Scientific Journal Impact Factor (SJIF) : 1.711
International Journal of Modern Trends in Engineering and Research
ISSN (ONLINE) : 2349 - 9745
ISSN (PRINT) : 2393 - 8161

Top-k Similarity Join Algorithm: An Indexing based approach

Sneha Kolhe, Prof. V. B. Patil
Computer Science and Engineering, MIT, Aurangabad
Computer Science and Engineering, MIT, Aurangabad

Abstract— There is a wide range of applications that require finding the top-k most similar pairs of records in a given database. However, computing such top-k similarity joins is a challenging problem today, as there is an increasing trend of applications that expect to deal with vast amounts of data. An algorithm, top k-join, is proposed to answer top-k similarity join efficiently. It is based on the prefix filtering principle and employs tight upper bounding of similarity values of unseen pairs. It also uses inverted indexes to find out candidates pairs i.e an indexed based approach. Experimental results demonstrate the efficiency of the proposed algorithm on large-scale real datasets.

Keywords- Similarity join, top-k join, prefix filtering principle, All-pairs algorithm, top-k processing

I. INTRODUCTION

Similarity join is a useful primitive operation underlying many applications, such as near duplicate Web page detection, data integration, and pattern recognition. Existing similarity search methods [2] require users to specify a similarity function and a similarity threshold. They find those strings with similarities to the query string within the given threshold. However it is rather hard to give an appropriate threshold, as a small threshold will involve many dissimilar answers and a large threshold may lead to few results. To address this problem, in this paper we study the problem of top-k string similarity search, which, given a collection of strings and a query string, returns the top-k most similar strings to the query string. There are many similarity functions to quantify the similarity of two strings, such as Jaccard similarity, Cosine similarity, and edit distance. In this paper, we use jaccard similarity.

Top-k similarity join poses several challenges to designing efficient query processing algorithms. Although many similarity join algorithms have been proposed [3], [4], their efficiency heavily depends on a constant, given similarity threshold, and hence they cannot be directly applied to the top-k similarity join problem. On the other hand, the top-k similarity join problem in vector spaces with Euclidean distance functions was studied in spatial databases [5]. However, their algorithms are specially tailored for Euclidean distance functions and not applicable to similarity (or distance) functions such as Jaccard and cosine similarities.

The contributions of this paper can be summarized as follows:

• We propose an Indexing based algorithm to answer top-k similarity join queries. Unlike traditional similarity joins algorithms, the proposed algorithm can progressively compute join results without need of a prior given similarity threshold.

• We first develop similarity join algorithm using prefix filtering principle and then extend it to find out top-k results i.e. ranking.

• The experimental results show that the new algorithm outperforms alternative algorithms in most cases and is applicable to the interactive application scenarios.

Paper Structure The rest of the paper is organized as follows: Section II introduces preliminaries and background on traditional similarity join algorithms. Section III introduces our basic top-k similarity join algorithm. We discuss implementation details and extensions to other

similarity functions in Section IV. Experimental results are given in Section VI and related work appears in Section VII. Section VIII concludes the paper.

II. PRILIMINARY

2.1. Problem Definition

In this paper, we consider only one set of records which finds records similar to given query, by calculating similarity.

We consider several commonly used similarity functions for sets and vectors:

- *Jaccard similarity* is defined as, $J(x,y)=\frac{|x\cap y|}{|x\cup y|}\geq t$
- *Cosine similarity* is defined as, $C(x,y)=\frac{\vec{x}\cdot\vec{y}}{\|\vec{x}\|\cdot\|\vec{y}\|}\geq t$
- *Dice similarity* is defined as, $D(x,y)=\frac{2|x\cap y|}{|x|+|y|}\geq t$

Here, we consider jaccard similarity. Suppose x is given query record and y is dataset record then jaccard similarity function Sim(x, y), where t is threshold value.

2.2. Prefix Filtering Principle

A common method to reduce the number of pairs needed to be checked for similarity when approaching the similarity join problem is to apply the prefix filtering principle. Prefix filtering makes use of the fact that if the elements in the records are ordered, for example by frequency, then some parts of two sets must intersect in order for them to have a similarity equal to or larger than a specified threshold. As Xiao et. al puts it: "Consider an ordering *O* of token universe *U* and a set of records, each with tokens sorted in the order of *O*. Let the p-prefix of a record x be the first p tokens of x. The prefix filtering principle is useful when generating pairs to be checked for exact similarity. It is only necessary to calculate pairs which share a token in their prefixes if they should be able to reach this threshold. The pairs that satisfy this condition are called *candidate pairs*.

Existing approaches usually follow the following filter-and-refine framework:

Indexing Phase Inverted indices are built for tokens that appear in the prefixes of the records.

Candidate Generation Phase The inverted indices for tokens in the prefix of each record are probed to generate a set of *candidate pairs*. The two records of a candidate pair are guaranteed to share at least one token in their prefixes. The candidate pairs are guaranteed to be a superset of the final results due to the prefix filtering principle.

Verification Phase Each candidate pair is evaluated against the similarity constraint and added to the final results if its similarity is no less than the threshold.

III. SIMILARITY JOIN ALGORITHMS

3.1. All-Pairs Algorithm

We now review All-Pairs [1], which is a state-of-the-art, prefix-filtering-based algorithm for processing similarity joins. The theoretical algorithm for All-Pairs is given below. It takes as the input a collection of records already sorted in the ascending order of their sizes. It iterates through each record x, looking for candidates that intersect *x*'s prefix (Lines 2-3). Afterwards, x and all its candidates will be verified against the similarity threshold to return the correct join results (Lines 4).

Algorithm 1: All-Pairs

Input: Canonicalized records, query record *x*, threshold value *t*.
Output: All similar pairs such that $(x,y) \geq t$.

1. Tokenize each record into a set.
2. Create inverted indexes using prefix filtering principle.
3. Generate candidate pairs *(x,y)* such that *x* and *y* shares at least one token.
4. Find all pairs of records, <*x*,*y*>, such that sim$(x,y) \geq t$.

All-pair algorithm only retrieves similar pairs of records. The drawback with the algorithms for the similarity join problem is the similarity threshold that needs to be specified. If we knew the exact threshold needed to produce k pairs, this algorithm would be useful to us, but as it is, we would have to test different thresholds until we returned the number of pairs that we wanted. This would be an inefficient solution.
The Top-k Join algorithm, which is based on the All-Pairs algorithm, but does not require a threshold, retrieves similar records with ranking so the topmost record has higher similarity value than of its next.

3.2. Top-k Join Algorithm

The Top-k Join algorithm is just like the All-Pairs algorithm based on prefix filtering, and we will only consider pairs which share a word in their prefixes. The difference from the problem discussed in the previous section is that the threshold will not be constant in this case - it will be gradually increased. We keep a min-heap, called the result heap, in which we will store the *k* currently most similar pairs that have been calculated exactly. The heap is initialized with k arbitrarily chosen pairs. In order for a new pair to be put on the heap, it needs to be more similar than the pair that is currently the k^{th} best, and we will denote this similarity *sk*. This value corresponds to the threshold in the similarity join problem.

Algorithm 2: Top-k-Join(R)

Input: R is a collection of records; each record has been canonicalized by a global ordering *O*, query string *x*.
Output: Top-k pairs of records ranked by their similarity value.

```
1.  E ← InitializeEvents(R);
2.  T ← InitializeTempResults(R); /* Store any k pairs as temp results in T */;
3.  Ii ← 0;  (1 ≤ i ≤ |U|);
4.  while E ≠ 0; do
5.     (x, px, spx) ← E.pop();
6.     if spx ≤ sk then
7.     break;  /* stop here */;
8.     w ← x[px];
9.     sk ← T[k].sim;
10.    for j = 1 to |Iw| do
11.        y ← Iw[j];
12.        sim(x, y) ← CalcSimilarity(x, y);
13.        T.add((x, y), sim(x, y));
14.    Iw ← Iw U { x };  /* index the current prefix */;
15.   px ← SimilarityUpperBoundProbe(x, px + 1);
```

16. E.push(x, $p_x + 1$, s'_{px})
17. Return T

Algorithm 2: InitializeEvents(R)

1. E ← 0;
2. **for each** x in R **do**
3. E.push(x, 1, 1.0); /* E is a max-heap */;
4. Return E;

Algorithm 4: UpperBoundProbe

Input: A record x and a prefix length p.
Output: The upper bound of the similarity value of x and another record provided that $x[p]$ is the first common token of x and the other record.

1. $s_{px} \leftarrow 1 - \frac{px - 1}{|x|}$
2. Return s_{px}

Algorithm 2 describes this incremental similarity join algorithm. A fixed sized min-heap T is used to keep the largest k pairs seen so far. $T[k]$ gives the pair with the k-th largest similarity. For the ease of illustration, we can initialize T with any k pairs so that it is full. The prefix length for each record is initialized as 1 first. For each record x, we insert (x, 1, 1.0) into a max heap based on the upper bounds. The similarity upper bound for these prefix events are initialized as 1.0 (Line 1). The algorithm then iteratively pulls the next prefix event (x, p_x, s_{px}) and probes the inverted lists of tokens in the probing prefix $x[1 . . p_x]$. x is paired with the records that share at least one token with x in their prefixes. The pairs and their similarity values returned by the similarity function are regarded as temporary results and added to T. Next, we extend x's prefix length to p_x+1, and recalculate the similarity upper bound for x with the current prefix length of p_x+1 (Line 17). Finally, the new event (x, $p_x + 1$, s'_{px}) is pushed into the max heap for prefix events. The algorithm stops when the max heap for prefix events is empty or all the remaining events in the heap has a lower similarity upper bound than that of the k-th result in T.

IV. RESULTS AND EXPERIMENTATION

We have developed all-pair and top-k algorithms and used DBLP dataset to perform various experiments. DBLP is a snapshot of the bibliography records from the DBLP Web site http://www.informatik.uni-trier.de/~ley/db. Our dataset contains about 7,847 records; each record is a concatenation of author name(s) and the title of a publication. Here, we used query record string Q, which is joined with the records in our DBLP dataset depending on similarity value.

Case No.	Search String	Threshold Value	Running Time (In milliseconds)	No. of Records Returned
1	switching networks	0.001	621	23
2	networks switching	0.001	429	23
3	SWITCHING NETWORKS	0.04	100	21
4	switching networks	0.04	671	21
5	switching networks	0.07	669	11

6	switching networks	0.1	356	06
7	switching networks	0.4	754	00
8	Network	0.01	333	17
9	Networks	0.01	551	19
20	Networking	0.01	341	05

Case No.	Search String	Order/Result	Running Time		No. of Records Returned	
			Jaccard	Cosine	Jaccard	Cosine
1	algorithm chen	Changed	328	283	613	613
2	algorithms chen		584	543	349	349
3	simulation algorithms guang	Changed	228	157	336	336
4	Simulation algorithms ke chen		638	602	977	977
4	Algorithm	-	451	276	451	443
5	Guang	-	89	84	44	44

4.1. All-Pairs with different threshold values

In this section, we will see some experiments on All-Pairs algorithm with different threshold values. Here, from above observation, we can see that as threshold value increases the numbers of returned records are decreased. Table 1 shows detail results of all-pairs algorithm. We analyzed that if we run algorithm for various query records, then we get same result using jaccard and cosine. We analyzed that if we run algorithm for various query records, then we get same result using jaccard and cosine. Our observation leads to think that cosine takes less running time than jaccard. Following figures shows measures for top-k join algorithm for parameters: running time and number of records returned.

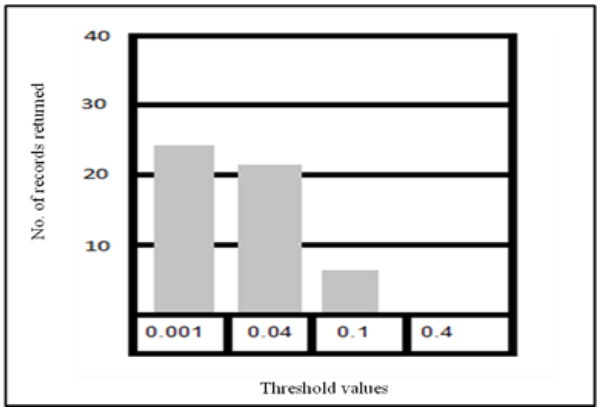

Fig. 1 All-Pairs result with different threshold values

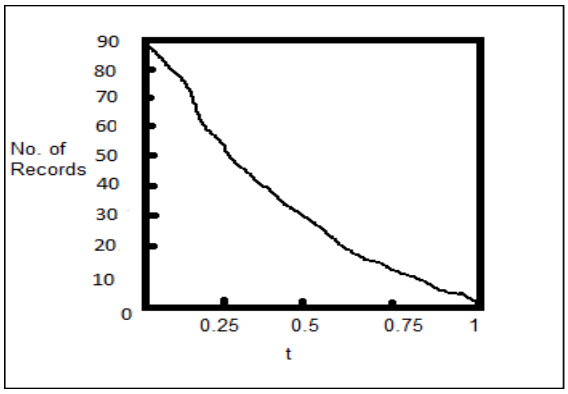

Fig. 2 Relation between threshold and no. of records returned

4.2. Top-K Join with Jaccard and Cosine

From table 1, we analyze that if we run same algorithm for same query record multiple time using jaccard and cosine similarity measures then, using both measures we get same result each time.
From table 2, we analyze that if we run algorithm for various query records, then we get same result using jaccard and cosine. Our observation leads to think that cosine takes less running time than jaccard. Following figures shows measures for top-k join algorithm for parameters: running time and number of records returned.

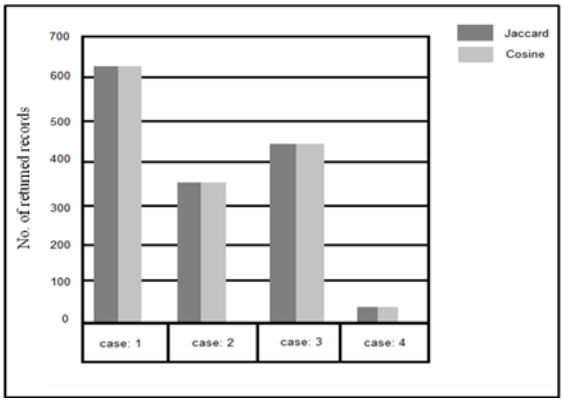

Fig. 3 Running time, jaccard and cosine, DBLP

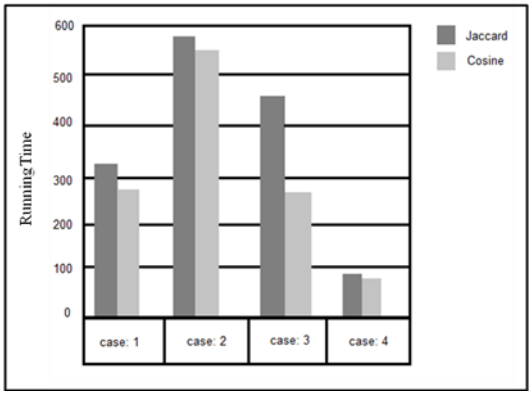

Fig. 4 Running time, jaccard and cosine, DBLP

V. CONCLUSIONS

Here, All-pairs algorithm simply generates all similar records without any ranking. But, top-k algorithm returns all similar records ranked by their similarity value. The drawback with the all pair

algorithm for the similarity join problem is the similarity threshold that needs to be specified. But, the Top-k Join algorithm, which is based on the All-Pairs algorithm, does not require a threshold. Top-k algorithm gives exact match at the top positions and rest of all at the next, according to similarity values. By extending single-attribute search to multi-attribute search, we can retrieve high similarity records. Another observation is that, even jaccard and cosine similarity measures returns same result still their running time is different. Cosine takes less running time than jaccard.

REFERENCES

[1] Chuan Xiao, Wei Wang, Xuemin Lin Haichuan Shang, IEEE 25th international conference on "Top-k Set Similarity Joins", pages 914-927, March 2009.
[2] R. J. Bayardo, Y. Ma, and R. Srikant, Scaling up all pairs similarity search," in *WWW*, 2007.
[3] C. Xiao, W. Wang, X. Lin, and J. X. Yu, "Efficient similarity joins for near duplicate detection," in *WWW*, 2008.
[4] S. Sarawagi and A. Kirpal, "Efficient set joins on similarity predicates," in SIGMOD, 2004.
[5] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos, "Closest pair queries in spatial databases," in *SIGMOD Conference*, 2000, pp. 189–20.
[6] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman "A Survey of Top-k Query Processing Techniques in Relational Database Systems" ACM Computing Surveys, Vol. 40, Article 11, October 2008.
[7] C. Li, J. Lu, and Y. Lu. "Efficient merging and filtering algorithms for approximate string searches", in *ICDE*, pages 257–266, 2008.

