Journal of Electrical and Computer Engineering Innovations
(JECEI)
Journal homepage: http://www.jecei.sru.ac.ir

Research paper

Parallel and Exact Method for Solving n-Similarity Problem

M. Mirhosseini, M. Fazlali*

Department of Data and Computer Science, Faculty of Mathematical Sciences, Shahid Beheshti University, Tehran, Iran.

Article Info

Article History:
Received 24 September 2019
Reviewed 20 November 2019
Revised 18 January 2020
Accepted 21 April 2020

Keywords:
n-Similarity
Parallel programming
Open-MP
Text document similarity

*Corresponding Author's Email Address:
fazlali@sbu.ac.ir

Abstract

Background and Objectives: n-similarity problem defined as measuring the similarity among $n \geq 3$ objects and finding a group of n objects from a dataset that have the most similarity to each other. This problem has been become an important issue in information retrieval and data mining. Theory of this concept is mathematically proven, but it practically has high memory complexity and is so time consuming. Besides, the solutions found by metaheuristics are not exact.
Methods: This paper is conducted to propose an exact method to solve n-similarity problem reducing the memory complexity and decreasing the execution time by parallelism using Open-MP. The experiments are performed on the application of text document resemblance.
Results: It has been shown that the memory complexity of the proposed method is decreased to $O(N^2)$, and the experimental results show that this method accelerates the speed of the computations about 5 times.
Conclusion: The simulated results of the proposed method display a good improvement in speed, the used memory space, and scalability compared with the previous exact method.

Introduction

The similarity is defined as the resemblance degree between two or more objects, phenomena or concepts. In various applications, designers want to build classes of objects based on the similarity criteria. So, it has an important role in automated classification, clustering methods, decision making, approximate reasoning, and diagnosis systems [1], [2], [3]. The similarity between two objects, known as 2-similarity, is mathematically defined as a function on domain U as $S_2: U \times U \longrightarrow [0,1]$ *or* $[-1,1]$. The value of 1 shows that the objects are identical, whereas 0 or -1 indicates that the objects are completely different. Various kinds of similarity measures including classic and fuzzy types have been suggested by researchers. Some well-known classic similarity/dissimilarity measures are Euclidean distances, Jaccard, Overlap, Dice, Pearson and Cosine coefficients that their effects are studied in data clustering [4].

Kaufman and Rousseeuw [4] compared several similarity measures on hierarchical clustering. Also, different types of fuzzy similarity measures have been developed. As an instance, a similarity measure between two fuzzy sets has been proposed in [6]. Its main idea is that if the intersection between two fuzzy sets is high, their similarity is identical. Two other fuzzy similarity measures based on the relative sigma count have been suggested in [7]. Using the Intuitionistic Fuzzy Sets (IFS) theories a fuzzy Cosine similarity measure and its weighted kind are proposed in [8]. These similarity measurements compute the similarity degree between two objects. But, in some applications, computing the similarity value among $n \geq 3$ objects or finding the most similar n-group among a dataset is required. The concept of n-similarity is mathematically defined by Keshavarzi et al. [1] at first. But, this method is infeasible due to the high time and memory complexities.

Therefore it is unable to handle the large datasets. To decrease the execution time of the algorithm and solving the problem in a reasonable time, a binary genetic algorithm has been exploited in [9]. Since the performance of metaheuristics varies in different problems, the effect of some other metaheuristics including, particle swarm optimization (PSO), gravitational search algorithm (GSA), imperialist competitive algorithm (ICA) and fuzzy imperialist competitive algorithm (FICA) has been studied and compared in solving the n- similarity problem and finding the most similar n-group of a dataset. The experiments show that the FICA has the best results [10].

Multi-core processors gain the market with increasing the number of cores per processor. But, the sequential programming model does not exploit multi-core systems well. Parallel programming techniques such as Open-MP present more effectively using of multiple processor cores to accelerate the speed of the algorithm [11]. Open-MP is utilized in various applications to reduce the execution time. For example, in [11], authors used Open-MP library to accelerate finding the maximum weighted clique. Also, the authors in [13] parallelized genetic algorithm to present a scalable method to solve large integer linear programming models derived from high-level synthesis of digital circuits. In [14] a new method for implementing the parallel breadth-first search algorithm for graph exploration on multi-core CPUs has been presented to accelerate the algorithm. In [15] the authors applied Open-MP to parallelize the quadrivalent quantum-inspired gravitational search algorithm in WSNs. This leads to improve the speedup faster than 4 times. Also, authors in [16] increased the speed of running the ant colony algorithm using Open-MP about 4.5 times. These researches encourage us to apply parallelism by multi-core CPU and Open-MP to improve the speed of our algorithm. Although metaheuristics [9], [10] can find a near optimal solution in a reasonable time, their results are not exact and they do not explore and compute the n-similarity value of all possible n-groups. Since Keshavarzi's method [1] is an exact method but suffers from high space complexity and its high execution time. Besides, the results of the metaheuristic methods [9], [10] are not exact. So, we are motivated to develop Keshavarzi's method as an exact one and improve its space complexity and decrease its execution time by parallelism using Open-MP. In fact, in the proposed approach, we perform some changes on recursive n-similarity relations introduced by Keshavarzi, et al. [1], and by that, the space complexity decreases from $O(N^n)$ to $O(N^2)$ for computing n-similarity for a dataset in the size of N. By this approach, it is not required to keep all similarity matrixes in the memory to be accessible in the recursive order. Moreover, since the proposed method involves several $nested\ for\ loops$, we apply the parallelism technique on the most time-consuming part of the algorithm to reduce the execution time. By this way, we would have an exact and executable method with lower space complexity that can run in a more reasonable time. This proposed method is better than Keshavarzi's [1] method from three aspects:

- The space complexity is low.
- Its execution time is less than the Keshavarzi's method.
- This method is scalable for larger datasets.

Also, it has two advantages in comparison with metaheurstics [9], [10] including:

- This method is an exact one.
- This method computes the n-similarity values for all possible groups of n objects.

The remainder of this study is organized as follows: the next section provides some preliminaries and related definitions. Then, the proposed parallel and exact method in solving the n-similarity problem is presented, and shows how this method can improve the space complexity. The experimental results as a case study on text document similarity are presented; and finally the paper is concluded in the last section.

Preliminaries

In this section some required definitions are presented from [1], [9].

Definition 1. Triangular norm (T-norm) is defined as a function $T: [0,1] \times [0,1] \longrightarrow [0,1]$ which satisfies the following conditions for all $x, y, w, z \in [0,1]$:

1. Commutatively: $T(x,y) = T(y,x)$,
2. Monotonicity: $T(x,y) \leq T(w,z), if\ x \leq w, and\ y \leq z$,
3. Associativity: $T(x, T(y,w)) = T(T(x,y), w)$,
4. Boundary : $T(x,0) = 0,\ T(x,1) = x$.

The minimum T-norm is a well-known instances of T-norms which has been applied in [1], [2] to n-similarity definition; $T_{min}(x,y) = \min(x,y)$.

Definition 2. The 2-similarity on domain U is a function $S: U \times U \longrightarrow [0,1]$ satisfying the following conditions:

1. Reflexivity: $for\ any\ x \in U, S(x,x) = 1$,
2. Symmetry: $for\ any\ x, y \in U, S(x,y) = S(y,x)$,
3. Transitivity: $for\ any\ x, y, z \in U, S(x,z) \geq S(x,y) \wedge S(y,z)$, where ∧ is the minimum operator.

Definition 3. The 3-similarity on domain U is a function $S: U \times U \times U \longrightarrow [0,1]$ satisfying the following conditions:

1. Reflexivity: $for\ any\ x \in U, S(x,x,x) = 1$.
2. Symmetry: $for\ any\ x_1, x_2, x_3 \in U,\ S(x_1, x_2, x_3) = S(x_{i1}, x_{i2}, x_{i3})$ where i_1, i_2, i_3 is an arbitrary permutation of (1,2,3).
3. Transitivity property:

$for\ any\ t, x_1, x_2, x_3 \in U, S(x_1, x_2, x_3) \geq S(t, x_2, x_3) \wedge S(x_1, t, x_3) \wedge S(x_1, x_2, t)$, where $\wedge$ is the minimum T-norm.

The minimum T-norm is applied to generalize the 2-similarity to 3-similarity as (1). This equation satisfies all conditions of Definition 3.

$$S_3(x_1, x_2, x_3) = \min\left(S_2(x_1, x_2), S_2(x_1, x_3), S_2(x_2, x_3)\right) \quad (1)$$

Definition 4. $S: U \times U \times \ldots \times U \rightarrow [0,1]$ is the n−similarity function satisfying the following conditions:

1. Reflexivity: $for\ any\ x \in U, S(x, x, \ldots, x) = 1$,
2. Symmetry: $S(x_1, x_2, \ldots, x_n) = S(x_{i1}, x_{i2}, \ldots, x_{in})$ for all permutations $(i_1, i_2, \ldots, i_n)$ of $(1,2, \ldots, n)$.
3. Transitivity:
   $for\ all\ x_1, x_2, \ldots, x_n, z \in U, S(x_1, x_2, \ldots, x_n) \geq \min\{S(z, x_2, \ldots, x_n), \ldots, S(x_1, x_2, \ldots, x_{n-1}, z)\}$.

It was shown in [1] that the n-similarity can be achieved from the (n-1)-similarity satisfying all conditions of Definition 4. If S_{n-1} is the $(n-1)$-similarity on U and $x_1, x_2, \ldots, x_n \in U$, then:

$$S_n(x_1, x_2, \ldots, x_n) = \min\begin{pmatrix} S_{n-1}(x_2, x_3, \ldots, x_n), S_{n-1}(x_1, x_3, \ldots, x_n) \\ , \ldots, S_n(x_1, x_2, \ldots, x_{n-1}) \end{pmatrix} \quad (2)$$

The pseudo codes of 3-similarity, 4-similarity and n-similarity are presented in Algorithms 1, 2 and 3, respectively [1]. In these algorithm computing the similarity for each value of n is dependant on the previous similarities including $(n-1)$, $(n-2), \ldots, 3\ and\ 2$. So, these matrixes are required to save and this leads to increase the space complexity of this method and the program is unable to run for datasets which have more than 100 objects. In addition for several *nested for* the execution time of these algorithms are high and increase more by growing the size of dataset. In the next section, it is shown how we handle these problems.

Algorithm 1. The pseudo codes of 3-similarity

Input: *2-sim* matrix
Output: Max, *3-sim matrix*
Max=0;
for *i=1* to size of dataset
  for *j=i+1* to size dataset
    for *k=j+1* to size of dataset
      3-sim(i, j, k)=min{2-sim(i ,j),2-sim(i ,k),2-sim(j ,k)};
      If(3-sim(i, j, k)>Max)
        Max=3-sim(i, j, k);
end of for *i, j* and *k*
End

Algorithm 2. The pseudo codes of 4-similarity

Input: *3-sim* matrix
Output: Max, *3-sim matrix*
for *i=1* to size of dataset
  for *j=i+1* to size dataset
    for *k=j+1* to size of dataset
      for l=k+1 to size of dataset
4-sim(i, j, k, l)=min{3-sim(i, j, k),3-sim(i, j, l),3-sim(i, k ,l),3-sim(j, k, l)};
If(4-sim(i, j, k, l)>Max)
Max=4-sim(i, j, k, l);
end of for *i, j, k* and *l*
End

Algorithm 3. The pseudo codes of n-similarity

Input: *(n-1)-sim* matrix
Output: Max, *n-sim* matrix
for i_1=1 to size of dataset
  for i_2=i_1+1 to size of dataset
    .
    .
    for i_n=$i_{(n-1)}$ to size of dataset
      n-sim(i_1,i_2,…,i_n)=min{(n-1)-sim(i_2,i_3,…,i_n),(n-1)-sim(i_1,i_3,…,i_n),…, (n-1)-sim(i_1,i_2,…,i_{n-1})};
      If(n-sim(i, j, k)>Max)
        Max=n-sim(i, j, k);
end of for i_1 to i_n
End

The Proposed method

Regarding Algorithms 1, 2 and 3 proposed in [1], it can be shown that their executing time and the space complexity are $O(N^n)$, and the execution time and required memory space are grown by increasing the size of dataset (N) and n. Therefore, using this method for large datasets is practically impossible. Let N be the size of dataset. For computing the n-similarity, it is needed to calculate and store the $(n-1)$-similarity matrix in the size of N^{n-1}; as the same way, it is needed to calculate and store the $(n-2)$-similarity matrix in the size of N^{n-2} and so on. Therefore, the space complexity for computing the n-similarity would be equal to $N^n + N^{n-1} + N^{n-2} + \cdots + N^3 + N^2 = O(N^n)$. By the following proposed method, the space complexity would be decreased to $O(N^2)$. Let S_2, S_3, S_4, S_n be the 2-similarity, 3-similarity, 4-similarity and n-similarity respectively. We know:

$$S_4(x_1, x_2, x_3, x_4) = \min\begin{pmatrix} S_3(x_1, x_2, x_3), S_3(x_1, x_2, x_4), S_3(x_1, x_3, x_4), \\ S_3(x_2, x_3, x_4) \end{pmatrix} \quad (3)$$

By replacing S_3 from (1) to S_4 in (3), we have S_4 in terms of S_2 as (4).

$$S_4(x_1, x_2, x_3, x_4) = \min\begin{pmatrix} S_2(x_1, x_2), S_2(x_1, x_3), S_2(x_1, x_4), S_2(x_2, x_3), \\ S_2(x_2, x_4), S_2(x_3, x_4) \end{pmatrix} \quad (4)$$

In a similar way, S_n would be achieved as (5) by replacing S_{n-1} in terms of S_2. So, S_n can be computed regardless of previous $(n-1)$ similarity matrices and it just needs 2-similarity matrix as input instead of 2-similarity, 3-similarity,…,$(n-1)$-similarity matrixes. By this way, the space complexity is decreased from $O(N^n)$ to $O(N^2)$.

$$S_n(x_1, x_2, \ldots, x_n) = \min\left(S_2(x_1, x_2), S_2(x_1, x_3), \ldots, S_2(x_{n-1}, x_n)\right). \quad (5)$$

Besides, to decrease the execution time of the algorithms, we use parallelism technique using Open-MP. Open-MP is a portable implementation of parallel programs for shared memory multiprocessors. Recently, many algorithms are implemented in parallel using Open-MP achieving good speedup in addition to its simple implementation compared to other parallel methods like CUDA applied on GPU [17]. In this problem we have *nested for* loops. Either for loop can be run in parallel but, we make outer loop parallel to reduce number of forks and joins. Also, each thread gets its own private copy of variables j, k, etc. Also, since the iteration number of inner loops gradually is decreased, the dynamic scheduling is exploited, in which only some iterations of loop are allocated to threads and remaining iterations allocated to threads that complete their assigned iterations. Algorithms 4, 5 and 6 present the proposed method for computing the 3-similarity, 4-similarity and n-similarity respectively. In all these algorithms, we have n nested for loop for computing n-similarity. Although the time complexity does not change compared with [1], the execution time is improved using parallelizing. Instruction *#pragma omp parallel for* causes to divide the index of for i among running threads and each thread works on a part of outer loop. As an example if the size of dataset is 1000, and we have 4 threads, 250 tasks can be assigned to each thread. Each thread computes the related n-similarities and finds their maximum and stores the related maximum in *MaxTread[tid]* cell, where *tid* is the index of the thread. Finally, the maximum of array *MaxTread* is computed and stored as output *MAX*. Also, it is observable that all these pseudo codes just need the 2-similarity matrixes composing of the similarity between all possible pair wise objects.

This matrix is a symmetric one with the size of $N \times N$ that N is the size of dataset. The elements on the main diagonal are one; and to avoid producing results with frequent objects, the elements below the main diagonal are set to zero. So, each inner for loop starts from the next value of previous loop index.

Algorithm 4. The pseudo codes of the proposed 3-similarity

```
Input: 2-sim matrix
Output: Max, 3-sim of each group
#pragma omp parallel for
 for i=1 to size of dataset
  for j=i+1 to size dataset
    for k=j+1 to size of dataset
      3-sim=min{2-sim(i , j),2-sim(i ,k),2-sim(j ,k)};
      Write:3-sim
        If(3-sim>MaxTread[tid])
          MaxTread[tid]=3-sim;
 end of for i, j and k
MAX=maximum of MaxTread
End
```

Algorithm 5. The pseudo codes of the proposed 4-similarity

```
Input: 2-sim matrix
Output: Max;
#pragma omp parallel for
for i=1 to size of dataset
  for j=i+1 to size dataset
    for k=j+1 to size of dataset
      for l=k+1 to size of dataset
 begin
4-sim=min{2-sim(i, j),2-sim(i, k),2-sim(i, l),2-sim(j, k),2-sim(j,l),2-sim(k,l)};
Write:4-sim
If(4-sim> MaxTread[tid])
          MaxTread[tid]=4-sim;
 end of for i, j, k and l
MAX=maximum of MaxTread
End
```

Algorithm 6. The pseudo codes of the proposed n-similarity

```
Input: 2-sim matrix
Output: Max
#pragma omp parallel for
 for i1=1 to size of dataset
  for i2=i1+1 to size of dataset
    .
      .
     for in=i(n-1) to size of dataset
       n-sim=min{2-sim(i1,i2),2- sim(i1,i3),..., 2-sim(in-1,in)};
       Write:n-sim
        If(n-sim> MaxTread[tid])
          MaxTread[tid]=n-sim;
end of for i1 to in
MAX=maximum of MaxTread
End
```

Results and Discussion

To compare the proposed method with the exact method suggested in [1], some experiments are conducted. One of the applications of n-similarity is in text similarity. The purpose is to find n documents among N texual documents in such away they have the most similarity to each other. Also the similarity among all possible n permutations is computed. The Re0 dataset from the Reuters repository [18] are used in this work, contained 1504 newspaper articles, 2886 key words and 31 classes. To work with the textual datasets some preprocessing steps are required to do. This procedure performs as follows and the output is the 2-similarity matrix which is given as input to the n-similarity algorithms.

Step 1: The first is tokenization in which, all digits and symbols are removed and the strings occurred before $space$, . , :, ; , − , ? , ! , (,) [,] etc. are extracted and considered as a word.

Step 2: in this step, all stop-words like *a, the, is, are,* etc. are eliminated using a pre-supplied list named Weka machine learning workbench [19] included 527 stop-words.

Step 3: There are some words with similar theme and different morphological concept. These words would be

mapped into their stem to treat as a single word. For instance both words *computing* and *computation* are maped into the stem *comput*. The Porter's suffix-stripping algorithm [20] is applied for stemming the words.

Step 4: In the weighting step, each extracted term gets a numerical weight. $T = \{t_1, t_2, \dots, t_l\}$ is all occurred words in the dataset and $d_i = \{w_{1i}, w_{2i}, \dots, w_{li}\}$ is the feature vector of document d_i, where w_{ki} is the weight of word t_k in document d_i. The weighting process is done by the TFIDF as (6).

$$w_{ki} = TFIDF(d_i, t_k) = TF(d_i, t_k) \times \log(\frac{|D|}{DF(t_k)}), \quad (6)$$

In this relation, *D* is the size of the dataset; $TF(d_i, t_k)$ is the frequency of term t_k in the document d_i and $DF(t_k)$ is the number of documents in which term t_k occurs in [21].

Step 5: In this step, the terms with the weight less than a predefined threshold are removed.

Step 6: In this step, to compose the 2-Similarity matrix, the similarity between all documents is computed by Cosine similarity measurement, which is a usual measurement for text document applications. Equation (7) shows this similarity measurement which computes the similarity degree between documents d_i and d_j using their feature vectors as $d_i = \{w_{1i}, w_{2i}, \dots, w_{li}\}$ [21].

$$Cosine - Sim(d_i, d_j) = \frac{\sum_{k=1}^{n} w_{ki} w_{kj}}{\sqrt{\sum_{k=1}^{n} w_{ki}^2 \sum_{k=1}^{n} w_{kj}^2}} \quad (7)$$

Finally, the 2-similarity matrix give as input to Algorithms 4, 5 or 6 to compute the 3-similarity, 4-similarity or n-similarity respectively. The simulations are performed on a laptop with CPU: Intel(R) Core(TM) i7-4702 MQ CPU @ 2.20GHz, 4 GB RAM. The results are averaged over 5 running of the algorithms. Fig. 1, Fig. 2 and Fig. 3 respectively show the results for 3-similarity, 4-similarity and 5-similarity. Each table presents the speedup of using the proposed method for different size of Re0 dataset from 200 to 1504 by changing the number of cores from 2 to 8 (showing by $S2$ to $S8$). In Fig. 1, for dataset with the size of 200, the speedup is enhanced from 1.45 to 4.29 by changing the number of cores from 2 to 8. Also, for dataset in the size of 1,504 the speedup is enhanced from 1.89 to 5.43 in average. The averaged speedup for different size of dataset is increased from 1.66 to 5 by changing the number of cores. Fig. 2 shows the results in the case of 4-similarity problem. As an example, for dataset with 400 documents, the speedup is gradually enhanced from 1.47 to 5.21 for different number of threads. Also, in the case of dataset with the size of 1,504, the speedup is 1.73 using 2 cores and is 5.68 for 8 cores. The averaged speedup for different size of dataset varies from 1.68 to 5.38 using 2 cores and 8 cores respectively. Similarly, Fig. 3 demonstrates the results for 5-similarity problem. The averaged speedup increased from 1.61 to 4.82 using different number of cores starting from 2 to 8. Regarding these figures, the speedup of 5, 5.38 and 4.82 are averagely achieved for 3-similariy, 4-similarity and 5-similarity problems by 8 cores. Fig. 1, Fig. 2 and Fig. 3 show that the speedup is generally improved by increasing the size of dataset and increasing the number of threads for 3-similarity, 4-similarity and 5-similarity problems. In fact, for larger datasets, increasing the number of threads leads to more speedup.

Because in the case of smaller datasets dividing the data on more number of cores has more overhead in comparison with dividing large datasets on a same number of cores. Fig. 4 represents the efficiency using 8 cores for 3, 4 and 5-smilarity problems for different size of datasets. It can be seen that for 3-similarity problem, the efficiency is increased from 53.6% for a dataset with the size of 200 to 67.8% for the dataset with the size of 1,504. In addition, for 4-similarity problem, the efficiency of using 8 cores changes as 57.7% to 71.0% by changing the size of dataset from 200 to 1,504. Also, the efficiency of using 8 cores is computed as 51.3% for a dataset with the size of 200 and increased to 67.3% for dataset with 1,504 documents. It is observable that by growing the size of dataset, the efficiency is increased.

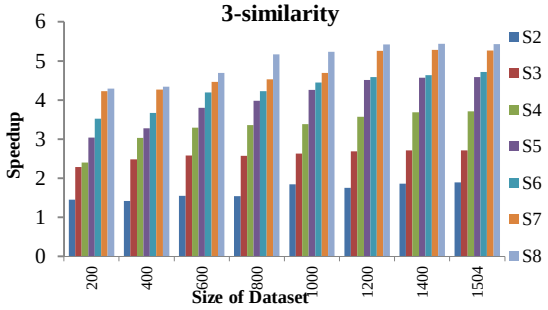

Fig. 1: The speedup of the proposed method in solving the 3-similarity for different number of threads and different size of datasets.

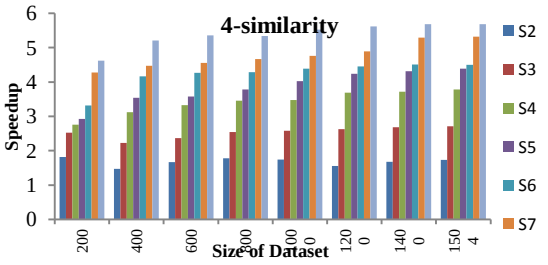

Fig. 2: The speedup of the proposed method in solving the 4-similarity for different number of threads and different size of datasets.

Table 1 presents the execution time of the Keshavarzi's method [1] and our proposed method in sequential and parallel with 8 threads for 3-similarity, 4-similarity and 5-similarity by changing the size of dataset. In this table "-" means that the method is unable to solve the problem.

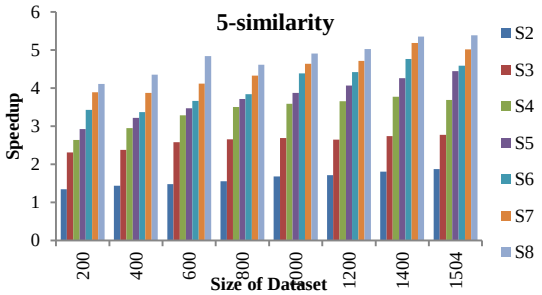

Fig. 3: The speedup of the proposed method in solving the 5-similarity for different number of threads and different size of datasets.

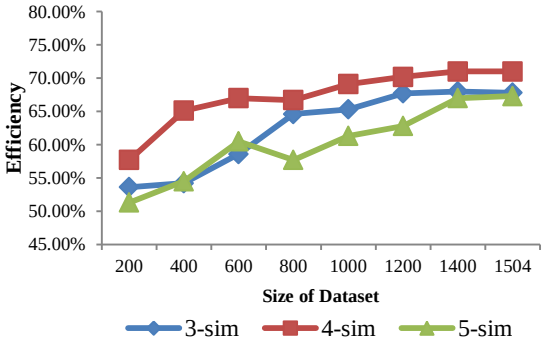

Fig. 4: The of efficiency using 8 threads for different size of dataset for 3, 4 and 5-similarity problems.

The columns entitled by "[1]" shows that although the Keshavarzi's method [1] is theatrically true, it is practically unable to solve the n-similarity problem for $n \geq 5$. Besides, for $n = 4$ the proposed method in [1] just can work on datasets with the size of less than 200. For the 3-similarity problem, the Keshavarzi's method [1] is applicable for datasets which are contained less than 1400 documents. These are due to the high memory space which this method needs to keep matrixes to be accessible in recursive order. While, our proposed method is applicable for all examined size of datasets. Because, it just needs to keep the 2-similarity matrix in the size of N^2 to solve the 3, 4, 5 and generally n-similarity problems. Moreover, in this table, it clearly illustrates the decreasing execution time using parallelism in comparison with sequential version of the proposed method. So, our proposed method is a scalable version of exact method proposed in [1]. Table 2 shows the approximate required space memory to keep the matrixes in Keshavarzi's method [1] and our suggested approach. It can be clearly seen that the proposed method needs very low memory in comparison with [1].

Besides, the used space memory of the proposed method is constant for 3, 4 and 5-similarity problems. This table truly shows why the proposed method in [1] is unable to work for large datasets and bigger amount of n as have been specified by "-" at Table 1.

Table 1: Comparison of the expectation time (sec.) of the proposed method in sequential and parallel for 3, 4 and 5-similarity

Dataset size	$n = 3$			$n = 4$			$n = 5$		
	[1]	Seq.	Par.	[1]	Seq.	Par.	[1]	Seq.	Par.
200	0.032	0.025	0.005	1.76	1.669	0.320	-	211.652	51.496
400	0.31	0.252	0.058	-	25.268	4.849	-	2899.12	664.935
600	0.882	0.868	0.185	-	128.207	23.919	-	4576.58	945.574
800	1.741	1.698	0.328	-	406.095	76.047	-	9543.86	2065.771
1000	3.621	3.570	0.682	-	991.948	179.375	-	14763.76	3006.875
1200	6.39	6.315	1.165	-	2057.19	366.048	-	18673.91	3712.506
1400	9.024	8.773	1.612	-	3824.16	673.267	-	24983.13	4661.031
1504	-	10.38	1.912	-	5262.34	926.468	-	26459.87	4909.066

Table 2: Comparison of approximate size of required memory for the proposed method and Keshavarzi's method [1] in solving n-similarity for $n = 3, 4$ and 5

Dataset size	$n = 3$		$n = 4$		$n = 5$	
	[1]	Proposed method	[1]	Proposed method	[1]	Proposed method
200	8 MB	**40 KB**	1.6 GB	**40 KB**	320 GB	**40 KB**
400	64 MB	**160 KB**	25.6 GB	**160 KB**	10.2 TB	**160 KB**
600	216 MB	**360 KB**	129.8 GB	**360 KB**	77.8 TB	**360 KB**
800	512 MB	**640 KB**	410.1 GB	**640 KB**	328 TB	**640 KB**
1000	1 GB	**1 MB**	1 TB	**1 MB**	1 PB	**1 MB**
1200	1.7 GB	**1.4 MB**	2 TB	**1.4 MB**	2.4 PB	**1.4 MB**
1400	2.7 GB	**2 MB**	3.8 TB	**2 MB**	5.3 PB	**2 MB**
1504	3.4 GB	**2.2 MB**	5.1 TB	**2.2 MB**	7.7 PB	**2.2 MB**

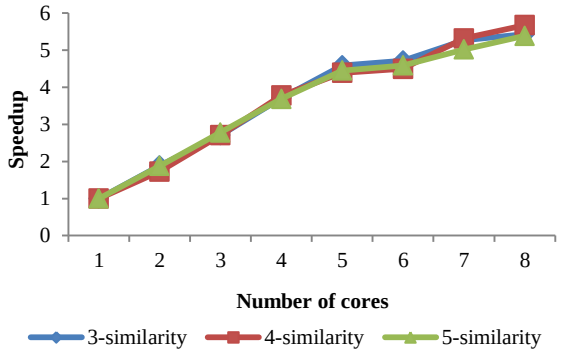

Fig. 5: The speedup of proposed method for Re0 dataset in the size of 1504 by changing the number of cores.

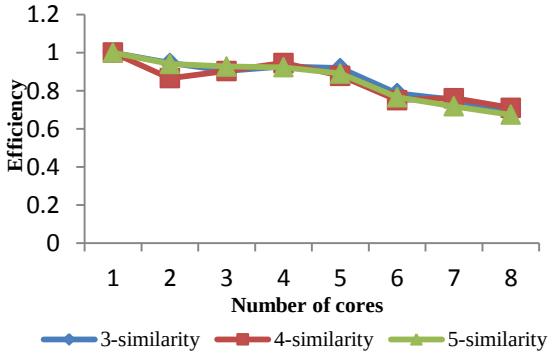

Fig. 6: The efficiency of the proposed method for Re0 dataset in the size of 1504 by changing the number of cores.

Fig. 5 shows the speedup by changing the number of cores from 2 to 8 for Re0 dataset with the size of 1,504, for 3-similarriy, 4-similarity and 5-similarity problems. For the 3-similrity, the speedup is changed from 1 to 5.43. For 4-similarity it is reached from 1 to 5.68 and in the case of 5-similarity, the speedup increases from 1 to 5.39 by changing the number of threads from 1 to 8.

It is observable that the trend of speedup for these three problems is so close to each other. Also, Fig. 6 shows the efficiency for Re0 dataset in the size of 1,504 for different number of cores from 1 to 8, for 3-similarity, 4-similarity and 5-similarity problems. The efficiency respectively reaches to 67.8%, 71.0% and 67.3% for 3-similarity, 4-similarity and 5-similarity by an 8-core CPU. The trend of this figure shows decreasing the efficiency from 100% to 68% in average. Also, as in Fig. 5, the efficiency trend is for the three problems is similar. Generally, the proposed parallel method leads to decreasing the execution time and increasing the speedup especially for larger datasets. As a consequence our method is able to deliver the exact result in a practically feasible time in comparison with the state-of-the-art exact method.

Conclusion

In this study, the theory of the n-similarity problem is reviewed. The executing time and the space complexity of the previous exact method to solve this problem are high and increase more by growing the size of dataset. In addition, the metaheuristics proposed to solve this problem generate non-deterministic solutions and are unable to compute the n-similarity values for all possible groups of n objects. Therefore, in this paper we proposed an exact method with low space complexity and improve its running time using parallelism by Open-MP. The experiments performed on a textual dataset by varying its size and reported results show that the proposed parallel method averagely accelerate the speed of the method as 5, 5.38 and 4.82 times for 3-similariy, 4-similarity and 5-similarity problems respectively. Also, it was mathematically proved that for each value of n, the space complexity was improved to $O(N^2)$, in which N is the size of dataset.

Author Contributions

M. Mirhosseini carried out the experiment, and wrote the manuscript. M. Fazlali helped supervise and write the paper.

Acknowledgment

The authors gratefully express sincere thanks to dear reviewers and editors for their guidance and valuable comments.

Conflict of Interest

The author declares that there is no conflict of interests regarding the publication of this manuscript. In addition, the ethical issues, including plagiarism, informed consent, misconduct, data fabrication and/or falsification, double publication and/or submission, and redundancy have been completely observed by the authors.

Abbreviations

(n-1)-sim	(n-1)-similarity
d_i	Document i
w_{ki}	the weight of word t_k in document d_i
2-sim	2-similarity
3-sim	3-similarity
Cosine_Sim	Cosine similarity measurement
MAX	maximum
MaxTread	Found maximum by each thread
min	minimum
N	Size of dataset
n-Sim	n-similarity
S	similarity
S_2	2-similarity
S_3	3-similarity
S_4	4-similarity
S_n	n-similarity
T	The set of terms
TF	Term frequency
tid	Thread ID
DF	number of documents

References

[1] M. Keshavarzi, M. A. Dehghan, M. Mashinchi, "Applications of classification based on similarities and dissimilarities," Fuzzy Information and Engineering, 4(1): 75-92, 2012.

[2] M. Keshavarzi, M. A. Dehghan, M. Mashinchi, "Classification based on 3-similarity, Iranian Journal of Mathematical Sciences and Informatics," 6(1): 7-21, 2011.

[3] M. Keshavarzi, "Classification based on similarity and dissimilarity", PhD thesis, Shahid Bahonar University of Kerman, Iran, 2010.

[4] S. Theodoridis, K. Koutroumbas, Pattern recognition, Academic Press, 2003.

[5] L. Kaufman, P. J. Rousseeuw, Finding Group in Data An Introduction to Cluster Analysis, Wiley, New York, 2005.

[6] W. J. Wang, "New similarity measure on fuzzy sets and on elements", Fuzzy Sets and Systems, 85(3): 305-309, 1997.

[7] H. Rezaei, M. Emoto, M. Mukaidono, "New similarity measure between two fuzzy sets," Journal of Advanced Computational Intelligence and Intelligent Informatics, 10(6): 946-953, 2006.

[8] J. Ye, "Cosine similarity measures for intuitionistic fuzzy sets and their applications," Mathematical and Computer Modeling, 53: 91–97, 2011.

[9] M. Mirhoseini, M. Mashinchi, H. Nezamabadi-pour, "Improving n-Similarity problem by genetic algorithm and its application in text document resemblance," Fuzzy Information and Engineering, 6: 263-278, 2014.

[10] M. Mirhoseini, H. Nezamabadi-pour, "Metaheuristic Search Algorithms in Solving the n-Similarity Problem," Fundamenta Informaticae, 152(2): 145-166, 2017.

[11] K. Lakshmanan, S. Kato, R. Rajkumar, "Scheduling Parallel Real-Time Tasks on Multi-core Processors," in Proc. 2010 31st IEEE Real-Time Systems Symposium: 259-268, 2010.

[12] M. K. Fallah, V. S. Keshvari, M. Fazlali, "A Parallel Hybrid Genetic Algorithm for Solving the Maximum Clique Problem," in Proc. High-Performance Computing and Big Data Analysis. TopHPC 2019. Communications in Computer and Information Science, 891: 378-393, 2019.

[13] M. K. Fallah, M. Mirhosseini, M. Fazlali, M. Daneshtalab, "Scalable Parallel Genetic Algorithm For Solving Large Integer Linear Programming Models Derived From Behavioral Synthesis," in Proc. 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP): 390-394, 2020.

[14] S. Hong, T. Oguntebi, K. Olukotun, "Efficient Parallel Graph Exploration on Multi-Core CPU and GPU", 2011 International Conference on Parallel Architectures and Compilation Techniques, Galveston, TX, pp. 78-88, 2011.

[15] M. Mirhosseini, M. Fazlali, G. Gaydadjiev, "A Parallel and Improved Quadrivalent Quantum-Inspired Gravitational Search Algorithm in Optimal Design of WSNs," High-Performance Computing and Big Data Analysis. TopHPC 2019. Communications in Computer and Information Science, 891: 352-366, 2019.

[16] P. Delisle, M. Krajecki, M. Gravel, C. Gagné, "Parallel implementation of an ant colony optimization metaheuristic with OpenMP," In Proceedings of the 3rd European Workshop on OpenMP (EWOMP'01): 1-7, 2001.

[17] L. Dagum, M. Menon. "OpenMP: an industry standard API for shared-memory programming," IEEE Computational Science and Engineering, 5(1): 46-55, 1998

[18] http://www.daviddlewis.com/resources/testcollections/reuters21578/.

[19] http://www.ai.mit.edu/projects/jmlr/papers/volume5/lewis04a/a11-smart-stop-list/.

[20] M. F. Porter, "An algorithm for suffix stripping", Program, 14(3): 130–137, 1980.

[21] C. Qimin, G. Qiao, W. Yongliang, W. Xianghu, "Text clustering using VSM with feature clusters", Neural Computing and Applications, vol 26, pp. 995- 1003, 2015.

Biographies

Mina Mirhosseini received her B.Sc. and M.Sc. degrees from Shahid Bahonar University, Kerman, Iran, in Computer Engineering and Computer Science, respectively. Currently, she is working towards a Ph.D. degree in Computer Science at Shahid Beheshti University, Tehran, Iran. Her research interests include parallel processing, metaheuristic, text processing and wireless sensor networks.

Mahmood Fazlali received B. Sc in computer engineering from Shahid Beheshti University (SBU) in 2001. Then he received M.Sc from University of Isfahan in 2004, and PhD from SBU in 2010 in computer architecture. He performed researches on reconfigurable computing systems in computer engineering lab at Technical University of Delft (TUDelft) as a postdoc researcher. Now, he is working as assistant professor at department data and computer sciences at SBU. His research interest includes parallel processing, reconfigurable computing, and data sciences.

How to cite this paper:
M. Mirhosseini, M. Fazlali, "Parallel and Exact Method for Solving n-Similarity Problem," Journal of Electrical and Computer Engineering Innovations, 8(2): 193-200, 2020.

DOI: 10.22061/JECEI.2020.7247.377

URL: http://jecei.sru.ac.ir/article_1461.html