Efficient Implementation of Phrase Relatedness using Overlapping Context and its Application

Md Rashadul Hasan Rakib *

Abstract

This paper describes an efficient implementation of an existing phrase relatedness method ($TrWP$) that uses overlapping bi-gram contexts extracted from the Google-n-gram corpus. In order to implement $TrWP$ efficiently, a data structure has been designed so that the data extracted from the Google-n-gram corpus can be fitted into memory so as to compute the relatedness score between two phrases within a reasonable amount of time (i.e. $\sim 10^{-5}$ second). The efficiency is measured in terms of how many phrase relatedness scores are computed within a single unit of time by the efficient implementation of $TrWP$ and by $TrWP$ itself respectively. This paper also shows how the new implementation of $TrWP$ is used for one of the most popular Natural Language Processing (NLP) tasks such as document relatedness. In order to do that, this paper delineates a parallel algorithm that computes a large number of phrase relatedness scores generated from two large documents. The objective of the parallel algorithm is to compute relatedness between two large documents as fast as possible. The parallel algorithm has been tested on multi-core machine using different performance measures such as relative speed-up, size-up and scale-up.

1 Introduction

Generally, a phrase is an ordered sequence of multiple words [15, 17] that all together refer to a particular meaning. Phrases are treated as more informative feature terms [5] than words in the documents. Phrase relatedness quantifies how two phrases relate to each other. It plays an important role in different Natural Language Processing (NLP) tasks; for instance, document similarity [1], classification, clustering are performed on the documents composed of phrases. Several document clustering methods [3, 5, 6, 7, 13, 16] used phrase similarity to determine the similarity between documents so as to improve the clustering result. SpamED [14] used the n-gram [2] (i.e. bi-gram and tri-gram) phrase similarity between an incoming e-mail message and a previously marked spam, in order to enhance the accuracy of spam detection.

This paper mainly focuses on the scalability of an existing phrase relatedness method ($TrWP$) [15] that takes about one second to compute relatedness score between two phrases. In order to make $TrWP$ usable to the real world, we are motivated to implement an efficient version of it. To compute relatedness between two phrases, $TrWP$ uses the overlapping bi-gram contexts extracted from the Google-n-gram [4] corpus. It considers the bi-grams as

*Faculty of Computer Science, Dalhousie University, Halifax, Canada B3H 1W5, ychen@cs.dal.ca.

[1] We use 'relatedness' and 'similarity' interchangeably in our thesis, albeit 'similarity' is a special case or a subset of 'relatedness'.

[2] An n-gram consists of n consecutive words.

phrases. It also considers a word as phrase [17] when the relatedness is computed between a word and a bi-gram. This paper deals with only the bi-gram phrase pairs.

Given two bi-gram phrases to $TrWP$, for each phrase it elicits a set of Google-4-grams where each Google-4-gram contains both the target bi-gram phrase and a bi-gram context. Then it finds the overlapping bi-gram contexts from these two sets in order to compute the relatedness score between them. Bi-gram context [15] of a phrase $P(w_1, w_2)$ is a bi-gram (z_1, z_2) such that either of the following augmented 4-grams

$$(w_1, w_2, z_1, z_2) = (P, z_1, z_2)$$
$$(z_1, w_1, w_2, z_2) = (z_1, P, z_2)$$
$$(z_1, z_2, w_1, w_2) = (z_1, z_2, P)$$

exist in the Google-4-gram corpus. Sample bi-gram contexts for the bi-gram phrase "large number" are extracted by placing it at the left most, middle and right position within the Google-4-grams, as shown in Table 1.

Phrase position	Bi-gram phrase in Google-4-gram	Bi-gram context
Left most	large number **of files**	of files
Middle	**very** large number **generator**	very..generator
Right most	**multiply a** large number	multiply a

Table 1: Positions of the bi-gram phrase ("large number") in Google-4-grams and corresponding bi-gram contexts marked bold.

The overlapping bi-gram context [15] is the bi-gram which is overlapped between two Google-4-grams that contain two target phrases at the same position. Consider the Google-4-grams "large number of data" and "vast amount of data" where "large number" and "vast amount" are the target phrases situated at the left most position and "of data" is the overlapping bi-gram context. $TrWP$ takes into account three different positions of a phrase within the Google-n-grams [4] whereas this paper considers only the left most position of a phrase within the Google-4-grams.

We follow the same phrase relatedness approach described in $TrWP$ [15]. The only difference is that $TrWP$ performs On-disk computation (i.e. For each phrase, $TrWP$ loads the bi-gram contexts into memory from the Google-n-gram corpus) and we perform In-memory computation (i.e. All the phrases and their associated bi-gram contexts are loaded into memory before computing relatedness score between them). To perform In-memory computation, a data structure has been designed so that the required data extracted from the Google-4-gram corpus can be fitted into memory to allow us to speed up the phrase relatedness computation.

We show that using multi-core machine, a large number of phrase relatedness scores are computed which let us to compute relatedness score between two large documents. A parallel algorithm has been designed that creates multiple threads where each thread computes $\sim 10^5$ phrase relatedness scores within a second.

There is an efficient document relatedness method (GTM) [9] that uses the word pair relatedness computed from the co-occurrences of two words in Google-3-grams (`http://ares.research.cs.dal.ca/gtm/texttextform.html`). GTM computes 10^6 word relatedness scores per scored. In $TrWP$, it is shown that using word relatedness together with phrase relatedness improves the document relatedness result. However, phrase relatedness computed by $TrWP$ is fairly slow that makes the document relatedness computation even

much slower. Hence we are motivated to improve the the phrase relatedness computation so that we can provide the better document relatedness score to the real world.

The rest of the paper is organized as follows: a brief overview of the naive phrase relatedness method ($TrWP$) is presented in section 2. The data structure for efficient phrase relatedness computation is described in section 3. The proposed In-memory Phrase Relatedness algorithm is discussed in section 4. The Parallel Document Relatedness algorithm using In-memory Phrase Relatedness is delineated in section 5. Evaluation and experimental results of the phrase and document relatedness algorithms are discussed in section 6. We summarize contributions and future related work in section 7.

2 Naive Phrase Relatedness Method ($TrWP$)

$TrWP$ extracts two sets of Google-4-grams for two bi-gram phrases. Then it finds the overlapping bi-gram contexts between these two sets by the placing the phrases at the left most, middle and right most position. For each pair of 4-grams having overlapping bi-gram contexts, it calculates the sum-ratio (SR) [15] value using the frequencies of the corresponding 4-grams. The sum-ratio value represents the strength of association between two overlapping 4-grams. For instance, the 4-grams "gentle man marry woman" and "bachelor person marry woman" have frequencies 200 and 300 respectively where the target phrases are "gentle man" and "bachelor person" and "marry woman" is the overlapping bi-gram context. The SR value is $(200 + 300) \times (200/300) = 333.33$, representing the strength of association between these two Google-4-grams which is used to compute relatedness between the phrases "gentle man" and "bachelor person". Sample extracted (non) overlapping bi-gram contexts for the phrases P_1="large number" and P_2="vast amount" are given in Table 2. The frequencies (counts) of the Google-4-grams and the phrases are shown here.

large number, count=1000		vast amount, count=2000		Bi-gram context	
Google-4-gram	Count	Google-4-gram	Count	Overlap	Non-overlap
large number **of death**	100	vast amount **of death**	200	of death	
consider the large number	300	**consider the** vast amount	400	consider the	
very large number **generator**	150	**very** vast amount **generator**	300	very..generator	
large number **of data**	200	vast amount **of data**	300	of data	
large number **of items**	100	vast amount **of land**	200		of items, of land

Table 2: Sample (non) Overlapping Google-4-grams for phrases "large number" and "vast amount".

For each pair of overlapping Google-4-grams, the SR value is calculated; then some of them are pruned [15] using the mean and standard deviation of all SR values. After that, $TrWP$ calculates the sum of the remaining SR values; and the sum is normalized to measure the relatedness between two phrases. From the above discussion, we can synthesize that $TrWP$ measures relatedness using the common items from two sets.

3 Data Structure for Efficient Phrase Relatedness Computation

$TrWP$ does not consider the order of the items within the sets that causes $m \times n$ computation where m and n is the number of items in two sets respectively. Therefore, we sort

the items (i.e. bi-gram contexts) of each phrase so as to find the common (i.e. overlapping) items between two sets within $minimum(m, n)$ iterations. The data structure is created only once using the all Google bi-grams and their corresponding bi-gram contexts extracted from the Google-4-gram corpus. The steps of constructing the data structure are given below.

- Tokenization of Google uni-grams and bi-grams.
- Constructing the data structure using the bi-gram phrases as well as their contexts extracted from the Google-4-grams.

3.1 Tokenization of Google uni-grams and bi-grams

At first all the Google uni-grams are assigned unique IDs. The uni-gram frequencies are ignored because we consider only the bi-gram phrases. Each uni-gram of a Google bi-gram is given an uni-gram ID. After that each bi-gram is assigned an unique bi-gram ID. The purpose of tokenization is to reduce the memory usage and fast loading of data into memory. The structure of the tokenized uni-gram and bi-gram files are shown in Fig. 1 and Fig. 2 respectively. The unique IDs of uni-gram and bi-gram are represented as uniID and biID respectively. After tokenization the number of uni-grams and bi-grams are 7,058,483 and 188,577,760 respectively.

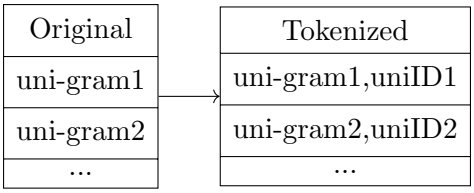

Figure 1: Tokenized uni-gram files.

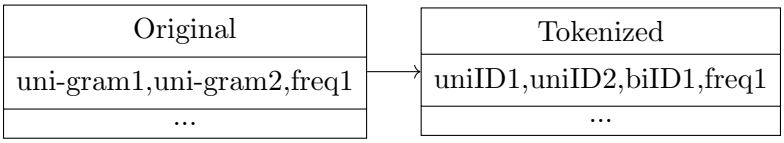

Figure 2: Tokenized bi-gram files.

3.2 Constructing the Data Structure from Google-4-grams

At first all the 4-grams are grouped by bi-gram phrases where the phrases are located at the left most position within the 4-grams. After grouping, we have a list of 38,054,622 bi-gram phrases and for each bi-gram phrase there is a list of bi-gram contexts. The bi-gram contexts of a list are sorted (ascending order) based on their biIDs. The average number of bi-gram contexts for each phrase is 12.97. For each bi-gram context there are two values: an unique biID and the frequency of the Google-4-gram that contains both the phrase as well as bi-gram context. The structure of the tokenized 4-gram files is shown in Fig. 3. biID1 is the ID of the bi-gram phrase ("uni-gram1,uni-gram2"). biID2, biID3 are the associated bi-gram contexts of the phrase biID1 and freq2, freq3 are the frequencies of the Google-4-grams where (biID1,biID2) and (biID1,biID3) appears respectively.

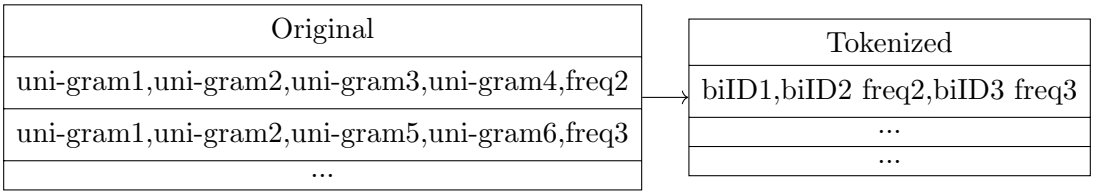

Figure 3: Tokenized 4-gram files.

4 Efficient In-memory Phrase Relatedness Computation

At first all the tokenized uni-grams, bi-grams and 4-grams are loaded into memory. Then for each of the bi-gram phrases P_1 and P_2, the uni-grams are mapped into IDs which are combined to look up the bi-gram ID from Google bi-grams. Each bi-gram ID is used to look up the bi-gram contexts from Google-4-grams. The relatedness score between the phrases P_1 and P_2 is computed using their associated bi-gram contexts. The efficient in-memory phrase relatedness computation is described in the following Algorithm 1 that takes two phrases P_1 and P_2 as input and computes their relatedness score. The idea behind Algorithm 1 is to process the common items between two sorted lists [11].

Procedure 1 EfficientPhraseRelatedness(P_1, P_2)
Input: P_1 = First Phrase, P_2 = Second Phrase
Output: $relPh$ = Relatedness score between P_1 and P_2
Required Variables: $hm1g$ = HashMap of Tokenized Google uni-grams
$hm2g$ = HashMap of Tokenized Google bi-grams
$hm4g$ = HashMap of Tokenized Google-4-grams
Assumption: All the data are loaded into memory before running this algorithm
(1) $uniID1$ = First uni-gram ID of P_1 from $hm1g$
$uniID2$ = Second uni-gram ID of P_1 from $hm1g$
$uniID3$ = First uni-gram ID of P_2 from $hm1g$
$uniID4$ = Second uni-gram ID of P_2 from $hm1g$
(2) $biID1$ = Bi-gram ID of P_1 from $hm2g$ by $uniID1$ and $uniID2$
$biID2$ = Bi-gram ID of P_2 from $hm2g$ by $uniID3$ and $uniID4$
$freqP1$ = Frequency of P_1 from $hm2g$ by $uniID1$ and $uniID2$
$freqP2$ = Frequency of P_2 from $hm2g$ by $uniID1$ and $uniID2$
(3) $list1$ = List of sorted bi-gram contexts of P_1 from $hm4g$ by $biID1$
$list2$ = List of sorted bi-gram contexts of P_2 from $hm4g$ by $biID2$
(4) $c1 = 0$ //counter for list1
$c2 = 0$ //counter for list2
$size1$ = size of $list1$
$size2$ = size of $list2$
$sumSR$ =0
$listSR$ = Empty List
While $c1 < size1$ and $c2 < size2$
- $item1$ = context item from $list1[c1]$, $item2$ = context item from $list2[c2]$
$biIDC1$ = Bi-gram ID from $item1$
$biIDC2$ = Bi-gram ID from $item2$
$freq1$ = Frequency of 4-gram from $item1$

$freq2$ = Frequency of 4-gram from $item2$

- If $biIDC1 = biIDC2$
  - $min = minimum(freq1, freq2)$
    $max = maximum(freq1, freq2)$
    $sumSR = min \div max \times (min + max)$
    Add $sumSR$ to $listSR$

(5) avg = Average of the $sumSR$ values from $listSR$
sd = Standard Deviation of the $sumSR$ values
(6) Prune the $sumSR$ values which are greater than $avg + sd$ or less than $avg - sd$
$sumNonPrunSR$ = Sum of the non-pruned $sumSR$ values
(7) $relPh = sumNonPrunSR/max(freqP1, freqP2)$

The time complexity of Step 3 of Algorithm 1 is $O(m + n)$ and for each step among 4, 5 and 6, the time complexity is $O(min(m, n))$ where m and n is the number of bi-gram contexts of P_1 and P_2 respectively. Time complexity for other steps is $O(1)$. Considering the above steps the time complexity is linear which is approximately equal to the average number of bi-gram contexts of each phrase (i.e. 12.97).

5 Document Relatedness using Efficient In-memory Phrase Relatedness

Most works on documents relatedness are abstracted as a function of word relatedness [8]. To the best of our knowledge, TrWP [15] explicitly shows that using phrase relatedness along with word relatedness improves the documents relatedness result. This paper measures document relatedness using only phrase relatedness since we want to explicitly demonstrate the performance of document relatedness task using the proposed In-memory phrase relatedness method. The performance is measured in terms of how fast the document relatedness is computed when two documents have large number of phrase pairs.

A parallel algorithm 2 has been designed that runs on multi-core machine of 16 physical cores with hyper-threading enabled (32 logical cores). The input of this algorithm is a number of phrase pairs generated from two documents D_1 and D_2 and the output is the relatedness score between them using phrase pair relatedness. In order to generate phrases from a document the bi-grams are linearly scanned. A bi-gram is a phrase when its frequency is greater than a threshold [15] which is precomputed only once. The phrase generation steps are ignored in algorithm 2.

The total phrase pairs are divided into T chunks where each chunk is assigned to a single thread that computes the phrase relatedness scores of that particular chunk independently. Each thread calculates the average of the relatedness scores and return it to the main thread. The main thread sum the T averages and normalizes the sum which is the ultimate relatedness score between the documents D_1 and D_2.

Procedure 2 DocumentRelatedness(PL)
Input: PL = List of phrase pairs generated from the documents D_1 and D_2
T = Number of threads
Output: $relDoc$ = Relatedness score between D_1 and D_2
Assumption: All the data are loaded into memory before running this algorithm

(1) N = size of PL
$nt = N/T$ //Number of phrase pairs per thread
(2) For $i = 0$ to $T - 1$
- $start = i \times nt$ //Start index of the chunk i
  $end = (i + 1) \times nt - 1$ //End index of the chunk i
- Execute Thread i with $start$ and end index
  Thread i reads each phrase pair from $start$ to end and executes the algorithm:$EfficientPhraseRelatedness$

(3) Gather T averages from T threads
$relDoc$ = average of T values/Maximum of T values

The time complexity of Algorithm 2 is $N\times$ Time complexity of $EfficientPhraseRelatedness$ 1 where N is the number of phrase pairs. Since the time complexity of phrase relatedness is linear with small value (i.e. 12.97), therefore the time complexity of Algorithm 2 is linear ($\sim N$).

6 Experimental Results and Evaluation

The sequential In-memory phrase relatedness method computes relatedness scores of $\sim 10^5$ phrase pairs within a second whereas the Naive phrase relatedness method takes about one second to compute relatedness score of one pair. Therefore the In-memory Phrase Relatedness method is $\sim 10^5$ times faster than the Naive phrase relatedness method.

To measure the performance of Parallel Document Relatedness algorithm, the datasets from SemEval-2012 [1], SemEval-2013 [2], STS131 [12] and ABC1225 [10] are used that contain in total 6714 document pairs. The average of the average number of words of each document pair is 24.67. So there are about 24.67×24.67=608.68 bi-gram pairs for each document pair. However, not all the bi-grams are treated as phrases because their frequencies are lower than the threshold [15]. If the frequency of a bi-gram is low, then the probability of finding bi-gram contexts of that particular bi-gram is also low. This is because we randomly divide the 6714 document pairs into two subsets and each subset is considered as a single document. Rest of this paper performs different experiments using these two subsets (i.e. two documents).

There are 846951 phrase pairs between these two documents. Among them 500000 phrase pairs are randomly selected and given to the Parallel Document Relatedness algorithm 2 which is evaluated using different performance measures such as Running-time vs Threads, Relative speed-up, Size-up and Scale-up.

The Running-time vs Threads curve is shown Fig. 4. The running time for 500000 phrase pairs declines up to 16 threads running on 16 physical cores that clearly shows the necessity of using multiple cores when the volume of data is big. After that the running time up to 32 threads declines very slowly since there is no physical core after 16 and all the threads are using the same shared memory.

The Relative speed-up performance of an application on multi-core machine is the ratio between the running time on single-core and the running time on multiple cores. Liner speed-up refers to the speed-up which is equal to the number of cores. The Relative speed-up performance of the Parallel Document Relatedness algorithm for 500000 phrase pairs improves by 53% up to 16 threads, as shown in Fig. 5. The relative speed-up improves at small scale between the threads 16 to 32 due to hyper-threading.

The size-up performance of an application on multi-core machine shows how the running time increases with increase of the volume of data for a fixed number of cores. The

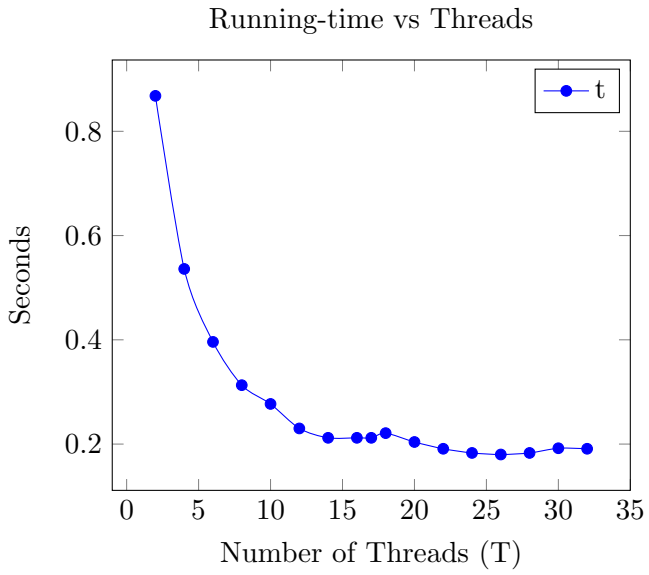

Figure 4: Running time of Parallel Document Relatedness on 500000 phrase pairs (extracted from two documents) using between 2 to 32 threads on 16 core machine with hyper-threading.

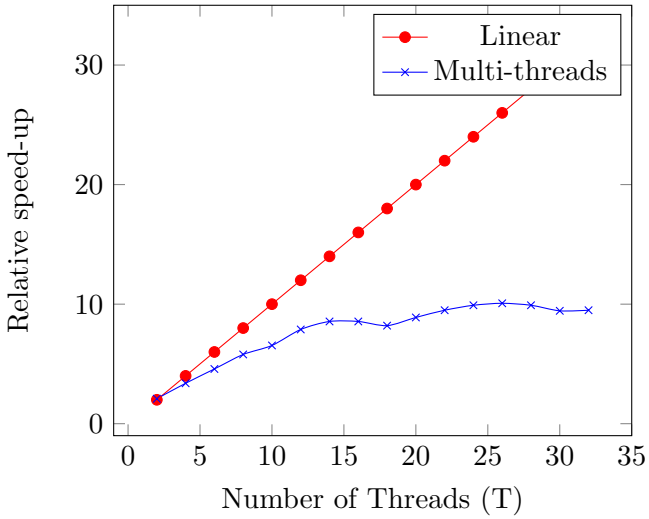

Figure 5: Relative speed-up performance of Parallel Document Relatedness on 500000 phrase pairs (extracted from two documents) using between 2 to 32 threads on 16 core machine with hyper-threading.

size-up performance of the Parallel Document Relatedness algorithm on 50000 to 500000 phrase pairs for 32 threads is depicted in Fig. 6. It is shown that, the trend of the size-up performance is almost liner (i.e. the running time increases as the number of phrase pair increases).

The scale-up performance shows how the application is scalable with the increase of input size. The scalability is quantified based on the running time of the application while the ratio of the input size per core is fixed. The scale-up performance of the Parallel

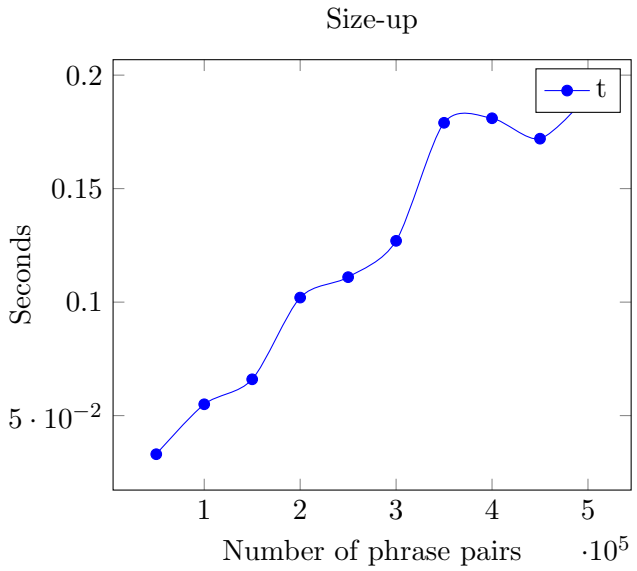

Figure 6: Size-up performance of Parallel Document Relatedness on 50000 to 500000 phrase pairs using 32 threads on 16 core machine with hyper-threading.

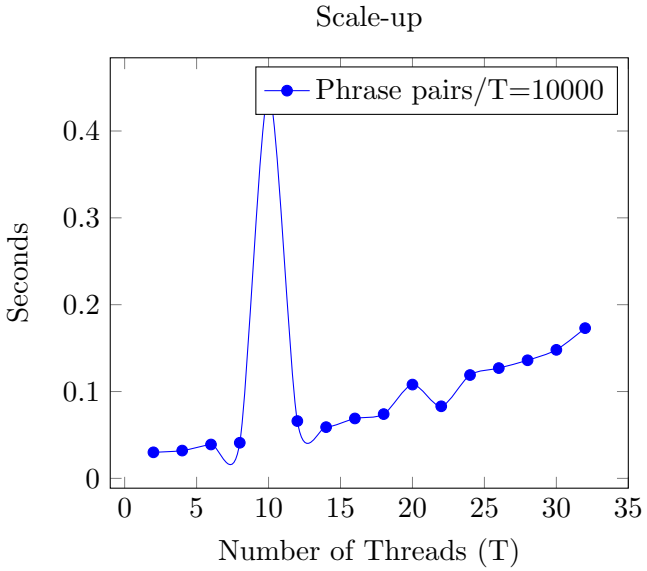

Figure 7: Scale-up performance of Parallel Document Relatedness using between 2 to 32 threads on 16 core machine with hyper-threading. Number of phrase pairs per Thread is 10000.

Document Relatedness algorithm using 2 to 32 threads for fixed 10000 phrase pairs per thread is shown in Fig. 7 that shows, the parallel algorithm is not scalable with the input size since the running time almost linearly increases as the number of threads increases.

7 Conclusion and Future Work

We propose an efficient In-memory Phrases Relatedness method using overlapping bi-gram contexts that takes about 10^{-5} second to compute relatedness between two phrases while the existing Naive approach takes one second per phrase pair. A data structure has been designed so that the required data extracted from the Google-n-gram corpus is fitted into memory. We show the application of phrase relatedness in one of the most popular Natural Language Processing tasks (i.e. document relatedness). A Parallel Document Relatedness algorithm has been designed to compute relatedness between the documents when two documents have large number of phrase pairs. The Parallel Document Relatedness algorithm exhibits the expected Relative Speed-up and Size-up performance.

In the future, we plan to augment the In-memory Phrases Relatedness method so that it can compute relatedness score between the phrases of different number of words along with different phrase positions within the Google-n-grams. We also plan to evaluate the Parallel Document Relatedness algorithm on large number of document pairs.

References

[1] Eneko Agirre, Johan Bos, Mona Diab, Suresh Manandhar, Yuval Marton, and Deniz Yuret, editors. **SEM 2012: The First Joint Conference on Lexical and Computational Semantics – Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*. Association for Computational Linguistics, Montréal, Canada, 7-8 June 2012.

[2] Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. *SEM 2013 shared task: Semantic textual similarity. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, pages 32–43, Atlanta, Georgia, USA, June 2013. Association for Computational Linguistics.

[3] A.M. Bakr, N.A. Yousri, and M.A. Ismail. Efficient incremental phrase-based document clustering. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 517–520, Nov 2012.

[4] T. Brants and A. Franz. Web 1T 5-gram corpus version 1.1. *Linguistic Data Consortium*, 2006.

[5] Hung Chim and Xiaotie Deng. Efficient phrase-based document similarity for clustering. *IEEE Trans. on Knowl. and Data Eng.*, 20(9):1217–1229, September 2008.

[6] Khaled M. Hammouda and Mohamed S. Kamel. Phrase-based document similarity based on an index graph model. In *ICDM*, pages 203–210. IEEE Computer Society, 2002.

[7] K.M. Hammouda and M.S. Kamel. Efficient phrase-based document indexing for web document clustering. *Knowledge and Data Engineering, IEEE Transactions on*, 16(10):1279–1296, Oct 2004.

[8] Chukfong Ho, Masrah Azrifah Azmi Murad, Rabiah Abdul Kadir, and Shyamala C. Doraisamy. Word sense disambiguation-based sentence similarity. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, COLING '10, pages 418–426, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[9] Aminul Islam, Evangelos Milios, and Vlado Kešelj. Text similarity using google trigrams. In *Proceedings of the 25th Canadian conference on Advances in Artificial Intelligence*, Canadian AI'12, pages 312–317, Berlin, Heidelberg, 2012. Springer-Verlag.

[10] Michael D. Lee, Brandon Pincombe, and Matthew Welsh. An empirical evaluation of models of text document similarity. In *Proceedings of the 27th Annual Conference of the Cognitive Science Society (CogSci2005)*, pages 1254–1259. Erlbaum, 2005.

[11] Wei Lu, Chuitian Rong, Jinchuan Chen, Xiaoyong Du, G.P.C. Fung, and Xiaofang Zhou. Efficient common items extraction from multiple sorted lists. In *Web Conference (APWEB), 2010 12th International Asia-Pacific*, pages 219–225, April 2010.

[12] James O'Shea, Zuhair Bandar, and Keeley Crockett. A new benchmark dataset with production methodology for short text semantic similarity algorithms. *ACM Trans. Speech Lang. Process.*, 10(4):19:1–19:63, January 2014.

[13] Maria Soledad Pera and Yiu-Kai Ng. Utilizing phrase-similarity measures for detecting and clustering informative rss news articles. *Integr. Comput.-Aided Eng.*, 15(4):331–350, December 2008.

[14] Maria Soledad Pera and Yiu-Kai Ng. Spamed: A spam e-mail detection approach based on phrase similarity. *J. Am. Soc. Inf. Sci. Technol.*, 60(2):393–409, February 2009.

[15] Md Rashadul Hasan Rakib. Text relatedness using word and phrase relatedness. Technical report, 2014.

[16] Shailendra Kumar Shrivastava, J. L. Rana, and R. C. Jain. Article: Text document clustering based on phrase similarity using affinity propagation. *International Journal of Computer Applications*, 61(18):38–44, January 2013. Published by Foundation of Computer Science, New York, USA.

[17] Oren Zamir and Oren Etzioni. Grouper: A dynamic clustering interface to web search results. In *Proceedings of the Eighth International Conference on World Wide Web*, WWW '99, pages 1361–1374, New York, NY, USA, 1999. Elsevier North-Holland, Inc.