DESIRE: An Efficient Dynamic Cluster-based Forest Indexing for Similarity Search in Multi-Metric Spaces

Yifan Zhu
Zhejiang University
xtf_z@zju.edu.cn

Lu Chen
Zhejiang University
luchen@zju.edu.cn

Yunjun Gao
Zhejiang University
gaoyj@zju.edu.cn

Baihua Zheng
Singapore Management University
bhzheng@smu.edu.sg

Pengfei Wang
Zhejiang University
wangpf@zju.edu.cn

ABSTRACT

Similarity search finds similar objects for a given query object based on a certain similarity metric. Similarity search in metric spaces has attracted increasing attention, as the metric space can accommodate any type of data and support flexible distance metrics. However, a metric space only models a single data type with a specific similarity metric. In contrast, a multi-metric space combines multiple metric spaces to simultaneously model a variety of data types and a collection of associated similarity metrics. Thus, a multi-metric space is capable of performing similarity search over any combination of metric spaces. Many studies focus on indexing a single metric space, while only a few aims at indexing multi-metric space to accelerate similarity search. In this paper, we propose DESIRE, an efficient dynamic cluster-based forest index for similarity search in multi-metric spaces. DESIRE first selects high-quality centers to cluster objects into compact regions, and then employs B$^+$-trees to effectively index distances between centers and corresponding objects. To support dynamic scenarios, efficient update strategies are developed. Further, we provide filtering techniques to accelerate similarity queries in multi-metric spaces. Extensive experiments on four real datasets demonstrate the superior efficiency and scalability of our proposed DESIRE compared with the state-of-the-art multi-metric space indexes.

PVLDB Reference Format:
Yifan Zhu, Lu Chen, Yunjun Gao, Baihua Zheng, Pengfei Wang. DESIRE: An Efficient Dynamic Cluster-based Forest Indexing for Similarity Search in Multi-Metric Spaces. PVLDB, 15(10): 2121 - 2133, 2022.
doi:10.14778/3547305.3547317

1 INTRODUCTION

Given a query object, similarity search finds its similar objects based on a certain similarity metric. As metric space is able to accommodate any type of data (e.g., locations, strings, and images) and support flexible similarity metrics (e.g., L_p-norm distance, edit distance, and cosine similarity), similarity search in metric spaces is becoming increasingly important in a wide spectrum of real-life applications such as multi-media retrieval, decision making, data analysis, and personalized recommendation [1, 25, 26, 34, 39, 40]. However, existing studies mostly focus on a single metric space, resulting in the limitation that they can only support the similarity search over a single type of data with a specific similarity metric. Recently, many database vendors start developing multi-model database systems (e.g., OrientDB[1], ArangoDB[2], Azure Costmos DB[3], and GeminiDB[4]), which aims to support a range of models in a single system. Such systems easily become very complex due to various data types and flexible distance metrics. In contrast, multi-metric spaces combine multiple metric spaces to simultaneously model different features of objects that are presented by data of various types and support a collection of associated similarity metrics. It is able to perform similarity search in any combination of metric spaces. Thus, multi-metric index offers a method to reduce this complexity, which benefits the development of vector database.

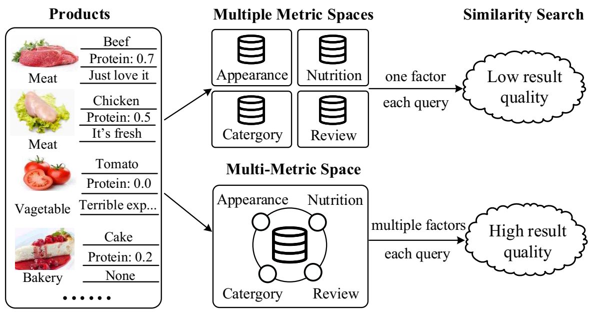

Figure 1: Food Products

In real-life applications, it is easy to gather a variety of information to capture different properties of objects, especially in the era of big data. For example, as shown in Fig. 1, customers consider many factors such as appearance, nutrition, category, and customer review when buying food. In this case, appearance, nutrition, category, and customer review of each food item are represented by images, vectors, strings, and texts respectively, and hence food items can be modeled as objects in four different metric spaces. Given a customer who wants to find the meat with positive review and high protein content, if we perform similarity search in a single metric space that contains category information, the result quality may be low. This is because products belonging to the meat category could have significantly different nutrition/reviews, bringing a large amount

Proceedings of the VLDB Endowment, Vol. 15, No. 10 ISSN 2150-8097.
doi:10.14778/3547305.3547317

[1]OrientDB. https://orientdb.org (2010)
[2]ArangoDB. https://www.arangodb.com (2014)
[3]Microsoft. https://azure.microsoft.com/services/cosmos-db/ (2017)
[4]Huawei. https://www.huaweicloud.com/intl/product/geminidb.html (2019)

of unwanted candidates that require further verification from the customer (e.g., the chicken that has median protein content is an irrelevant result). In contrast, multi-metric space can simultaneously model multiple data types by combining metric spaces, and thus, it can support high quality similarity search. In the example, we can provide customers with more accurate recommendations by considering multiple criteria simultaneously (including category, nutrition, and customer reviews). That is to say we only recommend beef, which is the most appropriate option. Therefore, multi-metric space can support flexible similarity search for any combination of metric spaces to obtain high quality result.

Many existing studies [2–4, 9–15, 18, 23, 24, 27–29, 31–33, 35, 41] focus on indexing single metric space, while a few [5–8, 16, 17, 19, 22, 30, 36, 37] target at multi-metric spaces. With the increasing demand to manage and explore data objects across different metric spaces, recently researchers have investigated the problem of indexing multi-metric space and supporting similarity search in any combination of metric spaces. They can be generally classified into two categories, namely, *combined methods* [6, 7, 16, 17, 19] and *separate methods* [5, 8, 19, 22, 30, 36, 37]. Combined methods use a single index structure to manage all the metric spaces by treating different metric spaces equally, and they measure the similarity between objects via a linear combination of individual distance metrics to support similarity search. Separate methods construct an index structure for each individual metric space, and support the similarity search by combining the candidate results from each metric space. Combined methods cannot efficiently support flexible combinations of metric spaces, but separate methods incur high construction and search costs. Hence, we aim to design an efficient multi-metric space index following separate methods to support flexible combinations of metric spaces, while having low construction and query costs. Three challenges exist below.

Challenge I: How to effectively index multi-metric space? Objects in different metric spaces are usually with different types. Combined methods transform multiple metric spaces into a single metric space, which fails to preserve the individuality of each metric space. Although separate methods can capture the properties of each metric space, they incur high construction cost to build each separate index and high query cost to search in each separate index. To this end, we propose a cluster-based forest index for multi-metric spaces. We select high quality centers for each metric space and then cluster objects based on the selected centers into compact regions, which can well capture the characteristics of each metric space. Next, we use B^+-trees to effectively manage each cluster by indexing the distances between objects and their corresponding cluster center, which forms a cluster-based forest. Note that, only distance values are stored in each B^+-tree, but the detailed object information is stored together and only once in a random access file (RAF), which effectively reduces the storage cost.

Challenge II: How to efficiently perform similarity search in multi-metric spaces? Similarity search in multi-metric spaces needs to combine the metrics when computing the similarity. Existing studies design distance bounds w.r.t. the weights of each metric to accelerate the similarity search. However, it is difficult for users to determine the weights in real-life applications. Take Fig. 1 as an example. It is difficult to set the weight for each metric (i.e., appearance, nutrition, category, and customer review) to digitize the preference. In addition, users have various preferences. For instance, nutritionists have high preference on the nutrition and the category, but restaurants have high preference on the appearance and the customer review. Thus, we first search among cluster centers in queried metric spaces to prune unnecessary metrics and shrink the search regions of corresponding spaces. Then, we develop filtering techniques to accelerate the search for candidate objects in B^+-trees. Finally, we integrate and validate candidate objects from different metric spaces to obtain the actual result.

Challenge III: How to efficiently support dynamic scenarios? In real-life applications, new objects may come while existing objects may change. As depicted in Fig. 1, new customer reviews are submitted, and the appearance of food may change over time. The updates bring great challenge to multi-metric space indexing. To address this, we design efficient index update strategies to support dynamic scenarios. Specifically, we dynamically choose cluster centers according to the distribution change among objects, and then efficiently update the corresponding B^+-trees, which supports efficient updates on objects and metric spaces.

In this paper, we present a $\underline{d}$ynamic clust$\underline{e}$r-based fore$\underline{s}$t $\underline{i}$ndex for multi-met$\underline{r}$ic spac$\underline{e}$s, called DESIRE, which can well capture the characteristics of individual metric space, and support efficient updates and flexible similarity search with low construction cost. To sum up, our key contributions are as follows:

- *Effective indexing framework for multi-metric spaces.* We develop a dynamic cluster-based forest indexing framework DESIRE for multi-metric spaces to simultaneously capture the characteristics of each individual space and support efficient index construction and flexible similarity search.
- *Dynamic cluster-based forest.* DESIRE first selects high quality centers to cluster objects into compact regions for each metric space. It then builds a B^+-tree for each cluster, and all those B^+-trees form a cluster-based forest. Moreover, we present efficient update techniques with theoretical analysis to support dynamic scenarios.
- *Efficient similarity search.* We design filtering techniques to accelerate similarity search in multi-metric spaces, including multi-metric range and multi-metric k nearest neighbour queries for any combination of metric spaces.
- *Extensive experiments.* We conduct extensive experimental evaluation on three real datasets and one synthetic dataset. The results demonstrate that DESIRE achieves flexible and efficient updates, supports efficient multi-metric similarity search, and scales well with the data size.

The rest of this paper is organized as follows. We review the related work in Section 2, and present the problem statement in Section 3. We introduce the dynamic cluster-based forest index for multi-metric spaces in Section 4, and detail the similarity search with filtering techniques in Section 5. Comprehensive experiments and our findings are reported in Section 6. Finally, Section 7 concludes the paper, and offers directions for future work.

2 RELATED WORK

In this section, we review the previous work on indexing single metric space and indexing multi-metric space.

Table 1: Symbols and description

Notation	Description
q, o	A query, an object in a metric space
$\bar{q}$, $\bar{o}$	A query, an object in a multi-metric space
o^i	The data part of $\bar{o}$ in the i-th metric space
C^i, c^i	A cluster, a cluster center in the i-th metric space
$\bar{O}$	An object set in a multi-metric space
n	The number of objects in a multi-metric space
m	The number of metric spaces
$d_i(\cdot,\cdot)$	A distance metric in a single metric space
$\mathbb{W}$, $d^{\mathbb{W}}(\cdot,\cdot)$	A weight vector, a multi-metric distance
$\mathbb{D}$	A set of distance metrics d_i $(1 \le i \le m)$
$MMRQ(\bar{q}, \mathbb{W}, r)$	A multi-metric range query w.r.t. a query object q, a weight vector $\mathbb{W}$, and a radius r
$MMkNNQ(\bar{q}, \mathbb{W}, k)$	A multi-metric kNN query w.r.t. a query object q, a weight vector $\mathbb{W}$, and an integer k

2.1 Indexing Single Metric Space

Indexing single metric space for similarity search has been widely studied, and existing index structures can be generally classified into three categories [12], i.e., *compact partitioning methods*, *pivot-based methods*, and *hybrid methods* that combine the previous two techniques [15].

Compact partitioning methods (including GHT [32], SAT [9, 28], M-tree [18], LC [10, 11], etc.) partition the space into compact sub-regions, and propose filtering techniques to prune unqualified sub-region(s) during the search. Pivot-based methods (including LAESA [27], VPT [32, 35], MVPT [2, 3], OmniR-tree [24], SPB-tree [13, 14], etc.) store pre-computed distances from every object in the database to a set of pivots, and then, they utilize these distances and the triangle inequality to prune unqualified objects during search. Hybrid methods combine compact partitioning with the use of pivots. GNAT [4] and pivoting metric tree [31] combine the partitioning technique and cut-regions defined by pivots to accelerate similarity search. M-index [29] generalizes the iDistance [23] technique for general metric spaces, and uses the B$^+$-tree to store pre-computed distances. Note that, iDistance is designed for vector spaces, and cannot support dynamic updates of objects.

Although the aforementioned techniques achieve high similarity search efficiency in a single metric space, they are not able to support similarity queries in dynamically combined multiple metric spaces, i.e., similarity search in multi-metric spaces.

2.2 Indexing Multi-Metric Space

As various data types co-exist and can be gathered to improve the quality of similarity search, a single metric space can no longer be applied to simultaneously process multiple data types. In view of this, recently many studies have targeted at multi-metric space indexing to accelerate the similarity search on various data types, which can be classified into *combined methods* and *separate methods*.

Combined methods treat different metric spaces equally, and linearly combine distance metrics from different metric spaces into a single metric. QIC-M-tree [17] applies the user defined distance that can be regarded as a combination of multiple metrics to build the index. M^2-tree [16] and M^3-tree [7] modify the structure of M-tree by using integrated distance metric to choose routing objects, and storing a vector of partial distances to estimate weighted distances in node entries. A general methodology is presented to

Table 2: A multi-metric space dataset (Apartment Set)

	Price	Room	Location	Date	Review
a	340	(2,1)	(40.71,-74.01)	16-04-01	Wonderful
b	925	(1,2)	(40.61,-75.47)	19-02-24	Dark
c	2180	(3,4)	(40.72,-74.00)	19-05-12	Terrible
d	520	(1,1)	(40.70,-73.99)	18-12-25	Great

adapt existing single metric space index (e.g., GNAT and LC) to multi-metric space index [6]. Nonetheless, combined methods fail to preserve the individuality of each metric space. RR*-tree [19] captures the characteristic of each metric space via reference-object embedding, and utilizes R-tree to index the embedded objects. However, it suffers from the curse of dimensionality, especially for a larger number of metric spaces. In addition, combined methods incur a high cost of rebuilding the whole index whenever a new metric is combined into the current multi-metric space.

Separate methods index objects in each individual metric space separately. As pivots are verified having strong pruning power [12, 15, 41], C-forest [8] and pivot-based index [5] select high quality pivots to index objects in each metric space. However, they are mainly designed for static multi-metric spaces, i.e., the subset of metric spaces that are combined during the similarity search is supposed to be known in advance. Similarly, Spectra [38] utilizes the pivots to embed and index each metric space, and metric spaces with low correlations are indexed together. Nevertheless, these methods are main-memory indexes, which are difficult to be efficiently implemented in secondary-indexes to support large-scale datasets. Also, they are not suitable for dynamic scenarios, as the indexes need to be rebuilt whenever pivots are changed. Many studies [22, 30, 36, 37] build single metric index for each metric space, and investigate preference top-k queries. For example, PM-tree [19] designs parallel M-trees to index multiple metric spaces, and uses the threshold to boost the range queries. Nonetheless, they suffer from high construction cost to build index for each metric space and high query overhead to search in each separated index.

3 PROBLEM FORMULATION

We proceed to introduce the multi-metric space and the definitions of similarity search. Table 1 summarizes frequently used notations.

A metric space is represented by a tuple (M, d), where M is the domain of objects, and d is a distance metric to measure the similarity between any pair of objects q and o in this space. The distance metric d satisfies: (i) symmetry: $d(q, o) = d(o, q)$; (ii) non-negative: $d(q, o) \ge 0$; (iii) identity: $d(q, o) = 0$ iff $q = o$; and (iv) triangle inequality: $d(q, o) \le d(q, o') + d(o, o')$. Note that, in a single metric space, M only contains data objects belonging to a single type. A multi-metric space is represented by a tuple $(\mathbb{M}, \mathbb{D})$ that combines m metric spaces, where $\mathbb{M}$ is a collection of domains M_i $(1 \le i \le m)$ and $\mathbb{D}$ is a set of distance metrics d_i $(1 \le i \le m)$ for each metric space M_i. Hence, the multi-metric space can simultaneously model multiple data types with various distance metrics. Here, an object in a multi-metric space is denoted as $\bar{o} = \{o^i | 1 \le i \le m\}$, where o^i is an object in metric space domain M_i.

Table 2 gives an example of multi-metric space object set to describe the apartment information, i.e., $\bar{O}$=$\{a, b, c, d\}$. Each apartment includes the information of i) rental price, ii) the number of bathrooms and bedrooms, iii) location, iv) publish date, and v) a brief review. Take apartment a as an example. The monthly rental is \$340; it has two bathrooms and one bedroom; it locates at New York with geomagnetic coordinates 74.01°W and 40.71°N; it is published on 1 April 2016; and one customer left a comment “Wonderful”. Hence, the multi-metric space in Table 2 combines five single metric spaces $(M_i, d_i)(1 \leq i \leq 5)$, where M_1 is a domain of one-dimensional values to describe the rental prices, M_2 is a domain of two-dimensional vectors to describe the room information, M_3 is a domain of two-dimensional vectors to state the location, M_4 is a domain of date, and M_5 is a domain of strings to describe the reviews. In addition, metric spaces use different distance metrics. For instance, L_1-norm, L_2-norm, and edit distances can be used as a distance metric for M_1, M_3, and M_5, respectively.

Formally, we define the multi-metric distance to measure the similarity between objects as follows.

Definition 1. **(Multi-Metric Distance)** *Given a weight vector* $\mathbb{W} = (\omega_1, \cdots, \omega_m)$, $\omega_i \in \mathbb{R} \wedge \omega_i \in [0, 1]$ *to describe the importance of each metric space, the multi-metric distance* $d^{\mathbb{W}}(\cdot, \cdot)$ *between two objects* $\bar{q}$ *and* $\bar{o}$ *is defined as* $d^{\mathbb{W}}(\bar{q}, \bar{o})=\sum_{d_i \in \mathbb{D}} \omega_i \cdot d_i(q^i, o^i)$.

Here, ω_i indicates the importance of metric d_i that participates into the similarity measurement. In Table 2, a weight vector $\mathbb{W} = (0.5, 0, 0.5, 0, 0)$ means that we only consider price and location information equally when computing the similarity. In real-life applications, users may have various preferences during similarity search. For example, some users choose apartments according to the location and review information; while others only consider price information. Thus, the similarity computation in a multi-metric space, i.e., Definition 1, is flexible to support any combination of metrics. Following the previous study [19], the distances $d_i(q^i, o^i)$ in each metric space (M_i, d_i) are divided by two times the median of all occurring distances for normalization. For simplicity, we use $d_i(\bar{q}, \bar{o})$ to denote $d_i(q^i, o^i)$ in the rest of the paper.

The multi-metric distance metric $d^{\mathbb{W}}(\cdot, \cdot)$ also satisfies symmetry, non-negative, identity, and triangle inequality properties. Detailed proofs are omitted due to the space limitation. Based on the multi-metric distance, we define two types of similarity search in multi-metric spaces, i.e., multi-metric range query and multi-metric k nearest neighbour (kNN) query.

Definition 2. **(Multi-Metric Range Query)** *Given an object set* $\bar{O}$, *a query object* $\bar{q}$, *a weight vector* $\mathbb{W}$, *and a search radius* r *in a multi-metric space, a multi-metric range query (MMRQ) finds objects in* $\bar{O}$ *with their distances to* $\bar{q}$ *no larger than* r, *i.e.,* $MMRQ(\bar{q}, \mathbb{W}, r) = \{\bar{o} | \ \bar{o} \in \bar{O} \wedge d^{\mathbb{W}}(\bar{q}, \bar{o}) \leq r\}$.

Definition 3. **(Multi-Metric kNN Query)** *Given an object set* $\bar{O}$, *a query object* $\bar{q}$, *a weight vector* $\mathbb{W}$, *and an integer* k *in a multi-metric space, a multi-metric kNN query (MMkNNQ) finds* k *objects in* $\bar{O}$ *that are most similar to* $\bar{q}$, *i.e.,* $MMkNNQ(\bar{q}, \mathbb{W}, k) = \{\bar{S} | \ \bar{S} \subseteq \bar{O} \wedge |\bar{S}| = k \wedge \forall \bar{s} \in \bar{S}, \forall \bar{o} \in (\bar{O} - \bar{S}), d^{\mathbb{W}}(\bar{q}, \bar{s}) \leq d^{\mathbb{W}}(\bar{q}, \bar{o})\}$.

Suppose $\mathbb{W} = (0.5, 0, 0.5, 0, 0)$ in Table 2, the multi-metric range query $MMRQ(a, \mathbb{W}, 0.3)$ retrieves apartments in $\bar{O}$ that are within the multi-metric distance 0.3 to a, s.t. $MMRQ(a, \mathbb{W}, 0.3) = \{d\}$. The multi-metric kNN query $MMkNNQ(a, \mathbb{W}, 2)$ finds two apartments that are most similar to a, yielding $MMkNNQ(a, \mathbb{W}, 2) = \{c, d\}$.

4 THE FRAMEWORK DESIRE

In this section, we present an overview of the dynamic cluster-based forest indexing framework DESIRE for multi-metric spaces, and then introduce its corresponding structure. Next, we detail the construction and update strategies to support dynamic scenarios. Finally, we analyze the complexities of DESIRE, including the space consumption, the time complexity, and the I/O cost.

Our construction framework of DESIRE contains three steps. (i) We cluster the multi-metric objects in each metric space. Specifically, we select high quality centers in each metric space in order to preserve the properties of individual metric space, and then, we group objects into clusters based on the selected centers in every metric space. (ii) For each cluster, we employ a B$^+$-tree to efficiently index the distances from objects belonging to this cluster to the cluster center, which forms the B$^+$-forest. Here, B$^+$-trees store the one-dimensional distances and the partial data information instead of the entire multi-metric objects, which reduces the storage cost significantly. Besides, we also leverage the pre-computed distances to prune the objects during the search, which avoids the unnecessary distance computations and improves the search performance. (iii) DESIRE dynamically updates the cluster centers, and adjusts the corresponding B$^+$-forest structure to support dynamic scenarios.

4.1 Indexing Structure

DESIRE consists of three components, i.e., lists of clusters, the B$^+$-forest, and the random access file (RAF), as depicted in Fig. 2. To be more specific, for a given object set $\bar{O} = \{\bar{o}_1, \bar{o}_2, ..., \bar{o}_{10}\}$, Fig. 2(a) shows the clustering result in each metric space, Fig. 2(b) reports the lists of clusters, and Fig. 2(c) plots the B$^+$-forest and the RAF.

Lists of clusters. Fig. 2(b) illustrates an example of cluster lists according to clustering shown in Fig. 2(a). For each metric space, we have a list to store the clusters obtained in this metric space. For instance, in the first list corresponding to metric space (M_1, d_1), we have four entries to describe the obtained four clusters C_1^1, C_2^1, C_3^1, and C_4^1. Similarly, in the m-th list, we have four entries to describe the clusters C_1^m, C_2^m, C_3^m, and C_4^m. Specifically, each entry for describing C_j^i stores i) the cluster center (denoted as $C_j^i.c$), and ii) the pointer (denoted as $C_j^i.ptr$) to the B$^+$-tree that indexes the underlying objects. Note that, for a center c in the i-th list, we only store the center information c in a specific metric space (M_i, d_i) instead of its information in all m spaces. One B$^+$-tree is used for a cluster to index objects in this cluster, as detailed below.

B$^+$-forest Indexing. We use the secondary memory B$^+$-tree to support efficient construction, update, and search. Each B$^+$-tree indexes the distances between objects and the cluster center. Fig. 2(c) shows the B$^+$-forest to index clusters in Fig. 2(b), where two detailed B$^+$-trees are built for C_4^1 and C_2^m, respectively. Each entry E in a non-leaf node (e.g., N_0^m) records the key value (denoted as $E.key$), the minimum and the maximum keys (denoted as $E.min$ and $E.max$). As an example, $E_4.key$ in node N_0^m is $d_m(o_2^m)$ (the short form of $d_i(c^i, o_j^i)$), $E_4.min$ is $d_m(o_2^m)$, and $E_4.max$ is $d_m(o_5^m)$. Each entry E in a leaf node (e.g., N_0^1, N_1^m, and N_2^m) records the key value (denoted as $E.key$), the object (denoted as $E.o$) and the pointer (denoted as $E.ptr$) to the RAF file. Instead of storing the entire object $\bar{o}$ in each metric space, $E.o$ only records the partial data o^i in (M_i, d_i). Since a B$^+$-tree is built for each cluster, all the B$^+$-trees form the B$^+$-forest.

RAF File. All the B$^+$-trees share the same RAF file, which stores all the multi-metric objects. In a B$^+$-tree leaf entry, it stores the partial data (i.e., the object in a particular metric space), and also stores

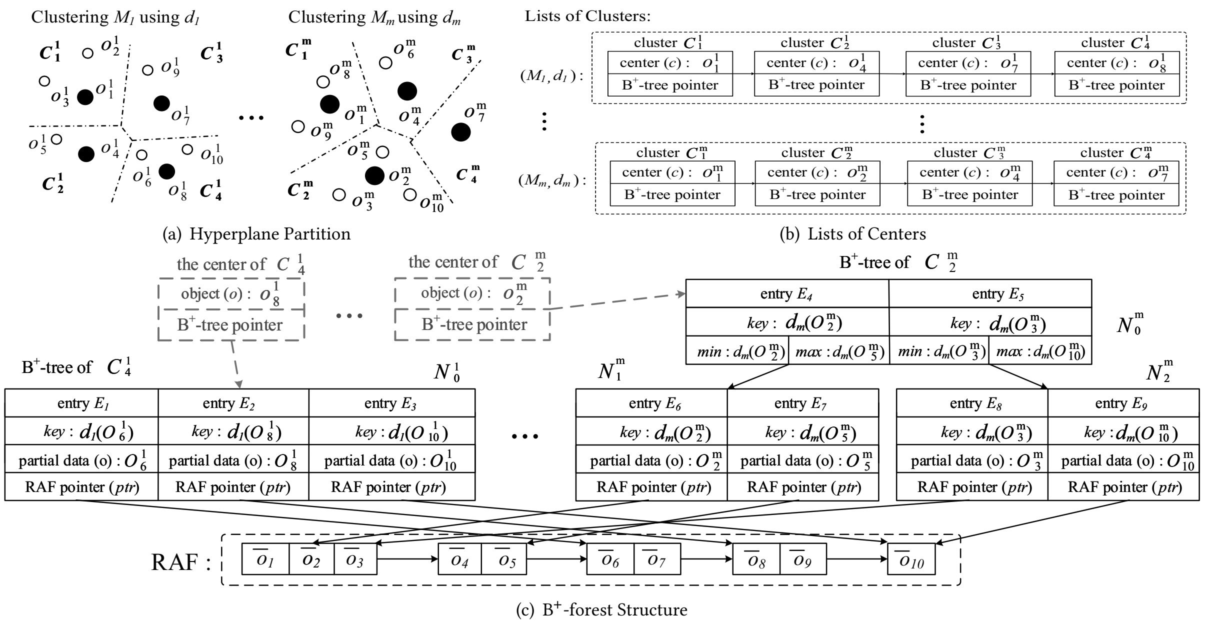

Figure 2: Structure of the DESIRE

the pointer to the entire multi-metric object in RAF. For example, in Fig. 2(c), the first leaf entry of node N_1^m stores the partial data (i.e., o_2^m) and the pointer to the address of $\bar{o}_2$ in RAF.

4.2 Construction and Update

In a dynamic scenario, new objects may come, and existing objects may expire. Accordingly, we design proper object insertion/deletion operations such that DESIRE can support efficient multi-metric queries even in a dynamic setup. **Clustering objects.** For each metric space (M_i, d_i), we adopt the generalized hyperplane partitioning technique to cluster objects, i.e., we assign the newly coming object $\bar{o}$ to a cluster C_j^i whose center c_j^i is *nearest* to o^i, which is defined below.

DEFINITION 4. **(Generalized Hyperplane Partitioning)** *Given a list LC^i of clusters, a set O^i of objects, and a distance metric d_i, let c_i be the center of the cluster $C^i \in LC^i$. The cluster C^i grouped by c^i using generalized hyperplane partitioning technique is defined as* $\{o^i | o^i \in O^i \wedge \forall c^{i\prime} \neq c^i, d_i(o^i, c^i) \leq d_i(o^i, c^{i\prime})\}$.

The reason why we adopt the generalized hyperplane partitioning is as follows. The distance computation (e.g., edit distance, L_p-norm for high dimensional vectors) is usually costly in metric spaces. To avoid unnecessary distance computations, DESIRE estimates the distance bounds via triangle inequality. Given a query object $\bar{q}$, let c_j^i be the center of the cluster where $\bar{o}$ is located in (M_i, d_i). We have $|d_i(\bar{q}, c_j^i) - d(c_j^i, \bar{o})| \leq d_i(\bar{o}, \bar{q}) \leq d_i(\bar{q}, c_j^i) + d_i(c_j^i, \bar{o})$, where $d_i(c_j^i, \bar{o})$ is pre-computed and stored in DESIRE. Hence, we need to compute only $d_i(\bar{q}, c_j^i)$ to estimate the upper and lower bounds of $d_i(\bar{o}, \bar{q})$ for any object $\bar{o}$ in the cluster. According to the estimation, the smaller the $d(c_j^i, \bar{o})$ is, the tighter the bounds will be. Motivated by this, we assign $\bar{o}$ to a cluster whose center c_j^i is the closest to it. Note that, if an object shares an equal shortest distance to multiple centers, it will be assigned to the cluster of the center with the smallest index number i based on Definition 4.

Center selection. Given a cluster C_j^i in a metric space (M_i, d_i) with the center c_j^i and the maximum distance max_j^i between the center and the objects in the cluster, for any object pairs (o_x^i, o_y^i) in this cluster, $\max(|d_i(o_x^i, c_j^i) - max_j^i|, |d_i(o_y^i, c_j^i) - max_j^i|) \leq d_i(o_x^i, o_y^i) \leq \min(d(o_x^i, c_j^i) + max_j^i, d(o_y^i, c_j^i) + max_j^i)$. Therefore, the smaller the max_j^i, the tighter the bounds of $d_i(o_x^i, o_y^i)$ will be. In order to obtain well-distributed centers, mainstream solutions look for objects in each cluster that can minimize max_j^i values and set those objects as new centers. Consequently, they incur extremely high computation cost as they have to calculate pairwise distances for all the objects in each cluster. To this end, we apply a simple yet effective way [21] to choose objects having maximum distances to existing centers as new centers to form new clusters (to note, the first inserted object is the first center).

To note, the number of centers (which is equivalent to the number of clusters) is critical. If we have only a few centers, the maximum distance max_j^i of the cluster will be large, resulting in poor distance estimation. If we have a large number of centers, the computation cost of inserting a new object $\bar{o}$ will be high, as we need to compute the distances between $\bar{o}$ and all the centers to find the nearest one. Thus, we use a tuning parameter λ $(0 \leq \lambda \leq 1)$ to control the number of centers, which will be analyzed in Section 4.3, and its impact will be evaluated in Section 6.1. Specifically, let num_c^i be the number of centers in a metric space, num_o be the number of indexed multi-metric objects, the clustering requires a update when $num_c^i \neq \lceil (num_o)^\lambda \rceil$. For simplicity, we assume that all the metric spaces share a common λ value, although it is not necessary as each metric space can have its own λ setting.

Cluster deletion. When we need to reduce the number of clusters, we find a cluster pair having the minimum distance among all the cluster pairs, and delete the smaller cluster in the pair. Specifically, assume clusters C_j^i and C_l^i form the cluster pair with the smallest distance, and C_j^i with the center c_j^i and the distance max_j^i is the one

Algorithm 1: Object Insertion Algorithm

```
Input: a new multi-metric object $\bar{o}$, a DESIRE index, and a
       parameter $\lambda$
1:  insert $\bar{o}$ into RAF, $loc \leftarrow$ the address of $\bar{o}$ in RAF
2:  $num_o \leftarrow num_o + 1$ // update the number of multi-metric objects
3:  foreach metric space $(M_i, d_i)$ do
4:      $LC^i \leftarrow$ the list of clusters in $(M_i, d_i)$
5:      $C_{min} \leftarrow \arg\min_{C^i_j \in LC^i} d_i(o^i, C^i_j.c)$
6:      $E \leftarrow$ new B$^+$-tree leaf entry $(d_i(C_{min}.c, o^i), o^i, loc)$
7:      insert $E$ into the B$^+$-tree of $C_{min}$
8:      if $num^i_c < \lceil (num_o)^\lambda \rceil$ then Add_center($LC^i$, $d_i$)
9:  Function Add_center($LC^i, d_i$)
10: $C_{cand} \leftarrow \emptyset$, $dis_{max} \leftarrow 0$
11: $num^i_c \leftarrow num^i_c + 1$ // update the number of centers
12: foreach cluster $C^i_j \in LC^i$ do
13:     $N \leftarrow$ the root node of the B$^+$-tree for $C^i_j$
14:     $max^i_j \leftarrow$ the maximum distance in last entry of $N$
15:     if $max^i_j > dis_{max}$ then $C_{cand} \leftarrow \{C^i_j\}$, $dis_{max} \leftarrow max^i_j$
16:     else if $max^i_j = dis_{max}$ then $C_{cand} \leftarrow C_{cand} \cup \{C^i_j\}$
17: $C_{max} \leftarrow \arg\max_{C \in C_{cand}} |C|$ // cluster with largest size
18: $E \leftarrow$ get_last_entry($C_{max}$) // last leaf entry in B$^+$-tree of $C_{max}$
19: remove $E$ from $C_{max}$
20: $C_{new} \leftarrow (E.o$, 0, the pointer to an empty B$^+$-tree)
21: foreach cluster $C^i_j \in LC^i$ do
22:     $E \leftarrow$ get_last_entry($C^i_j$)
23:     while $d_i(E.o, C_{new}.c) < d_i(E.o, C^i_j.c)$ do
24:         move $E$ from $C^i_j$ to $C_{new}$, $E \leftarrow$ get_last_entry($C^i_j$)
25: $LC^i \leftarrow LC^i \cup C_{new}$
```

to be deleted from the metric space (M_i, d_i). This deletion strategy is motivated by our consideration of the changes to the maximum distances of other clusters in this metric space. For any object o^i_j originally located inside C^i_j, it needs to be moved to the new closest cluster (e.g., C^i_k), i.e., $d_i(o^i_j, c^i_k) \le d_i(o^i_j, c^i_l)$. In addition, according to the triangle inequality and the definition of max^i_j, we can derive that $d_i(o^i_j, c^i_l) \le d_i(o^i_j, c^i_j) + d_i(c^i_j, c^i_l) \le max^i_j + d_i(c^i_j, c^i_l)$. After object o^i_j is inserted into cluster C^i_k, the maximum distance max^i_k of C^i_k needs to be updated to $max(d_i(c^i_j, c^i_l) + max^i_j, max^i_k)$ in the worst case. As max^i_j and max^i_k are fixed, in order to minimize $d_i(c^i_j, c^i_l)$ and tighten the radii of cluster centers, we find a cluster pair having the minimum distance between the centers, and then delete the cluster with smaller size for higher efficiency (as fewer objects will be updated).

Object Insertion operation. We develop the object insertion operation, with the pseudo-code shown in Algorithm 1. Due to space limitation, traditional operations for B$^+$-trees and clusters are omitted. First, the algorithm inserts the new multi-metric object in the RAF file to get the address loc (line 1), and updates the number num_o of multi-metric objects (line 2). Then, in each metric space (M_i, d_i), it finds the closest cluster C_{min} for o^i (lines 3–5), creates corresponding leaf entry E (line 6), and inserts E into the B$^+$-tree of C_{min} (line 7). Next, the algorithm calls function Add_center to include a new cluster to LC^i if the number of centers num^i_c is smaller than $\lceil (num_o)^\lambda \rceil$(line 8). The function first finds all the candidate clusters C_{cand} having the maximum distance (lines 10–16). Then, it finds the cluster C_{max} in C_{cand} with the largest size, and uses the farthest object to the center in C_{max} as the new center to create the new cluster C_{new} (lines 17–20). Here, $get\_last\_entry(\cdot)$ is to get the last entry in the B$^+$-tree of C_{max} that contains the farthest object. In addition, for all the other clusters in LC^i, the objects will be re-assigned if they are nearer to the new cluster (lines 21–24). Finally, LC^i is updated to include a new cluster C_{new} (line 25).

Algorithm 2: Object Deletion Algorithm

```
Input: an object $\bar{o}$, a DESIRE index, and a parameter $\lambda$
1:  $num_o \leftarrow num_o - 1$ // update the number of multi-metric objects
2:  foreach metric space $(M_i, d_i)$ do
3:      $LC^i \leftarrow$ the list of clusters in $(M_i, d_i)$
4:      $E \leftarrow$ the B$^+$-tree entry of $o^i$ indexed in $LC^i$
5:      remove $E$ from $LC^i$
6:      if $num^i_c > \lceil (num_o)^\lambda \rceil$ then Delete_center($LC^i$, $d_i$)
7:  Function Delete_center($LC^i, d_i$)
8:  $(C^i_{m1}, C^i_{m2}) \leftarrow \arg\min_{C^i_x, C^i_y \in LC^i, |C^i_x| \le |C^i_y|} d_i(C^i_x.c, C^i_y.c)$
9:  $LC^i \leftarrow LC^i - \{C^i_{m1}\}$, $num^i_c \leftarrow num^i_c - 1$
10: foreach leaf entry $E \in C^i_{m1}$ do
11:     $C_{min} \leftarrow \arg\min_{C^i_j \in LC^i} d(C^i_j.c, E.o)$
12:     move $E$ from $C^i_{m1}$ to $C_{min}$
```

Object Deletion operation. Algorithm 2 lists the pseudo-code of object deletion operation. For a multi-metric object $\bar{o}$ to be deleted, the algorithm first updates the number num_o of objects (line 1). Then, for each metric space (M_i, d_i), it finds the leaf entry E of o^i, and removes E from the list LC^i of clusters (lines 3–5). Finally, it invokes Delete_center function to remove a cluster from LC^i if the number num^i_c of centers is larger than $\lceil (num_o)^\lambda \rceil$ (line 6). $Delete\_center$ first finds a cluster pair (C^i_{m1}, C^i_{m2}) having the minimum distance among all the center pairs (lines 8). It then removes the cluster C^i_{m1} with the smaller size, updates the number num^i_c of centers, and reallocates the objects in C^i_{m1} (lines 9–12).

Metric Space Insertion/Deletion. As a separate method, DESIRE also supports efficient metric space updating. Specifically, when a metric space is removed, DESIRE only needs to delete the clusters and B$^+$-forests in that space, and updates the RAF file. Similarly, if a new space is added, DESIRE inserts the objects of the new space to the index, without making any changes to the existing part. These operations can be easily implemented in a way similar to object insertion/deletion, and thus, are omitted here.

4.3 Complexity Analysis

Space Consumption. DESIRE consists of three components, i.e., the lists of clusters, the B$^+$-forest, and the RAF file. Let n be the size of the object set (i.e., $|\bar{O}|$), and m be the number of metric spaces. The storage cost of the lists of clusters is $O(mn^\lambda)$, as the number of centers is controlled by n^λ. For each metric space (M_i, d_i), let n_j denote the number of the objects in cluster C^i_j. Then, the estimated space cost for the B$^+$-forest is:

$$F(n_1, n_2, \cdots) = \sum_{j=1}^{n^\lambda} O(n_j) = O(n) \qquad (1)$$

According to Eq. (1) and the fact that each metric space is managed by n^λ B$^+$-trees, the space cost of each B$^+$-tree is $O(n^{1-\lambda})$. The size

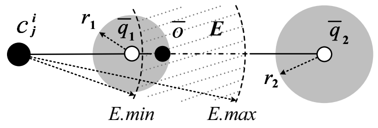

Figure 3: Illustration of Pruning Non-leaf Entries

of RAF file equals to the object set size $O(ns)$, where s represents the average size of an object in a multi-metric space. In summary, as $s \gg 1$, the total space cost of DESIRE is $O(ns + mn^{\lambda})$.

Time Complexity of Object Insertion. We need to insert the new object into m metric spaces, which requires two steps: i) finding the cluster to be inserted with the cost $O(n^{\lambda} \cdot dcmp)$; and ii) inserting the object in B$^+$-tree of the cluster with the cost $O(\log n^{1-\lambda})$. Here, $dcmp$ denotes the cost of a distance computation, and each B$^+$-tree contains $n^{1-\lambda}$ objects on average. Next, we analyze the cost of Add_center function, which adds a center for each of m spaces and includes three steps: i) finding the farthest object in each cluster with the cost $O(n^{\lambda} \log n^{1-\lambda})$; ii) finding the best center and creating a new cluster with the cost $O(n^{\lambda})$; and iii) redistributing objects in old clusters to the new cluster with the cost $O(n^{1-\lambda} \log n^{1-\lambda} + n^{1-\lambda} \cdot dcmp)$, as $O(n^{1-\lambda})$ objects on average are moved to the new cluster. Thus, as $0 \le \lambda \le 1$, the total cost of an insertion operation is $O(m(n^{\lambda} + n^{1-\lambda}) \cdot (dcmp + \log n^{1-\lambda}))$.

Time Complexity of Object Deletion. Similar to the object insertion, the cost of deleting an object in each metric space is $O(\log n^{1-\lambda} + n^{\lambda} \cdot dcmp)$. Next, the cost for deleting a center includes $O(n^{2\lambda})$ distance computations to find the cluster to be deleted, $O(n^{\lambda})$ distance computations to find the nearest cluster of each object in the deleted cluster, and $O(\log n^{1-\lambda})$ cost to add each object in the deleted cluster to the B$^+$-tree of its new cluster (as each B$^+$-tree contains $n^{1-\lambda}$ objects on average). Hence, the total cost of a deletion operation is $O(mn^{2\lambda} \cdot dcmp + mn^{\lambda} \log n^{1-\lambda})$.

I/O Cost of Object Insertion/Deletion. The I/O cost includes three parts: i) the I/O cost for lists of clusters is $O(|LC|_p)$, where $|LC|_p$ denotes the number of pages to store the lists of clusters, as we visit and update the lists of clusters only once in order; ii) the I/O cost of B$^+$-forest is $O(m(n^{\lambda} + n^{1-\lambda}) \log n^{1-\lambda})$ for insertion and $O(mn^{\lambda} \log n^{1-\lambda})$ for deletion according to the above time complexity analysis; and iii) the I/O cost of RAF file is O(1).

5 SIMILARITY SEARCH

In this section, we propose efficient similarity search algorithms in multi-metric spaces using DESIRE to support multi-metric range query and multi-metric kNN query, respectively.

5.1 Multi-Metric Range Query

As defined in Definition 2, a multi-metric range query aims to find multi-metric objects $\bar{o}$ whose multi-metric distances $d^{\mathbb{W}}(\bar{q}, \bar{o})$ to a query object $\bar{q}$ are within r. As DESIRE is a separate method (i.e., each metric space is indexed separately), we have to search the result in each metric space. To accelerate the search in single spaces, we borrow the idea from the pigeonhole principle, and propose the following filtering lemma.

Lemma 5.1. *Given a range query object $\bar{q}$, a query weight vector $\mathbb{W}$, the set $\mathbb{D}$ of distance metrics, and a radius r, object $\bar{o}$ is a result of MMRQ($\bar{q}$, $\mathbb{W}$, r) only if $\exists d_i \in \mathbb{D}, \omega_i > 0 \wedge d_i(\bar{q}, \bar{o}) \le \frac{r}{\sum_{d_i \in \mathbb{D}} \omega_i}$.*

Algorithm 3: RangeQuery Algorithm

```
   Input: a query object $\bar{q}$, the list of clusters $LC^i$, a distance metric
          $d_i$, and a search radius $\frac{r}{\sum \omega_i}$
   Output: the result set $Ans$ of entries
1  $Ans \leftarrow \varnothing$ // the result set
2  foreach $C \in LC^i$ do
3      $N \leftarrow$the B$^+$-tree root of $C$
4      $Ans \leftarrow Ans \cup TreeRangeQ(N, \bar{q}, d_i, \frac{r}{\sum \omega_i}, d(C.c, \bar{q}))$
5  return $Ans$
6  Function TreeRangeQ($N, \bar{q}, d_i, \frac{r}{\sum \omega_i}, dis_C$)
7  $Ans \leftarrow \varnothing$
8  foreach Entry $E$ of $N$ do
9      if $N$ is leaf node then
10         if $E.key \le \frac{r}{\sum \omega_i} + dis_C \wedge E.key \ge dis_C - \frac{r}{\sum \omega_i}$ then
11             $dis \leftarrow d(E.o, \bar{q})$
12             if $dis \le \frac{r}{\sum \omega_i}$ then $Ans \leftarrow Ans \cup \{(E.ptr, dis)\}$
13     else
14         if $E.min \le \frac{r}{\sum \omega_i} + dis_C \wedge E.max \ge dis_C - \frac{r}{\sum \omega_i}$ then
15             $N' \leftarrow$the B$^+$-tree node pointed by $E$
16             $Ans \leftarrow Ans \cup TreeRangeQ(N', \bar{q}, d_i, \frac{r}{\sum \omega_i}, dis_C)$
17 return $Ans$
```

Proof. Assume to the contrary that there is a result object $\bar{o}$ in MMRQ($\bar{q}$, $\mathbb{W}$, r) such that $\forall \omega_i \in \mathbb{W}, d_i(\bar{q}, \bar{o}) > \frac{r}{\sum_{d_i \in \mathbb{D}} \omega_i} \wedge d^{\mathbb{W}}(\bar{q}, \bar{o}) \le r$. According to the definition of MMRQ, $d^{\mathbb{W}}(\bar{q}, \bar{o}) = \sum_{d_i \in \mathbb{D}} \omega_i \cdot d_i(\bar{q}, \bar{o}) > \sum_{d_i \in \mathbb{D}} \omega_i \cdot \frac{r}{\sum_{d_i \in \mathbb{D}} \omega_i}$. Hence, we derive that $d^{\mathbb{W}}(\bar{q}, \bar{o}) > r$, which contradicts with our assumption. The proof completes. □

Based on Lemma 5.1, we conduct the range query with search radius of $\frac{r}{\sum \omega_i}$ in each single metric space with $\omega_i > 0$. If $\sum \omega_i$ remains unchanged, the search performance of DESIRE when answering MMRQ is very stable even when the weighted value of each metric space changes. During the search in each metric space, we can further use the triangle-inequality discussed in Section 4.2 to avoid unnecessary distance computations. As non-leaf entries in B$^+$-trees contain a set of objects in the sub-trees, we propose Lemma 5.2 to prune non-leaf entries.

Lemma 5.2. *Given a non-leaf entry E, the query object $\bar{q}$ in the metric space (M_i, d_i), and a search radius $\frac{r}{\sum \omega_i}$, let c denote the cluster center of this B$^+$-tree. If $E.min > \frac{r}{\sum \omega_i} + d_i(\bar{q}, c)$ or $E.max < d_i(\bar{q}, c) - \frac{r}{\sum \omega_i}$, E can be safely pruned for MMRQ in this single metric space.*

Proof. As $E.min$ and $E.max$ are the minimum and maximum key values (i.e., distances to the cluster center c) of all the objects in E, $\forall \bar{o} \in E$, $E.min \le d_i(\bar{o}, c) \le E.max$. If $E.max < d_i(\bar{q}, c) - \frac{r}{\sum \omega_i}$, we have $d_i(\bar{o}, c) < d_i(\bar{q}, c) - \frac{r}{\sum \omega_i}$, and then, $d_i(\bar{o}, \bar{q}) \ge d_i(\bar{q}, c) - d_i(\bar{o}, c) > \frac{r}{\sum \omega_i}$ due to the triangle inequality, indicating that all the objects in E can be discarded. Similarly, if $E.min > \frac{r}{\sum \omega_i} + d_i(\bar{q}, c)$, we have $d_i(\bar{o}, \bar{q}) \ge d_i(\bar{q}, c) - d_i(\bar{o}, c) > \frac{r}{\sum \omega_i}$, leading to the same conclusion that E can be pruned. The proof completes. □

Consider the example shown in Fig. 3. Given a cluster center c^i_j, an entry E in the cluster, a query object $\bar{q}_1$, and a radius r_1, E cannot be pruned for MMRQ in the single space by Lemma 5.2 due to $E.min < r_1 + d_i(\bar{q_1}, c^i_j)$ and $E.max > d_i(\bar{q}, c^i_j) - r_1$, i.e., E might contain objects (e.g., $\bar{o}$) that are query answers. However, given

Algorithm 4: MMRQ Algorithm

```
Input: a query object $\bar{q}$, a query weight vector $\mathbb{W}$, and a search
       radius $r$
Output: the result set $Ans$ of objects
1  $Ans \leftarrow \varnothing$ // the result set
2  foreach $\omega_i \in \mathbb{W}, \omega_i = 1$ do
3      $LC^i \leftarrow$ the list of clusters in $(M_i, d_i)$
4      $Res_i \leftarrow RangeQuery(\bar{q}, LC^i, d_i, \frac{r}{\sum \omega_i})$
5      foreach $(E.ptr, dis) \in Res_i$ do
6          $\bar{o} \leftarrow$the object pointed by $E.ptr$
7          if $\bar{o} \notin Ans$ then
8              $Ans \leftarrow Ans \cup \{\bar{o}\}$
9              $\bar{o}.dis[j] \leftarrow \infty, 1 \le j \le |\mathbb{D}|$
10         $\bar{o}.dis[i] \leftarrow dis$
11 foreach $\bar{o} \in Ans$ do
12     $dis\_tmp \leftarrow 0$
13     foreach $d_i \in \mathbb{D}, \omega_i = 1$ do
14         if $\bar{o}.dis[i] \neq \infty$ then $dis\_tmp \leftarrow dis\_tmp + \bar{o}.dis[i]$
15         else $dis\_tmp \leftarrow dis\_tmp + d_i(\bar{q}, \bar{o})$
16     if $dis\_tmp > r$ then remove $\bar{o}$ from $Ans$
17 return $Ans$
```

another query object $\bar{q}_2$ with the search radius r_2, E can be pruned safely as $E.max < d_i(\bar{q}_2, c^i_j) - r_2$.

Algorithm 3 depicts the pseudo-code of range query in a single metric space built on top of DESIRE. The algorithm takes as inputs a query object $\bar{q}$, the list of clusters LC^i, a distance metric d_i, a search radius $\frac{r}{\sum \omega_i}$, and outputs the result set *Ans*. It recursively calls TreeRangeQ function to search the result in the B$^+$-tree of each cluster (lines 2–4). Specifically, TreeRangeQ function takes as inputs a B$^+$-tree node N, $\bar{q}$, d_i, $\frac{r}{\sum \omega_i}$, and the distance dis_C between $\bar{q}$ and the cluster center. For each entry E in the node N, if N is a leaf node, the function first prunes E via the triangle inequality (line 10). If E cannot be pruned safely, it computes the distance *dis* between $E.o$ and the query object $\bar{q}$ (line 11), and inserts $(E.ptr, dis)$ into the answer set *Ans* if $dis \le \frac{r}{\sum \omega_i}$ (line 12). Here $E.ptr$ points to the entire multi-metric object in RAF for final verification. Otherwise, i.e., N is a non-leaf node, the function searches its sub-trees pointed by each entry E if E cannot be pruned by Lemma 5.2 (lines 14–16).

After searching in each single metric space to find the candidates, we apply the multi-metric range query algorithm to obtain the results, with the pseudo-code shown in Algorithm 4. The algorithm first initializes the result set *Ans* (line 1). For each queried metric space M_i with $\omega_i > 0$, it performs a range query (i.e., Algorithm 3) to obtain the candidate result set Res_i (lines 2–4). For each candidate object $\bar{o} \in Res_i$, it gets the multi-metric object pointed by the RAF pointer (lines 5–6). If the object $\bar{o}$ has not been added to *Ans*, the algorithm adds $\bar{o}$ to *Ans*, and initializes a vector to infinity that stores the distance $dis[j]$ between $\bar{o}$ and $\bar{q}$ under each metric d_j (lines 7–9). Next, $\bar{o}.dis[i]$ is updated to *dis* as it is already computed in Algorithm 3 (line 10). After the algorithm finishes the range queries w.r.t. all the queried metric spaces, it has a candidate set *Ans* and then evaluates every candidate $o \in Ans$ (lines 11–16). For each candidate $\bar{o}$, the algorithm computes the exact distance $d^{\mathbb{W}}(\bar{q}, \bar{o})$ stored in dis_tmp (lines 14–15). If $dis\_tmp > r$, it removes $\bar{o}$ from *Ans* (line 16). After all the candidate objects are evaluated, it returns the final result set *Ans* to complete the search (line 17).

Algorithm 5: *k*NNQ Algorithm

```
Input: a query object $\bar{q}$, a distance metric $d_i$, and an integer $k$
Output: the result set $Ans$ of entries
1  $Ans \leftarrow \varnothing$ // the answer set in ascending order of distance to $\bar{q}$
2  $Queue \leftarrow \varnothing$ // the priority queue to store candidate nodes
3  $dis_k \leftarrow \infty$ // the distance of the $k$-th NN object to $\bar{q}$
4  foreach $C^i_j \in LC^i$ do
5      $N \leftarrow$the B$^+$-tree root of $C^i_j$ // get the node $N$
6      $dis_C \leftarrow d_i(C^i_j.c, \bar{q})$ // the distance from the cluster center to $\bar{q}$
7      push $(N, dis_C, 0)$ into $Que$ // the last item 0 is the lower bound
         distance of $d_i(\bar{q}, E)$ derived in Lemma 5.3)
8  while $Queue \neq \varnothing$ do
9      $(N, dis_C, dis) \leftarrow Queue.pop()$
10     if $dis > dis_k$ then Continue // Lemma 5.3
11     foreach Entry $E$ of $N$ do
12         if $N$ is leaf node then
13             if $|E.key - dis_C| \le dis_k$ then
14                 $dis \leftarrow d(E.o, \bar{q})$
15                 if $dis \le dis_k$ then
16                     $Ans \leftarrow Ans \cup \{(E.ptr, dis)\}$
17                     while $|Ans| > k$ do
18                         remove the last entry in $Ans$
19                     $dis_k \leftarrow$the maximum distance in $Ans$
20         else
21             $dis \leftarrow max(E.min - dis_C, dis_C - E.max)$
22             $N' \leftarrow$the B$^+$-tree node pointed by $E$
23             push $(N', dis_C, dis)$ into $Queue$
24 return $Ans$
```

5.2 Multi-Metric *k* Nearest Neighbour Query

Similar to the multi-metric range query, a naive solution to multi-metric *k* nearest neighbour (MM*k*NN) query is to search *k*NN in each queried metric space and then combine the results. However, it is incorrect as the result objects might not be *k* nearest neighbours to the query object in a single metric space. Taking objects listed in Table 2 as an example. Given a query object *c* and a distance weight vector $\mathbb{W} = (0, 0, 1, 1, 0)$, the result to MM*k*NN query ($k = 1$) is $\{d\}$. Nevertheless, *a* is the nearest to *c* in M_3, while *b* is the closest to *c* in M_4. In this example, the naive solution does not work.

Although the results returned by the naive solution could be incorrect, they provide an upper bound for answering MM*k*NN query. Specifically, we calculate the maximum distance *maxdis* between the query object and the *k*NNs returned by the naive solution in a single space. We can further conduct a multi-metric range query with the radius *maxdis* to find the final MM*k*NN query result. If we search more single metric spaces, a tighter distance bound is obtained with more expensive query cost. In this paper, we only conduct the MM*k*NN query in a random single metric space to improve the efficiency of MM*k*NN query.

When answering *k*NN in a single metric space, we follow the best-first tree traversal strategy [20], which employs a priority queue to iteratively visit the sub-trees and verify the corresponding objects in ascending order of their distances to the query object until all the *k*NNs are found. In addition, in order to avoid unnecessary distance computations, a pruning rule is developed as follows.

Lemma 5.3. *Given a query object $\bar{q}$, a distance metric d_i, a cluster C^i_j, an entry E in the B$^+$-tree of C^i_j, and the distance dis_k between $\bar{q}$ and* *current k-th NN in space M_i, if $max(E.min - d_i(\bar{q}, C^i_j.c), d_i(\bar{q}, C^i_j.c) - E.max) > dis_k$, then E can be pruned safely.*

Proof. For any object $\bar{o}$ in E, $d_i(\bar{o}, \bar{q}) \geq |d_i(\bar{q}, C^i_j.c) - d_i(\bar{o}, C^i_j.c)|$ due to the triangle inequality. As mentioned in Section 4.1, $E.min \leq d_i(\bar{o}, C^i_j.c) \leq E.max$. Then, we derive $d_i(\bar{o}, \bar{q}) \geq E.min - d_i(\bar{q}, C^i_j.c)$ and $d_i(\bar{o}, \bar{q}) \geq d_i(\bar{q}, C^i_j.c) - E.max$. Hence, for any object $\bar{o}$ in E, $d_i(\bar{o}, \bar{q}) \geq max(E.min - d_i(\bar{q}, C^i_j.c), d_i(\bar{q}, C^i_j.c) - E.max)$. If $dis_k < max(E.min - d_i(\bar{q}, C^i_j.c), d_i(\bar{q}, C^i_j.c) - E.max)$, then $dis_k < d_i(\bar{o}, \bar{q})$ for any object $\bar{o}$, and thus, E can be safely pruned. □

Based on Lemma 5.3, we present the k nearest neighbour query algorithm in a single metric space, with the pseudo-code listed in Algorithm 5. It takes as inputs a query object $\bar{q}$, a distance metric d_i, and an integer k, and outputs the result set Ans. Initially, the algorithm initializes an answer set Ans and a priority queue $Queue$ to empty sets, and sets the distance dis_k between $\bar{q}$ and the current k-the NN object to infinity (lines 1–3). Then, all the B$^+$-tree root nodes of different clusters in the queried metric space are inserted into the priority queue (lines 4–7). Here, we set the lower bound distance of $d_i(\bar{q}, E)$ to 0 (derived in Lemma 5.3), as $E.min$ and $E.max$ are not available for the root node. Next, a while-loop is performed until the priority queue is empty (lines 8–23). In each iteration, the top entry (N, dis_C, dis) is popped from the priority queue. If $dis > dis_k$, N can be pruned directly using Lemma 5.3 (lines 9–10). Otherwise, we continue the evaluation. When N is a leaf node, for each leaf entry E of node N, the algorithm first computes the lower bound $|E.key - dis_C|$ of $d_i(E.o, \bar{q})$. If $|E.key - dis_C| \leq dis_k$, it proceeds to compute the exact distance $dis = d_i(E.o, \bar{q})$, and inserts $(E.ptr, dis)$ into Ans if $dis \leq dis_k$ (lines 12–16). In addition, Ans is updated if it contains more than k entries, and dis_k is updated according to the last entry of Ans (lines 17–19). When N is a non-leaf node, the algorithm computes the lower bound of $d_i(\bar{q}, E)$ according to Lemma 5.3 (line 21), and pushes the root node N' pointed by E into $Queue$ (lines 22–23). Finally, the result set Ans is returned (line 24).

After performing Algorithm 5 in a random single metric space, we have retrieved k objects stored in Ans. We then calculate their multi-metric distances to $\bar{q}$ to find the maximum value, i.e., $maxdis = \max_{\bar{o} \in Ans} d^{\mathbb{W}}(\bar{q}, \bar{o})$. Next, a $MMRQ(\bar{q}, \mathbb{W}, maxdis)$ is conducted to find all the mutli-metric objects having their distances to $\bar{q}$ within $maxdis$, and then return the k result objects with the smallest distances to $\bar{q}$. As at least k objects are retrieved by the range query, the correctness of this algorithm is guaranteed.

6 EXPERIMENTS

In this section, we conduct extensive experiments to evaluate the performance of our proposed index DESIRE, including the construction and update performance, the similarity search performance, and the scalability performance.

6.1 Experimental Settings

Datasets. We employ three real-life datasets in our experiments: (i) *Rental*[5] that consists of the price, the number of bedrooms and bathrooms, the location, the publish date, and a brief review for apartments in New York; (ii) *Air*[6] that contains data of prominent air pollutant (including PM2.5, NO, NO_2, CO, SO_2, O_3, Benzene, Toluene, and Xylene) at hourly level of various stations across multiple cities in India; and (iii) *Food*[7] that provides information of food products, including the number of additives, the nutrition facts (i.e., the salt, the energy, the fat, the proteins, and the sugars), the main category, a set of description labels, and the appearance. In addition, we generate a dataset *Synthetic*, which consists of geographical locations in Los Angeles[8], words taken from the Moby project[9], images from Flickr[10], and several randomly generated one dimensional features.

Table 3: Statistics of the datasets used

Dataset	Card. (n)	m	Distance Metrics
Rental	113,176	5	L_1-norm: price & location & date L_2-norm: numbers of rooms Word cosine distance: review
Air	1,150,000	13	L_1-norm: all data
Food	38,757	9	L_1-norm: additives & nutrition & picture Edit distance: category Word cosine distance: description label
Synthetic	200,000	50	L_1-norm: integer & picture L_2-norm: location Edit distance: words

Table 4: Evaluation parameters in our experiments

Parameter	Value
Integer k	1, 2, 4, **8**, 16, 32
search radius r (MMRQ selectivity)	1%, 2%, 4%, **8%**, 16%, 32%
Tuning parameter λ	0.1, **0.2**, 0.3, 0.4
Page Size (KB)	**8**, 16, 32, 64
Number num_m of queried metrics	1, **2**, 3, 4
Cardinality (%)	20, 40, 60, 80, **100**
Weight Ratio	0.1, 0.5, **1**, 5, 10

In our experiments, we use the following metric distance functions: (i) L_1-norm distance used for date, price, the number of bedrooms and bathrooms, prominent air pollutant, the number of additives, the nutrition facts, and the food product pictures and Flickr images (each picture is transformed to a vector of standard MPEG-7 image features); (ii) L_2-norm distance used for geographic locations; (iii) edit distance used for main category of food products and words from Moby; and (iv) word cosine distance[11] used for apartment reviews and food description labels, while each value is represented by 100 embedding features. Following the previous study [19], each distance is normalized by dividing two times the median of all occurring distances, in order to make the impact of each metric comparable. Table 3 lists the dataset statistics, where cardinality and the number of metric spaces are denoted as Card. and m, respectively. Here, a metric space contains a data type and associated distance metric.

Parameters and Performance Metrics. We investigate the performance of our indexes and similarity search algorithms by varying parameters k, r, the page size, the tuning parameter λ, the page size, the number num_m of the queried metric spaces, the cardinality (i.e., the percentage w.r.t. the entire dataset), and the weight ratio for the queried metrics, where k is used for MMkNNQ, r is used for MMRQ, and λ is used to control the number of cluster centers. Here, we use the selectivity of multi-metric range queries to set

[5]https://www.kaggle.com/c/two-sigma-connect-rental-listing-inquiries
[6]https://www.kaggle.com/datasets/rohanrao/air-quality-data-in-india
[7]https://world.openfoodfacts.org/data
[8]https://www.dbs.ifi.lmu.de/cms/
[9]http://icon.shef.ac.uk/Moby/
[10]http://cophir.isti.cnr.it/
[11]https://code.google.com/archive/p/word2vec

Table 5: Construction costs and storage sizes

	Rental				Air				Food				Synthetic			
	Time (s)	compdists ($x10^6$)	PA ($x10^6$)	Stor. (MB)	Time (s)	compdists ($x10^7$)	PA ($x10^7$)	Stor. (MB)	Time (s)	compdists ($x10^6$)	PA ($x10^6$)	Stor. (MB)	Time (s)	compdists ($x10^7$)	PA ($x10^6$)	Stor. (MB)
M^3-tree	**17.02**	25.51	1.85	328.8	**134.0**	97.48	1.51	1205	29.47	33.71	2.59	1095.3	89.3	52.68	5.30	1562
RR^*-tree	27.47	**1.13**	**1.10**	**15.8**	273.1	**2.99**	**1.61**	**416**	**13.08**	**0.70**	**0.36**	**9.4**	**74.6**	**2.00**	**3.85**	**292.9**
PM-tree	43.91	60.38	4.37	298.5	1114	267.55	8.96	1082	33.40	34.70	2.81	372	824	139.68	62.25	1991
DESIRE	**33.58**	**4.84**	**2.60**	**104.1**	**1089**	**20.32**	**8.11**	**713**	**20.35**	**2.41**	**1.57**	**87.4**	**665**	**9.66**	**46.78**	**685.3**

Table 6: Average update costs of deleting one object and inserting one object

	Rental			Air			Food			Synthetic		
	Time ($x10^{-3}$s)	compdists ($x10^4$)	PA ($x10^3$)	Time ($x10^{-3}$s)	compdists ($x10^4$)	PA ($x10^3$)	Time ($x10^{-2}$s)	compdists ($x10^3$)	PA ($x10^3$)	Time ($x10^{-3}$s)	compdists ($x10^4$)	PA ($\times10^3$)
M^3-tree	92.20	93.20	7.63	208.85	68.28	15.26	15.23	125.89	12.22	850.81	382.00	42.00
RR^*-tree	**1.71**	**0.03**	**0.12**	**16.83**	**0.01**	**1.29**	**0.14**	**0.01**	**0.10**	**6.19**	**0.03**	**0.61**
PM-tree	43.80	31.65	3.91	**130.92**	**57.40**	**3.13**	10.91	32.88	9.84	415.04	8.81	36.37
DESIRE	**10.99**	**30.03**	**0.46**	146.05	57.61	4.32	**1.99**	**16.59**	**0.31**	**33.91**	**1.10**	**1.17**

Table 7: Average update costs of deleting a metric space and inserting a metric space

	Rental			Air			Food			Synthetic		
	Time (s)	compdists ($x10^6$)	PA ($x10^5$)	Time (s)	compdists ($x10^8$)	PA ($x10^4$)	Time (s)	compdists ($x10^6$)	PA ($x10^5$)	Time (s)	compdists ($x10^7$)	PA ($\times10^5$)
M^3-tree	52.56	48.00	35.95	**320.7**	10.70	**16.05**	54.72	58.45	47.66	242.92	53.72	**54.17**
RR^*-tree	**50.63**	**1.70**	**21.81**	529.7	**0.44**	31.87	**19.80**	**1.05**	**7.20**	**123.94**	**2.98**	77.57
PM-tree	8.30	12.75	8.92	84.0	2.08	6.91	4.16	3.97	3.22	19.06	2.81	12.54
DESIRE	**6.44**	**0.97**	**5.26**	**78.6**	**0.16**	**6.25**	**2.14**	**0.27**	**1.77**	**11.63**	**0.19**	**9.39**

the search radius r that controls the search region. In particular, the value of r denotes the percentage of objects in the dataset that are result objects of a MMRQ. Table 4 lists the key parameters and their detailed values, where the defaults are shown in bold. Note that, the page sizes for compared methods are 4KB. However, in order to store the entire multi-metric object in the leaf node of M^3-tree [7], we set the default page size for all methods to 8KB. The main performance metrics include the number of page accesses (*PA*), the number of distance computations in single metric spaces (*compdists*), and the running time. Each measurement we report is the average of 100 random queries.

Baselines. We compare our DESIRE against three state-of-the-art multi-metric indexes that belong to two different categories respectively, i.e., two combined methods (i.e., M^3-tree [7] and RR^*-tree [19]) and a separate method PM-tree [19], where one reference point is used for each metric space in RR^*-tree. We implemented the multi-metric indexes in C++. All experiments were conducted on an Intel Core i7-7700 3.6GHz PC with 32GB memory. All source code of the implemented algorithms is publicy available[12].

6.2 Construction and Update Performance

We first evaluate the construction and update performance of our DESIRE and its state-of-the-art competitors. Here, we use the running time, compdists, PA, and the storage size (denoted as Stor. for short) as the performance metrics.

Table 5 lists the construction costs of all the indexes corresponding to four datasets. It is observed that RR^*-tree performs the best, as it stores the entire multi-metric objects together by indexing their distances to the reference points, and then indexes the vector of distances by a single index. However, this nature of RR^*-tree does not allow it to support flexible combinations of metric spaces. Moreover, it also brings the curse of dimensionality to RR^*-tree index. Since distances between each object and reference points are indexed together, the dimensionality of RR^*-tree is equivalent to

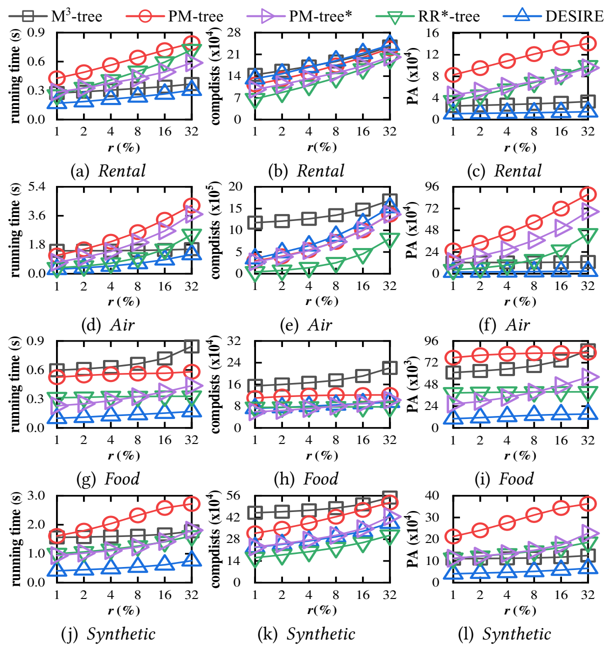

Figure 4: MMRQ Performance vs. Search Radius r

the total number of reference points. In the two separated methods, our proposed DESIRE outperforms PM-tree in all the evaluated aspects, as we employ an effective hyperplane partitioning technique to cluster the objects.

Tables 6 and 7 report the update costs of all the indexes. The results show that RR^*-tree performs the best for updating objects while DESIRE performs the best for updating metric spaces. When inserting/deleting a single object, as RR^*-tree stores distance vectors instead of real multi-metric objects in the index, it only needs to update the distance vectors and verify the found objects, resulting in low number of distance computations. However, when a metric space needs to be inserted/deleted, combined methods including RR^*-tree have to rebuild all the index, incurring high costs. In

[12]https://github.com/ZJU-DAILY/DESIRE

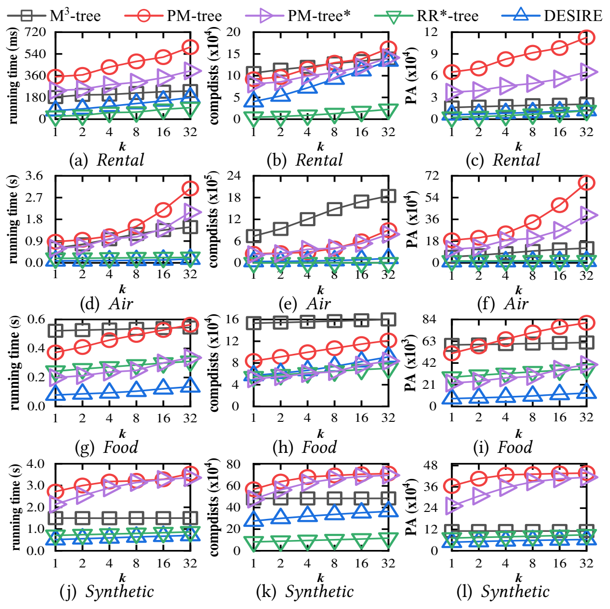

Figure 5: MM*k*NNQ Performance vs. *k*

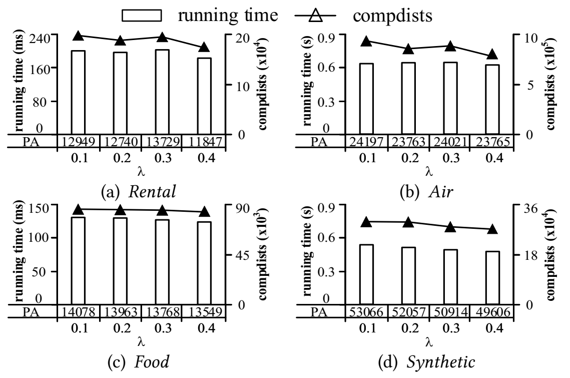

Figure 6: MMRQ Performance vs. λ

contrast, separate methods only need to update the corresponding space. In the two separate methods, our proposed index DESIRE performs better in most cases. The only exception is on *Air* dataset whose objects share the same data type in all spaces. Hence, we would like to claim that DESIRE can well support dynamic scenarios where objects and metric spaces could be changed.

6.3 Similarity Search Performance

We proceed to evaluate the multi-metric similarity search performance of the indexes under three parameters, including (i) the search radius r for MMRQ, (ii) the desired number k for MMkNNQ, and (iii) the tuning parameter λ. In order to demonstrate the effectiveness of our proposed indexing structure, we employ our proposed filtering techniques on PM-tree (denoted as PM-tree*), and compare its results with DESIRE.

Effect of r. Fig. 4 plots the performance of multi-metric range queries (MMRQ) under different selectivity values. In terms of *compdists*, RR*-tree always performs the best, as it uses reference points to achieve more precise distance estimations to avoid unnecessary distance calculations. Nonetheless, as the dimensionality of datasets in our experiments grows from 5 to 50, the distance vector dimensionality of each multi-metric object in RR*-tree also increases, even though each metric space uses only one reference point. Thus, due to the curse of dimensionality, RR*-tree needs to traverse nearly the entire index to find answers, leading to higher CPU and I/O costs than DESIRE. However, as RR*-tree only stores the vector of reference points in the index while the detailed objects are stored in RAF files, RR*-tree has small index structure, and exceeds M^3-tree, PM-tree, and PM-tree*. In addition, PM-tree* incurs lower number of distance computations than our proposed DESIRE on *Rental* and *Air*, while DESIRE has comparable or better performance on *Food* and *Synthetic*. This is because, PM-tree* stores the real data in each node entry to achieve more precise distance estimations. However, as DESIRE only stores partial information of objects and employs efficient B$^+$-tree to manage referenced distances, DESIRE is able to obtain both relatively low number of distance computations and low I/O costs. Thus, DESIRE has the lowest CPU and I/O costs on all the datasets when performing MMRQ.

Effect of k. Fig. 5 shows the performance of multi-metric k nearest neighbour queries (MMkNNQ). Since separate methods (including PM-tree and DESIRE) need additional MMRQ to find the accurate MMkNNQ results, RR*-tree performs the best on *Rental* that only has 5 metric spaces , while DESIRE has the least CPU and I/O costs on all other datasets that are more complex than *Rental*. As CPU cost is the overall performance metric, the above results confirm the efficiency and effectiveness of DESIRE in most cases except when MMkNNQ is performed on low dimensionality datasets.

Effect of λ. Fig. 6 illustrates the MMRQ performance under various λ values. The experimental results of MMkNNQ confirms a similar trend. In the rest of experiments, we only present the MMRQ results due to space limitation and similar performance. As observed, the performance could improve or drop with the growth of λ. This is because, a larger number of centers offer stronger pruning power, but need higher CPU cost. In our experiments, we fix λ to 0.2. Although it cannot achieve the best performance in all the cases, we believe 0.2 is a proper value.

Effect of Page Size. Fig. 7 plots the performance of MMRQ under *Air* and *Food*. Note that, M^3-tree cannot run under the page size of 4KB, and thus, we report the results by varying page size from 8KB to 64KB. The results show that as the page size grows, the I/O cost drops, but both the CPU and distance computation costs first drop and then grow. The reasons are that, as the page size grows, i) on the one hand, more objects are stored in each node, such that fewer nodes are visited and more objects in each node can be filtered, leading to lower I/O cost and fewer distance computations; and ii) on the other hand, fewer nodes can be used to filter objects, decreasing the pruning power which results in more distance computations. As a result, the running time first drops and then increases with the growth of page size. Besides, we also observe that the best page size for separate methods is around 16KB. This is because separate methods only store single metric objects. As page size grows from 8KB to 64KB, the structure of separate methods degenerates from m-way trees to arrays, resulting in worse search efficiency.

Effect of Weight Ratio. In order to evaluate the influence of the weight of each single metric space, we vary the weight ratio of searched spaces from 0.1 to 10, and the results are shown in Fig. 8.

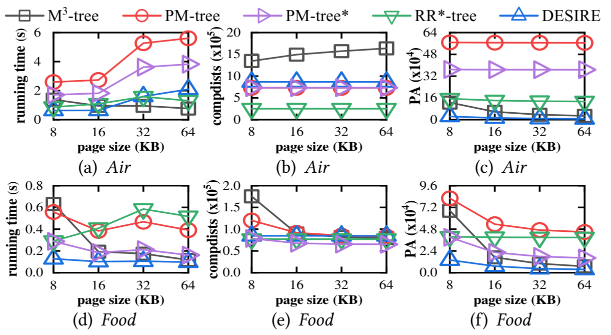

Figure 7: MMRQ Performance vs. Page Size

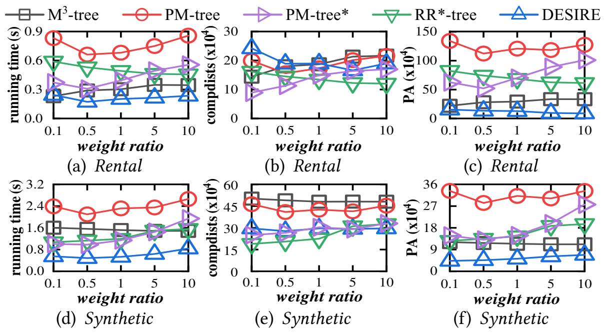

Figure 8: MMRQ Performance vs. Weight Ratio

As observed, DESIRE achieves the best performance compared to other indexes in terms of CPU and I/O costs on all the datasets. Meanwhile, although RR*-tree leverages the reference points to reduce unnecessary distance computations and has the least number of distance computations on *Rental*, its performance fluctuates with the change of the weight ratio. Differently, DESIRE keeps relatively stable, which demonstrates the effectiveness of DESIRE.

6.4 Scalability Analysis

In this subsection, we study the scalability of DESIRE by varying the number of combined metrics and cardinality.

As multi-metric similarity search supports the combination of any number num_m of metrics, we vary the number num_m of metrics, and report the MMRQ results on *Rental* and *Synthetic* in Fig. 9. Once num_m is fixed, we randomly generate a weight vector $\mathbb{W}$ with num_m bits setting to one, and apply the weight vector to all the queries. We have made a few observations. First, our DESIRE performs the best in terms of CPU and I/O costs. This shows that DESIRE is flexible and efficient to support the combination of small number of metrics. However, PM-tree achieves smaller compdists than DESIRE on *Rental*. This is because PM-tree organizes the clusters in the tree structure, which can prune the entire sub-trees to reduce the compdists. Second, the query costs of separate methods (i.e., PM-tree, PM-tree* and DESIRE) increase with the growth of num_m, as more indexes for queried spaces are combined. Differently, the combined indexes (i.e., M^3-tree and RR*-tree) transform all the metrics to a single metric to index objects, and thus, they estimate more accurate distances between objects when more metrics are queried, resulting in less PA and running time.

In order to explore the scalability, we change the cardinality of the datasets from 20% to 100%. Fig. 10 depicts the performance

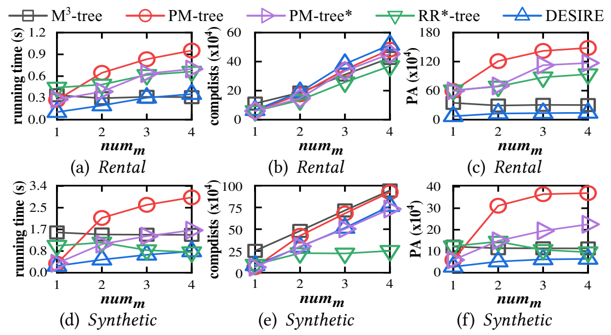

Figure 9: MMRQ Performance vs. num_m

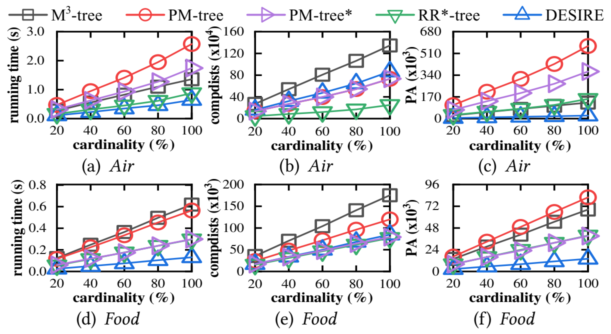

Figure 10: MMRQ Performance vs. Cardinality of Dataset

of MMRQ results w.r.t. cardinality of *Air* and *Food*. As observed, the running time, the number of distance computations, and the number of page accesses increase linearly as the size of the dataset grows, because the search space grows as cardinality ascends. The results mean that the DESIRE framework offers good scalability.

7 CONCLUSIONS

In this paper, we propose DESIRE, an efficient dynamic cluster-based forest index to support similarity search in multi-metric spaces, which is a combination of multiple metric spaces. DESIRE chooses high quality centers to cluster objects into compact regions; employs the B$^+$-tree to effectively indexing distances between centers and multi-metric objects; and leverages efficient and flexible update strategies to support dynamic scenarios. In addition, we develop efficient similarity search algorithms based on filtering techniques. Extensive experiments show that, compared with state-of-the-art multi-metric indexes, our DESIRE supports more efficient and stable similarity search on flexible combinations of metrics, and achieves more efficient updates in terms of both object level and metric space level. The experiments demonstrate the superior efficiency and scalability of DESIRE. Thus, DESIRE has great potential in real applications such as multi-model databases. In the future, we plan to extend DESIRE for distributed environments and new hardware platforms. Also, it is of interest to find appropriate number of centers for different metric spaces, in order to further improve the performance of DESIRE.

ACKNOWLEDGMENTS

This work was supported by the NSFC under Grants No. (62102351, 62025206, and 61972338). Yunjun Gao is the corresponding author of the work.

REFERENCES

[1] Callista Bee, Yuan-Jyue Chen, Melissa Queen, David Ward, Xiaomeng Liu, Lee Organick, Georg Seelig, Karin Strauss, and Luis Ceze. 2021. Molecular-level similarity search brings computing to DNA data storage. *Nature communications* 12, 1 (2021), 1–9.

[2] Tolga Bozkaya and Z. Meral Özsoyoglu. 1997. Distance-Based Indexing for High-Dimensional Metric Spaces. In *SIGMOD*. 357–368.

[3] Tolga Bozkaya and Z. Meral Özsoyoglu. 1999. Indexing Large Metric Spaces for Similarity Search Queries. *ACM Trans. Database Syst.* 24, 3 (1999), 361–404.

[4] Sergey Brin. 1995. Near Neighbor Search in Large Metric Spaces. In *VLDB*. 574–584.

[5] Benjamin Bustos, Daniel A. Keim, and Tobias Schreck. 2005. A pivot-based index structure for combination of feature vectors. In *SAC*. 1180–1184.

[6] Benjamin Bustos, Sebastian Kreft, and Tomás Skopal. 2012. Adapting metric indexes for searching in multi-metric spaces. *Multim. Tools Appl.* 58, 3 (2012), 467–496.

[7] Benjamin Bustos and Tomás Skopal. 2006. Dynamic similarity search in multi-metric spaces. In *Proceedings of the 8th ACM SIGMM International Workshop on Multimedia Information Retrieval*. 137–146.

[8] Cengiz Celik. 2006. *New Approaches to Similarity Searching in Metric Spaces*. Ph.D. Dissertation. University of Maryland, College Park, MD, USA.

[9] Edgar Chávez, Verónica Ludueña, Nora Reyes, and Patricia Roggero. 2016. Faster proximity searching with the distal SAT. *Inf. Syst.* 59 (2016), 15–47.

[10] Edgar Chávez and Gonzalo Navarro. 2000. An Effective Clustering Algorithm to Index High Dimensional Metric Spaces. In *Seventh International Symposium on String Processing and Information Retrieval*. 75–86.

[11] Edgar Chávez and Gonzalo Navarro. 2005. A compact space decomposition for effective metric indexing. *Pattern Recognit. Lett.* 26, 9 (2005), 1363–1376.

[12] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Maproquín. 2001. Proximity searching in metric spaces. *Comput. Surveys* 33, 3 (2001), 273–321.

[13] Lu Chen, Yunjun Gao, Xinhan Li, Christian S. Jensen, and Gang Chen. 2015. Efficient metric indexing for similarity search. In *ICDE*. 591–602.

[14] Lu Chen, Yunjun Gao, Xinhan Li, Christian S. Jensen, and Gang Chen. 2017. Efficient Metric Indexing for Similarity Search and Similarity Joins. *IEEE Trans. Knowl. Data Eng.* 29, 3 (2017), 556–571.

[15] Lu Chen, Yunjun Gao, Baihua Zheng, Christian S. Jensen, Hanyu Yang, and Keyu Yang. 2017. Pivot-based Metric Indexing. *PVLDB* 10, 10 (2017), 1058–1069.

[16] Paolo Ciaccia and Marco Patella. 2000. The M^2-tree: Processing Complex Multi-Feature Queries with Just One Index. In *DELOS*, Vol. 01/W001.

[17] Paolo Ciaccia and Marco Patella. 2002. Searching in metric spaces with user-defined and approximate distances. *ACM Trans. Database Syst.* 27, 4 (2002), 398–437.

[18] Paolo Ciaccia, Marco Patella, and Pavel Zezula. 1997. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *VLDB*. 426–435.

[19] Maximilian Franzke, Tobias Emrich, Andreas Züfle, and Matthias Renz. 2016. Indexing multi-metric data. In *ICDE*. 1122–1133.

[20] Gísli R. Hjaltason and Hanan Samet. 2003. Index-driven similarity search in metric spaces. *ACM Trans. Database Syst.* 28, 4 (2003), 517–580.

[21] Dorit S. Hochbaum and David B. Shmoys. 1985. A Best Possible Heuristic for the k-Center Problem. *Math. Oper. Res.* 10, 2 (1985), 180–184.

[22] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. 2008. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.* 40, 4 (2008), 11:1–11:58.

[23] H. V. Jagadish, Beng Chin Ooi, Kian-Lee Tan, Cui Yu, and Rui Zhang. 2005. iDistance: An adaptive B$^+$-tree based indexing method for nearest neighbor search. *ACM Trans. Database Syst.* 30, 2 (2005), 364–397.

[24] Caetano Traina Jr., Roberto F. Santos Filho, Agma J. M. Traina, Marcos R. Vieira, and Christos Faloutsos. 2007. The Omni-family of all-purpose access methods: a simple and effective way to make similarity search more efficient. *VLDB J.* 16, 4 (2007), 483–505.

[25] Yongjiang Liang and Peixiang Zhao. 2017. Similarity Search in Graph Databases: A Multi-Layered Indexing Approach. In *ICDE*. 783–794.

[26] Wei Lu, Jiajia Hou, Ying Yan, Meihui Zhang, Xiaoyong Du, and Thomas Moscibroda. 2017. MSQL: efficient similarity search in metric spaces using SQL. *VLDB J.* 26, 6 (2017), 829–854.

[27] Luisa Micó, José Oncina, and Enrique Vidal. 1994. A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements. *Pattern Recognit. Lett.* 15, 1 (1994), 9–17.

[28] Gonzalo Navarro. 2002. Searching in metric spaces by spatial approximation. *VLDB J.* 11, 1 (2002), 28–46.

[29] David Novak, Michal Batko, and Pavel Zezula. 2011. Metric Index: An efficient and scalable solution for precise and approximate similarity search. *Inf. Syst.* 36, 4 (2011), 721–733.

[30] Kostas Patroumpas and Dimitrios Skoutas. 2020. Similarity search over enriched geospatial data. In *Proceedings of the Sixth International ACM SIGMOD Workshop on Managing and Mining Enriched Geo-Spatial Data*. 1:1–1:6.

[31] Tomás Skopal, Jaroslav Pokorný, and Václav Snásel. 2004. PM-tree: Pivoting Metric Tree for Similarity Search in Multimedia Databases. In *ADBIS*. 803–815.

[32] Jeffrey K. Uhlmann. 1991. Satisfying General Proximity/Similarity Queries with Metric Trees. *Inf. Process. Lett.* 40, 4 (1991), 175–179.

[33] Enrique Vidal-Ruiz. 1986. An algorithm for finding nearest neighbours in (approximately) constant average time. *Pattern Recognit. Lett.* 4, 3 (1986), 145–157.

[34] Qitong Wang and Themis Palpanas. 2021. Deep Learning Embeddings for Data Series Similarity Search. In *KDD*. 1708–1716.

[35] Peter N. Yianilos. 1993. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. In *SODA*. 311–321.

[36] Albert Yu, Pankaj K. Agarwal, and Jun Yang. 2012. Processing a large number of continuous preference top-k queries. In *SIGMOD*. 397–408.

[37] Albert Yu, Pankaj K. Agarwal, and Jun Yang. 2016. Top-k Preferences in High Dimensions. *IEEE Trans. Knowl. Data Eng.* 28, 2 (2016), 311–325.

[38] Guilherme F. Zabot, Mirela T. Cazzolato, Lucas C. Scabora, Agma J. M. Traina, and Caetano Traina. 2019. Efficient Indexing of Multiple Metric Spaces with Spectra. In *ISM*. 169–176.

[39] Huayi Zhang, Lei Cao, Yizhou Yan, Samuel Madden, and Elke A. Rundensteiner. 2020. Continuously Adaptive Similarity Search. In *SIGMOD*. 2601–2616.

[40] Mingdong Zhu, Derong Shen, Lixin Xu, and Xianfang Wang. 2021. Scalable Multi-grained Cross-modal Similarity Query with Interpretability. *Data Sci. Eng.* 6, 3 (2021), 280–293.

[41] Yifan Zhu, Lu Chen, Yunjun Gao, and Christian S Jensen. 2021. Pivot selection algorithms in metric spaces: a survey and experimental study. *VLDB J.* (2021). https://doi.org/10.1007/s00778-021-00691-4