PEXESO: Finding Joinable Tables by Distance-based Similarities

Yuyang Dong[†] Kunihiro Takeoka[†] Masafumi Oyamada[†]
[†]*NEC Corporation, Japan*
{y-dong@aj, k-takeoka@az, m-oyamada@cq}.jp.nec.com

***Abstract*—Finding joinable tables from massive data lakes is an important problem with many applications in data integration, data (feature) augmentation, data analysis, and data market. For example, when we hold a POS (point-of-sale) table of a store and want to build a machine learning model of sales prediction, we can search for local customer profile tables or weather tables, then join them together to add features and train a high-quality model. Existing works on finding joinable tables only focus on the exact or similarity join of columns with string data, and no one involves the numerical data. However, there are also many applications for numerical data such as similarity join on the columns with location (spatial) data, image data, and the numerical vectors which are transformed from textual data by (word, document, graph, etc.) embedding techniques. The similarity between numerical data is normally measured by distance-based similarity functions (e.g. L2 distance).**

If two numerical columns have enough matched values, it may make sense to say that they are joinable. This problem can be formalized to a threshold similarity search problem on a huge collection of sets, and each set contains a collection of numerical vectors. In this paper, we define a novel searching problem to find joinable tables with distance-based similarities on numerical data. To solve this problem, we developed PEXESO, a general framework that handles arbitrary threshold values and a large space of similarity functions. PEXESO groups similar vectors with a pivot-base group structure to reduce a huge number of distance computations when determining joinable between two sets. To further reduce the computations, PEXESO takes advantage of the dominating relations between groups in different sets to prune candidate sets and groups. Experiments on real and synthetic datasets demonstrate the efficiency of our method and its superiority over baseline solutions.

I. Introduction

Join is a fundamental and essential operation that connects two or more tables together, and it is also a significant technique applied to relational database management systems and business intelligence tools for data analysis. The benefits of joining tables are not only in the database fields (data integration, data mining, data indexing) but also in the machine learning fields such as feature augmentation and data enrichment [15], [18], [19], [27]. For example, by combining multiple tables together with the join operation, there are more features (columns) and more samples (rows) so that we can build a better quality machine learning model [18].

Traditionally, the main problem of join is how to efficiently combine known-to-joinable tables connected by key-foreign key definitions. However, recent trends on open data movements by governments and dissemination of data lake solutions in industries increase the opportunities to obtain possibly-joinable tables joinable that could improve the quality of data. To leverage such an external "treasure-trove", it is necessary to check relevance of user-owned tables and external tables to decide if the tables are joinable or not because tables from open data repositories and data lakes do not have explicit key-foreign key definitions. Since computing relevance for every table in a huge amount of external data repositories is computationally heavy, efficient methods for finding joinable tables from a large data source are recently proposed [7], [22], [31], [35], [37].

Although finding joinable tables from a large data source is a popular research problem in the database community recently, all existing works focus on the textual (strings) columns. For example, join the "Name" column in Table Ia with the "Host" column in Table Ib. Existing approaches including exact overlap-based set similarity search [22], [35], [37] and fuzzy token similarity [7], [30] cannot support similarity with the distance-based metric between numerical columns. The numerical columns are discarded in their system. However, numerical columns also account for a large proportion of data, e.g., 77% in UCI machine learning repository [1]. It is wasteful to just discard them.

Name	Income	Age	Address
Tom	50	45	414 EAST 10TH STREET, 4E
S.Bruce	34	24	206 EAST 7TH STREET, 17,18
Jerr.	24	30	616 EAST 9TH STREET, 4W
Tyke	100	20	303 EAST 8TH STREET, 3F

(a) Membership table

Name	Host	Location	Price
Cozy Clean	Tom	$\langle 35.24, 139.87\rangle$	125
Cute & Cozy	Jerry	$\langle 35.31, 139.35\rangle$	540
Central Manhattan	Spike	$\langle 35.63, 139.74\rangle$	1265
Sweet and Spacious	Tyke	$\langle 35.66, 139.63\rangle$	1000

(b) Airbnb information table

Apartment ID	Lat-long	Sale Price
A345	$\langle 35.67, 139.65\rangle$	5400
F885	$\langle 35.64, 139.74\rangle$	6200
G245	$\langle 35.33, 139.38\rangle$	2140
K982	$\langle 35.57, 139.65\rangle$	1020

(c) House sales table

TABLE I: Examples of similarity joinable tables with textual and numerical columns.

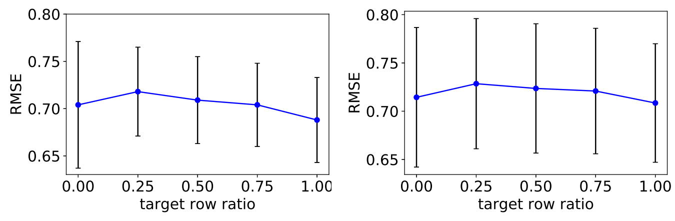
[1]http://archive.ics.uci.edu/ml/index.php

In this paper, we consider finding joinable tables with numerical columns for allowing three reasons:

(1) Similarity join with numerical values is important and essential. For example, Table Ib is a table of Airbnb rooms and Table Ic is a table of House price. We can join an Airbnb room to a closed house by computing the distance between the coordinates in "location" and "Lat-long" columns, then execute a prediction on the price of Airbnb rooms [2]. We conduct a preliminary experiment to test the effect of similarity join for feature augmentation with machine learning techniques. We execute similarity join to connect the Airbnb data [3] with House price data [4] in New York city. We join the tables with the columns of "latitude and longitude" numerical coordinates with Euclidean distance. Two types of similarity joins are executed: (a) Top-1: a room of Airbnb table is joined with an apartment in House data which have the smallest distance; and (b) Range Aggregation: a room of Airbnb table is set as a center, and it joins with aggregation information from the apartment in a distance range from this center. We sample a part of Airbnb's rows as a query table, then join a target table which sampled from House's rows with different ratio to the query table, so that we can test the effect of a different number of rows in the target table. After joining, combined table are used to predict the price of Airbnb rooms with the random forest regression model. Table II and Figure 1 show the RMSE (Root Mean Square Error) of the prediction, and note that the less RMSE value means the better performance. The RMSE first increases when joining with a small target table then decreases with the target table gets large. Therefore, when the target table has enough similar values to be joined, similarity join helps to build a better model.

Target row ratio	RMSE-0.02%-Threshold	RMSE-Top-1
0	0.704 ± 0.067	0.714 ± 0.072
0.25	0.718 ± 0.047	0.728 ± 0.067
0.50	0.709 ± 0.046	0.723 ± 0.067
0.75	0.704 ± 0.044	0.720 ± 0.065
1.0	$\mathbf{0.688 \pm 0.045}$	$\mathbf{0.708 \pm 0.061}$

TABLE II: Prediction RMSE values with different target rows in our Airbnb price prediction experiment.



(a) Threshold join. (b) Top-1 join.

Fig. 1: Prediction RMSE trends with different target rows in our Airbnb price prediction experiment.

(2) In order to better discover the relation between data, people loves to transform the non-numerical data into numerical one. For example, we can transform the "Address" column in Table Ia from string to numerical coordinates, then it can join with the other tables. With the developing of the technologies in data lake, we can not only store the huge meta-data but also index them into different representations. Embedding is a vector representation technology which is a popular data pre-processing because it has superior performance on machine learning processing than traditional models. By now, almost everything can be represented as a numerical vector by word, document, graph, image etc, embedding techniques. Therefore, it is an important issue to support finding joinable tables with considering the numerical data and similarities.

(3) Even though there are many techniques in the similarity join field, they can not solve the problem of searching joinable tables. Many research studies the similarity join between two sets of numerical data [3], [5], [9], [10], [13], [17], [24], [25]. However, the key different is that they are design for similarity join two sets rather than searching joinable tables. [3], [5], [17], [17], [25] is proposed in a fashion that on pre-indexing both two sets, it is not allowed in the searching problem since we can not know query table in advance. [9], [24] can not deal with an arbitrary condition of query and requiring heavy index rebuilding in some cases. [10], [13] support one time join with two sets without underline indexing, which can not fit for the searching problem that issues queries repeatedly.

Based on the above motivation, we study the problem of finding joinable tables on numerical columns with distance-based similarities. To the best of our knowledge, this is the first work targeting the numerical data with finding joinable tables. We proposed a framework named PEXESO[5] to solve this problem. PEXESO is an efficient framework can deal with any query condition and a huge space of similarity functions, and mainly addresses the following two challenges:

Expensive similarity computations. The distance computing between numerical cells is expensive because they are always multiple dimensional vectors, e.g., word2vec data [11] has 50 dimensions to 200 dimensions. Therefore, it is time-consuming if we naively compute the similarity between all pairs of cells. To reduce the distance computation of this challenge, PEXESO splits a set into groups and processes them in a divide and conquer way. Two filtering methodologies, lower bound filtering and reverse filtering are proposed to prune the sets and early terminate when a set is clearly joinable or unjoinable. PEXESO also carries out a pivot-based filtering on an MVP-tree to filter the groups in a branch-and-bound way.

Large table size. Normally, there are huge number of tables in the data lake, so it is time-consuming to check these tables joinable or not one by one. To filter the search space of the

[2]Intuitively, an Airbnb room has expensive price when the house is expensive.

[3]https://www.kaggle.com/dgomonov/new-york-city-airbnb-open-data

[4]https://www.kaggle.com/new-york-city/nyc-property-sales

[5]PEXESO is a card game and the objective is to find matched pairs.https://en.wikipedia.org/wiki/Concentration_(card_game)

table, PEXESO takes advantage of the dominating relations between groups and further prune groups in other tables. Also, PEXESO uses a greedy algorithm to select the next table to process, so that maximum prunes the unchecked tables.

Our contributions are summarized as follows.

- We propose and study a novel and important problem that searching for joinable tables on numerical columns with distance-based similarities. We formalize the searching problem to a set similarity search problem.
- We conduct a preliminary experiment and show that the similarity join can be an effective tool for feature augmentation with building a machine learning model.
- We propose PEXESO, a framework of our searching problem which can deal with all metric distance-based similarities. PEXESO splits data into groups and uses pivot-based filtering to reduce the distance computations between two sets. Moreover, PEXESO further reduces the computations by pruning sets with group dominating relations while searching.
- We prove the problem of determining the processing order in PEXESO is NP-hard, and propose a greedy algorithm for approximation. The proposed greedy algorithm shorten 1.2 to 2.1 times of processing time than a random order processing one.
- We conduct sufficient experiments on different types of real data including spatial data, image data and word2vec data. The performance of PEXESO outperforms baseline methods by up to 5.8 times speed-up.

The rest of the paper is organized as follows. Section II introduces basic concepts and defines the searching problem. Section III introduces the group structure and pivot-based filtering techniques. Section IV presents the dominating relations and dominating filter techniques. Experimental results are reported and analyzed in Section V. Section VI reviews related work, and Section VII concludes the paper.

II. Preliminaries

In this section, we present a formal definition and notations of the joinable table search problem, and some background material on metric space similarity, pivot-based filtering and inverted index. Table III lists the notations frequently used in this paper.

A. Metric Space Similarity

Metric space is a universal and versatile model of similarity that can be applied in various areas of information retrieval. A metric space is represented as a two-tuple (O, d), in which O is an object universe and d is a distance function for measuring the "similarity" between two objects in O. The distance function d satisfies four properties: (1) symmetry: $d(p, q) = d(q, p)$; (2) non-negativity: $d(p, q) \geq 0$; (3) identity: $d(p, q) = 0$ iff $p = q$; and (4) triangle inequality: $d(p, q) \leq d(p, o) + d(q, o)$.

In this paper, we assume that a cell data in columns of the table can be normalized or embedded as an object in a metric space. An object o is represent as a numerical vector $o = \{o_1, o_2, o_3, ..., o_n\}$, where o_i is the element value in i-th dimension. For example, table cells with the longitude and latitude coordinates, can be normalized an object in (0,1] metric space, and we can use Euclidean distance (L2-norm) to compute the similarity between two cells.

TABLE III: Frequently used notations.

Symbol	Description
$\mathbb{S}$	Collection of sets
Q	Query set
d	distance function
f	Set similarity function
τ	Distance threshold
T	Similarity Threshold
$M_\tau(a, b)$	Objects a and b are matched, that $d(a, b) \leq \tau$
g	Group
G_X	collection of groups for set X
P	Pivot set
C	Candidate group center
$Dom(g_a, g_b)$	g_1 dominates g_2
$Dom_\tau(g_a, g_b)$	g_1 dominates g_2 with a threshold τ

B. Problem Definition

In a searching system for joinable tables, users are asked to input a table and specify a column, then the system returns tables that can be joined with the input table on the specified column. In this paper, we focus on the case of that the join column contains objects in a metric space. In particular, each cell of a column is an object in the metric space [6]. We convert every column of the source tables into a set of objects. The collection of all sets is called $\mathbb{S}$ and is managed in the searching system.

The converted set of objects is used to measure the joinability between tables. We first define the matching of two objects as follows:

Definition 1 (Object Matching)**.** *Given two objects o_1 and o_2 and a distance threshold τ in a metric space, o_1 and o_2 are matched in τ, denoted as $M_r(o_1, o_2)$, when the distance between o_1 and o_2 is equal or less than r, i.e., $\{M_\tau(o_1, o_2) | d(o_1, o_2) \leq \tau\}$.*

Normally, if two sets have enough number of matched objects, we can say that these two sets are joinable, and the tables which they belong to are also joinable. We also learned this from the preliminary experiment in Section I that we should consider the number of similar cells as a measure of the joinability between tables. Therefore, the joinable table search problem can be formalized as set similarity search problems, including the threshold set similarity search and the top-k set similarity search nearest as follows:

Definition 2 (Threshold Set Similarity Search)**.** *Given a collection of sets $\mathbb{S}$, a query set Q, a distance threshold τ, and a set similarity threshold T, find the sets X whose similarity*

[6]Or the values in a cell can be transformed into an object in the metric space by embedding technologies

to Q is large or equal than T w.r.t a set similarity function f, i.e., $\{\mathbb{X} | f(Q, X, \tau) \geq T,\ X \in \mathbb{X}, \mathbb{X} \subset \mathbb{S}\}$

Definition 3 (Top-k Set Similarity Search)**.** *Given a collection of sets $\mathbb{S}$, a query set Q, a distance threshold τ, and an integer k, find top-k sets from $\mathbb{S}$ that are most similar to Q, i.e., $\{\mathbb{X} | f(Q, X, \tau) \geq f(Q, A, \tau),\ X \in \mathbb{X}, A \in \mathbb{S} - \mathbb{X}, \mathbb{X} \subset \mathbb{S}\}$*

For the set similarity function f, we define two functions for different measures. One is the **ALL** function $f_{all}(Q, X, \tau)$ that counts the number of all matched pairs of objects from sets Q and X:

$$f_{all}(Q, X, \tau) = |M_\tau(q, x)|,\ q \in Q, x \in X \qquad (1)$$

another is the **HIT** function $f_{hit}(Q, X)$ that counts the number of objects in Q which can match an object in X:

$$f_{hit}(Q, X, \tau) = |q|,\ \exists M_\tau(q, x), q \in Q, x \in X \qquad (2)$$

In this paper, we focus on the threshold set similarity search because the top-k search can be answered by using the threshold-based technique. Specifically, we can set a k-size result buffer and compute the similarity of the first k accessed sets to fill up this buffer. Then, we use the smallest similarity value in this buffer as the threshold value and execute the threshold search, during searching, the threshold value and k-size buffer are updated according to the incrementally computed results.

C. Pivot-based Filtering and iDistance index

Pivot-based filtering uses pre-computed distances to prune objects based on the triangle inequality in metric distance functions. Specifically, the distance from each object to a set of pivots P are pre-computed and stored. Given two objects q and o, for any pivot $p \in P$, the distance $d(q, o)$ cannot be smaller than $|d(q, p) - d(o, p)|$ based on the triangle inequality. Here, we adjust the pivot-based filtering to the object matching problem in Definition 1.

Lemma 1 (Pivot Filtering)**.** *Given a set P of pivots, an object set O, a query object q, and a distance threshold τ, if $d(o, p) \notin [d(q, p) - \tau, d(q, p) + \tau],\ o \in O, p \in P$, then o is not satisfied with $M_\tau(q, o)$.*

Proof. We prove by contradiction. Assume that there exists an object o satisfies $M_\tau(q, o)$, i.e., $d(q, o) \leq \tau$, but $d(o, p) \notin [d(q, p) - \tau, d(q, p) + \tau]$. According to the triangle inequality, $d(q, o) \geq |d(q, p) - d(o, p)| > \tau$, which contradicts the assumption. □

iDistance [14] is an indexing technique that takes advantage of pivot-based filtering. iDistance partitions the objects into clusters and selects a reference point for each partition. The data in each cluster are transformed into a single dimensional space based on their distance to the reference point. This allows the objects to be sorted with their distances' values and indexed using a tree structure (e.g. B$^+$-tree). Therefore, a similarity search can be performed by using one dimensional range search towards this tree structure.

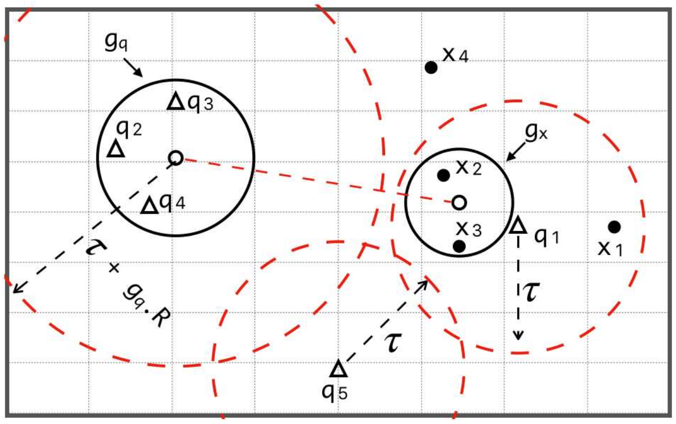


Fig. 2: An example of matching and filtering between objects and groups.

In this paper, we also utilize the concept of transforming data with their distance values to a reference point to efficiently execute a range search with a group of objects. Different from the existing index structure, we create the common reference points as the global group centers, we use a multi-vantage-point tree (MVP tree) [2] to management these global group centers with multiple vantage points (pivot points) and avoid the unnecessary computations with pivot-based filtering.

III. Group Structure

In this section, we present the definition of a group for a collection of objects from a set, and the filtering and matching methodologies to efficiently check the matched objects. We also propose a two-step indexing structure to efficient searching and pruning unnecessary computing with groups.

A. Group definition

Given a query set Q and an object set X, we need to count the pair number of matched objects then compute the set similarity $f(.)$. To avoid checking the pairwise objects from Q and X one by one, objects from a set are partitioned into groups, and it allows to count the number of matched pairs efficiently in a divide and conquer way. The group is defined as:

Definition 4 (Group)**.** *A group g has three elements: $g =< C, L, R >$. $g.C$ is the center point of this group; $g.L$ is a sorted list of objects; and $g.R$ is the radius of this group, i.e., the maximum distance value between objects in $g.L$ to the center $g.C$; According to the specific distance function, objects in L are sorted by the value of distance to $g.C$.*

For a set X, the objects are partitioned into several groups G_X, denoted as $G_X = \{g_1, g_2, ..., g_n\}$. Note that objects in each groups $g_i \in G_X$ are not overlapped, i.e., $g_i.L \cap g_j.L = \emptyset$, and $\cup g_i.L = X$. According to the density and distribution of the objects in a set, different sets may have different a number of groups, as well as different group centers. We present how to select the centers in Section IV-A.

B. Group-level Filtering

In metric space, if a query object located far from a group, we may filter all objects in this group by only checking the distance between the query and the group center. We develop an object-group filtering technique based on basic pivot filter in Lemma 1 and the range-pivot filtering in [34].

Lemma 2 (Object-group Filtering). *Given a group g, a query object q, and a distance threshold τ, if $d(q, g.C) > g.R + \tau$, then every $x \in g.L$ is not satisfied with $M_\tau(x, q)$, and g can be pruned safely.*

Proof. For any object x in group $g.L$, it holds that $d(x, g.C) \leq g.R$ based on Definition 4. If $d(q, g.C) > g.R + \tau$, then $d(q, o) \geq |d(q, g.C) - d(x, g.C)| > g.R + \tau - d(x, g.C)$ because the triangle inequality. Then $d(q, x) > g.R + \tau - g.R = \tau$. Hence, every $x \in g.L$ is not satisfied with $M_\tau(x, q)$, and g can be pruned safely. □

In addition, we can also carry out a pivot filtering with a group of query objects:

Lemma 3 (Group-group Filtering). *Given a group g_x, a query group g_q, and a distance threshold τ, if $d(g_q.C, g_x.C) > g_x.R + g_q.R + \tau$, then for all $x \in g_x.L$ and for all $q \in g_q.L$, it is not satisfied with $M_\tau(x, q)$, and g_x can be pruned safely.*

Proof. For any object $x \in g_x$ and any object $q \in g_q$, then $d(x, q) \geq |d(x, g_x.C) - d(q, g_x.C)|$ based on the triangle inequality. We can also know that $d(q, g_x.C) \geq |d(g_x.C, g_q.C) - d(q, g_q.C)|$ based on the triangle inequality, therefore, $d(x, q) \geq |d(x, g_x.C) - |d(g_x.C, g_q.C) - d(q, g_q.C)||$. If $d(g_q.C, g_x.C) > \tau + g_x.R + g_q.R$, then $d(x, q) > |d(x, g_x.C) - |\tau + g_x.R + g_q.R - d(q, g_q.C)||$. therefore, $d(x, q) > \tau$ since $d(x, g_x.C) \leq g_x.R$ and $d(q, g_q.C) \leq g_x.R$. Hence, for all $x \in g_x.L$ and for all $q \in g_q.L$, it is not satisfied with $M_\tau(x, q)$, and g_x can be pruned safely. □

On the contrary, if an query object (group) locate very close to a group in metric space, we can also infer that the query object (group) and group are matched based on the following Lemmas.

Lemma 4 (Object-group Matching). *Given a group g, a query object q, and a distance threshold τ, if $d(q, g.C) \leq \tau - g.R$, then every $x \in g.L$ is satisfied with $M_\tau(x, q)$.*

Proof. For any object x in group $g.L$, it holds that $d(x, g.C) \leq g.R$ based on Definition 4. If $d(q, g.C) \leq \tau - g.R$, then $d(q, o) \leq |d(q, g.C) + d(x, g.C)| \leq \tau - g.R - d(x, g.C)$ because the triangle inequality. Then $d(q, x) \leq g.R + \tau - g.R = \tau$. Hence, every $x \in g.L$ is satisfied with $M_\tau(x, q)$. □

Lemma 5 (Group-group Matching). *Given a group g_x, a query group g_q, and a distance threshold τ, if $d(g_q.C, g_x.C) < \tau - g_x.R - g_q.R$, then for all $x \in g_x.L$ and for all $q \in g_q.L$, it holds $M_\tau(x, q)$.*

Proof. For any object $x \in g_x$ and any object $q \in g_q$, then $d(x, q) \leq d(x, g_x.C) + d(q, g_x.C)$ based on the triangle inequality. We can also know that $d(q, g_x.C) \leq d(g_x.C, g_q.C) + d(q, g_q.C)$ based on the triangle inequality, therefore, $d(x, q) \leq d(x, g_x.C) + d(q, g_q.C) + d(g_x.C, g_q.C)$. If $d(g_q.C, g_x.C) \leq \tau - g_x.R - g_q.R$, then $d(x, q) \leq d(x, g_x.C) + d(q, g_q.C) + \tau - g_x.R - g_q.R$. Therefore, $d(x, q) \leq \tau$, because of $d(x, g_x.C) \leq g_x.R$ and $d(q, g_q.C) \leq g_x.R$. Hence, for all $x \in g_x.L$ and for all $q \in g_q.L$, it holds $M_\tau(x, q)$. □

Example 1. *In Figure 2, query object q_5 can filter a group g_x since their distance exceeds $g_x.R + \tau$. Similarly, g_p can filter g_x, and q_1 matches g_x.*

For other cases that the group of objects can not be filtered or matched by above Lemmas, we can also process an efficient pivot filtering to reduce distance computations. Specifically, for each group g_i, the distance values between object to center $g_i.C$ are pre-computed and sorted in $g_i.L$. For a query object q, we can carry out an efficient binary search with the range $[d(q, g_i.C) - R, d(q, g_i.C + R)]$ on $g_i.L$, and only compute the distance of the searched object to verify if they are matched based on Lemma 1.

C. Set-level Filtering

According to our searching problem in Definition 2, a set X is determined as joinable to a query set Q if the set similarity value $f(.)$ exceeds a given threshold T. Therefore, we can early terminate the processing for those sets that are clearly joinable or unjoinable to Q.

Clearly Joinable Case. For a query set Q and a specific object set X, the objects in query groups in G_Q are possible to match the corresponding object group in G_X which has the common center. Therefore, we can first check the common center groups from G_Q and G_X, then compute a lower bound of set similarity $f(Q, X)$ and compare to the threshold T. Here is the Lemma of this lower bound filtering.

Theorem 1 (Lower Bound Filtering). *Given a collection of groups $G_A \subseteq G_Q$ for query set Q, a collection of groups $G_B \subseteq G_X$ for set X, a distance threshold τ, and a set similarity threshold T, if $\sum_{g_a.C=g_b.C} f(g_a, g_b, \tau) \geq T, g_a \in G_A, g_b \in G_B$, then X is joinable to Q.*

Proof. Since sets are partitioned into non-overlapped groups, so $f(Q, X, \tau) = \sum_i^{|G_X|} \sum_j^{|Q_X|} f(g_i, g_j, \tau), g_i \in G_X, g_j \in G_Q$. Because of $G_A \subseteq G_Q$ and $G_B \subseteq G_X$, and the pairs of common center groups is a subset of all pairs of groups between G_A and G_B, therefore, $\sum_{g_a.C=g_b.C} f(g_a, g_b, \tau)$ is a lower bound of $f(Q, X, \tau)$. Hence, if $\sum_{g_a.C=g_b.C} f(g_a, g_b, \tau) \geq T, g_a \in G_A, g_b \in G_B$, then X is joinable to Q. □

Algorithm 1 shows an implementation for processing lower bound filtering in 1. For a query group $g_q \in G_Q$ and a common center group $g_x \in G_X$, if $g_q.R \leq \frac{1}{2}\tau \wedge g_x.R \leq \frac{1}{2}\tau$, all objects in g_q are matched with all objects in g_x based on Lemma 5. Hence, we can compute $f(g_a, g_b, \tau)$ directly and without any distance computation (Lines 3 and 4). However, we can not use Lemma 5 when the radius of the group is large than

```
Algorithm 1: LBfilter($X, Q, \tau, T$)
Input : $X$, $Q$, $\tau$, $T$, $Sim$
1  foreach $g_q \in G_Q$ do
2      Find the common center $g_x$ in Hash Table of $G_X$;
3      if $g_q$ and $g_x$ are matched with Lemma 5 then
4          update $Sim$ w.r.t matched number and $f$;
5      else
6          $A \leftarrow$ binarySearch on $g_x.L$:
             $\{x|d(x, g_x.C) \leq \frac{1}{2}\tau\}$;
7          update $Sim$ w.r.t matched number and $f$;
8      if $Sim \geq T$ then
9          Add $X$ into joinable set list;
10         Further add joinable sets w.r.t. Theorem 4;
11         break
```

```
Algorithm 2: PGfilter($X, Q, \tau, T$)
Input : $X$, $Q$, $\tau$, $T$, $Sim$
1  foreach $g_q \in G_Q$ do
2      $Stack.push$(MVP-tree.$root$);
3      while $Stack$ is not empty do
4          $Node \leftarrow Stack.pop()$;
5          if $Node$ is a non-leaf node then
6              if $Node$ is pruned w.r.t. Theorem 3 then
7                  Further prune the groups in $Node$ w.r.t.
                     Lemma 6;
8              else
9                  $Stack.push$(Node.child);
10         if $Node$ is a leaf node then
11             if $Node$ matches $g_q$ w.r.t. Lemmas 5 then
12                 Update $Sim$ w.r.t. $f$;
13                 Further match groups w.r.t. Lemma 7;
14             else if $Node$ is filtered w.r.t. Lemmas 3 then
15                 Further prune groups w.r.t. Lemma 6;
16             else
17                 Verify with Lemmas 1, 2, 4; and Update
                     $Sim$ w.r.t $f$ ;
18         if $Sim \geq T$ then
19             Add $X$ into joinable set list;
20             break
21         if $X$ is unjoinable w.r.t. Theorem 2 then
22             Further prune groups w.r.t Theorem 5 and $U$;
23             break
```

the distance threshold τ. In this case, we compute a lower bound of $f(g_q, g_x)$ by checking a part of objects which are clearly matched with query objects. Since g_q and g_x has a same group center, i.e., $g_q.C = g_x.C$, we can infer that for all $o \in g_x.L \wedge d(o, g_x.C) \leq \frac{1}{2}\tau$ and for all $q \in g_q.L \wedge d(q, g_q.C) \leq \frac{1}{2}\tau$, it holds $M_\tau(o, q)$. The objects in g_x which has less or equal distance than $\frac{1}{2}\tau$ can be retrieved by a binary range search on $g_x.L$ (Line 6).

For query objects, we only compute once of the objects in

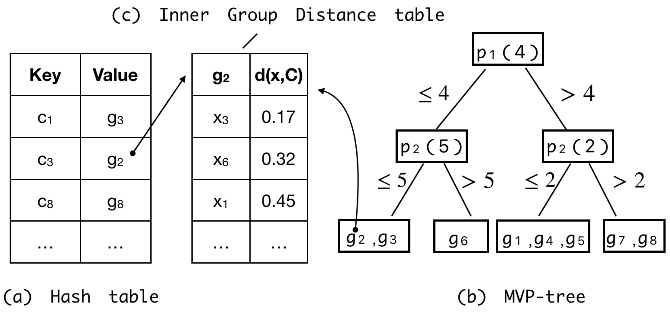


Fig. 3: The two-step index structure with: (a) Hash table; (b) MVP-tree, $m = 2$ $|P| = 2$; (c) Group.

g_q which has a less or equal distance than τ to $g_q.C$, and memorized the result in order to reuse for the future checking with Theorem 1 of all $X \in \mathbb{S}$. On the other hand, we also memorized those query objects which has a large distance than $g_q.C$, and use them in the verification cases of checking Lemma 2, Lemma 4 and Lemma 1. It worthy to note that even Q and X do not pass the Theorem 1, the result of the lower bound of the set similarity $f(Q, X)$ can be memorized as an internal result. We can only consider the unchecked objects in Q in future processing.

Clearly unjoinable case. During the checking of objects in Q and X are matched or not, we also monitoring the maximum possible set similarity between Q and X w.r.t to the current matched number. If this maximum possible set similarity is reduced to smaller than the threshold T, it means that it will not achieve T even the all unchecked pairwise objects are matched, therefore, Q and X are unjoinable, and we can filter X. We propose the following reverse filtering which is checked when the Lemmas 2 and 3 filter all objects in set X.

Theorem 2 (Reverse Filtering). *Given a query set Q, a set X, let U be the set of query objects from Q that can not match any object in X, if $Max(f(Q-U, X, \tau)) < T$, then X is not a joinable set to Q.*

Proof. We prove by contradiction. Assume that X is joinable to Q and $Max(f(Q-U), X, \tau) \leq T$, there must exist mathced objects in U, which contradicts that U is the set of query objects from Q that can not match any object in X. □

D. Two-step Index Structure: Hash and MVP-tree

Let us introduce the index structure for the groups in a set. We use a two-step index structure which contains a hash and an MVP-tree, and it works well with the above filtering lemmas and theorems. We assume all data are in-memory, we can store detailed information of the groups and objects in the group and object tables, respectively. In the two-step index, we only index the id referring to the detailed information in tables, and detailed information can be accessed by id with random access.

Hash table for quick computing lower bound. According to Theorem 1, for query groups in G_Q and object groups in

G_X, we should quick check the pairwise groups which have a common group center, i.e., $g_q.C = g_q.C, g_q \in G_Q, g_x \in G_X$. Hash table is a good choice that maps keys to values, and it can quickly access a value for a specific key with an $O(1)$ search complexity on average. Because the groups and its' centers are unique in a set, so we can set a group center as a hash key and the group id as a value.

The hash table search are used in Algorithm 1 Line 2. Benefiting from using the hash table, the compute complexity of processing Theorem 1 can be $O(|G_Q| \cdot log\overline{|L_X|})$, where $|G_Q|$ is the number of groups in query set and $\overline{|L_X|}$ is the average number of objects in a single group of set X. Figure 3 shows a hash structure.

MVP-tree for pruning groups. For the case that X and Q do not pass the lower bound filter of Theorem 1, we should check the pairwise groups and objects in a double loop between the groups in G_X and G_Q. To avoid checking pairwise group one by one, we use an MVP-tree [2] for further pruning the similar groups in a branch-and-bound way.

MVP-tree (Multi-Vantage-Point tree) is a balanced tree and each non-leaf node has m child nodes. It chooses a pivot as the root node, and selects $i-1$ medium values: ν_1, ν_2, ..., ν_{i-1}. The entries e with $d(e,p) \leq \nu_1$ are put into the first child node, the entries e with $\nu_1 < d(e,p) < \nu_2$ are put into the second child node, etc. The child node will be further partitioned when the number of entries exceeds a threshold. We can use many pivot selection methods [5], [6], [16] to select pivots, in this paper, we use the HF algorithm [16] to select outliers as pivots. Figure 3 shows an example of MVP-tree where $|P| = 2$, $m = 2$.

For each group in a set, we treat the group center as an entry and index into the MVP-tree. To correctly pruning with the MVP-tree, we take advantage of Lemmas 1 and 3 and propose a pivot-based group filtering in the following theorem.

Theorem 3 (Pivot Group Filtering). *Given a set P of pivots, a query group g_q, a group g_x, and a distance threshold τ, if $d(g_x.C,p) \notin [d(g_q.C,p) - (\tau + g_q.R + g_x.R), d(g_q.C,p) + (\tau + g_q.R + g_x.R)], p \in P$, then for all $x \in g_x.L$ and for all $q \in g_q.L$, it is not satisfied with $M_\tau(q,x)$.*

Proof. We prove by contradiction. Assume that there exists an object $x \in g_x.L$ satisfies $M_\tau(q,x)$ where $q \in g_q.L$. but $d(g_x.C,p) \notin [d(g_q.C,p) - (\tau + g_q.R + g_x.R), d(g_q.C,p) + (\tau + g_q.R + g_x.R)], p \in P$. According to the triangle inequality, $d(g_x.C, g_q.C) > \tau + g_q.R + g_x.R$, and according to Lemma 3, for all objects in $g_x.L$ and all object in $g_q.L$, they do not hold $M_\tau(q,x)$. which contradicts the assumption. □

Algorithm 2 shows the process of pivot group filtering in Theorem 3. To search the matched objects for a group of query objects, for each group of query objects, the nodes in the MVP-tree are traversed in the depth-first fashion. Specifically, when accessing a non-leaf node, we prune the child nodes with Theorem 3 (Lines 5 and 6). When accessing a leaf node (Line 10), we verify the groups in the candidate list with Lemma 5 (Line 11), Lemma 3 (Line 14), and Lemmas 1, 2, 4 (Line 17). If the current similarity exceeds T, X is a result of joinable set (Lines 18 and 19). On the other hand, we stop traversing if X is determined as unjoinable by the reverse filter in Theorem 2 (Lines 21 and 23).

Because the groups are accessed in a branch-and-bound way, the complexity of search with MVP-tree is $O(|G_Q| \cdot log|G_X| \cdot log\overline{|L_X|})$.

IV. Dominating relation

Section III introduces the group structure to efficient check two sets are joinable or not. Recall that our problem is to retrieve the joinable sets from a large collection of sets $\mathbb{S}$, so it is inefficient to check the candidate set one by one. Therefore, in this section, we introduce the method to reduce the computations upon all candidate sets. We present the group center selection, the definitions of dominating relations between the groups from different sets, and the corresponding filtering methodologies. We also propose an inverted index based structure to index the dominating relations, as well as the efficient filtering process.

A. Group Center Selection

To support the group structure in Section III, for each set of objects, we need to split the objects into groups with respect to some group centers. We use a set of global centers for all sets in $\mathbb{S}$ because of we want to make the groups from different sets have commonality. We can make use of this commonality to summarize the dominating relationship between the groups, and make further filtering on other candidate sets.

There are two ways to select centers, the space-based partitioning, and the data-based partitioning [14]. We use space-based partitioning since we need to generate centers independent of the objects in all sets. In this paper, we use a simple way that generates a set of candidate global centers with the uniform distribution on the space. We select a subset from this candidate global centers for each set. A group with a center is associated with the objects that are nearest to it. Moreover, we can also partition the space into hyperplanes [7], and select the centers of hyperplanes as the group center.

B. Group Dominating Definition and Filtering

We define the following dominating relations for those two groups which have a common group center.

Definition 5 (Group Dominating). *Given two groups g_1 and g_2 which have the same center, i.e., $g_1.C = g_2.C$, if $g_1.R \leq g_2.R$, then g_1 dominates g_2, denoted as $Dom(g_1, g_2)$.*

Definition 6 (Group Threshold Dominating). *Given two groups g_1 and g_2 which have the same center, i.e., $g_1.C = g_2.C$, and a distance threshold τ, let $g_1.A$ be the set of objects whose distance to $g_1.C$ is less or equal than $\frac{1}{2}\tau$, i.e., $\{g_1.A | o \in g_1.A \wedge d(o, g_o.C) \leq \frac{1}{2}\tau, g_1.A \subseteq g_1.L\}$. if $|g_1.A| \geq |g_2.A|$, then g_1 dominates g_2 with threshold τ, denoted as $Dom_\tau(g_1, g_2)$.*

[7]The space is partitioned into pyramids in [14].

Assume that a group g from a set are filtered by Lemma 3 w.r.t a query group, we can further filter the groups in other sets that dominate g with Definition 5.

Lemma 6 (Group Dominating Filtering). *Given a query group g_q, a group g_1 and a group g_2 which dominated by g_1 with radius, i.e., $Dom(g_1, g_2)$, if g_2 can be filtered to g_q by Lemma 3, then g_1 also can be filtered to g_q by Lemma 3.*

Proof. If g_2 can be filtered to q_q by Lemma 3, then $d(g_2.C, g_q.C) > g_2.R + g_q.R + \tau$. Since $Dom(g_1, g_2)$, we can know that $g_1.C = g_2.C$ and $g_1.R \leq g_2.R$, therefore, $d(g_1.C, g_q.C) > g_1.R + g_q.R + \tau$. Hence, g_1 also can be filtered to g_q by Lemma 3. □

Similarly, we can also further match the groups in other sets that dominate g with Definition 5.

Lemma 7 (Group Dominating Matching). *Given a query group g_q, a group g_1 and a group g_2, which dominated by g_1 with radius, i.e., $Dom(g_1, g_2)$, if g_2 can be matched to g_q by Lemma 5, then g_1 also can be matched to g_q by Lemma 5.*

Proof. If g_2 can be matched to q_q by Lemma 5, then $d(g_2.C, g_q.C) \leq g_2.R + g_q.R + \tau$. Since $Dom(g_1, g_2)$, we can know that $g_1.C = g_2.C$ and $g_1.R \leq g_2.R$, therefore, $d(g_1.C, g_q.C) \leq g_1.R + g_q.R + \tau$. Hence, g_1 also can be matched to g_q by Lemma 5. □

Theorems 1 and 2 are two set-level filtering methodologies to filter a set directly. If a set X are filtered by Theorem 1, we can further filter the other sets that dominate X with τ based on Definition 6.

Theorem 4 (Set Lower Bound Dominating Filtering). *Given two set X_1 and X_2, a query set Q, and a distance threshold τ, let G_A be a collection of groups in G_{X_1}. If X_1 and Q is determined as joinable by Theorem 1 w.r.t G_A, and all groups in G_A are dominated by a corresponding groups $G_B \in G_{X_2}$ with τ, i.e., $\{Dom_\tau(g_a, g_b) | g_a \in G_A, g_b \in G_B, G_A \subseteq G_{X_1}, G_B \subseteq G_{X_2}\}$, then X_2 and Q can also be determined as joinable by Theorem 1 w.r.t G_B.*

Proof. If X_1 and Q is determined as joinable by Theorem 1 w.r.t G_A, we can know that $\sum_{g_a.C = g'_a.C} f(g_a, g'_a, \tau) \geq T, g_a \in G_A, g'_a \in G'_A$, where G'_A is the corresponding common center group in Q. Since all groups in G_A are dominated by a corresponding groups $G_B \in G_{X_2}$ with τ, therefore $\sum_{g'_a.C = g_b.C} f(g'_a, g_b, \tau) \geq T, g'_a \in G'_A, g_b \in G_B$. Hence, X_2 and Q can also be determined as joinable by Theorem 1 w.r.t G_B. □

On the other hand, if a set X is filtered by Theorem 2, we can further filter the other sets that dominate X based on Definition 5.

Theorem 5 (Set Reverse Dominating Filtering). *Given two set X_1 and X_2, a query set Q, and a distance threshold τ, let U be a collection of objects in Q that can not match any object in X_1. If X_1 and Q is determined as unjoinable by Theorem 2 w.r.t U, and each group in G_{X_1} dominates a corresponding groups G_{X_2}, i.e., $\{Dom(g_a, g_b) | \forall g_a \in G_{X_1} \wedge \exists g_b \in G_{X_2}\}$, then X_2 and Q can also be determined as unjoinable by Theorem 2 w.r.t U.*

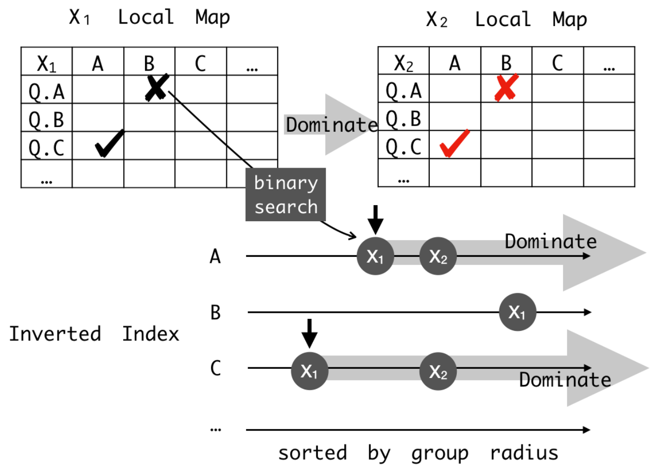


Fig. 4: The inverted index and dominating filtering.

Proof. If X_1 and Q is determined as unjoinable by Theorem 2 w.r.t U, we can know that for all objects $u \in U$ and all $g_{x_1} \in G_X$, $d(u, g_{x_1}) > g_{x_1}.R + \tau$. Since each group in G_{X_1} dominates a corresponding groups G_{X_2}, therefore, for each group $g_{x_2} \in G_{X_2}$, it holds $d(u, g_{x_2}) > g_{x_2}.R + \tau$, where $u \in U$. Hence, X_2 and Q can also be determined as unjoinable by Theorem 2 w.r.t U. □

C. Inverted Index for Dominating Filtering

Let us introduce the index structure for the dominating relations of the groups from all sets. To quickly find the dominated groups by a specific group, we use an inverted index to assemble the groups from all sets. An inverted index contains a collection of posting lists. Each posting lists represent a unique element (a group in this paper) from the elements universe of all sets. A posting list contains a list of pointers to the sets that contain this group. We sort each posting list by groups' radii w.r.t. the contained sets, and this works well with the above filtering lemmas and theorems.

Lemmas 6 and 7 can filter the dominated groups in Definition 5 which have less radii. Therefore, we can execute a binary search on the posting list of a specific group g_a, and retrieve the groups which have fewer radii than $g_a.R$, then mark them as filtered or matched. Algorithm 2 further prunes groups in other sets w.r.t 6 in Lines 7 and 15, and further matches groups in other sets w.r.t 7 in Line 13.

During the marking, if all groups of a set are marked as filtered, this set is an unjoinable set and can be skipped in the future process based on Theorem 5. Algorithm 2 execute a further set pruning in Line 22.

Example 2. *Figure 4 shows examples of processing above dominating filtering techniques. In this figure, we use "A", "B", "C" etc. to denote the groups, and X.A means the A* *group in set X. Assume that when checking $Q.A$ group with a set X_1, and we filter the group $X_1.B$, a local map keeps the relations of filter or not. Then we execute a binary search with the radius of $X_1.A$ and found the position of X_1 with the posting list of A. Just like the $X_2.B$ also marked in a red cross, according to Lemma 6, we can mark the group B of the local maps from all sets which after X_1 in the posting list. Similarly, if $Q.C$ matches $X_1.A$, we can also mark $X_2.A$ as matched. When all groups of this row in a local map are marked as filtered, then we can check the $|Q.A|$ so that filter a whole set with Theorem 5.*

Let see the case that a set is determined as joinable by Theorem 1 w.r.t a collection of groups G_A. To achieve the dominating filter in Theorem 4, for each group $g_a \in G_A$, we need to check the posting list of g_a and find the groups which are dominated by g_a with τ in Definition 6. We can not know the groups dominated by g_a in advance since τ is an input value. Therefore, we execute a lossy but quick check to find the groups which are clearly dominated by g_a with τ. Specifically, for each $g_a \in G_A$, we execute a binary search on the posting list of g_a to retrieve the groups which have fewer radii than $\frac{1}{2}\tau$, then discard the retrieved groups which have fewer objects than $|g_a.L|$. For each $g_a \in G_A$, we got a collection of sets corresponding to the dominated groups, then we take the intersection of all collection, and the result is the sets can be filtered based on Theorem 4. Algorithm 1 executes this in Line 10.

D. Next Set Selection

According to the dominating relation between groups and sets, the processing order of sets has a huge impact on the searching performance. For example, in Figure 4, we can see that processing X_1 first can filter more groups than processing X_2 first.

Therefore, how to select the next set to process is an important problem. In the processing of our searching problem, we assume that the checked processed sets are a sub-collection of the source sets $\mathbb{S}$, and the other sets are skipped by our dominating filtering techniques. We define the following dominating cover problem to solve the next set selection problem.

Problem 1 (Dominating Cover Problem)**.** *Let $g.Dom$ be the collection of dominating groups by group g. Given a universe collection of groups $\mathbb{S}_G$, a collection of sets $\mathbb{S}$ such that the union of the groups contained in $\mathbb{S}$ is equal to $\mathbb{S}_G$. Find a smallest sub-collection M from $\mathbb{S}$ such that $\cup_{g.Dom} = \mathbb{S}_G, g \in M$.*

It is easy to see that the dominating cover problem is exactly a set cover problem [8]. Hence it is NP-hard and can be solved by a greedy algorithm with an approximation ratio of $O(ln|\mathbb{S}_G|)$ [32]. One greedy algorithm for polynomial time approximation is that: for each time, choose the set that dominated the largest number of unchecked groups. However, due to the procedure of finding the overall largest one requires to check all unchecked sets, we propose a local greedy algorithm for an approximation that selects the set from the first k elements of the posting lists in the inverted index. This is because the order of the inverted index reacts the dominating power, and we can easily update the unchecked length of each posting lists. Algorithm 3 shows the proposed greedy algorithm.

[8] https://en.wikipedia.org/wiki/Set_cover_problem

Algorithm 3: SelectNextSet(k)

```
Input  : $k$
Output: next set to check
1 foreach Posting list in Dominating Inverted Index do
2     $cand \leftarrow$ the first $k$ set ids;
3 $X_{next} \leftarrow$ the most frequent set id in $cand$;
4 return $X_{next}$
```

Algorithm 4: PEXESO($Q, \mathbb{S}, \tau, T$)

```
Input  : $Q$, $\mathbb{S}$, $\tau$, $T$, $k$
Output: joinable set list
1  while exist unchecked set in $\mathbb{S}$ do
2      $Sim \leftarrow 0$;
3      $X \leftarrow$ SelectNextSet($k$);
       // Algorithm 3
4      LBfilter($X, Q, \tau, T, Sim$) ;
       // Algorithm 1
5      if $X$ is joinable by Theorem 1 then
6          Further filter sets with Theorem 4;
7      else
8          PGfilter($X, Q, \tau, T, Sim$) ;
           // Algorithm 2
9          if $X$ is unjoinable by Theorem 2 then
10             Further filter sets with Theorem 5;
11 return joinable set list
```

Finally, the PEXESO algorithm is shown in Algorithm 4 which assemble the filtering algorithms and selection algorithm to process a joinable set search.

V. Experiments

A. Setting

Datasets. We have three real datasets and one synthetic dataset. Table IV summarizes the statistics of datasets.

- **LOGO**: a dataset transformed from company images logos in [20]. LOGO contains 194 unique logo classes and over 2M logo images. We extract the nine-dimensional HSV features from the images. We divide the total 2M vectors as 2000 sets, and each set contains 800 to 1200 objects.
- **TAXI**: a dataset of spatial coordinates. TAXI [23] contains 55M rows of yellow taxi trip in New York City. For each row, we extract the latitude and longitude of the pick-up location, and form a two-dimensional vector. We divide the total 55M vectors as 10000 sets, and each set contains 5000 to 6000 objects.

Dataset	Size	Dim	Set #	Avg Obj #	distance function
LOGO	2M	9	2K	1K	L2
TAXI	55M	2	10K	5.5K	L2
W2V	400K	50	0.4K	1K	L2, Angular
SYN	100M	20	10K	10K	L2, Angular

TABLE IV: Datasets statistics

- **W2V**: a dataset transformed from words in Wikipedia 2004. W2V is transformed by the GloVe word2Vec technique [11]. It contains 50 dimensional pre-trained 400K vectors. We divide it into 400 sets, and each sets contains 800 to 1200 objects.
- **SYN**: a synthetic dataset generated by us. The data set is a clustered dataset, we first generate cluster center with uniform distribution in the space, then use this center as a means to generate 20-dimensional objects with a normal distribution. SYN contains totally 100M objects, up to 10K sets, and each set contains up to 100K objects.

Algorithms. According to the reason we mention in Section I, we implement the baseline with the state-of-the-art index based methods. We consider the following methods: (1) RTREE, index each set with R-tree [3], [17], and all sets are processed one by one; (2) EPT[9] , index each set with EPT pivot table [26], and all sets are processed one by one. (3) PNODOM, our proposed method without dominating filter and next set selection, and all sets are processed one by one. (4) PEXESO, our proposed method in Algorithm 4.

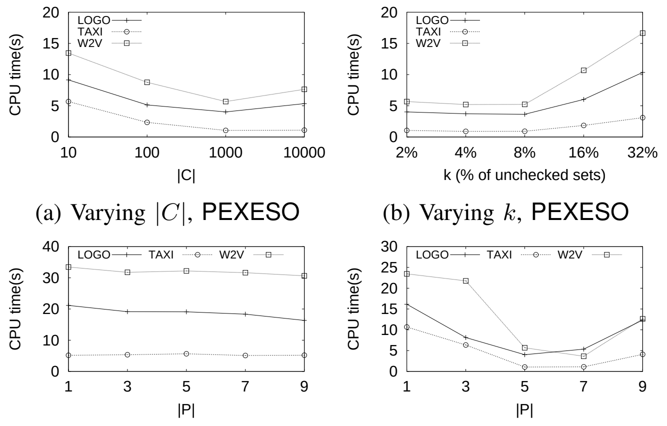
Environments and measures. The experiments were run on a Server with an Intel Xeon CPU E7-8890 2.20GHz and 630GB RAM. All the algorithms were implemented in C++ and in the main memory fashion. We randomly selected 100 sets as queries from data sets and reported the average processing time and distance computation times per each query. We default to use the f_{hit} function. The default τ is 6% of the maximum distance in the space, and the default T is 60% of the maximum set similarity value $f(.)$. we only test the RTREE method in TAXI and LOGO data, since it is well known that R-tree performs extremely bad in high-dimensional data with the curse of dimensionality.

B. Parameter Tuning

There are three parameters in our method, reading number k for greedy selection, the center number $|C|$ for generating candidate global centers for groups, and pivot number $|P|$ for the MVP-tree.

Varying $|C|$. Figure 5a shows the processing time with varying $|C|$, note the $|C|$ is the candidate number to select rather than a actual groups number in each sets. In this experiment, about $30 \sim 200$ centers are selected as group center with the increasing $|C|$. The processing time tend to stable but become a little slow with a large $|C|$ (10K), this is because more groups

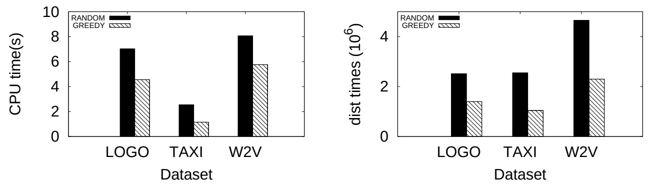
[9]The reason we select EPT as a baseline method is because [6] summarizes EPT and MVP-tree methods are performed better than other existing methods with in-memory processing, and our PNODOM and PEXESO is based on the MVP-tree.



(a) Varying $|C|$, PEXESO (b) Varying k, PEXESO

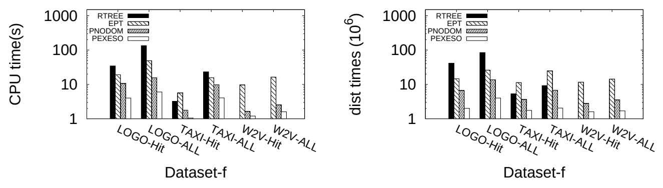
(c) Varying $|P|$, EPT (d) Varying $|P|$, PEXESO

Fig. 5: Parameter tuning.



(a) Average processing time. (b) Average distance computation.

Fig. 6: Compare a random algorithm with the proposed greedy algorithm on the next set selection.



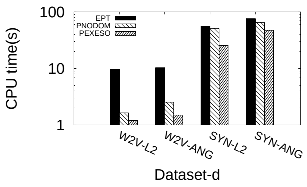
(a) Average processing time. (b) Average distance computation.

Fig. 7: Compare set similarity function f_{hit} and f_{all} on all real datasets.

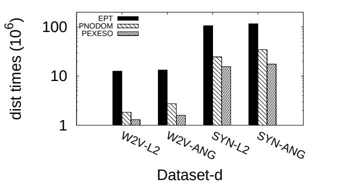
make each group tighter to filter more objects, but it also leads to more overhead of the pairwise group checking. We choose $|C| = 1000$.

Varying k. Figure 5b shows the processing time with varying k, as our expected, large k leads more overhead on the computing the most dominating set in the set selection. Therefore, we choose $k = 8\%$.

Varying $|P|$. Figures 5d and 5d show the processing time with varying $|P|$. EPT are affected slightly with different pivot number, while the processing time of PEXESO first drop and then increase with the increasing $|P|$, this is because more pivots make MVP-tree tightly partition the objects, but also makes MVP-tree has more levels and more traversal overhead. We choose $|P| = 7$ for both EPT and PEXESO.

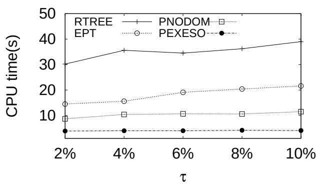


(a) Average processing time.

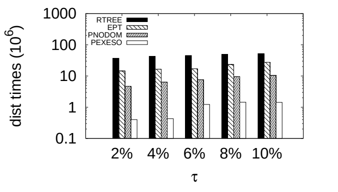


(b) Average distance computation.

Fig. 8: Compare distance function d, L2 and Angular distance on all real datasets.

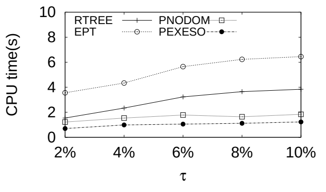


(a) Average processing time.

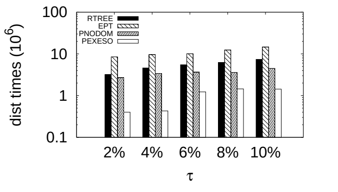


(b) Average distance computation.

Fig. 9: Varying τ on LOGO data.

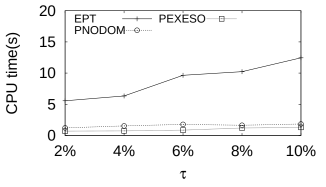


(a) Average processing time.

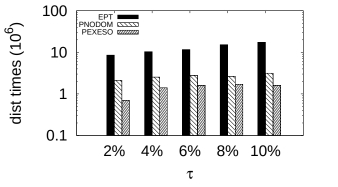


(b) Average distance computation.

Fig. 10: Varying τ on TAXI data.



(a) Average processing time.



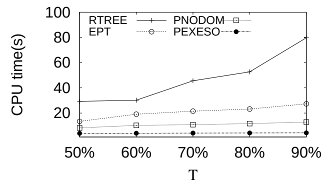
(b) Average distance computation.

Fig. 11: Varying τ on W2V data.

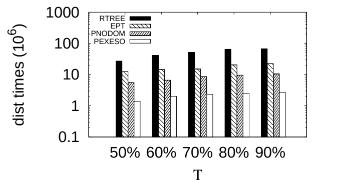
C. Searching Performance

Next set selection. We first test the effectiveness of the proposed greedy algorithm to a random one for selecting the next set. Figure 6 shows the experimental results. The proposed greedy algorithm shorten 1.2 to 2.1 times of processing time than a random order processing one in all real datasets.

Set similarity function: f_{hit} and f_{all}. We compare the processing time of f_{hit} and f_{all} with all real datasets. Figure 7a shows the processing time and Figure 7b shows the distance computation times. We can see that f_{all} always cost more time than f_{hit} since it needs check all pairs between objects in query sets and target objects. While f_{hit} is related to the $|Q|$ and each object in Q is only checked once. PEXESO is the best method in both set similarity functions.

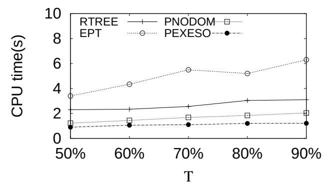


(a) Average processing time.

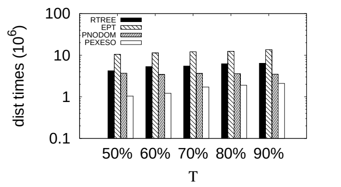


(b) Average distance computation.

Fig. 12: Varying T on LOGO data.

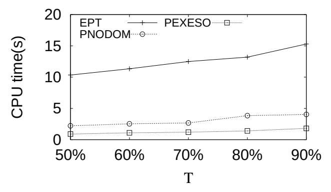


(a) Average processing time.

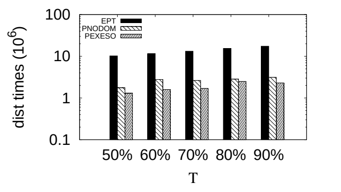


(b) Average distance computation.

Fig. 13: Varying T on TAXI data.

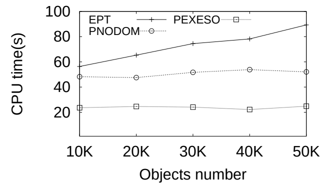


(a) Average processing time.

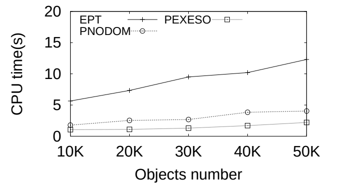


(b) Average distance computation.

Fig. 14: Varying T on W2V data.



(a) SYN data.

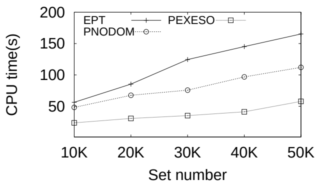


(b) TAXI data.

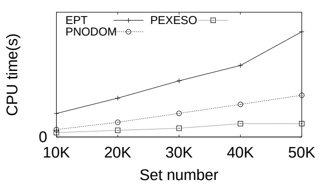
Fig. 15: Varying $\overline{|X|}$.

L2 and Angular distance. Figure 8 shows comparison results of L2 and Angular distance on the W2V data and SYN data. These experimental results show that the performance is stable and not affect by distance functions. this is because all pivot-based methods (EPT, PNODOM, PEXESO) which are independent with the distance functions. We can also see that PEXESO is the best method in both distance functions.

Varying τ. Figures 9, 10 and 11 show the comparison results on real datasets with different distance search range (threshold) τ. The processing time and distance computations of RTREE and EPT increase linearly with the large search range τ. The proposed PNODOM and PEXESO increased slightly since they have both filtering technique and matching technique so that it can perform well in any range of τ. The PEXESO and PNODOM are the best and the second best methods in all real datasets, and PEXESO is 2 to 4.6 times faster than



(a) SYN data.



(b) TAXI data.

Fig. 16: Varying $|\mathbb{S}|$.

the other baselines. We found that RTREE is better than EPT in the taxi data, RTREE are powerful to split the data with less overlap rectangle range (also called MBR) in the two dimensional spatial space. However, RTREE faces the curse of the dimensionality that most nodes are overlapped with each other. Compare to this, the pivot-based methods EPT, PNODOM and PEXESO transform the high-dimensional values to the one-dimensional distance value w.r.t pivot points, and avoid the curse of dimensionality.

Varying T. Figures 12, 13 and 14 show the comparison results on real datasets with different set similarity threshold T. Same to the above results, PEXESO is the best method in all cases on both processing time and distance computation times. PEXESO is 2.4 to 5.8 times faster than other baselines. According to the results, the processing time of both RTREE and EPT are linear increasing with the large threshold T, this is because large T requires to check more pairwise objects during the searching. However, benefiting from the reverse filter technique in Theorem 2, the processing time of both PNODOM and PEXESO are stable or increase slightly.

D. Scalability

We study the scalability with the object number and the set number with two large scale dataset TAXI and SYN.

Varying object number. According to the experimental results in Figure 15, we can see that EPT are affected with the varying number of objects while PNODOM and PEXESO are stable. This is because EPT is a method with the pivot table, the large objects leads to more comparison between query objects and target objects. On the other hand, PNODOM and PEXESO use MVP-tree to index groups, and the processing time of MVP-tree has a logarithmic growth with respect to the object number. PEXESO is still the best one, about 4.5 to 5.7 times faster than baselines.

Varying set number. Figure 16 shows the experimental results with different set number. We can see that PEXESO is still the best method, about 3.2 to 5.4 times faster than other baselines. and its processing time and distance computation times increase slightly with the increasing set number. This is because the dominating filtering techniques can reduce the computation of similar candidate sets. This experiment shows that the proposed dominating filtering techniques are effective. PEXESO is able to deal with the large scale tables in the data lake.

VI. Related Work

We review the most related work as follows.

A. Search tables from the data lake and set similarity join

For the topic of searching tables from the data lake, Zhu *et al.*. proposed an exact searching solution [35], which is a data-dependent solution that uses a cost model to adapt to the data distribution. Zhu *et al.* also proposed an approximate solution [37] that use a min-hash LSH to efficiently search web tables. Nargesian et al. [22] proposed min-hash and sim-hash based techniques to search for unionable tables from data lakes. Deng *et al.* [7] proposed a related set (table) searching system that finds sets with the maximum bipartite matching metrics.

If we treat a set as an identity of a table, then the set similarity join problem (or called all pair set similarity search) can also apply to the problem of finding tables. Xiao *et al.* proposed the pp-join [1], [31]. Deng *et al.* proposed a partition based method [8]. Wang *et al.* designed a similarity which combines token and characters and proposed a solution for fuzzy join [28], [29]. Wang *et al.* proposed MF-join [30] which performs a fuzzy join with multi-level filtering. We also refer readers to [21] for various problem setting and methods. Zhu *et al.* study the Auto-join [36] to join two tables with leveraging transformation on string columns. He *et al.* propose SEMA-join [12] using a data-driven method that leverages a big table corpus to determine statistical correlation between cell values at row-level and column-level, then find joinable pairs between two tables.

The above works only consider the textual columns with the string-based similarity functions, while We consider the numerical columns with distance-based metric similarity functions, and their methods can not solve our problem.

B. Metric similarity search and join

There also have research on similarity search on join with numerical vectors, including compact data (space) partition methods [3], [17], and pivot-based methods [4], [9], [10], [13], [25], [33]. Yu *et al.* [33] applied iDistance technique to the kNN join problem. Fredriksson *et al.* [10] improved quick join algorithm [13] for similarity joins. We also refer readers to [5], [6] for various pivot-based indices and methods.

However, since the above methods are design for the problem of similarity join rather than our joinable searching problem. They can not deal with our joinable searching problem for the reasons: (1) Only working with specific cases and requiring heavy index rebuilding for other cases [9], [24]; (2) Pre-build sophisticate index for both sets [3], [5], [17], [25]; and (3) One time join without indexing [10], [13].

VII. Conclusion

In this paper, we investigate a novel problem of searching joinable tables with numerical columns. We propose a framework PEXESO which can reduce expensive distance computations by group-level pivot-based filtering, set-level lower bound filtering and reverse filtering. PEXESO also takes advantage of dominating relations between groups and further reducing computations. We conducted extensive experiments on real and synthetic datasets. Experimental results showed that the PEXESO outperforms baseline methods by 2 to 5.8 times speed-up.

For future work, we'd like to build an end-to-end joinable searching system based on PEXESO.

References

[1] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *WWW*, pages 131–140, 2007.

[2] T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, SIGMOD '97, pages 357–368, New York, NY, USA, 1997. ACM.

[3] T. Brinkhoff, H. Kriegel, and B. Seeger. Efficient processing of spatial joins using r-trees. In *SIGMOD*, pages 237–246, 1993.

[4] L. Chen, Y. Gao, X. Li, C. S. Jensen, and G. Chen. Efficient metric indexing for similarity search. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, pages 591–602, 2015.

[5] L. Chen, Y. Gao, X. Li, C. S. Jensen, and G. Chen. Efficient metric indexing for similarity search and similarity joins. *IEEE Trans. Knowl. Data Eng.*, 29(3):556–571, 2017.

[6] L. Chen, Y. Gao, B. Zheng, C. S. Jensen, H. Yang, and K. Yang. Pivot-based metric indexing. *PVLDB*, 10(10):1058–1069, 2017.

[7] D. Deng, A. Kim, S. Madden, and M. Stonebraker. Silkmoth: An efficient method for finding related sets with maximum matching constraints. *PVLDB*, 10(10):1082–1093, 2017.

[8] D. Deng, G. Li, H. Wen, and J. Feng. An efficient partition based method for exact set similarity joins. *PVLDB*, 9(4):360–371, 2015.

[9] V. Dohnal, C. Gennaro, and P. Zezula. Similarity join in metric spaces using ed-index. In *DEXA 2003*, pages 484–493, 2003.

[10] K. Fredriksson and B. Braithwaite. Quicker similarity joins in metric spaces. In *SISAP*, pages 127–140, 2013.

[11] GloVe. Glove: Global vectors for word representation. https://nlp.stanford.edu/projects/glove/, 2019.

[12] Y. He, K. Ganjam, and X. Chu. SEMA-JOIN: joining semantically-related tables using big table corpora. *PVLDB*, 8(12):1358–1369, 2015.

[13] E. H. Jacox and H. Samet. Metric space similarity joins. *ACM Trans. Database Syst.*, 33(2):7:1–7:38, 2008.

[14] H. V. Jagadish, B. C. Ooi, K. Tan, C. Yu, and R. Zhang. idistance: An adaptive b^+-tree based indexing method for nearest neighbor search. *ACM Trans. Database Syst.*, 30(2):364–397, 2005.

[15] V. Jahns. Principles of data integration by anhai doan, alon halevy, zachary ives. *ACM SIGSOFT Software Engineering Notes*, 37(5):43, 2012.

[16] C. T. Jr., R. F. S. Filho, A. J. M. Traina, M. R. Vieira, and C. Faloutsos. The omni-family of all-purpose access methods: a simple and effective way to make similarity search more efficient. *VLDB J.*, 16(4):483–505, 2007.

[17] Y. J. Kim and J. M. Patel. Performance comparison of the r^*-tree and the quadtree for knn and distance join queries. *IEEE Trans. Knowl. Data Eng.*, 22(7):1014–1027, 2010.

[18] A. Kumar, J. F. Naughton, and J. M. Patel. Learning generalized linear models over normalized data. In *SIGMOD*, pages 1969–1984, 2015.

[19] A. Kumar, J. F. Naughton, J. M. Patel, and X. Zhu. To join or not to join?: Thinking twice about joins before feature selection. In *SIGMOD*, pages 19–34, 2016.

[20] logdata. Weblogo-2m dataset. http://www.eecs.qmul.ac.uk/~hs308/WebLogo-2M.html/, 2018.

[21] W. Mann, N. Augsten, and P. Bouros. An empirical evaluation of set similarity join techniques. *PVLDB*, 9(9):636–647, 2016.

[22] F. Nargesian, E. Zhu, K. Q. Pu, and R. J. Miller. Table union search on open data. *PVLDB*, 11(7):813–825, 2018.

[23] nyctaxi. Nyc taxi trip data. https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page, 2019.

[24] R. Paredes and N. Reyes. Solving similarity joins and range queries in metric spaces with the list of twin clusters. *J. Discrete Algorithms*, 7(1):18–35, 2009.

[25] S. S. Pearson and Y. N. Silva. Index-based R-S similarity joins. In *SISAP*, pages 106–112, 2014.

[26] G. Ruiz, F. Santoyo, E. Chávez, K. Figueroa, and E. S. Tellez. Extreme pivots for faster metric indexes. In *SISAP*, pages 115–126, 2013.

[27] V. Shah, A. Kumar, and X. Zhu. Are key-foreign key joins safe to avoid when learning high-capacity classifiers? *PVLDB*, 11(3):366–379, 2017.

[28] J. Wang, G. Li, and J. Feng. Fast-join: An efficient method for fuzzy token matching based string similarity join. In *ICDE*, pages 458–469, 2011.

[29] J. Wang, G. Li, and J. Feng. Extending string similarity join to tolerant fuzzy token matching. *ACM Trans. Database Syst.*, 39(1):7:1–7:45, 2014.

[30] J. Wang, C. Lin, and C. Zaniolo. Mf-join: Efficient fuzzy string similarity join with multi-level filtering. In *ICDE*, pages 386–397, 2019.

[31] C. Xiao, W. Wang, X. Lin, J. X. Yu, and G. Wang. Efficient similarity joins for near-duplicate detection. *ACM Trans. Database Syst.*, 36(3):15:1–15:41, 2011.

[32] N. E. Young. Greedy set-cover algorithms. In *Encyclopedia of Algorithms*. 2008.

[33] C. Yu, B. Cui, S. Wang, and J. Su. Efficient index-based KNN join processing for high-dimensional data. *Information & Software Technology*, 49(4):332–344, 2007.

[34] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search - The Metric Space Approach*, volume 32 of *Advances in Database Systems*. Kluwer, 2006.

[35] E. Zhu, D. Deng, F. Nargesian, and R. J. Miller. JOSIE: overlap set similarity search for finding joinable tables in data lakes. In *SIGMOD*, pages 847–864, 2019.

[36] E. Zhu, Y. He, and S. Chaudhuri. Auto-join: Joining tables by leveraging transformations. *PVLDB*, 10(10):1034–1045, 2017.

[37] E. Zhu, F. Nargesian, K. Q. Pu, and R. J. Miller. LSH ensemble: Internet-scale domain search. *PVLDB*, 9(12):1185–1196, 2016.