A Transformation-based Framework for KNN Set Similarity Search (Extended Abstract)

Yong Zhang†, Jiacheng Wu†, Jin Wang‡, Chunxiang Xing†
† RIIT, TNList, Dept. of Computer Science and Technology, Tsinghua University, Beijing, China.
‡ Computer Science Department, University of California, Los Angeles.
{zhangyong05,xingcx}@tsinghua.edu.cn; wu-jc18@mails.tsinghua.edu.cn; jinwang@cs.ucla.edu;

I. INTRODUCTION

Set similarity search is a fundamental operation in a variety of applications [3], [5], [2]. There is a long stream of research on the problem of set similarity search. Given a collection of set records, a query and a similarity function, the algorithm will return all the set records that are similarity with the query. There are many metrics to measure the similarity between two sets, such as Overlap, Jaccard, Cosine and Dice. In this paper we use the widely applied Jaccard to quantify the similarity between two sets, but our proposed techniques can be easily extended to other set-based similarity functions. Previous approaches require users to specify a threshold of similarity. However, in many scenarios it is rather difficult to specify such a threshold. For example, when users types some keywords in the search engine, they will pay more attention for the results which rank in the front, say the top five ones. In this case, if we use threshold-based search instead of KNN similarity search, it is difficult to find the results that are more attractive for users.

In this paper, we study the problem of KNN set similarity search, which given a collection of set records, a query and a number $k$, returns the top-$k$ results with the largest Jaccard similarity to the query. We will use “KNN search” for short in the paper without ambiguity. There are already some existing approaches for threshold-based set similarity search and join, one straightforward solution is to extend them to support KNN search as following. This can be done by initializing the similarity threshold as 1 and decreasing it by a fixed step (say 0.05) every time. For each threshold, we apply existing threshold-based approaches to obtain the similar records. This step is repeated until we obtain $k$ results. However, this simple strategy is rather expensive as we need to execute multiple search operations during enumeration. Besides, as there is infinite number of thresholds, it is difficult to select a proper value of step. A large step will result in more than $k$ results, which include many dissimilar records; while a small step will lead to more search operations and thus bring heavy overhead. There are also some previous studies on the KNN similarity search with edit distance constraint on string data. They adopt filter-and-verify frameworks and propose effective filtering techniques to avoid redundant computing on dissimilar records. As verifying the edit distance between two strings requires $O(n^2)$ time, they devise complex filters to reduce the number of verifications. However, as the verification time for set similarity metrics is just $O(n)$, it is not proper to adopt such techniques for edit distance to support our problem due to their heavy filter cost. Similar phenomenon has also been observed in a previous study of exact set similarity join: it reports that the main costs are spent on the filtering phase, while the verifications can be done efficiently. Therefore, it calls for novel techniques to support KNN search.

To address above issues, we propose a transformation based framework to efficiently support KNN set similarity search. To perform effective filtering, we transform all set records with variant lengths to representative vectors with fixed length. By carefully devising the transformation, we can guarantee that the representative vectors of similar records will be close to each other. We first provide a metric to evaluate the quality of transformations. As achieving optimal transformation is NP-Hard, we devise a greedy algorithm to generate high quality transformation with low processing time. Next we use a $R$-Tree to index all the representative vectors. Due to the properties of $R$-Tree, our work can efficiently support both memory and disk based settings. Then we propose an efficient KNN search algorithm by leveraging the property of $R$-Tree to prune dissimilar records in batch. We further propose a dual-transformation based algorithm to capture more information from the original set records so as to achieve better pruning power.

Moreover, as in many situations it is not required to return the exact KNN results, we also propose an approximate KNN search algorithm which is much faster than the exact algorithm and with high recall at the same time. To reach this goal, we devise an iterative estimator to model the data distribution. We evaluate our proposed methods using three popular datasets, on both memory and disk based settings. Experimental results show that our framework significantly outperforms state-of-the-art methods.

II. METHODOLOGY

We first give the formal definition of the KNN Set Similarity Search as shown in Definition 1. Here we use Jaccard as the similarity metrics between two set records.

Definition 1: Given a collection of set records $U$ and a query $Q$, the KNN Set Similarity Search returns a subset $R ⊆ U$ such that $|R| = k$ and for $∀X ∈ R$ and $Y ∈ U \setminus R$, we have $Jaccard(X,Q) ≥ Jaccard(Y,Q)$. 
Next we introduce the transformation based framework to support KNN set similarity search. It first transforms all records in the dataset into fixed-length vectors. Then such vectors are indexed with an R-Tree structure. The KNN search is performed by traversing the R-Tree and dissimilar records can be pruned in batch in this process.

**Records Transformation** The purpose of performing transformation is to eliminate redundancy in the index structure and reduce filter cost. The basic idea is that for each record \( X \) in the dataset, we transform it into a representative vector with fixed length \( m \) denoted as \( \omega[X] \). Then we can deduce a bound of set similarity from the transformation distance between representative vectors. To this end, we group all tokens from records in the dataset into \( m \) groups, \( \{G_1, G_2, ..., G_m\} \). For a record \( X \), we have the number of tokens that belong to group \( i \) as \( \sum_{t \in G_i} \mathbf{1}\{t \in X\} \), which will be the value on its \( i^{th} \) dimension. Then we can define the transformation distance between two records by looking at the representative vectors. We have shown that obtain an optimal transformation is NP-Complete. Therefore, we propose a greedy grouping algorithm to generate the transformation for all records.

**Exact KNN Algorithm** After obtaining all the transformed vectors, we index them into the R-Tree index, which can naturally support both in-memory and disk settings. Then we can use the property of R-Tree to perform batch pruning when conducting the KNN similarity search. Specifically, given a query \( Q \), we can estimate the upper bound of Jaccard similarity between \( Q \) and all records in a node of R-Tree with the lower bound of the transformation distance. This is realized by considering the Query-Node Minimum Transformation Distance as shown in Definition 2.

**Definition 2 (Query-Node Minimum Transformation Distance):** Given a record \( Q \) and a node \( N \), the minimum transformation distance between \( \omega[Q] \) and \( N \), denoted as MinDist(\( \omega[Q], N \)), is the distance between the vector and the nearest plane of hyper rectangle of \( B_N \).

\[
\text{MinDist}(\omega, Q, N) = 1 - \left( \frac{n(\omega, Q, N)}{d(\omega, Q, N)} - 1 \right)^{-1}
\]

where

\[
n(\omega, Q, N) = \sum_{i=1}^{m} \begin{cases} 
\omega_i[Q] + B_i^\perp & \omega_i[Q] < B_i^\perp \\
\omega_i[Q] + \omega_i[Q] & B_i^\perp \leq \omega_i[Q] < B_i^\top \\
\omega_i[Q] + B_i^\top & B_i^\perp \leq \omega_i[Q] \end{cases}
\]

and

\[
d(\omega, Q, N) = \sum_{i=1}^{m} \begin{cases} 
\omega_i[Q] & \omega_i[Q] < B_i^\perp \\
\omega_i[Q] & B_i^\perp \leq \omega_i[Q] < B_i^\top \\
B_i^\perp & B_i^\top \leq \omega_i[Q]
\end{cases}
\]

The basic idea of performing KNN search on a dataset is as following: we maintain a priority queue \( R \) to keep the current \( k \) promising results. Let UB denote the largest Jaccard distance between the records in \( R \) to the query \( Q \). Obviously UB is an upper bound of the Jaccard distance for KNN results to the query. In other words, we can prune an object if its Jaccard distance to the query is no smaller than UB. The query will terminate when UB is larger than the Query-Node Minimum Transformation Distance between \( Q \) and any node in the R-Tree index.

**Optimizations** To accelerate the processing of KNN similarity search, we propose two additional techniques: Multiple Transformation and Approximate KNN algorithm.

With a single transformation, we can only reveal a facet of the features in a set record. Therefore, the pruning power could be weaken due to loss of information. To address this problem, we can utilizing multiple independent transformations to excavate more information of data distribution. And then the upper bound of Jaccard similarity will be the lowest one among the Query-Node Minimum Transformation Distance computed with multiple transformations, which will definitely be tighter. We also show that multiple transformations can be generated from the first transformation with the idea similar with greedy grouping algorithm.

The general idea of approximate algorithm is to estimate the KNN results by considering the distribution of data. Then we can find the KNN results according to the “density” of data without traversing the index. This is realized by partitioning the whole dataset into several buckets and then incrementally search from the query point to identify the similar records. In this way, we can successfully find all the similar records within a circle having a predefined radius, which might not be the exact KNN results but will definitely make a good approximation. The buckets used in this process can be decided smartly using the MBRs of the existing R-Tree index.

### III. Evaluation

We conduct extensive set of experiments on three real world datasets: DBLP, PubMed and LiveJournal. We compare with two existing method Flamingo [1], Tree [6] and one approach extended from state-of-the-art method [4]. We also conduct experiments on the disk based setting as well as the approximate algorithm. The results show that our proposed method outperforms existing methods by up to an order of magnitude. Detailed results can be found in the full version [7].

**Acknowledgment** This work was supported by National Key R&D Program of China (2018YFB1404401, 2018YFB1402701) NSFC (91646202).

### References


