Aggregate k Nearest Neighbor Queries in Metric Spaces

Xin Ding[1], Yuanliang Zhang[1], Lu Chen[2], Keyu Yang[1], and Yunjun Gao[1](✉)

[1] College of Computer Science, Zhejiang University, Hangzhou, China
{dingxin,yuanlz,kyyang,gaoyj}@zju.edu.cn

[2] Department of Computer Science, Aalborg University, Aalborg, Denmark
luchen@cs.aau.dk

Abstract. Aggregate k nearest neighbor (AkNN) queries are useful in many areas, such as multimedia retrieval and resource allocation, to name but a few. Most of existing works on AkNN query only focus on Euclidean space or specific metric space, which employ properties of particular data to accelerate the query. However, due to the complex data types involved and the needs for flexible similarity criteria seen in real applications, properties of particular data cannot be used for general case. Hence, in this paper, we investigate AkNN search in metric spaces, termed as metric AkNN (MAkNN) search, as metric spaces can support any type of data and flexible similarity criteria as long as satisfying triangle inequality. To efficiently answer MAkNN queries, we develop several pruning techniques and corresponding algorithms based on SPB-tree. Extensive experiments using three real data sets verify the efficiency of our MAkNN algorithms.

Keywords: Metric space · Aggregate k nearest neighbor query Algorithm

1 Introduction

Aggregate k nearest neighbor (AkNN) retrieval is an interesting type of spatial queries, which finds k objects similar to all the specified query objects using an aggregate similarity criterion. It is useful in a variety of applications, such as resource allocation, recommender systems, etc. Here, we give two examples below.

Resource Allocation. Consider the carpooling, i.e., carpoolers want to take the same taxi to save money. An AkNN query can be utilized to help find candidate taxis for the carpoolers with smallest aggregate distances. Here, with the objective to save time, the aggregate distance summarizes all the distances from the taxi to each carpooler.

Y. Cai et al. (Eds.): APWeb-WAIM 2018, LNCS 10988, pp. 317–333, 2018.
https://doi.org/10.1007/978-3-319-96893-3_24

Table 1. Symbols and description

Notation	Description
q	A query object
Q or O	The set of objects in metric spaces
P	The set/table of pivots
o or p	An object in O, a pivot in P
$\|Q\|, \|O\|, \|P\|$	The cardinality of Q, O or P
$d()$	The distance function for the generic metric space
$D()$	The L_∞-norm metric for the mapped vector space
$d_{agg}(Q, o)$	The aggregate distance between Q and o in generic metric space
$\phi(o)$	The data point for o in the mapped vector space
$SFC(o)$	The space-filling curve value of an object o
$MAkNN(Q, O, k)$	The result set of an MAkNN query w.r.t. the query set Q and the object set O
$curAND_k$	The current k-th nearest neighbor distance

Recommender Systems. An image recommender system can generate personalized recommendations (i.e., the images that the user may be interested in) based on the images the user already reviewed. Here, the aggregate distance could be the minimum distance between the image to be recommended and the images reviewed.

Considering the wide range of data types in the above application scenarios, e.g., taxis and images, a generic model is desirable that is capable of accommodating not just a single type, but a wide spectrum. In addition, the distance metrics for comparing the similarity of objects, such as road network distance used for taxis and L_p-norm used for images, are not restricted to the Euclidean distance (i.e., L_2-norm). To accommodate a wide range of similarity notions, we investigate AkNN retrieval in metric spaces, termed as metric AkNN (MAkNN) search, where *no* detailed representations of objects are required and where any similarity notion that satisfies the *triangle inequality* can be accommodated.

Most of existing works on AkNN search focus on Euclidean space or particular metric space (e.g., road network, graph), where properties of particular data (e.g., geometric property for Euclidean space) are used to improve the query efficiency. However, these properties cannot be used for the general case, i.e., these approaches cannot answer MAkNN search efficiently. Motivated by this, we develop several pruning lemmas based on the triangle inequality property of metric spaces, and present corresponding algorithms. To sum up, the key contributions of this paper are as follows:

- We develop several pruning lemmas based on SPB-tree for sum, min, and max aggregate functions to accelerate the search.
- We present an efficient algorithm designed for MAkNN search by integrating the designed pruning lemmas.
- We conduct extensive experiments using three real data sets to verify the efficiency of our proposed algorithms, compared with a baseline algorithm extended from the state-of-the art MAkNN framework.

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 describes the SPB-tree. Section 4 defines MAkNN search and presents corresponding algorithms. Considerable experimental results and findings are reported in Sect. 5. Finally, Sect. 6 concludes the paper with some directions for future work.

2 Related Work

In this section, we survey existing work on metric access methods, and AkNN search algorithms. Table 1 summarizes the notations frequently used throughout this paper.

2.1 Metric Access Methods

Two broad categories of metric access methods (MAMs) exist, namely, compact partitioning methods and pivot-based approaches, to accelerate query processing in metric spaces. Compact partitioning methods partition the space as compact as possible, and try to prune unqualified regions during search. Many indexes, e.g., BST [1], GHT [2], GANT [3], SAT [4], M-tree [5] family, D-Index [6], LC [7], BP [8] exist. Pivot-based methods store pre-computed distances from every object in the database to a set of pivots, and then utilize these distances and the triangle inequality to prune objects during search. Many indexes, e.g., LAESA [9], EP [10], BKT [11], FQT [12], MVPT [13], the Omni-family [14] exist.

Although pivot-based methods clearly outperform compact partitioning approaches in terms of the number of distance computations (i.e., CPU cost) [14–17], they generally have high I/O cost because objects are not well clustered on disk. Recently, hybrid methods that combine compact partitioning with the use of pivots have appeared in the literature. PM-tree [18] uses cut-regions defined by pivots to accelerate query processing on the M-tree. M-Index [19] generalizes the iDistance technique for metric spaces, which compacts the objects by using pre-computed distances to their closest pivots. SPB-tree [20] utilizes the two mapping phase to further improve the efficiency. Hence, in this paper, we use SPB-tree as the underlying index.

2.2 AkNN Search Algorithm

Aggregate k nearest neighbor (AkNN) retrieval generalizes kNN search, which considers multiple query objects. Consequently, the distances from each query object to an object must be aggregated (min, max or sum) according to an optimization goal, in order to offer the similarity measure employed to rank answered objects. Many works [21,22] only focus on AkNN in Euclidean space, where geometric properties are used to accelerate the search. In addition, AkNN in particular metric space (e.g., road network [23], graphs [24], trajectories [25]) are also investigated. However, all these approaches cannot solve our MAkNN search problem, due to the general case we focus on.

Razente et al. [26] study circumscription-constrained aggregate similarity (CCAS) queries in metric spaces, where the region circumscribed by the query objects limits the search space. However, algorithms developed for CCAS queries can not be efficiently extended to solve MAkNN search. This is because, they utilize the circumscription-constrained region to significantly prune search space. Without the circumscription constraint, they have to scan the whole object set to obtain the final query result, which is costly. In addition, Ranzente et al. [27] also develop a framework for MAkNN search that can be adaptive to all kinds of MAMs. Flowing the framework of [27], we develop a baseline algorithm (BL) based on the-state-of-the art MAM SPB-tree.

3 The SPB-tree

In this section, we describe the SPB-tree used as the underlying index.

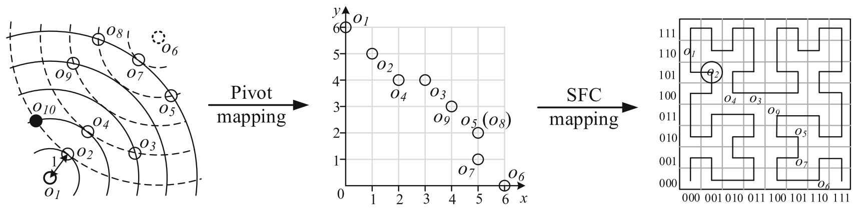


Fig. 1. Pivot mapping and space-filling curve mapping

3.1 Construction Framework

The construction framework of a SPB-tree is based on a two-stage mapping. The first stage maps the objects in a metric space to data points in a vector space using well-chosen pivots. The vector space offers more freedom than the metric space when designing search approaches, since it is possible to utilize the geometric information that is unavailable in the metric space. The second stage uses the space-filling curve (SFC) to map the data points in the vector space into integers in an one-dimensional space. Finally, a B$^+$-tree with MBB information is employed to index the resulting integers.

Pivot Mapping. Given a pivot set $P = \{p_1, p_2, \ldots, p_n\}$, a metric space (M, d) can be mapped to a vector space (R^n, L_∞). Specifically, an object o in the metric space is represented as a point $\phi(o) = \langle d(o, p_1), d(o, p_2), \ldots, d(o, p_n)\rangle$ in the vector space. For instance, consider the example in Fig. 1, where $O = \{o_1, o_2, \ldots, o_9\}$ and L_2-*norm* is used. If $P = \{o_1, o_6\}$, O can be mapped to a two-dimensional vector space, in which the x-axis denotes $d(o_i, o_1)$ and the y-axis represents $d(o_i, o_6)$, $1 \le i \le 9$.

Given objects o_i, o_j, and p in a metric space, $d(o_i, o_j) \geq |d(o_i, p) - d(o_j, p)|$ according to the triangle inequality. Hence, for a pivot set P, $d(o_i, o_j) \geq \max\{|d(o_i, p_i) - d(o_j, p_i)| \mid p_i \in P\} = D(\phi(o_i), \phi(o_j))$, in which $D(\ )$ is the L_∞-norm. Consequently, we can conclude that the distance in the mapped vector space is a *lower bound* on that in the metric space. For example, in Fig. 1, $d(o_2, o_3) > D(\phi(o_2), \phi(o_3)) = 2$.

Space-Filling Curve Mapping. Given a vector $\phi(o)$ after pivot mapping and assume that the range of $d(\ )$ in the metric space is *discrete* integers (e.g., edit distance), SFC can directly map $\phi(o)$ to an integer $SFC(\phi(o))$. Consider the SFC mapping examples in Fig. 1, where SFC value $SFC(\phi(o_2)) = 18$ for the Hilbert curve. As a default, we use the Hilbert curve for SPB-tree. If the range of $d(\ )$ in the metric space is continuous real numbers, we can partition the range of $d(\ )$ into discrete integers.

3.2 Indexing Structure

An SPB-tree used to index an object set in a generic metric space contains three parts, i.e., the pivot table, the B$^+$-tree, and the random access file (RAF). Figure 2 shows an SPB-tree example to index the object set $O = \{o_1, \ldots, o_9\}$ in Fig. 1. A pivot table stores selected objects (e.g., o_1 and o_6) to map a metric space into a vector space.

A B$^+$-tree is employed to index the SFC values of objects after a pivot mapping. Each leaf entry in the leaf node (e.g., N_3, N_4, N_5, and N_6) of the B$^+$-tree records (1) the SFC value *key*, and (2) the pointer *ptr* to a real object, which is the address of the actual object kept in the RAF. For example, in Fig. 2, the leaf entry E_7 associated with the object o_2 records the Hilbert value 18 and the storage address 0 of o_2. Each non-leaf entry in the root or intermediate node (e.g., N_0, N_1, and N_2) of the B$^+$-tree records (1) the minimum SFC value *key* in its subtree, (2) the pointer *ptr* to the root node of its subtree, and (3) the SFC values *min* and *max* for $\langle L_1, L_2, \ldots, L_{|P|}\rangle$ and $\langle U_1, U_2, \ldots, U_{|P|}\rangle$, to represent the MBB $M(= \{[L_i, U_i] | i \in [1, |P|]\})$ of the root node N of its subtree. Specifically, an MBB M denotes the axis aligned *minimum bounding box* to contain all $\phi(o)$ with $SFC(\phi(o)) \in N$, and thus, L_i and U_i represent the minimum and maximum

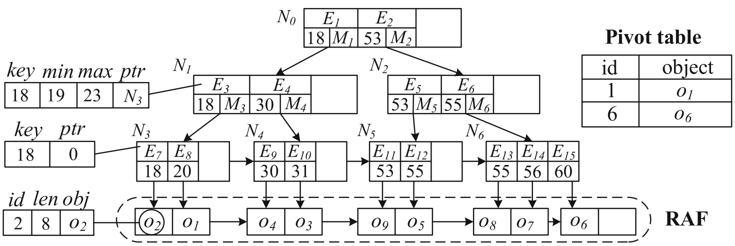


Fig. 2. Example of an SPB-tree

values of $\phi(o)$ on dimension i. For instance, the non-leaf entry E_3 uses *min* ($= SFC(\langle 0,5\rangle) = 19$) and *max* ($= SFC(\langle 1,6\rangle) = 23$) to represent the M_3 (= {[0, 1], [5, 6]}) of N_3.

RAF is sorted to store the objects in ascending order of SFC values as they appear in the B$^+$-tree. Each RAF entry records (1) an object identifier *id*, (2) the length *len* of the object, and (3) the real object *obj*. In Fig. 2, the RAF entry associated with an object o_2 records the object identifier 2, the object length 8, and the real object o_2.

4 Metric Aggregate *k* Nearest Neighbor Search

In this section, we first formalize A*k*NN retrieval in metric spaces, and then propose an efficient algorithm for processing metric A*k*NN queries based on the SPB-tree.

4.1 Problem Definition

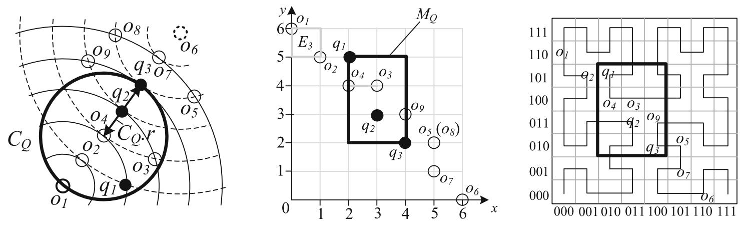
A metric space is a tuple (M, d), in which M is the domain of objects and d is a distance function which defines the similarity between the objects in M. In particular, the distance function d has four properties: (1) *symmetry*: $d(q, o) = d(o, q)$, (2) *non-negativity*: $d(q, o) \geq 0$, (3) *identity*: $d(q, o) = 0$ iff $q = o$, and (4) *triangle inequality*: $d(q, o) \leq d(q, p) + d(p, o)$. Based on the properties of the metric space, A*k*NN queries in metric spaces have been investigated.

Definition 1. (**MA*k*NN Query**). *Given a query object set Q, an object set O, and an integer k , an MAkNN query finds k objects in O with the smallest aggregate distances* $d_{agg}(Q, o)$*, i.e., MAkNN(Q, O, k)* $= \{o_i | o_i \in O \wedge 1 \leq i \leq k \wedge \forall o_j (\neq o_i) \in O, d_{agg}(Q, o_j) \geq d_{agg}(Q, o_i)\}$. *In particular,* $d_{agg}(Q, o)$ *can be computed as* $f(d(q_1, o), d(q_2, o), \ldots, d(q_{|Q|}, o))$*, in which the aggregate function f might be sum, min, or max.*

Consider two English word sets $Q =$ {“defoliate”, “defoliates”} and $O =$ {“citrate”, “defoliation”, “defoliating”, “defoliated”}, for which the edit distance is the similarity measurement. Suppose $k = 2$, an MA*k*NN query *MAkNN* (Q, O, 2) finds the two words in O having the smallest aggregate distances from Q. If f is *sum* function, the query result is {“defoliated”, “defoliation”}; if f is *min* function, the query result is {“defoliated”, “defoliation”}; and if f is *max* function, the query result is {“defoliated”, “defoliating”}. It is worth noting that *MAkNN*(Q, O, k) may be not unique due to the distance tie. Nonetheless, the target of our presented algorithms is to find one possible instance.

4.2 MA*k*NN Query Processing

MA*k*NN search generalizes the form of M*k*NN queries, in which there are multiple (instead of one) query objects. Consider a running example of MA*k*NN



(a) Metric space (b) Vector space (c) Hilbert curve

Fig. 3. Illustration of $MAkNN(Q, O, k)$

retrieval depicted in Fig. 3, where $Q = \{q_i | 1 \leq i \leq 3\}$ and $O = \{o_j | 1 \leq j \leq 9\}$. Assume that $k = 2$ and L_2-norm is utilized, the result of $MAkNN(Q, O, 2)$ is $\{o_4, o_3\}$ if *sum* function is used to compute the aggregate distance; the query result is $\{o_4, o_7\}$ if *min* function is used; and the query result is $\{o_4, o_3\}$ if *max* function is used. To solve MAkNN search, a simple method BL is to use SPB-tree and follow the framework [27] developed for MAkNN retrieval. In particular, BL traverses the B$^+$-tree entries in ascending order of their minimum aggregate distances to Q in the mapped vector space. As discussed in Sect. 3, the distance in the mapped vector space is the lower bound distance of the original metric space, we develop Lemma 1 below for MAkNN search, to avoid unnecessary verifications of B$^+$-tree entries.

Lemma 1. *Given a query set Q and a B$^+$-tree entry E, E can be safely pruned if $MIND_{agg}(Q, E) \geq curAND_k$, where $MIND_{agg}(Q, E)$ denotes the minimum aggregate distance between E and Q in the mapped vector space, and $curAND_k$ represents the current k-th aggregate NN distance from Q.*

Proof. Since the aggregate function is monotonically increasing, the aggregate distance in the mapped vector space is still the lower bound distance of that in the original metric space. Then, we can get that $mind_{agg}(E, Q) \geq MIND_{agg}(E, Q)$, with $mind_{agg}(E, Q)$ denoting the minimum aggregate distance between E and Q in the original metric space. If $MIND_{agg}(E, Q) \geq curAND_k$, then for each o ($\in E$), $d_{agg}(o, Q) \geq mind_{agg}(E, Q) \geq curAND_k$. Consequently, E can be discarded safely. □

Note that, $curAND_k$ used in Lemma 1 is obtained and updated during MAkNN search. In particular, after computing the aggregate distance of an object, we can update immediately the result set and $curAND_k$ if necessary. Consider the example depicted in Fig. 3 with the corresponding SPB-tree in Fig. 2. Assume that $curAND_k = 1$ and *min* function is used, E_3 and E_6 can be safely pruned as $MIND_{agg}(E_3, M_Q) = MIND_{agg}(E_6, M_Q) = curAND_k$.

Since $MIND_{agg}(E, Q)$ is computed as $f(MIND(E, q_1), MIND(E, q_2), \ldots, MIND(E, q_{|Q|}))$, it is costly (because it needs $|Q|$ computations of $MIND$).

Motivated by this, we build MBB M_Q $(= \{[L_{Qi}, U_{Qi}] \mid 1 \leq i \leq |P|\})$ for Q in the mapped vector space, to reduce $MIND_{agg}(E, Q)$ computation cost. Back to the running example illustrated in Fig. 3, the thick black rectangle in Fig. 3(b) represents MBB M_Q $(= \{[2, 4], [2, 5]\})$ for Q in the mapped vector space using $P = \{o_1, o_6\}$. Let $MIND_i(E, q_t)$ be the minimum distance between E and q_t $(\in Q)$ on dimension i $(1 \leq i \leq |P|)$, and $MIND_i(E, Q)$ be the minimum aggregate distance between E and Q on dimension i.

$$
\begin{aligned}
& MIND_{agg}(E, Q) \\
& = f(MIND(E, q_1), \ldots, MIND(E, q_{|Q|})) \\
& = f(max\{MIND_i(E, q_1) | 1 \leq i \leq |P|\}, \\
& \quad \ldots, max\{MIND_i(E, q_{|Q|}) | 1 \leq i \leq |P|\}) \\
& \geq max\{f(MIND_i(E, q_1), \ldots, \\
& \quad MIND_i(E, q_{|Q|})) | 1 \leq i \leq |P|\} \\
& = max\{MIND_i(E, Q) | 1 \leq i \leq |P|\}
\end{aligned} \tag{1}
$$

According to Eq. (1), the lower bound distance of $MIND_{agg}(E, Q)$, termed as $EMIND_{agg}(E, Q)$, can be computed as $max\{MIND_i(E, Q) | 1 \leq i \leq |P|\}$. To obtain $EMIND_{agg}(E, Q)$, we only need to compute $MIND_i(E, Q)$ on each dimension i, with the detailed computations stated below for *sum*, *min*, or *max* function, respectively.

***Sum* function.** If $L_{Ei} \geq U_{Qi}$ (as shown in Fig. 4(a)), $MIND_i(E, Q) = \sum_{1 \leq t \leq |Q|} MIND_i(E, q_t) = \sum_{1 \leq t \leq |Q|} (L_{Ei} - d(q_t, p_i)) = |Q| \times L_{Ei} - \sum_{1 \leq t \leq |Q|} d(q_t, p_i)$. If $U_{Ei} \leq L_{Qi}$ (as depicted in Fig. 4(b)), then $MIND_i(E, Q) = \sum_{1 \leq t \leq |Q|} MIND_i(E, q_t) = \sum_{1 \leq t \leq |Q|} (d(q_t, p_i) - U_{Ei}) = \sum_{1 \leq t \leq |Q|} d(q_t, p_i) - |Q| \times U_{Ei}$. Otherwise, i.e., M_Q and M_E are intersected on dimension i, $MIND_i(E, Q)$ is estimated as 0.

Note that, $\sum_{1 \leq t \leq |Q|} d(q_t, p_i)$ used in $MIND_i(E, Q)$ computation for *sum* function can be obtained and stored for reuse when building M_Q. Hence, for *sum* function, the computational cost of $EMIND_{agg}(E, Q)$ is O(1), which is much smaller than O(|Q|) of $MIND_{agg}(E, Q)$ computation. For example, in Fig. 3, and assume that *sum* function is used on dimension x, as $U_{E_3 x} < L_{Qx}$, $MIND_x(E_3, Q) = d(q_1, o_1) + d(q_2, o_1) + d(q_3, o_1) - 3 \times U_{E_3 x} = 6$. Thus, we can get that $EMIND_{agg}(E_3, Q) = \max\{MIND_x(E_3, Q), MIND_y(E_3, Q)\} = 6$, which is a tight lower bound of $MIND(E_3, Q)$ $(= 6)$.

***Min* function.** If $L_{Ei} \geq U_{Qi}$ (as shown in Fig. 4(a)), then $MIND_i(E, Q) = \min_{1 \leq t \leq |Q|} MIND_i(E, q_t) = L_{Ei} - U_{Qi}$. If $U_{Ei} \leq L_{Qi}$ (as depicted in Fig. 4(b)), then $MIND_i(E, Q) = \min_{1 \leq t \leq |Q|} MIND_i(E, q_t) = L_{Qi} - U_{Ei}$. Otherwise, i.e., M_Q and M_E are crossed on dimension i, $MIND_i(E, Q)$ is estimated as 0.

Similarity, the $EMIND_{agg}(E, Q)$ computational cost is also reduced to O(1) for *min* function. Back to the example shown in Fig. 3 and suppose that *min* function is used, since $U_{E_3 x} < L_{Qx}$ on dimension x, $MIND_x(E_3, Q) = L_{Qx} -$

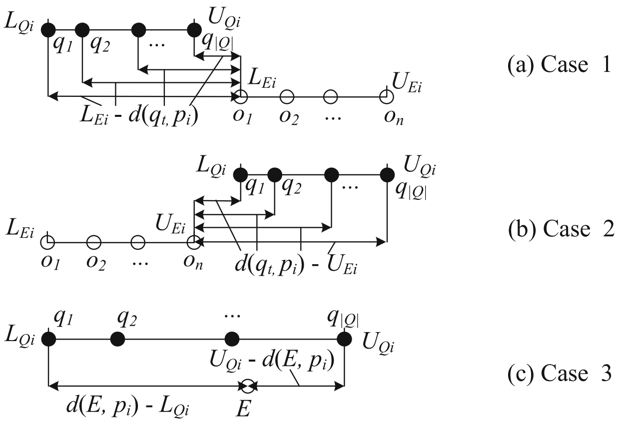


Fig. 4. $MIND_i(Q, E)$ computation

$U_{E_3x} = 1$. Hence, we can get that $EMIND_{agg}(E_3, Q) = \max\{MIND_x(E_3, Q), MIND_y(E_3, Q)\} = 1$, which is a tight lower bound of $MIND(E_3, Q)$ $(= 1)$.

***Max* function.** If $L_{Ei} \geq U_{Qi}$ (as shown in Fig. 4(a)), then $MIND_i(E, Q) = \max_{1 \leq t \leq |Q|} MIND_i(E, q_t) = L_{Ei} - L_{Qi}$. If $U_{Ei} \leq L_{Qi}$ (as depicted in Fig. 4(b)), then $MIND_i(E, Q) = \max_{1 \leq t \leq |Q|} MIND_i(E, q_t) = U_{Qi} - U_{Ei}$. Otherwise, i.e., M_Q and M_E are intersected on dimension i, $MIND_i(E, Q)$ is estimated as 0. Note that, for the case when M_E is intersected with M_Q, if E is a leaf entry (as illustrated in Fig. 4(c)), then $MIND_i(E, Q) = \max\{d(E, p_i) - L_{Qi}, U_{Qi} - d(E, p_i)\}$.

For *max* function, the $EMIND_{agg}(E, Q)$ computational cost is also reduced to O(1). Back to the example depicted in Fig. 3 and assume that *max* function is used, on dimension x, as $U_{E_3x} < L_{Qx}$, $MIND_x(E_3, Q) = U_{Qx} - U_{E_3x} = 3$; on dimension y, $MIND_y(E_3, Q) = d(E_{10}, o_6) - L_{Qy} = 2$. Thus, we can get that $EMIND_{agg}(E_3, Q) = \max\{MIND_x(E_3, Q), MIND_y(E_3, Q)\} = 3$, which is a tight lower bound of $MIND(E_3, Q)$ $(= 3)$. For object o_3, on dimension y, since $L_{Qy} < d(o_3, o_6) < U_{Qy}$, $MIND_y(o_3, Q) = \max\{d(o_3, o_6) - L_{Qi}, U_{Qi} - d(o_3, o_6)\} = 2$. therefore, we can get that $EMIND_{agg}(o_3, Q) = \max\{MIND_x(o_3, Q)), MIND_y(o_3, Q)\} = 2$, which is also a tight lower bound of $MIND(o_3, Q)$ $(= 2)$.

Based on $EMIND_{agg}(E, Q)$ derived, we develop a lemma to avoid unnecessary $MIND_{agg}(Q, E)$ computations.

Lemma 2. *Given a query set Q and a B^+-tree entry E, E can be safely pruned if $EMIND_{agg}(Q, E) \geq curAND_k$.*

Proof. Since $EMIND_{agg}(Q, E) \leq MIND_{agg}(Q, E)$, if $EMIND_{agg}(Q, E) \geq curAND_k$, then $MIND_{agg}(Q, E) \geq curAND_k$. Hence, E can be safely pruned due to Lemma 1, which completes the proof. □

Consider the example depicted in Fig. 3 with the corresponding SPB-tree in Fig. 2. Assume that *sum* function is used and $curAND_k = 5$, E_3 can be safely discarded due to $EMIND_{agg}(E_3, M_Q) > curAND_k$.

Lemma 2 utilizes MBB to reduce the computational cost in the mapped vector space. In order to further reduce the computational cost of the aggregate distance $d_{agg}(Q, o)$ between the object o and the query set Q, we can also build a minimum bounding circle (MBC) for Q in original metric space. The MBC C_Q is centered at $C_Q.o$ with the radius $C_Q.r$ equaling to the maximum distance $d(q, C_Q.o)$ ($q \in Q$). Consider the example illustrated in Fig. 3(a), the thick black circle, centered at object o_4 with the radius $C_Q.r = d(o_4, q_3)$, denotes the MBC for Q. With the assistant of MBC, we can get the lower bound $ed_{agg}(Q, o)$ of $d_{agg}(Q, o)$, with the detailed derivation stated as follows for *sum*, *min*, and *max* function, respectively.

***Sum* function.** According to the triangle inequality,

$$\begin{aligned} d_{agg}(Q,o) &= \sum_{1 \le t \le |Q|} d(o, q_t) \\ &\ge \sum_{1 \le t \le |Q|} |d(o, C_Q.o) - d(C_Q.o, q_t)| \\ &\ge |\sum_{1 \le t \le |Q|} (d(o, C_Q.o) - d(C_Q.o, q_t))| \\ &= |d(o, C_Q.o) \times |Q| - \sum_{1 \le t \le |Q|} d(C_Q.o, q_t)| \end{aligned} \tag{2}$$

Hence, $ed_{agg}(Q, o)$ can be computed as $|d(o, C_Q.o) \times |Q| - \sum_{1 \le t \le |Q|} d(C_Q.o, q_t)|$ for *sum* function. Note that, $\sum_{1 \le t \le |Q|} d(C_Q.o, q_t)$ can be computed and stored for reuse when building MBC C_Q. For example, in Fig. 3(a), suppose that *sum* function is used, $ed_{agg}(Q, o_6) = 3 \times d(o_6, o_4) - \sum_{1 \le t \le 3} d(o_4, q_t) = 9 - d(o_4, q_1) = 7.2$, which is a lower bound value of $d_{agg}(Q, o_6)$ $(= 5 + d(q_1, o_6) = 10)$.

***Min* function.** Based on the triangle inequality,

$$\begin{aligned} d_{agg}(Q,o) &= \min_{1 \le t \le |Q|} d(o, q_t) \\ &\ge \min_{1 \le t \le |Q|} |d(o, C_Q.o) - d(C_Q.o, q_t)| \\ &\ge \min_{1 \le t \le |Q|} (d(o, C_Q.o) - d(C_Q.o, q_t)) \\ &= d(o, C_Q.o) - \max_{1 \le t \le |Q|} d(C_Q.o, q_t) \\ &= d(o, C_Q.o) - C_Q.r \end{aligned} \tag{3}$$

Thus, $ed_{agg}(Q, o)$ can be computed as $d(o, C_Q.o) - C_Q.r$ for *max* function. Back to the example shown in Fig. 3(a), and assume that *min* function is used, $ed_{agg}(Q, o_6) = d(o_6, o_4) - C_Q.r = 2$, which is a tight lower bound of $d_{agg}(Q, o_6)$ $(= 2)$.

***Max* function.** According to the triangle inequality,

$$\begin{aligned} d_{agg}(Q, o) &= \max_{1 \le t \le |Q|} d(o, q_t) \\ &\ge \max_{1 \le t \le |Q|} |d(o, C_Q.o) - d(C_Q.o, q_t)| \\ &= max\{\max_{1 \le t \le |Q|} d(C_Q.o, q_t) - d(o, C_Q.o), \\ &\quad d(o, C_Q.o) - \min_{1 \le t \le |Q|} d(C_Q.o, q_t)\} \\ &= max\{C_Q.r - d(o, C_Q.o), d(o, C_Q.o) - \\ &\quad min\{d(C_Q.o, q_t) | 1 \le t \le |Q|\}\} \end{aligned} \tag{4}$$

Therefore, $ed_{agg}(Q, o)$ can be computed as $\max\{d(o, C_Q.o) - C_Q.r, d(o, C_Q.o) - min\{d(C_Q.o, q_t) | 1 \le t \le |Q|\}\}$ for *min* function. Note that, $min\{d(C_Q.o, q_t) | 1 \le t \le |Q|\}$ can be computed and stored for reuse when building C_Q. Back to the example depicted in Fig. 3(a), and suppose that *max* function is used, $ed_{agg}(Q, o_6) = d(o_6, o_4) - d(o_4, q_2) = 3$, which is a lower bound value of $d_{agg}(Q, o_6) (= d(q_1, o_6) = 5)$.

According to Eqs. 2–(4), it only needs one distance computation for $ed_{agg}(Q, o)$ calculation, instead of $|Q|$ distance computations for $d_{agg}(Q, o)$ calculation, which reduces significantly the computational cost. Thus, we develop a new lemma based on $ed_{agg}(Q, o)$ derived, to avoid unnecessary computations of $d_{agg}(Q, o)$.

Lemma 3. *Given a query set Q and an object o, o can be safely pruned if $ed_{agg}(Q, o) \ge curAND_k$.*

Proof. As $ed_{agg}(Q, o) \le d_{agg}(Q, o)$, $d_{agg}(Q, o) \ge curAND_k$ if $ed_{agg}(Q, o) \ge curAND_k$. Hence, o can be safely pruned due to the definition of the aggregate kNN query, which completes the proof. □

Consider the example shown in Fig. 3 with the corresponding SPB-tree in Fig. 2. Assume that *max* function is used and $curAND_k = 5$, object o_6 can be safely discarded due to $EMIND_{agg}(o_6, M_Q) > curAND_k$, without any further verification.

Algorithm 1 Aggregate kNN Algorithm (AkNNA)

Input: a query set Q, an integer k, an object set O indexed by a SPB-tree
Output: the result set $MAkNN(Q, O, k)$ of an aggregate kNN query
1: $curAND_k = \infty$, $C_Q = H = \varnothing$ // H stores the intermediate and leaf entries of B^+-tree in ascending order of $MIND_{agg}(Q, E)$
2: compute $\phi(q)$ for each object q in Q using P and obtain M_Q
3: push the root entries of B^+-tree onto H
4: **while** $H \neq \varnothing$ **do**
5:   de-heap the top entry E from H
6:   **if** $MIND_{agg}(M_Q, E) \geq curAND_k$ **then** break // Lemma 1
7:   **if** E is a non-leaf entry **then**
8:     **for** each sub entry $e \in E$ **do**
9:       **if** $EMIND_{agg}(M_Q, e) < curAND_k$ **then** // Lemma 2
10:         **if** $MIND_{agg}(M_Q, e) < curAND_k$ **then** // Lemma 1
11:           push e onto H
12:   **else** // E is a leaf entry
13:     **if** $C_Q \neq \varnothing$ **then**
14:       **if** $ed_{agg}(Q, e.ptr) \geq curAND_k$ **then** continue // Lemma 3
15:     **if** $d_{agg}(Q, e.ptr) < curAND_k$ **then**
16:       insert $e.ptr$ into $MAkNN(Q, O, k)$
17:       update $curAND_k$ and C_Q if necessary
18: return $MAkNN(Q, O, k)$

To achieve the strongest pruning power of Lemma 3, i.e., the lower bound $ed_{agg}(Q, o)$ must approach to $d_{agg}(Q, o)$ as much as possible, we need to tight the MBC. In other words, we need to choose an MBC center to obtain the minimal MBC radius. A simple way to obtain the optimal center is to perform an $MAkNN(Q, O, 1)$ query using max function. However, it is costly to perform an additional aggregate NN query. Therefore, we can update the center of MBC using the object o ($\in O$) during MAkNN search when verifying whether o is contained in the final result.

Based on Lemmas 1 to 3, we present an efficient *Aggregate kNN Algorithm* (AkNNA), with the pseudo-code depicted in Algorithm 1. To begin with, AkNNA sets $curAND_k$ to infinity, and initializes the MBC C_Q and min-heap H to empty. Then, it computes $\phi(q)$ for each $q \in Q$ using P, and obtains the MBB M_Q in the mapped vector space. Next, the algorithm pushes the root entries of a B^+-tree into H. In the sequel, a while-loop is performed until H is empty (lines 4–17). In every while-loop, AkNNA de-heaps the top entry E from H, and stops searching if $MIND_{agg}(Q, E)$ is no smaller than $curAND_k$ by Lemma 1. If E is a non-leaf entry, the algorithm pushes all the qualified sub entries of E into H according to Lemmas 1 and 2 (lines 8–11). Otherwise (i.e., E is a leaf entry), if C_Q exists, AkNNA computes $ed_{agg}(Q, e.ptr)$ and prunes object $e.ptr$ without any further verification using Lemma 3 (lines 13–14). Thereafter, if $d_{agg}(Q, e.ptr)$ is smaller than $curAND_k$, the algorithm inserts $e.ptr$ into the result set $MAkNN(Q, O, k)$ (line 16), and updates $curAND_k$ and C_Q if necessary (line 17). In the end, the final query result set $MAkNN(Q, O, k)$ is returned.

Example 1. We illustrate AkNNA using the example depicted in Fig. 3 with the corresponding SPB-trees shown in Fig. 2. Assume that $k = 2$ and sum function is utilized. First of all, $curAND_k$ is initialized to infinity, and C_Q and the min-heap H are set to empty. Then, AkNNA computes $\phi(q_1) = \langle 2, 5\rangle$, $\phi(q_2) = \langle 3, 3\rangle$, and $\phi(q_3) = \langle 4, 2\rangle$ using P, obtains MBB $M_Q = \{[2, 4], [2, 5]\}$, and pushes the root entries into H ($= \{E_1, E_2\}$). Next, it performs a while-loop. In the first loop, AkNNA pops the top entry E_1 from H. Since E_1 is a non-leaf entry, the algorithm pushes its qualified sub entries E_3 and E_4 into H ($= \{E_4, E_2, E_3\}$), due to $EMIND_{agg}$ and $MIND_{agg}$ of E_3 and E_4 from Q are smaller than $curAND_k$. Similarly, in the second loop, AkNNA pops E_4 and pushes the qualified sub leaf entries into H($=\{E_9, E_{10}, E_2, E_3\}$). Then, A$k$NNA pops the leaf entry E_9 and inserts o_4 into $MAkNN(Q, O, 2)$ as $d_{agg}(o_4, Q) < curAND_k$. After that, $C_Q.o$ and $C_Q.r$ are set as o_4 and 2, respectively. In the sequel, it pops and evaluates entries in H similarly until $MIND_{agg}(E_3, Q) > curAND_k$, after which $MAkNN(Q, O, 2) = \{o_4, o_3\}$. Finally, A$k$NNA stops and returns $MAkNN(Q, O, 2)$ as the final result set. □

5 Performance Study

In this section, we experimentally evaluate the performance of MAkNN retrieval algorithms based on the SPB-tree. We implemented the algorithms in C++. All experiments were conducted on an Intel Core 2 Duo 2.93 GHz PC with 3 GB RAM.

5.1 Experimental Setup

We employ three real datasets, namely, *Words*, *Color*, and *DNA*, as depicted in Table 2. *Words*[1] contains proper nouns, acronyms, and compound words taken from the Moby Project, and the edit distance is used to compute the distance between two words. *Color*[2] denotes the color histograms extracted from an image database, and L_5-norm is utilized to compare the color image features. *DNA*[3] consists of 1 million DNA data, and the cosine similarity is used to measure its similarity under the tri-gram counting space.

We investigate the efficiency of MAkNN retrieval algorithms under various parameters, which are listed in Table 3. Note that, in every experiment, only one factor varies, whereas the others are fixed to their default values. The main

Table 2. Statistics of the datasets used

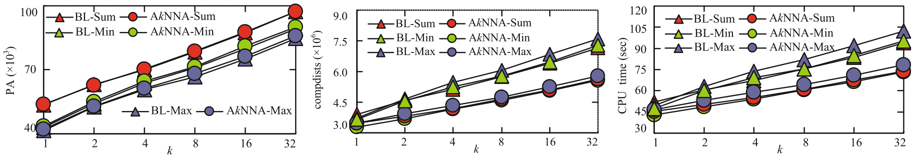
Dataset	Cardinality	Dim.	Ins. Dim.	Measurement
Words	611,756	1–34	4.9	*Edit distance*
Color	112,682	16	2.9	L_5-*norm*
DNA	1,000,000	108	6.9	*Cosine similarity under tri-gram counting space*

[1] *Words* is available at http://icon.shef.ac.uk/Moby/.
[2] *Color* is available at http://www.sisap.org/Metric_Space_Library.html.
[3] *DNA* is available at http://www.ncbi.nlm.nih.gov/genome.

Table 3. Parameter ranges and default values

Parameter	Setting	Default
k	1, 2, 4, 8, 16, 32	8
query set cardinality $\|Q\|$	4, 16, 64, 256, 1024	64
query set area A_Q of the whole space	2%, 4%, 8%, 16%, 32%	8%

performance metrics include the number of page accesses (*PA*), the number of distance computations (*compdists*), and the CPU time. Each measurement we report is the average of 500 queries.



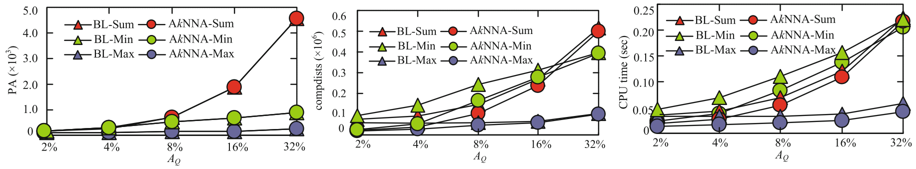
(a) *DNA* (b) *DNA* (c) *DNA*

Fig. 5. AkNN query performance vs. k

5.2 Results on AkNN Queries

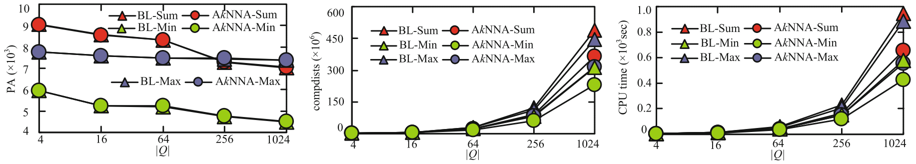
We verify the performance of our proposed algorithms (i.e., BL and AkNNA) in answering MAkNN queries in metric spaces. BL is a baseline method directly extended from MkNN framework [27] using SPB-tree. We inspect the influence of various parameters, containing (1) the area of query set A_Q, (2) the cardinality of query set $|Q|$, and (3) the value of k, i.e., the number of aggregate NNs required.

Figures 5, 6, and 7 show the experimental results w.r.t. k, A_Q, and $|Q|$, respectively. The first observation is that, AkNNA achieves better performance in terms of the number of distance computations and the CPU time, but has similar number of page accesses as BL. This is because, AkNNA employs Lemmas 2 and 3 to save the distance computational cost and avoid unnecessary distance computations, while BL only uses Lemma 1. However, the I/O cost of MAkNN search is related with the search region. In other words, the I/O cost is mostly related with the distribution of the query set and the dataset, which can hardly be reduced by Lemmas 2 and 3. Thus, BL and AkNNA have similar I/O cost. The second observation is that, the query cost increases with A_Q and k, due to the growth of search space. Note that, the query cost of AkNNA, including the number of distance computations and the CPU time, approaches to that of BL as A_Q grows. The reason is that, with the growth of A_Q, the minimum bounding box and minimum bounding circle for the query set becomes larger, and thus, the pruning power of Lemmas 2 and 3 decreases. In addition, the number of distance computations and the CPU time increase with $|Q|$. This is because, the



(a) *Color* (b) *Color* (c) *Color*

Fig. 6. AkNN query performance vs. query set area A_Q



(a) *Words* (b) *Words* (c) *Words*

Fig. 7. AkNN query performance vs. query set cardinality $|Q|$

aggregate distance computation needs more distance computations and becomes more costly as the number of query objects $|Q|$ ascends. Nevertheless, the I/O cost drops as $|Q|$ grows, since the search region decreases due to the dropping k-the aggregate NN distance (AND_k) value for min and max functions, and $AND_k/|Q|$ value for sum function.

6 Conclusions

Metric aggregation k nearest neighbor (MAkNN) search is useful in many areas of computer science, such as multimedia retrieval, resource allocation, and so forth, because it can support various data types and flexible similarity measurements as long as the measurements satisfy the triangle inequality. To answer MAkNN efficiently, we develop several pruning lemmas that utilizes the triangle inequality and present efficient algorithms based on SPB-tree. Extensive experiments show that, our MAkNN search algorithm is more efficient than the baseline algorithm extended from the state-of-the art MAkNN search framework. In the future, we plan to extend the MAkNN search algorithms to various distributed environments.

Acknowledgments. This work was supported in part by the 973 Program of China under Grant No. 2015CB352502, the NSFC under Grant No. 61522208, the NSFC-Zhejiang Joint Fund under Grant No. U1609217, and the ZJU-Hikvision Joint Project.

References

1. Kalantari, I., McDonald, G.: A data structure and an algorithm for the nearest point problem. IEEE Trans. Softw. Eng. **9**(5), 631–634 (1983)
2. Uhlmann, J.K.: Satisfying general proximity/similarity queries with metric trees. Inf. Process. Lett. **40**(4), 175–179 (1991)
3. Brin, S.: Near neighbor search in large metric spaces. In: VLDB, pp. 574–584 (1995)
4. Navarro, G.: Searching in metric spaces by spatial approximation. VLDB J. **11**(1), 28–46 (2002)
5. Ciaccia, P., Patella, M., Zezula, P.: M-tree: an efficient access method for similarity search in metric spaces. In: VLDB, pp. 426–435 (1997)
6. Dohnal, V., Gennaro, C., Savino, P., Zezula, P.: D-index: distance searching index for metric data sets. Multimed. Tools Appl. **21**(1), 9–33 (2003)
7. Chavez, E., Navarro, G.: A compact space decomposition for effective metric indexing. Pattern Recogn. Lett. **26**(9), 1363–1376 (2005)
8. Almeida, J., Torres, R.D.S., Leite, N.J.: BP-tree: an efficient index for similarity search in high-dimensional metric spaces. In: CIKM, pp. 1365–1368 (2010)
9. Mico, L., Oncina, J., Carrasco, R.C.: A fast branch & bound nearest neighbour classifier in metric spaces. Pattern Recogn. Lett. **17**(7), 731–739 (1996)
10. Ruiz, G., Santoyo, F., Chavez, E., Figueroa, K., Tellez, E.S.: Extreme pivots for faster metric indexes. In: SISAP, pp. 115–126 (2013)
11. Burkhard, W., Keller, R.: Some approaches to best-match file searching. Commun. ACM **16**(4), 230–236 (1973)
12. Baeza-Yates, R.A., Cunto, W., Manber, U., Wu, S.: Proximity matching using fixed-queries trees. In: CPM, pp. 198–212 (1994)
13. Bozkaya, T., Ozsoyoglu, M.: Distance-based indexing for high-dimensional metric spaces. In: SIGMOD, pp. 357–368 (1997)
14. Traina Jr., C., Filho, R.F.S., Traina, A.J.M., Vieira, M.R., Faloutsos, C.: The Omni-family of all-purpose access methods: asimple and effective way to make similarity search more efficient. VLDB J. **16**(4), 483–505 (2007)
15. Ares, L.G., Brisaboa, N.R., Esteller, M.F., Pedreira, O., Places, A.S.: Optimal pivots to minimize the index size for metric access methods. In: SISAP, pp. 74–80 (2009)
16. Chavez, E., Navarro, G., Baeza-Yates, R.A., Marroquin, J.L.: Searching in metric spaces. ACM Comput. Surv. **33**, 273–321 (2001)
17. Mosko, J., Lokoc, J., Skopal, T.: Clustered pivot tables for I/O-optimized similarity search. In: SISAP, pp. 17–24 (2011)
18. Skopal, T., Pokorny, J., Snasel, V.: PM-tree: pivoting metric tree for similarity search in multimedia databases. In: ADBIS, pp. 803–815 (2004)
19. Novak, D., Batko, M., Zezula, P.: Metric index: an efficient and scalable solution for precise and approximate similarity search. Inf. Syst. **36**(4), 721–733 (2011)
20. Chen, L., Gao, Y., Li, X., Jensen, C.S., Chen, G.: Efficient metric indexing for similarity search. In: ICDE (2015, to appear)
21. Papadias, D., Tao, Y., Mouratidis, K., Hui, C.K.: Aggregate nearest neighbor queries in spatial databases. ACM Trans. Database Syst. (TODS) **30**(2), 529–576 (2005)
22. Li, F., Yi, K., Tao, Y., Yao, B., Li, Y., Xie, D., Wang, M.: Exact and approximate flexible aggregate similarity search. VLDB J. **25**(3), 317–338 (2016)

23. Wang, H., Zheng, K., Su, H., Wang, J., Sadiq, S., Zhou, X.: Efficient aggregate farthest neighbour query processing on road networks. In: Wang, H., Sharaf, M.A. (eds.) ADC 2014. LNCS, vol. 8506, pp. 13–25. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08608-8_2
24. Liu, Z., Wang, C., Wang, J.: Aggregate nearest neighbor queries in uncertain graphs. World Wide Web **17**(1), 161–188 (2014)
25. Abbasifard, M.R., Naderi, H., Fallahnejad, Z., Alamdari, O.I.: Approximate aggregate nearest neighbor search on moving objects trajectories. J. Central South Univ. **22**(11), 4246–4253 (2015)
26. Razente, H.L., Barioni, M.C.N., Traina, A.J.M., Traina Jr., C.: Constrained aggregate similarity queries in metric spaces. In: SBBD, pp. 145–159 (2007)
27. Razente, H.L., Barioni, M.C.N., Traina, A.J.M., Faloutsos, C., Traina Jr., C.: A novel optimization approach to efficiently process aggregate similarity queries in metric access methods. In: CIKM, pp. 193–202. ACM (2008)