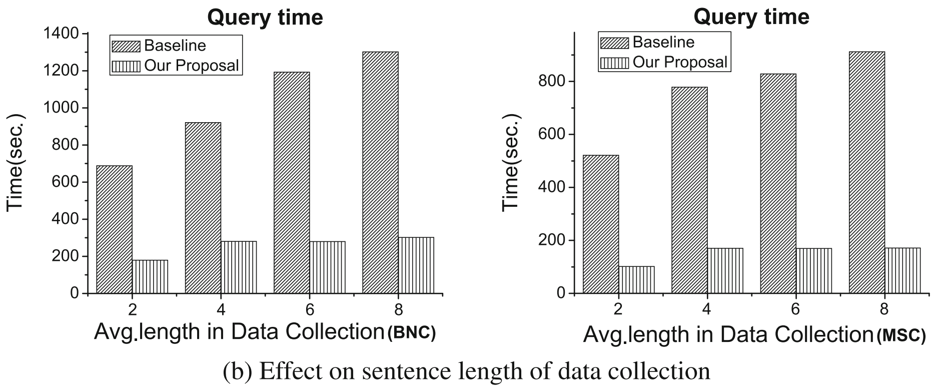
5.2.1 Evaluation on precisely labeled dataset

The *benchmark* dataset which was used in [24, 35] and it has been labeled by concrete value, i.e., we can easily understand the closeness between measured result and labeled data. Here, we denote **Distance** as the metric which measures the closeness between two sets of results, e.g., baseline and our proposal, where x_i and y_i are two sets of results respectively. Smaller value indicates that the two results are closer to each other.

$$\mathbf{Distance} = \frac{1}{n}\sum_{i=0}^{n-1}(x_i - y_i)^2$$



(a) Effect on query length



(b) Effect on sentence length of data collection

Figure 8 Effect on sentence length.

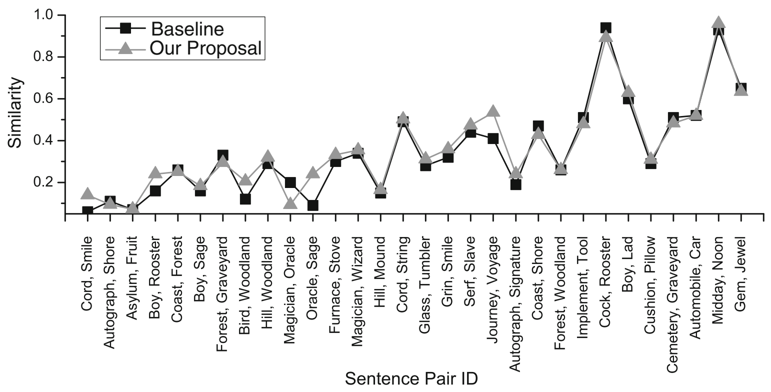


Figure 9 Evaluation on effectiveness under benchmark dataset.

Table 2 The selection of τ.

Labeled value	$\geq \tau$	$< \tau$
1	Hit	Miss
0	Miss	Hit

The experimental result conducted on the *benchmark* dataset is illustrated in Figure 9. From this figure we can see that the results of both algorithms are close to each other, which indicates that our proposal can obtain the same high precision as the state-of-the-art technique.

5.2.2 Evaluation on rawly labeled dataset

MSC is another labeled dataset which has been extracted from news sources on the web, along with human annotations indicating whether each pair captures a paraphrase-semantic equivalence relationship. They apply the value 1 as similar while the value 0 as dissimilar. However, such label data is rather raw and we cannot apply them to judge our measured results directly.

Raw data preprocessing Since such dataset has no concrete value while only “1” or “0”, we should distinct each value beforehand. Consider an example, there are two cases. Case (a): the similarity score between two sentences is 0.30 while the labeled data is 1. Case (b): the similarity score between two sentences is 0.60 while the labeled data is 0. From this example we can see that we cannot judge 0.30 or 0.60, which one represents “closeness”. So, how to measure our experimental results by using such raw data. To answer this question, we apply a threshold τ as the bound, i.e., if any measured value is greater than or equal to τ and the labeled data is “1” , we regard it as “hit”. Similarly, if one measured value is less than τ and the labeled data is “0”, we also regard it as “hit”, otherwise “miss”. Table 2 illustrates the strategy.

Randomly selecting 20 sentence pairs from MSC and set different τ value, we can obtain the result which is presented in Table 3. Here we set hit ratio be the number of hits divided by total results. With these different threshold τ, we select the optimal one that obtains the maximal hit ratio, i.e., $\tau = 0.3$ in this example.

Accordingly, we conducted extensive experiments to search the best τ value. Since we know that the similarity score comes from two different parts, i.e., string similarity and semantic similarity, and the weight α we apply before is 0.5. To evaluate the precise hit ratio under different τ, we evaluate the whole hit ratio by using MSC dataset under different τ and α value. Table 4 shows the experimental results.

Table 3 Different hit ratio under different selection of τ.

Threshold	Hit	Miss	Hit ratio (%)
0.7	0	20	0
0.6	1	19	5
0.5	9	11	45
0.4	12	8	60
0.3	17	3	85
0.2	15	5	75
0.1	14	6	70

Table 4 Hit ratio under different τ and α.

	$\tau = 0.1$ (%)	0.2 (%)	<u>0.3</u> (%)	0.4 (%)	0.5 (%)	0.6 (%)	0.7 (%)	0.8 (%)	0.9 (%)
$\alpha = 0$	20.55	21.20	36.25	41.29	36.06	34.18	18.86	16.89	16.43
0.1	17.74	24.67	41.29	41.39	33.24	28.44	18.74	15.14	12.27
0.2	56.73	55.25	53.51	51.44	46.13	34.80	30.32	26.62	22.76
0.3	69.28	72.33	68.70	63.39	55.34	49.53	34.67	22.51	18.69
0.4	77.71	78.57	68.85	57.32	51.46	47.66	36.36	28.44	22.31
0.5	77.81	79.16	77.59	74.37	63.37	51.27	40.41	32.77	32.77
<u>0.6</u>	71.02	76.87	**80.52**	71.70	63.21	53.28	39.05	33.86	22.29
0.7	66.30	71.07	72.32	63.06	61.42	39.91	36.05	30.01	18.82
0.8	34.84	67.63	69.92	57.30	50.99	36.29	34.22	30.10	21.32
0.9	29.62	36.32	61.92	53.28	46.18	41.13	36.29	30.81	17.95
1.0	20.84	22.44	21.34	18.86	16.95	15.14	12.72	11.58	10.36

From the table, we can see, when $\tau = 0.3$ under $\alpha = 0.6$, we obtain the best hit ratio as 80.52 %. However, the evaluation of each τ is under a fixed α, i.e., we cannot determine the combination of τ and α which can obtain the best hit ratio. Therefore, we should extensively train the value of α under different τ.

Training α under different τ We apply a k-fold cross-validation[5] strategy to check α under different τ. Firstly, we divide MSC dataset into k parts and apply the part from 1 to $k-1$ as the training data. Then we get α and τ under the best hit ratio and apply these two parameters to compute the hit ratio on the kth part. In each step i, we can get the best hit ratio H_i and the hit ratio (test) Ht_i. For example, in the 1st step, the best hit ratio occurs when $\alpha = 0.6$ and $\tau = 0.3$. Then we get the 81.24 % hit ratio in testing part which means there is no distinguish difference between training and testing. After k steps finish, we get a list of values of α, τ and hit ratios. The finial α and τ will be the average value. From Table 5, we can see that $\tau = 0.56$ and $\alpha = 0.31$ are the best parameters. We show the evaluation on the effectiveness for the MSC dataset in Figure 10 under these two parameters.

5.3 Evaluation on trade-off between efficiency and effectiveness

Because the introduced framework is built based on the aggregation of different features, i.e., string similarity and semantic similarity, the execution time is related to the number of features used. Therefore, if we apply only one feature, the execution time is shorter yet such strategy may affect the whole effectiveness. So we conduct a set of experiments (illustrated in Figure 11a) to study the trade-off between efficiency and effectiveness. In this set of experiments, we first evaluate the performance of single feature in the baseline strategy vs. our whole framework, as illustrated in Figure 11a. Then we explore the performance of single feature in the baseline strategy vs. single feature in our framework as illustrated in Figure 11b.

[5]We apply 10-fold cross-validation in this paper.

Table 5 10-fold cross-validation on τ under different α.

	Hit ratio H_i (%)	Parameter α	Threshold τ	Hit-ratio (test) Ht_i (%)
1st	81.25	0.7	0.3	81.58
3rd	78.72	0.7	0.4	82.79
4th	80.46	0.5	0.3	80.03
5th	80.67	0.4	0.3	84.51
6th	84.18	0.4	0.4	71.26
7th	79.10	0.5	0.3	79.35
8th	82.16	0.6	0.3	77.28
9th	82.07	0.6	0.2	75.39
10th	80.42	0.6	0.3	79.00
	Avg.	0.56	0.31	

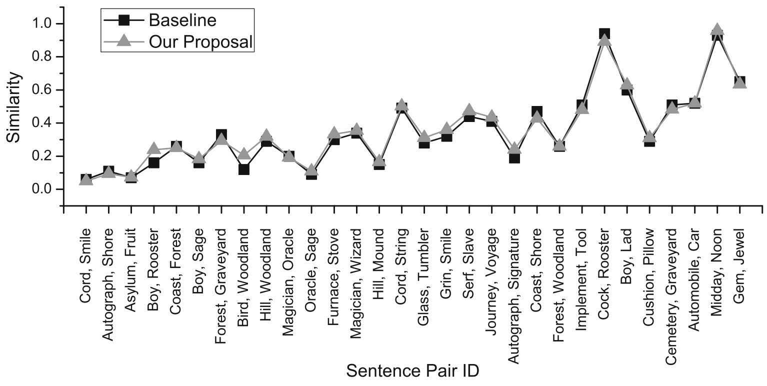


Figure 10 Evaluation on effectiveness under MSC.

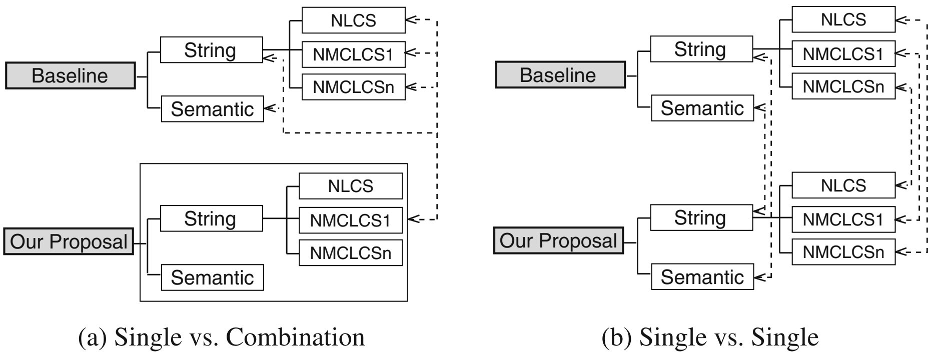
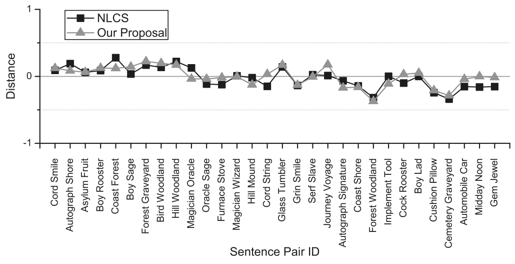
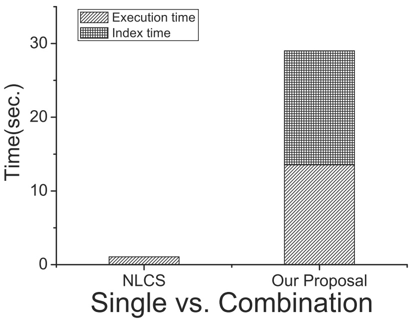


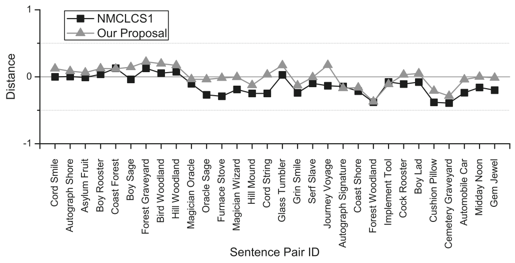
Figure 11 Two different evaluation strategies on trade-off between efficiency and effectiveness.



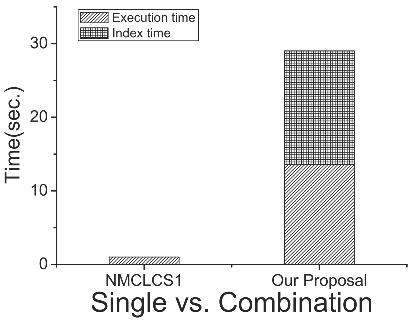
(a) NLCS vs. Our Proposal



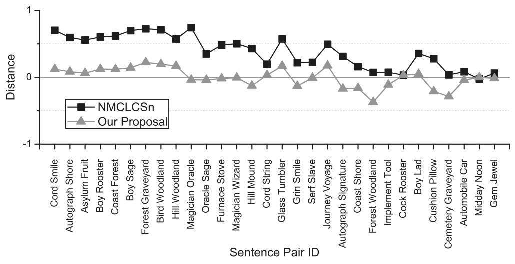
(b) NLCS vs. Our Proposal



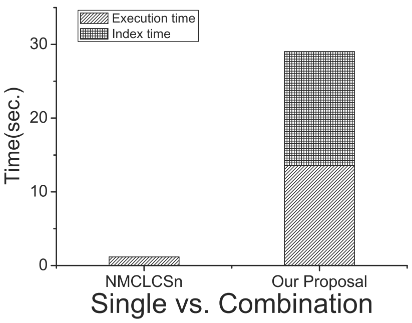
(c) NMCLCS$_1$ vs. Our Proposal



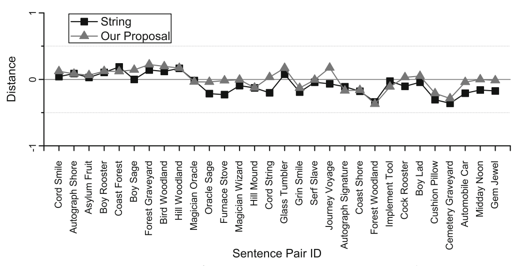
(d) NMCLCS$_1$ vs. Our Proposal



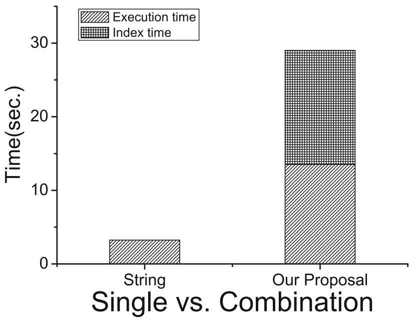
(e) NMCLCS$_n$ vs. Our Proposal



(f) NMCLCS$_n$ vs. Our Proposal

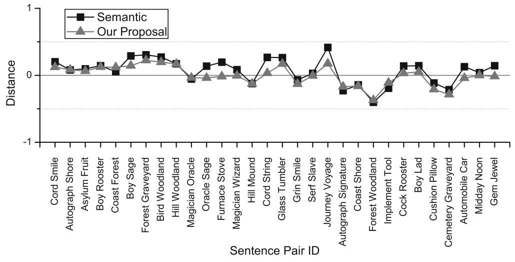


(g) String vs. Our Proposal

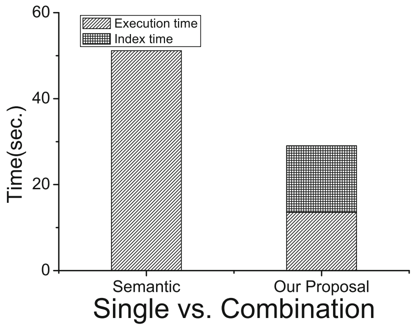


(h) String vs. Our Proposal

Figure 12 Evaluation on trade-off between efficiency and effectiveness (cont'd).



(i) Semantic vs. Our Proposal



(j)Semantic vs. Our Proposal

Figure 12 (continued)

5.3.1 Single strategy in the baseline vs. our proposal

To evaluate the effectiveness, we apply single strategy in baseline and combination strategy[6] in our proposal. Firstly, we compare the effectiveness between each strategy in baseline and combination strategy in our proposal. We also evaluate the execution time and index time of each pair under such strategy. The experimental result is illustrated in Figure 12. We report three different single string strategy in Figure 12a, c and e. String strategy and semantic strategy results are listed in Figure 12g and i, respectively. From the figure we can see that, single strategy beats our proposal in execution time while not in the effectiveness.

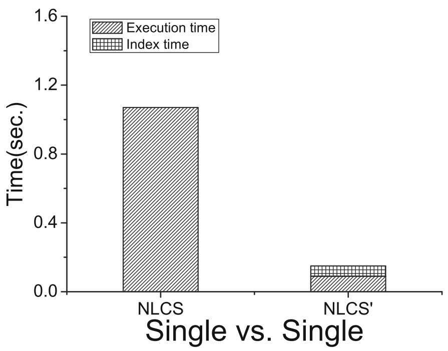
The former evaluation on effectiveness tells us that the combination strategy can obtain more precise results. Figure 12b, d, f, h, and j present the experimental results of execution time of single strategy in baseline and execution time of our proposal. Since the strategies in baseline do not need to index, we report the index time of combination strategy in our proposal. Note that in all the evaluation, we show the performance of extracting the top-5 results with 10 randomly selected queries. Here we take NLCS vs. combination strategy pair as an example. Figure 12b illustrates the execution time and the index time for NLCS (i.e., the left bar) and our proposal (i.e., the right bar). We can easily see that the execution time of NLCS is very fast while the combination strategy consumes more time. The similar results can be seen in Figure 12d, f and h. In Figure 12j, the execution time of single semantic strategy is longer than that of combination strategy. Such result tells us that the optimization on semantic similarity is crucial among all the optimization strategies.

5.3.2 Single strategy vs. single strategy

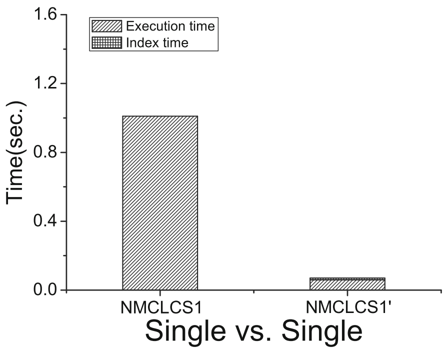
Evaluation on single strategy vs. combination strategy demonstrates the trade-off between effectiveness and efficiency. In this section, we study the performance of single feature in the baseline strategy vs. single feature in our framework (i.e., Figure 11b). Firstly, we compare the effectiveness between the two situations, as

[6]Combination strategy means the whole framework strategies.

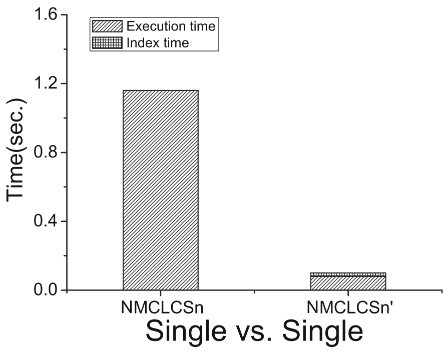
illustrated in Figure 13. From the figure we can see that the execution time of each single strategy in baseline is longer than that of in our proposal (i.e., including the execution time and index time). These results demonstrate that the optimizations of our proposal are efficient and make effect on each feature.



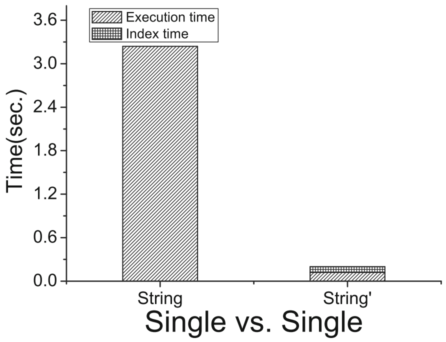
(a) NLCS vs. Our NLCS′



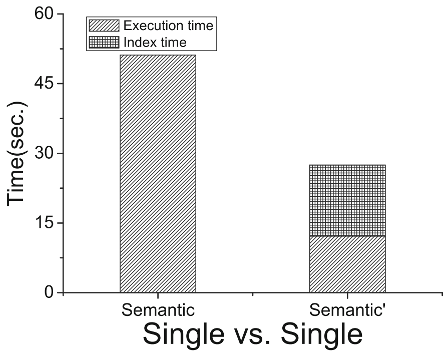
(b) NMCLCS_1 vs. NMCLCS_1'



(c) NMCLCS_n vs. NMCLCS_n'



(d) String vs. String′



(e) Semantic vs. Semantic′

Figure 13 Evaluation on efficiency and index cost between single strategy in baseline and combination strategy in our proposal.

6 Discussion

Based on the rank aggregation algorithm introduced in Section 4.4, most other (if not all) similarity metrics can be incorporated into our general framework. The essential issue is how to build an effective corresponding index for each similarity. In this section, we discuss the optimizations on these similarity measurement strategies.

ESA [13] is a method which represents the meaning of texts in a high-dimensional space of concepts derived from Wikipedia. It applies machine learning techniques to explicitly represent the meaning of any text as a weighted vector of Wikipeida-based concept. Since we compute the relatedness between two texts by semantic interpreter, i.e., each text fragment should be represented by weighted list of Wikipedia concepts, we can build an inverted list for all the words in the data collection by using these concepts. Optimization techniques, such as that proposed in [55, 60], e.g., divide-skip-merge, can be employed to efficiently extract the top-*k* similar sentences.

For the information content based measurement strategy [49], the authors present a semantic similarity measurement in an IS-A taxonomy (i.e., Wordnet). The similarity of two words is estimated as the correlation degree between the description texts on the words. We can first build a *q*-gram inverted list index. Each word in the information content can be considered as a unigram. Thereafter, the optimization strategies introduced in [66], e.g., count filtering, can be applied.

Edit distance (a.k.a, the Levenshtein distance) is a string metric for measuring the syntax difference between two sequences. Informally, the edit distance between two words is equal to the number of single-character edits required to change one word into the other. Many algorithms have used *q*-gram based strategies [42, 43, 59] to measure the edit distance between words. In [66] the authors proposed several strategies, e.g., adaptive *q*-gram selection, to efficiently retrieval the top-*k* results. These approaches can be utilized in our framework.

Edge counting strategy [26, 32, 45, 64] measures the similarity as the shortest path between words in some knowledge base (i.e., Wordnet). To efficiently discover the top-*k* similar words, in [65], the authors firstly build an index that associates nodes in WordNet to words in the data collection. Then they expand the nodes from the query word to retrieval the top-*k* results progressively. We can also apply the optimization method to efficiently obtain the top-*k* sentences in a similar way.

7 Conclusion and future work

In this paper, we proposed to tackle the efficiency issue of searching top-*k* similar sentences which has not been studied before. The issue is very important especially when the data is large. Several efficient strategies are introduced to test as few candidates as possible during the searching process. The comprehensive experimental evaluation demonstrates the efficiency of the proposed techniques while keeping the same high effectiveness as the state-of-the-art techniques. We conducted extensive experiments on different types of datasets. To evaluate the effectiveness, we conducted on two different types of datasets (labeled and unlabeled) and trained the parameters for the efficiency evaluation. To understand the trade-off between effectiveness and efficiency, we evaluated different combination of features between the baseline and our proposal. In the future, we will incorporate more similarity metrics to evaluate the efficiency and effectiveness of the framework.

References

1. Atallah, M.J., Fox, S.: Algorithms and Theory of Computation Handbook, 1st edn. CRC Press, Inc. (1998)
2. Blake, M.B., Cabral, L., König-Ries, B., Küster, U., Martin, D.: Semantic Web Services: Advancement through Evaluation. Springer (2012)
3. Bollegala, D., Matsuo, Y., Ishizuka, M.: Measuring semantic similarity between words using web search engines. In: Proceedings of the International Conference on World Wide Web, WWW'07, pp. 757–766 (2007)
4. Burgess, C., Livesay, K., Lund, K.: Explorations in context space: words, sentences, discourse. Discourse Process. **25**, 211–257 (1998)
5. Ceccarelli, D., Lucchese, C., Orlando, S., Perego, R., Silvestri, F.: Caching query-biased snippets for efficient retrieval. In: Proceedings of the International Conference on Extending Database Technology, EDBT/ICDT '11, pp. 93–104 (2011)
6. Chowdhury, G.G.: Introduction to Modern Information Retrieval, 3rd edn. Facet (2010)
7. Cohen, W.W.: Integration of heterogeneous databases without common domains using queries based on textual similarity. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD '98, pp. 201–212 (1998)
8. Cui, H., Sun, R., Li, K., Kan, M.Y., Chua, T.S.: Question answering passage retrieval using dependency relations. In: Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '05, pp. 400–407 (2005)
9. Damerau, F.J.: A technique for computer detection and correction of spelling errors. Commun. ACM **7**(3), 171–176 (1964)
10. Dolan, B., Quirk, C., Brockett, C.: Unsupervised construction of large paraphrase corpora: exploiting massively parallel news sources. In: Proceedings of the International Conference on Computational Linguistics, COLING '04, pp. 350–356 (2004)
11. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. In: Proceedings of the ACM SIGMOD symposium on Principles of Database Systems, PODS '01, pp. 102–113 (2001)
12. Foltz, P.W., Kintsch, W., Landauer, T.K.: The measurement of textual coherence with latent semantic analysis. Discourse Process. **25**, 285–307 (1998)
13. Gabrilovich, E., Markovitch, S.: Computing semantic relatedness using wikipedia-based explicit semantic analysis. In: Proceedings of the International Joint Conference on Artifical Intelligence, IJCAI'07, pp. 1606–1611 (2007)
14. Goyal, A., Daumé III, H.: Approximate scalable bounded space sketch for large data nlp. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP '11, pp. 250–261 (2011)
15. Goyal, A., Daumé III, H., Venkatasubramanian, S.: Streaming for large scale nlp: language modeling. In: Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics, NAACL '09, pp. 512–520 (2009)
16. Hamming, R.W.: Error detecting and error correcting codes. Bell Syst. Tech. J. **29**(2), 147–160 (1950)
17. Han, W.S., Lee, J., Moon, Y.S., Jiang, H.: Ranked subsequence matching in time-series databases. In: Proceedings of the International Conference on Very Large Databases, VLDB '07, pp. 423–434 (2007)
18. Hatzivassiloglou, V., Klavans, J.L., Eskin, E.: Detecting text similarity over short passages: Exploring linguistic feature combinations via machine learning. In: Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora, EMNLP/VLC '99, pp. 203–212 (1999)
19. Hellerstein, J.M., Avnur, R., Chou, A., Hidber, C., Olston, C., Raman, V., Roth, T., Haas, P.J.: Interactive data analysis: the control project. IEEE Comput. **32**(8), 51–59 (1999)
20. Hirschberg, D.S.: A linear space algorithm for computing maximal common subsequences. Commun. ACM **18**(6), 341–343 (1975)

21. Hua, M., Pei, J., Fu, A.W., Lin, X., Leung, H.F.: Top-*k* typicality queries and efficient query answering methods on large databases. VLDB J. **18**(3), 809–835 (2009)
22. Hua, M., Pei, J., Fu, A.W.C., Lin, X., Leung, H.F.: Efficiently answering top-*k* typicality queries on large databases. In: Proceedings of the International Conference on Very Large Databases, VLDB '07, pp. 890–901 (2007)
23. Islam, A., Inkpen, D.: Second order co-occurrence pmi for determining the semantic similarity of words. In: Proceedings of the International Conference on Language Resources and Evaluation, LREC '06, pp. 1033–1038 (2006)
24. Islam, A., Inkpen, D.: Semantic text similarity using corpus-based word similarity and string similarity. ACM Trans. Knowl. Discov. Data **2**(2), 1–25 (2008)
25. Jaro, M.A.: Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida. J. Am. Stat. Assoc. **84**(406), 414–420 (1989)
26. Jiang, J.J., Conrath, D.W.: Semantic similarity based on corpus statistics and lexical taxonomy. CoRR **cmp-lg/9709008** (1997)
27. Jones, K.S.: A statistical interpretation of term specificity and its application in retrieval. J. Doc. **28**(1), 11–20 (1972)
28. Kim, J.W., Kashyap, A., Li, D., Bhamidipati, S.: Efficient wikipedia-based semantic interpreter by exploiting top-k processing. In: Proceedings of the International Conference on Information and Knowledge Management, CIKM '10, pp. 1813–1816 (2010)
29. Koren, J., Zhang, Y., Liu, X.: Personalized interactive faceted search. In: Proceedings of the International Conference on World Wide Web, WWW '08, pp. 477–486 (2008)
30. Landauer, T.K., Dumais, S.T.: A solution to Plato's problem: The latent semantic analysis theory of the acquisition, induction, and representation of knowledge. Psychol. Rev. **104**, 211–240 (1997)
31. Landauer, T.K., Folt, P.W., Laham, D.: An introduction to latent semantic analysis. Discourse Process. **25**(2), 259–284 (1998)
32. Leacock, C., Chodorow, M.: Combining local context and wordnet similarity for word sense identification. In: Fellbaum, C. (ed.) WordNet: An Electronic Lexical Database, pp. 305–332. MIT Press (1998)
33. Levenshtein, V.: Binary codes capable of correcting deletions, insertions, and reversals. Sov. Phys. Dokl. **10**(8), 707–710 (1966)
34. Li, Y., Bandar, Z.A., McLean, D.: An approach for measuring semantic similarity between words using multiple information sources. IEEE Trans. Knowl. Data Eng. **15**(4), 871–882 (2003)
35. Li, Y., McLean, D., Bandar, Z., O'Shea, J., Crockett, K.A.: Sentence similarity based on semantic nets and corpus statistics. IEEE Trans. Knowl. Data Eng. **18**(8), 1138–1150 (2006)
36. Maguitman, A.G., Menczer, F., Roinestad, H., Vespignani, A.: Algorithmic detection of semantic similarity. In: Proceedings of the International Conference on World Wide Web, WWW '05, pp. 107–116 (2005)
37. Maynard, D., Greenwood, M.A.: Large scale semantic annotation, indexing and search at the national archives. In: Proceedings of the International Conference on Language Resources and Evaluation, LREC '12, pp. 3487–3494 (2012)
38. Meadow, C.T.: Text Information Retrieval Systems. Academic Press, Inc. (1992)
39. Metzler, D., Dumais, S.T., Meek, C.: Similarity measures for short segments of text. In: Proceedings of the European Conference on Information Retrieval, ECIR '07, pp. 16–27 (2007)
40. Mihalcea, R., Corley, C., Strapparava, C.: Corpus-based and knowledge-based measures of text semantic similarity. In: Proceedings of the AAAI Conference on Artificial Intelligence, AAAI'06, pp. 775–780 (2006)
41. Mihalcea, R., Tarau, P.: Textrank: Bringing order into text. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP'04, pp. 404–411 (2004)
42. Navarro, G.: A guided tour to approximate string matching. ACM Comput. Surv. **33**(1), 31–88 (2001)
43. Navarro, G., Baeza-Yates, R.A.: A practical q -gram index for text retrieval allowing errors. CLEI Electr. J. **1**(2) (1998)
44. Pantel, P., Crestan, E., Borkovsky, A., Popescu, A.M., Vyas, V.: Web-scale distributional similarity and entity set expansion. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP'09, pp. 938–947 (2009)
45. Rada, R., Mili, H., Bicknell, E., Blettner, M.: Development and application of a metric on semantic nets. IEEE Trans. Syst. Man Cybern. **19**(1), 17–30 (1989)
46. Radinsky, K., Agichtein, E., Gabrilovich, E., Markovitch, S.: A word at a time: computing word relatedness using temporal semantic analysis. In: Proceedings of the International Conference on World Wide Web, WWW '11, pp. 337–346 (2011)

47. Radlinski, F., Broder, A., Ciccolo, P., Gabrilovich, E., Josifovski, V., Riedel, L.: Optimizing relevance and revenue in ad search: a query substitution approach. In: Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '08, pp. 403–410 (2008)
48. Re, C., Dalvi, N.N., Suciu, D.: Efficient top-k query evaluation on probabilistic data. In: Proceedings of the International Conference on Data Engineering, ICDE'07, pp. 886–895 (2007)
49. Resnik, P.: Using information content to evaluate semantic similarity in a taxonomy. In: Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI'95, pp. 448–453 (1995)
50. Ryeng, N.H., Vlachou, A., Doulkeridis, C., Nørvåg, K.: Efficient distributed top-k query processing with caching. In: Proceedings of the International Conference on Database Systems for Advanced Applications, DASFAA'11, pp. 280–295 (2011)
51. Sahami, M., Heilman, T.D.: A web-based kernel function for measuring the similarity of short text snippets. In: Proceedings of the International Conference on World Wide Web, WWW '06 (2006)
52. Salton, G.: Automatic Text Processing. Addison-Wesley Longman Publishing Co., Inc. (1988)
53. Salton, G., Buckley, C.: Term-weighting approaches in automatic text retrieval. Inf. Process Manag. **24**(5), 513–523 (1988)
54. Salton, G., Wong, A., Yang, C.S.: A vector space model for automatic indexing. Commun. ACM **18**(11), 613–620 (1975)
55. Sarawagi, S., Kirpal, A.: Efficient set joins on similarity predicates. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD '04, pp. 743–754 (2004)
56. Shannon, C.E.: A mathematical theory of communication. Bell Syst. Tech. J. **27**, 379–423 (1948)
57. Tsatsaronis, G., Varlamis, I., Vazirgiannis, M.: Text relatedness based on a word thesaurus. J. Artif. Intell. Res. **37**, 1–39 (2010)
58. Turney, P.: Mining the web for synonyms: Pmi-ir versus lsa on toefl. In: Proceedings of the European Conference on Machine Learning, ECML'01, pp. 491–502 (2001)
59. Ukkonen, E.: Approximate string matching with q-grams and maximal matches. Theo. Comp. Sci. **92**(1), 191–211 (1992)
60. Vernica, R., Li, C.: Efficient top-k algorithms for fuzzy search in string collections. In: Proceedings of the International Workshop on Keyword Search on Structured Data, KEYS'09, pp. 9–14 (2009)
61. Vlachou, A., Doulkeridis, C., Nørvåg, K., Vazirgiannis, M.: On efficient top-k query processing in highly distributed environments. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD '08, pp. 753–764 (2008)
62. Wang, K., Ming, Z.Y., Hu, X., Chua, T.S.: Segmentation of multi-sentence questions: towards effective question retrieval in cqa services. In: Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '10, pp. 387–394 (2010)
63. Wei, F., Li, W., Lu, Q., He, Y.: Query-sensitive mutual reinforcement chain and its application in query-oriented multi-document summarization. In: Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '08, pp. 283–290 (2008)
64. Wu, Z., Palmer, M.: Verbs semantics and lexical selection. In: Proceedings of the annual meeting on Association for Computational Linguistics, ACL'94, pp. 133–138 (1994)
65. Yang, Z., Kitsuregawa, M.: Efficient searching top-k semantic similar words. In: Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI'11, pp. 2373–2378 (2011)
66. Yang, Z., Yu, J., Kitsuregawa, M.: Fast algorithms for top-k approximate string matching. In: Proceedings of the AAAI Conference on Artificial Intelligence, AAAI'10, pp. 1467–1473 (2010)
67. Zhang, X., Chomicki, J.: Semantics and evaluation of top-k queries in probabilistic databases. Distributed and Parallel Databases **26**(1), 67–126 (2009)
68. Zhuge, H.: The Web Resource Space Model. Springer (2008)
69. Zhuge, H.: Communities and emerging semantics in semantic link network: discovery and learning. IEEE Trans. Knowl. Data Eng. **21**(6), 785–799 (2009)
70. Zhuge, H.: Interactive semantics. Artif. Intell. **174**(2), 190–204 (2010)
71. Zhuge, H.: Special section: semantic link network. Future Gener. Comput. Syst. **26**(3), 359–360 (2010)
72. Zhuge, H.: Semantic linking through spaces for cyber-physical-socio intelligence: a methodology. Artif. Intell. **175**(5–6), 988–1019 (2011)

73. Zhuge, H.: The Knowledge Grid: Toward Cyber-Physical Society, 2nd edn. World Scientific Pub Co Inc. (2012)
74. Zhuge, H., Xing, Y.: Probabilistic resource space model for managing resources in cyber-physical society. IEEE Trans. Serv. Comput. **5**(3), 404–421 (2012)
75. Zhuge, H., Xing, Y., Shi, P.: Resource space model, owl and database: mapping and integration. ACM Trans. Internet Technol. **8**(4) (2008)