The VLDB Journal (2015) 24:415–439
DOI 10.1007/s00778-015-0383-4

REGULAR PAPER

Efficient *k*-closest pair queries in general metric spaces

Yunjun Gao[1,2] · Lu Chen[1] · Xinhan Li[1] · Bin Yao[3] · Gang Chen[1]

Received: 21 May 2014 / Revised: 10 February 2015 / Accepted: 13 March 2015 / Published online: 2 April 2015

Abstract Given two object sets P and Q, a *k-closest pair* (kCP) *query* finds k closest object pairs from $P \times Q$. This operation is common in many real-life applications such as GIS, data mining, and recommender systems. Although it has received much attention in the *Euclidean space*, there is *little* prior work on the *metric space*. In this paper, we study the problem of *kCP query processing in general metric spaces*, namely *Metric kCP* (MkCP) *search*, and propose several efficient algorithms using dynamic disk-based metric indexes (e.g., M-tree), which can be applied to arbitrary type of data as long as a certain metric distance is defined and satisfies the triangle inequality. Our approaches follow depth-first and/or best-first traversal paradigm(s), employ effective *pruning rules* based on metric space properties and the counting information preserved in the metric index, take advantage of *aggressive pruning and compensation* to further boost query efficiency, and derive a node-based *cost model* for MkCP retrieval. In addition, we extend our techniques to tackle two interesting variants of MkCP queries. Extensive experiments with both real and synthetic data sets demonstrate the performance of our proposed algorithms, the effectiveness of our developed pruning rules, and the accuracy of our presented cost model.

Keywords k-Closest pair query · Metric space · Query processing · Cost model · Algorithm

Electronic supplementary material The online version of this article (doi:10.1007/s00778-015-0383-4) contains supplementary material, which is available to authorized users.

✉ Yunjun Gao
gaoyj@zju.edu.cn

Lu Chen
luchen@zju.edu.cn

Xinhan Li
lixh@zju.edu.cn

Bin Yao
yaobin@cs.sjtu.edu.cn

Gang Chen
cg@zju.edu.cn

[1] College of Computer Science, Zhejiang University, Hangzhou, China

[2] Innovation Joint Research Center for Cyber-Physical-Society System, Zhejiang University, Hangzhou, China

[3] Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China

1 Introduction

Given two object sets P and Q, a *k-closest pair* (kCP) *query* finds k closest object pairs from $P \times Q$ according to a certain similarity metric (e.g., L_1-norm, L_2-norm, L_∞-norm, edit distance). Consider, for example, Fig. 1, in which $P = \{p_1, p_2, p_3\}$ and $Q = \{q_1, q_2, q_3, q_4\}$. Assume that the similarity metric between two objects is the Euclidean distance (i.e., L_2-norm), the final result of a 2CP ($k = 2$) query is $\{\langle p_2, q_2\rangle, \langle p_2, q_1\rangle\}$. The kCP query has received considerable attention from the database community, due to its importance in a wide spectrum of applications, such as GIS [17,18,24,39,40], data mining [6,21,30], and so forth. We give two representative examples here.

Application 1 (*GIS*) MkCP queries are helpful in many decision support and demanding data-handling problems. Considering efficient scheduling of tourist facilities, MkCP retrieval can be employed to retrieve k closest pairs of cultural landmarks and populated places. In this query, the *traveling distance* should take into account the underlying *road network*, i.e., it equals to the length of the *shortest path* connecting the cultural landmark and the populated place.

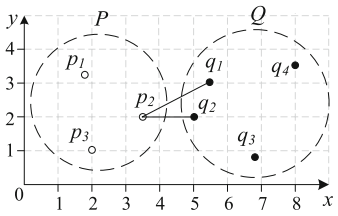

Fig. 1 Example of a kCP query

Application 2 (*Data Mining*) MkCP search can be used as a fundamental building block for a large number of data mining tasks, such as clustering [21,30], outlier detection [6], and so on. For instance, many clustering algorithms (e.g., Chameleon, C2P) can be improved by including MkCP retrieval as a primitive operation. It is worth mentioning that, in some clustering applications [2,34], the similarity of clusters may be *edit* (a.k.a. *Leven-shtein*) *distance*.

In view of flexible distance metrics in the above scenarios, e.g., network distance, edit distance, it requires more generic model than that specially used for a specific distance measurement. Consequently, in this paper, we aim at efficient methods for kCP retrieval in metric spaces, which requires *no* detailed representations of objects and can support any similarity metric satisfying the *triangle inequality*.

However, as to be discussed in Sect. 2, the existing solutions for kCP queries are *insufficient*, due to three reasons below. First, most approaches [17,18,25,35,40,45,52] are applicable only to the *Euclidean space*, where kCP search can be accelerated by exploiting various geometric properties (such as MBR [17,18,40] and plane-sweep [18,35,40]) to effectively prune the search space. Unfortunately, these geometric characteristics employed in the Euclidean space cannot be applied to the generic metric space, since complex data (e.g., strings, documents) do not have obvious Euclidean modeling. Second, existing similarity join algorithms in general metric spaces [22,27,32,33,42], which find object pairs with their distances within a distance threshold ϵ, cannot handle kCP search efficiently. This is because, it is difficult to choose a proper value of ϵ. For a smaller ϵ value, similarity joins cannot return k closet object pairs, resulting in false missing; for a larger ϵ value, similarity joins return more than k closet object pairs, incurring the significant query cost. Third, although there is little work [29,32] on kCP queries in metric spaces, it is insufficient. Specifically, [32] only supports the join for two fixed datasets and achieves its efficiency for datasets with large overlap percentage, which limits its applicability. [29] utilizes the divide-and-conquer technique, which requires high computational cost for dividing, and its efficiency degrades rapidly as the scale of input datasets increases. Moreover, both methods only focus on *main-memory* techniques, which cannot be efficiently applied for the datasets not fitting in memory. Hence, to achieve the scalability of database applications, we assume that the datasets are built external-memory indexes (e.g., M-tree and PM-tree), as the indexes can further speed up search.

Motivated by these, in this paper, we study the problem of *efficient* kCP *query processing in general metric spaces*, namely *Metric kCP* (MkCP) *search*. Intuitively, a straightforward solution is to compare every object pair from two datasets. This approach, nevertheless, is very *inefficient* because, for every object in an object set P, it needs to traverse another object set Q once, incurring *huge* I/O and CPU costs. In order to answer MkCP retrieval efficiently, two challenging issues have to be addressed.

Challenge I: How to minimize I/O overhead in terms of the number of node/page accesses? The I/O cost is an important metric in database search algorithms. We try to handle this issue from two aspects: avoiding unnecessary node accesses and reducing duplicated node accesses. We attempt to develop several effective *pruning rules* to avoid unqualified node accesses. In addition, we utilize LRU buffers and combine the *best-first* and *depth-first* traversal paradigms, in order to reduce duplicated node accesses.

Challenge II: How to minimize CPU cost in terms of the number of distance computations? In most applications, the distance is the dominant complexity measurement in metric spaces, and it is customary to just count the number of computed distances for comparing algorithms. Toward this, we exploit the properties of the metric space (e.g., the triangle inequality) and employ the *aggressive pruning* and *compensation technique*, in order to eliminate unqualified distance computations.

Based on these, we present, using the dynamic metric index M-tree [14], three algorithms for MkCP search with high-accuracy cost model. Note that, the proposed algorithms can also be extended to any other tree structure metric index (e.g., PM-tree [44]). Specifically, the first two algorithms follow depth-first and best-first traversal paradigms, respectively, and utilize pruning rules to efficiently handle MkCP queries. To further improve query efficiency, the third algorithm follows a hybrid traversal paradigm, which combines depth-first and best-first strategies, and uses the aggressive punning and compensation technique based on an estimated kCP distance value. In addition, we extend our techniques to tackle two natural variants of MkCP queries, i.e., (1) *Self* MkCP (SMkCP) search, which performs MkCP retrieval on a single dataset and (2) *Approximate* MkCP (AMkCP) search, which trades the quality of the MkCP query result for the search time.

In brief, the key contributions of this paper are summarized as follows:

– We propose several pruning rules, use the aggressive pruning and compensation technique, and combine best-first and depth-first traversal paradigms, in order to reduce I/O and CPU costs.

- We develop three efficient algorithms for processing the MkCP query and then analyze their corresponding correctness and I/O overhead.
- We derive a node-based cost model for MkCP retrieval using M-trees.
- We extend our techniques to handle two interesting variants of MkCP queries efficiently.
- Extensive experiments with both real and synthetic data sets demonstrate the performance of our proposed algorithms, the effectiveness of our presented pruning rules, and the accuracy of our derived cost model.

The rest of this paper is organized as follows. Sect. 2 reviews related work. Section 3 formalizes the problem and reveals its characteristics, presents the Count M-tree (COM-tree) used to speed up MkCP search, and proposes a series of pruning rules. Section 4 elaborates MkCP query processing algorithms based on COM-trees and devises a cost model for MkCP retrieval. Section 5 extends our techniques to solve two interesting MkCP query variations. Considerable experimental results and our findings are reported in Sect. 6. Finally, Sect. 7 concludes the paper with some directions for future work.

2 Related work

In this section, we overview the existing work related to MkCP queries, including kCP queries in Euclidean spaces, the M-tree, and querying metric spaces.

2.1 Euclidean kCP queries

The existing algorithms for kCP retrieval are mainly under the Euclidean space, which can be classified into two categories, namely the incremental algorithms and the non-incremental algorithms.

The first one [25,39,40] is the incremental alternative, which returns the result in ascending order of distances in a pipeline fashion. Hjaltason and Samet [25] present the incremental algorithms that can be used with a large class of hierarchical spatial data structures. Shin et al. [39,40] improves the incremental k-distance join algorithm, by utilizing bidirectional node expansion and plane-sweep techniques for fast pruning of distant pairs, and the plane-sweep is further optimized by novel strategies for selecting a good sweeping axis and direction. In addition, an adaptive multistage algorithm is developed for the incremental distance join when k is not known in advance.

The second one [17,18,52] is the non-incremental alternative, which assumes that k is known in advance, and reports the result all together at the end. Corral et al. [17,18] propose one pruning heuristic and two updating strategies for minimizing the pruning distance. Based on the pruning heuristic, three non-incremental branch-and-bound external-memory algorithms are developed, with two of them following the depth-first search strategy and one obeying the best-first policy. The plane-sweep technique and access ordering are used to further improve query efficiency. Yang and Lin [52] present a new index structure called bichromatic-Rdnn (b-Rdnn) tree, which utilizes the information about nearest neighbors to process kCP queries efficiently. In particular, the b-Rdnn tree built on an object set P preserves, for each object in P, its nearest neighbor in an answer object set Q. However, to build the b-Rdnn tree on P, we need to know Q in advance, which limits its applicability.

Recently, Cheema et al. [9] propose a unified method for top-k pairs, which allows the users to define a local scoring function for each attribute, and a global scoring function that computes the final score of each pair by combining its scores on different attributes. Nonetheless, the framework relies on the local scoring function defined on every attribute, which does not exist in generic metric spaces. Roumelis et al. [35] propose a new plane-sweep kCP search algorithm, termed *Reverse Run Plane-Sweep*, which minimizes the Euclidean and sweeping axis distance computations.

In addition, many variants of kCP queries have been investigated in the literature. In order to further reduce kCP query cost, some work [3,20,45] focuses on approximate kCP search, which forsakes some precision of the query result in exchange for improved efficiency. Angiulli and Pizzuti [3] study the approximate algorithm to calculate the top-k closest pairs join query, which employs the space-filling curve to establish an order between the points in the space, and performs at most $(d + 1)$ sorts and scans of the two data sets, where d denotes the dimensionality. Corral and Vassilakopulos [20] apply a combination of the approximation techniques, e.g., α-approximate, N-consider or Time-consider in the same query algorithm, i.e., hybrid approximation scheme. Tao et al. [45] utilize LSB technique to solve closest pair search in high dimensional space, which can be accomplished in (worst-case) time significantly lower than the quadratic complexity, yet still ensuring very good quality. Other interesting kCP query variants have also been well studied, such as self kCP queries [18], constrained kCP search [31,38], exclusive kCP retrieval [48], non-index-based kCP search [24], top-k set similarity join [51], kCP queries on moving objects [4,54] and spatial networks [10], respectively, cost models for kCP queries [19], performance comparisons of kCP queries [16,28], to name but a few.

Nonetheless, all the above approaches are unsuitable for MkCP search because, for boosting the query, they make use of some geometric properties (e.g., MBR [17,18,39,40], plane-sweep [18,35,39,40], and space-filling curve [3,45]) that are not available in generic metric spaces.

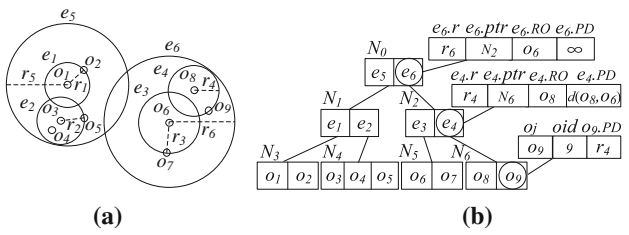

Fig. 2 Example of an M-tree. **a** The dataset placement. **b** The M-tree

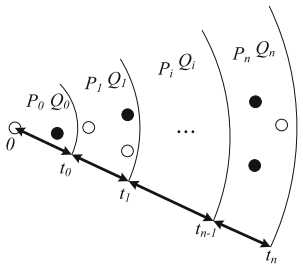

Fig. 3 Illustration of Adaptive Multi-Partitioning (AMP)

2.2 The M-tee

Many indexing technologies in metric spaces have been proposed [8,26,36,44]. However, following most approaches in the relevant literature [11,46,49,53], we assume, in this paper, M-tree [14], an external-memory metric index, is used as an underlying index structure. The M-tree is a balanced tree, and it can handle dynamic operations with reasonable costs, without requiring periodical restructuring.

Figure 2 depicts an example of M-tree for an object set $O = \{o_1, o_2, \ldots, o_9\}$. An intermediate (i.e., a non-leaf) entry e in a root node (e.g., N_0) or an intermediate node (e.g., N_1, N_2) records: (1) A routing object $e.RO$, which is a selected object in the subtree ST_e of e. (2) A covering radius $e.r$, which is the maximal distance between $e.RO$ and the objects in ST_e. (3) A parent distance $e.PD$, which equals the distance from $e.RO$ to the routing object of the parent entry e_p. Since a root entry e (e.g., e_6) has no any parent entry, $e.PD = \infty$. (4) An identifier $e.ptr$, which corresponds to the address of the root node of ST_e. For example, in Fig. 2, a non-leaf entry e_4 is associated with the routing object o_8 $(= e_4.RO)$, its covering radius $e_4.r$ is r_4, parent distance $e_4.PD$ corresponds to the distance $d(o_8, o_6)$ $(= d(e_4.RO, e_6.RO))$, and identifier $e_4.ptr = N_6$. On the other hand, a leaf entry o in a leaf node (e.g., N_3, N_4, N_5, N_6) records: (1) An object o_j, which stores the detailed information of o. (2) An identifier oid, which represents o's identifier. (3) A parent distance $o.PD$, which equals the distance between o and the routing object in o's parent entry. For instance, as shown in Fig. 2, the parent distance $o_9.PD$ of the object o_9 is r_4 $(= d(o_8, o_9))$, while $o_8.PD = 0$ because o_8 is the routing object of its parent entry e_4.

2.3 Querying metric spaces

A metric space is a space with an associated distance metric, which obeys certain simple properties. Several spatial queries in metric spaces have been investigated.

Except for Euclidean kCP queries, two alternatives for MkCP search are explored in [29,32]. Paredes and Reyes [32] answer MkCP retrieval based on a new metric index, termed coined List of Twin Clusters (LTC), which maintains two lists of overlapping clusters to index both object sets jointly. Thus, it is limited for the case when two object sets come from different datasets and have their own metric index independently. In addition, it only achieves the efficiency for the object sets with large overlap percentage, as to be verified in Sect. 6.2. Kurasawa et al. [29] propose a divide-and-conquer-based kCP query method in metric spaces, called Adaptive Multi-Partitioning (AMP), which repeatedly divides and conquers the objects from the sparser distance distribution space, and speeds up the convergence of the upper bound distance before partitioning the denser space. Each time, AMP randomly chooses a pivot o to divide the object set P into n regions: $P_0 = \{p \in P | 0 \leq d(o, p) < t_0\}$, $P_1 = \{p \in P | t_0 \leq d(o, p) < t_1\}, \ldots, P_i = \{p \in P | t_{i-1} \leq d(o, p) < t_i\}, \ldots, P_n = \{p \in P | t_{n-1} \leq d(o, p) < t_n\}$, as $|t_i - t_{i-1}| > u$ and u is the upper bound distance of kCP. Similarly, Q can also be partitioned by the pivot o, as shown in Fig. 3. Although AMP can avoid distance computations searching in $\langle P_i, Q_j \rangle$ with $|j - i| > 1$, the CPU cost required for dividing is still high, which increases rapidly with the scale of input datasets, as to be verified in our experiments. Moreover, the aforementioned two approaches only aim at in-memory techniques and hence cannot be applied to support external-memory MkCP query processing efficiently.

Similarity search in metric spaces, including range and nearest neighbor (NN) queries, has been well studied and summarized in [8,26,36]. Ciaccia and Patella [13] propose an approximation algorithm for kNN search, in which the error bound ε can be exceeded within a certain probability δ using the distance distribution of the query point. Zezula et al. [53] introduce three different approximation methods over M-trees, with the relative error is not bounded by any function of the input parameters. In addition, cost models [5,12,15,47] are also derived for metric similarity queries. More recently, Vlachou et al. [49] present a framework for distributed metric similarity search, where each participating peer preserves its own data using an M-tree.

A similarity join retrieves the object pairs with their distances bounded by a distance threshold ϵ. This operation has also been well studied in metric spaces, and there are many efficient solutions, as overviewed in [27]. Recently, Paredes and Reyes [32] handle similarity joins using LTC, which indexes both sets jointly. Silva and Pearson [33,41,42] develop a non-blocking similarity join database operator

DBSimJoin, and study index-based similarity joins. Fredriksson and Braithwaite [22] improve the Quicksort algorithm [27] for similarity joins. In addition, similarity joins using Map-Reduce have also been studied [37,50]. However, all the above solutions cannot handle MkCP search efficiently. This is because, it is difficult to choose a proper value of ϵ. For a smaller ϵ value, similarity join cannot return k closet object pairs, resulting in false missing. For a larger ϵ value, similarity join returns more than k closet object pairs, incurring the significant query cost. As an example, we have to run the similarity join algorithm twice to obtain the accurate result set [22], which is costly.

In addition, two other spatial queries, i.e., reverse kNN and skyline queries, are also studied in metric spaces. In particular, metric reverse kNN search [1,46] finds the objects in a given object set that have a specified query object q as one of their k NNs, and metric skyline query [11,23,43] retrieves the objects not dominated by any other object with respect to all query objects in a generic metric space.

3 Preliminaries

In this section, we first formally define metric kCP (MkCP) search, and then, we introduce the count M-tree (COM-tree), and present several pruning rules and lemmas that can facilitate the development of efficient MkCP search algorithms. Table 1 summarizes the notations used frequently throughout this paper.

3.1 Problem formulation

A metric space is a tuple (D, d), where D is the domain of feature values, and d is a distance function used to compare objects in D. The distance function must satisfy the four properties below: (1) symmetry: $d(p, q) = d(q, p)$; (2) non-negativity: $d(p, q) \geq 0$; (3) identity: $d(p, q) = 0$ iff $p = q$; and (4) triangle inequality: $d(p, q) \leq d(p, o) + d(o, q)$. Based on these properties, we formalize the MkCP query.

Table 1 Symbols and description

Notation	Description
P or Q	The object set P or Q
$\|P\|$ or $\|Q\|$	The cardinality of P or Q
M_P or M_Q	The COM-tree/COMdnn-tree/GMdnn-tree on P or Q
$\|M_P\|$ or $\|M_Q\|$	The cardinality of M_P or M_Q
ST_e	The subtree of an intermediate entry e
E_P or E_Q	The leaf/non-leaf entry in M_P or M_Q
PE_P of PE_Q	The parent entry of E_P or E_Q
$d()$	The function of a certain metric distance
S_R	The result set of MkCP/SMkCP/AMkCP search
CPD_k	The kth closest pair distance in the result set
$maxCPD_k$	The upper bound of CPD_k
$eCPD_k$	The estimation value of CPD_k

Definition 1 (*MkCP Search*). Given two object sets P and Q in a generic metric space, and an integer k ($1 \leq k \leq |P| \times |Q|$), a *metric k-closest pair* (MkCP) *query* finds k ordered different closest object pairs from $P \times Q$, i.e., $MkCP(P, Q) = \{\langle p_1, q_1\rangle, \langle p_2, q_2\rangle, \ldots, \langle p_k, q_k\rangle | p_1, p_2, \ldots, p_k \in P, q_1, q_2, \ldots, q_k \in Q, \langle p_i, q_i\rangle \neq \langle p_j, q_j\rangle, i \neq j, 1 \leq i, j \leq k$, and $\forall(p', q') \in P \times Q - \{\langle p_1, q_1\rangle, \langle p_2, q_2\rangle, \ldots, \langle p_k, q_k\rangle\}$ such that $d(p', q') \geq d(p_k, q_k) \geq \cdots \geq d(p_1, q_1)\}$.

Consider two English word sets $P = \{$"till," "thrill"$\}$ and $Q = \{$"ill," "doll," "nila"$\}$, and suppose the edit distance is used to measure the similarity between two words. If $k = 2$, $M2CP(P, Q) = \{\langle$ "till," "ill" $\rangle, \langle$"till," " doll" $\rangle\}$.

Based on Definition 1, $MkCP(P, Q)$ may be not unique due to the distance ties. However, the goal of our proposed algorithms was to find one possible instance. Thus, we randomly choose object pairs when the distance ties occur.

In this paper, we study the problem of MkCP retrieval. For simplicity, in all the illustrative examples used in the rest of this paper, we assume that the metric distance function is L_2-norm (i.e., Euclidean distance). In order to minimize the query cost, we introduce the COM-tree, and based on which we develop two pruning heuristics, as described in Sects. 3.2 and 3.3, respectively.

3.2 The count M-tree

To enable the search space pruning, we propose a variant of M-tree, termed *COUNT M-tree* (COM-tree), which includes $e.num$ in each intermediate entry e to represent the number of the objects contained in ST_e. For ease of understanding, the COM-tree on the object set depicted in Fig. 2a is illustrated in Fig. 4, where the number associated with every non-leaf entry denotes $e.num$. For instance, $e_6.num = 4$ as ST_{e_6} contains four objects o_6, o_7, o_8, o_9, and $e_4.num = 2$ since ST_{e_4} includes two objects o_8 and o_9. Note that, these digits are computed and stored during the construction of COM-tree.

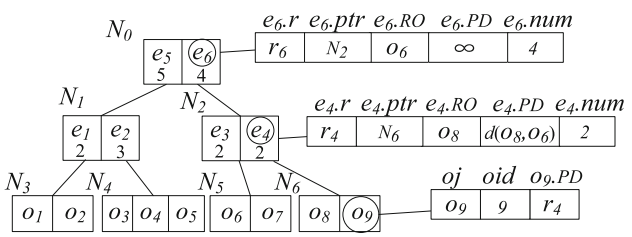

Fig. 4 Example of a COM-tree

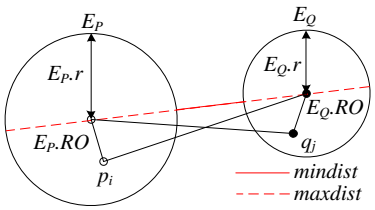

Fig. 5 Example for *mindist* and *maxdist*

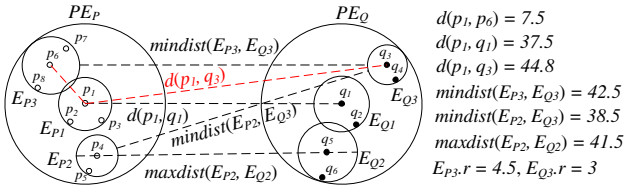

Fig. 6 Example for Lemmas 1 and 2

As M*k*CP search acts on the entry pairs from the COM-trees over P and Q (i.e., M_P and M_Q), the minimal distance and the maximal distance between entry pairs, which can accelerate the search space pruning, are defined as follows.

Definition 2 (*Mindist*). Given two intermediate entries E_P and E_Q, $mindist(E_P, E_Q) = \max\{d(E_P.RO, E_Q.RO) - E_P.r - E_Q.r, 0\}$.

Definition 3 (*Maxdist*). Given two intermediate entries E_P and E_Q, $maxdist(E_P, E_Q) = d(E_P.RO, E_Q.RO) + E_P.r + E_Q.r$.

Consider, for example, Fig. 5, in which the red solid line represents $mindist(E_P, E_Q)$, and the red dotted line denotes $maxdist(E_P, E_Q)$. Based on Definitions 2 and 3, $mindist(E_P, E_Q)$ and $maxdist(E_P, E_Q)$ offer the *lower* and *upper* bounds of the distances between object pairs (from E_P and E_Q), respectively.

3.3 Pruning heuristics

Querying metric spaces is inherently more difficult than that in Euclidean spaces, due to the lack of geometric properties. In the sequel, we devise alternative pruning rules based on an intuitive observation: an entry pair can be pruned if it cannot contain any qualified object pair.

Rule 1 *Given two leaf or non-leaf entries E_P and E_Q, if $mindist(E_P, E_Q) > maxCPD_k$, with $maxCPD_k$ denoting the upper bound distance of the kth closest pair, $\langle E_P, E_Q\rangle$ can be discarded safely.*

Proof The proof is straightforward according to the definitions of $mindist(E_P, E_Q)$ and $maxCPD_k$. □

Consider the example shown in Fig. 6. Assume that current $maxCPD_k = 20$, $\langle E_{P3}, E_{Q3}\rangle$ can be discarded as $mindist(E_{P3}, E_{Q3}) > maxCPD_k$. However, for every Rule 1 application, it needs one distance computation to calculate $mindist$. To this end, we give the definition of $emindist$ in Definition 4, which can utilize the triangle inequality to avoid unnecessary distance computations.

Definition 4 (*Emindist*). Given two intermediate entries E_P and E_Q, and suppose PE_P (PE_Q) is the parent entry of E_P (E_Q), then $emindist(E_P, E_Q)$ is defined as

$$\begin{cases} d_{PE_Q} & \text{only } d(E_P.RO, PE_Q.RO) \text{ is known} \\ d_{PE_P} & \text{only } d(E_Q.RO, PE_P.RO) \text{ is known} \\ \max\{d_{PE_Q}, d_{PE_P}\} & \text{both are known} \end{cases}$$

where $d_{PE_Q} = |d(E_P.RO, PE_Q.RO) - E_Q.PD| - E_P.r - E_Q.r$ and $d_{PE_P} = |d(E_Q.RO, PE_P.RO) - E_P.PD| - E_P.r - E_Q.r$, respectively.

Due to the triangle inequality, $|d(E_P.RO, PE_Q.RO) - E_Q.PD|$ and $|d(E_Q.RO, PE_P.RO) - E_P.PD|$ are no larger than $d(E_P.RO, E_Q.RO)$, and thus, $emindist(E_P, E_Q) \le mindist(E_P, E_Q)$, i.e., $emindist$ provides a lower bound of $mindist$. Take Fig. 6 as an example. If only $d(E_Q.RO, PE_P.RO)$ ($= d(q_3, p_1)$) is known, $emindist(E_{P3}, E_{Q3}) = d(q_3, p_1) - d(p_1, p_6) - E_{P3}.r - E_{Q3}.r = 29.8$, which is smaller than $mindist(E_{P3}, E_{Q3})$. Based on Definition 4, we present a new pruning rule as follows.

Rule 2 *Given two leaf or non-leaf entries E_P and E_Q, if $emindist(E_P, E_Q) > maxCPD_k$, then $\langle E_P, E_Q\rangle$ can be pruned away safely.*

Proof According to the definition of $emindist$, if $emindist(E_P, E_Q) > maxCPD_k$, $mindist(E_P, E_Q) \ge emindist(E_P, E_Q) > maxCPD_k$ holds. Therefore, $\langle E_P, E_Q\rangle$ can be discarded safely due to Rule 1. □

An example of Rule 2 is depicted in Fig. 6. Assume that current $maxCPD_k = 20$, $\langle E_{P3}, E_{Q3}\rangle$ can be pruned as $emindist(E_{P3}, E_{Q3}) > maxCPD_k$. Note that, although Rule 2 has weaker pruning power than Rule 1, it does not need any distance computation, as all the values required to compute $emindist$ for Rule 2 are directly available.

To deploy Rule 1 and Rule 2, we must derive values of $maxCPD_k$. Although their values are not unique, we know that $maxCPD_k$ should be as *small* as possible to achieve strong pruning power. A simple way to obtain $maxCPD_k$ is to use the distance of the kth closest object pair retrieved so far during M*k*CP query processing. However, we can get a tight $maxCPD_k$ using intermediate entry pairs, in order to efficiently shrink the search space as *early* as possible.

Lemma 1 *Given two intermediate entries E_P and E_Q, if $k = 1$, $maxCPD_1$ can be set to $d(E_P.RO, E_Q.RO)$; and if $k > 1$, $maxCPD_k$ can be set to*

$$\min \begin{cases} d(E_P.RO, E_Q.RO)+E_P.r & E_P.num \geq k \\ d(E_P.RO, E_Q.RO)+E_Q.r & E_Q.num \geq k \\ maxdist(E_P, E_Q) & E_P.num \times E_Q.num \geq k \end{cases}$$

Proof To prove this lemma, $maxCPD_k$ can be updated with r if there are k different object pairs having their distances bounded by r.

For $k = 1$, since the routing object of an intermediate entry is a real object, there exists an object pair $\langle E_P.RO, E_Q.RO\rangle$ with its distance bounded by $d(E_P.RO, E_Q.RO)$. Thus, $maxCPD_1$ can be set to $d(E_P.RO, E_Q.RO)$.

For $k > 1$, as shown in Fig. 5, there are (1) $E_P.num$ object pairs $\langle p_i, E_Q.RO\rangle$ $(p_i \in E_P)$ with their distances $d(p_i, E_Q.RO) \leq d(E_P.RO, E_Q.RO)+d(p_i, E_P.RO) \leq d(E_P.RO, E_Q.RO) + E_P.r$ (according to the triangle inequality), (2) $E_Q.num$ object pairs $\langle q_j, E_P.RO\rangle$ $(q_j \in E_Q)$ with their distances bounded by $d(E_P.RO, E_Q.RO)+E_Q.r$, and (3) $E_P.num \times E_Q.num$ object pairs $\langle p_i, q_j\rangle$ with their distances bounded by $maxdist(E_P, E_Q)$. Consequently, $maxCPD_k$ can be set as the corresponding minimum value according to k, and the proof completes. □

Consider the example illustrated in Fig. 6, where two intermediate entries E_{P2} and E_{Q2} with $E_{P2}.num = 2$ and $E_{Q2}.num = 2$. We can set $maxCPD_4$ $(k = 4)$ to $maxdist(E_{P2}, E_{Q2})$ based on Lemma 1. Hence, $\langle E_{P3}, E_{Q3}\rangle$ can be pruned by Rule 1, as $mindist(E_{P3}, E_{Q3}) > maxCPD_4$; whereas $\langle E_{P2}, E_{Q3}\rangle$ can not be discarded due to $mindist(E_{P2}, E_{Q3}) < maxCPD_4$. Note that, Lemma 1 considers only *one* intermediate entry pair. If we take into account *all* subentry pairs in the same parent entry pair, the value of $maxCPD_k$ can be tighter, as stated in Lemma 2.

Lemma 2 *Given two sets of intermediate entries E_{Pi} $(1 \leq i \leq m)$ and E_{Qj} $(1 \leq j \leq n)$, which are subentries of the parent entries PE_P and PE_Q, respectively, and we sort $d(E_{Pi}.RO, E_{Qj}.RO)$ and $maxdist(E_{Pi}, E_{Qj})$ in ascending order, and then get two ordered sequences $d(E_{Pt}.RO, E_{Qt}.RO)$ and $maxdist(E_{Pt}, E_{Qt})$ $(1 \leq t \leq mn)$. If $k > 1$, $maxCPD_k$ can be set to*

$$\min \begin{cases} d(E_{Pk}.RO, E_{Qk}.RO) & mn \geq k \\ maxdist(E_{Px}, E_{Qx}) & \sum_{1 \leq t \leq x \leq mn} E_{Pt}.num \times E_{Qt}.num \geq k \end{cases}$$

Proof Let r be the minimum between $d(E_{Pk}.RO, E_{Qk}.RO)$ and $maxdist(E_{Px}, E_{Qx})$. Similar as Lemma 1, to prove this lemma, we need to find k different object pairs with their distances bounded by r.

Since the routing object of an intermediate entry is a real object and $d(E_{Pt}.RO, E_{Qt}.RO)$ $(1 \leq t \leq mn)$ is sorted in ascending order, there are k object pairs $\langle E_{Pt}.RO, E_{Qt}.RO\rangle$ $(1 \leq t \leq k)$ with their distances bounded by $d(E_{Pk}.RO, E_{Qk}.RO)$ if $mn \geq k$; and if the xth $(x \leq mn)$ $maxdist(E_{Px}, E_{Qx})$ satisfies the fact that the total number of object pairs contained in $\langle E_{Pt}, E_{Qt}\rangle$ $(1 \leq t \leq x)$ is *larger* than k, there exist *at least* k object pairs with their distances bounded by $maxdist(E_{Px}, E_{Qx})$. Therefore, $maxCPD_k$ can be set as the corresponding minimum value according to k, and the proof completes. □

Back to the example shown in Fig. 6, where E_{Pi} $(1 \leq i \leq 3)$ and E_{Qj} $(1 \leq j \leq 3)$ are subentries of PE_P and PE_Q, respectively. To utilize Lemma 2, we sort $d(E_{Pi}.RO, E_{Qj}.RO)$ and $maxdist(E_{Pi}, E_{Qj})$ in ascending order, and then obtain two ordered sequences $d(p_4, q_5), d(p_1, q_5), d(p_4, q_1), d(p_1, q_1), \ldots, d(p_6, q_3)$; $maxdist(E_{P2}, E_{Q2})$, $maxdist(E_{P2}, E_{Q1})$, $maxdist(E_{P1}, E_{Q2}), \ldots, maxdist(E_{P3}, E_{Q3})$. Then, $maxCPD_4$ can be set to the *fourth* distance $d(p_1, q_1)$ that is *smaller* than $maxdist(E_{P2}, E_{Q2})$. Thus, $\langle E_{P2}, E_{Q3}\rangle$ can be pruned by Rule 1, as $mindist(E_{P2}, E_{Q3}) > d(p_1, q_1)$ holds, but it cannot be discarded if we only employ Lemma 1 to update $maxCPD_4$.

When expanding $\langle PE_P, PE_Q\rangle$, its subentry pairs $\langle E_{Pi}, E_{Qj}\rangle$ are processed *one by one*. In order to utilize Lemma 2 incurring *no* additional distance computations, instead of calculating $maxCPD_k$ using subentry pairs *all at once*, we update $maxCPD_k$ *gradually* with processed subentry pairs whose *mindist* are already computed. This is because both $d(E_{Pi}.RO, E_{Qj}.RO)$ and $maxdist(E_{Pi}, E_{Qj})$ can be easily obtained when computing $mindist(E_{Pi}, E_{Qj})$.

4 MkCP query processing

In this section, we elaborate three efficient algorithms for MkCP search, assuming that P and Q are indexed by COM-trees, and then present a cost model for MkCP retrieval. In the sequel, a running example (shown in Fig. 7) is employed to facilitate the understanding of different MkCP search

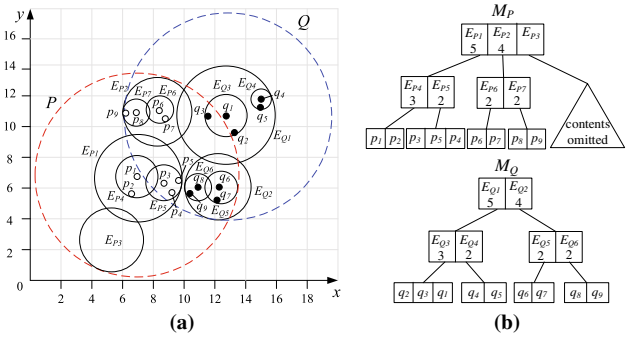

Fig. 7 A running example for an M2CP ($k = 2$) query. **a** The placement of P and Q. **b** The COM-trees

algorithms. Specifically, Fig. 7 shows the COM-trees M_P and M_Q for the object sets $P = \{p_1, p_2, \ldots, p_9\}$ and $Q = \{q_1, q_2, \ldots, q_9\}$, respectively, and the final result set $S_R = \{\langle p_5, q_9\rangle, \langle p_4, q_9\rangle\}$ for an M2CP ($k = 2$) query.

4.1 Recursive M*k*CP algorithm

Based on the pruning rules and lemmas presented in Sect. 3, we propose, as depicted in Algorithm 1, *Recursive MkCP Algorithm* (RMA) for M*k*CP retrieval, which follows a *depth-first* paradigm. First of all, $maxCPD_k$ is set as infinity (line 1). Then, RMA updates $maxCPD_k$ using Lemmas 1–2, and prunes the root entry pairs (line 2). Next, it calls the function RMA-PEP for the root entry pair $\langle E_P, E_Q\rangle$ not pruned in ascending order of its *mindist* until $mindist(E_P, E_Q) > maxCPD_k$ (line 3). Note that, if there is a tie of *mindist*, the intersection area between two entries serves as a tie-breaker, and the larger is better. However, it is also easy for RMA to use other tie-breakers (e.g., *maxdist*). Finally, the algorithm returns the result set S_R (line 4). Note that, S_R is maintained by using a max-heap, which keeps the k closest pairs found so far in descending order of their distances during M*k*CP search.

Algorithm 1 Recursive M*k*CP Algorithm (RMA)

Input: a COM-tree M_P, a COM-tree M_Q
Output: the result set S_R of an M*k*CP query
1: $maxCPD_k = \infty$
2: update $maxCPD_k$ by Lemmas 1-2 and prune root entry pairs by Rule 1
3: call RMA-PEP for the remaining root entry pairs $\langle E_P, E_Q\rangle$ in ascending order of *mindist* until $mindist(E_P, E_Q) > maxCPD_k$
4: **return** S_R

Function: RMA-PEP(E_P, E_Q)
/* H: a *local* min-heap used to store subentry pairs of $\langle E_P, E_Q\rangle$ not pruned in ascending order of *mindist*. */
5: **if** E_P and E_Q are intermediate entries pointing to non-leaf nodes **then**
6: $H = \emptyset$
7: PRUNE(E_P, E_Q, H)
8: **while** $H \neq \emptyset$ **do**
9: pop the top entry pair $\langle e_{Pi}, e_{Qj}\rangle$ from H
10: **if** $mindist(e_{Pi}, e_{Qj}) > maxCPD_k$ **then break**
11: RMA-PEP(e_{Pi}, e_{Qj}) // recursively call
12: **else if** E_P and E_Q are intermediate entries pointing to leaf nodes **then**
13: PRUNE(E_P, E_Q, S_R)

Function: PRUNE(E_P, E_Q, H)
14: **for** each entry $e_{Pi} \in E_P$ **do**
15: **if** $emindist(e_{Pi}, E_Q) > maxCPD_k$ **then** prune e_{Pi} // Rule 2
16: **else if** $mindist(e_{Pi}, E_Q) > maxCPD_k$ **then** prune e_{Pi} // Rule 1
17: **for** each entry $e_{Qj} \in E_Q$ **do**
18: **if** $emindist(e_{Qj}, E_P) > maxCPD_k$ **then** prune e_{Qj} // Rule 2
19: **else if** $mindist(e_{Qj}, E_P) > maxCPD_k$ **then** prune e_{Qj} // Rule 1
20: **for** each entry $e_{Pi} \in E_P$ not pruned **do**
21: **for** each entry $e_{Qj} \in E_Q$ not pruned **do**
22: **if** $emindist(e_{Pi}, e_{Qj}) \leq maxCPD_k$ and $mindist(e_{Pi}, e_{Qj}) \leq maxCPD_k$ **then**
23: insert $\langle e_{Pi}, e_{Qj}\rangle$ into H and update $maxCPD_k$ // Lemmas 1-2

For each currently visited intermediate entry pair $\langle E_P, E_Q\rangle$ pointing to non-leaf nodes, RMA-PEP uses *bidirectional* node expansion technique to expand $\langle E_P, E_Q\rangle$ (lines 6–11). Initially, it initializes a local min-heap H storing the subentry pairs of $\langle E_P, E_Q\rangle$ based on ascending order of their *mindist*. In order to minimize the effects quadratic cost of checking the Cartesian product of $\langle e_{Pi}, e_{Qj}\rangle$ ($e_{Pi} \in E_P$, $e_{Qj} \in E_Q$), it invokes a function PRUNE, which removes from consideration (1) all $e_{Pi} \in E_P$, for which $emindist(e_{Pi}, E_Q)$ or $mindist(e_{Pi}, E_Q)$ is larger than $maxCPD_k$ (lines 15–16), (2) all $e_{Qj} \in E_Q$, for which $emindist(e_{Qj}, E_P)$ or $mindist(e_{Qj}, E_P)$ is larger than $maxCPD_k$ (lines 18–19). Note that, $emindist(e_{Pi}, E_Q) = |d(E_P.RO, E_Q.RO) - e_{Pi}.PD| - e_{Pi}.r - E_Q.r$ and $emindist(E_P, e_{Qj}) = |d(E_P.RO, E_Q.RO) - e_{Qj}.PD| - E_P.r - e_{Qj}.r$, as $d(E_P.RO, E_Q.RO)$ is already known. For each remaining subentry pair $\langle e_{Pi}, e_{Qj}\rangle$, PRUNE adds it to H, and updates $maxCPD_k$ using Lemmas 1–2, if $\langle e_{Pi}, e_{Qj}\rangle$ can not be pruned by Rules 1–2 (lines 20–23), where $emindist(e_{Pi}, e_{Qj}) = \max\{|d(e_{Pi}.RO, E_Q.RO) - e_{Qj}.PD|, |d(e_{Qj}.RO, E_P.RO) - e_{Pi}.PD|\} - e_{Pi}.r - e_{Qj}.r$. Next, RMA-PEP is called recursively for every entry pair $\langle e_{Pi}, e_{Qj}\rangle$ in H until $mindist(e_{Pi}, e_{Qj}) > maxCPD_k$ or $H = \emptyset$ (lines 8–11). At each newly visited intermediate entry pair $\langle E_P, E_Q\rangle$ pointing to leaf nodes, it calls PRUNE to update the result set S_R and $maxCPD_k$, respectively, using each qualified object pair (line 13).

We illustrate RMA using the running example depicted in Fig. 7, and please refer to Appendix A for details.

Lemma 3 *The proposed algorithm RMA can return exactly the actual MkCP query result, i.e., the algorithm has no false negatives, no false positives, and the returned result set contains no duplicate objects.*

Proof First, no real answer object pairs are missed (i.e., *no false negatives*), as only unqualified (leaf and non-leaf) entry pairs are pruned by Rules 1–2. Second, assume, to the contrary, that a false answer object pair $\langle p'_i, q'_j\rangle$ is returned, then the returned CPD'_k must be larger than the actual CPD_k. For any real answer object pair $\langle p_i, q_j\rangle$, it cannot be pruned by Rules 1–2 due to $d(p_i, q_j) < CPD_k < CPD'_k \leq maxCPD_k$, which can be used to update the result set. Hence, all the actual answer object pairs are returned, which contradicts with our assumption, and thus, *no false positives* is ensured. Third, *no duplicate object pairs* are guaranteed because, all the qualified (leaf and non-leaf) entry pairs are pushed into each local min-heap in ascending order of their *mindist*s, and every entry pair is evaluated at most *once* and is popped right after evaluation. □

Treatment for different tree heights If the COM-trees M_P and M_Q have *different* heights, RMA needs to process entry pairs at different levels. In general, there are two approaches for treating different heights, i.e., "*fix-at-root*" and "*fix-at-leaf*." Following [18], in this paper, we take the "*fix-at-leaf*" strategy, which processes intermediate entry pairs as usual. However, for leaf and intermediate entry pairs, it stops propagating downwards the leaf entry, while propagates downwards the other intermediate entry.

Algorithm 2 Iterative MkCP Algorithm (IMA)

Input: a COM-tree M_P, a COM-tree M_Q
Output: the result set S_R of an MkCP query
/* H: a *global* min-heap storing entry pairs in ascending order of *mindist*. */
1: $H = \emptyset$ and $maxCPD_k = \infty$
2: update $maxCPD_k$ using Lemmas 1-2 and insert the root entry pairs $\langle E_P, E_Q\rangle$ not pruned by Rule 1 into H
3: **while** $H \neq \emptyset$ **do**
4: pop the top entry pair $\langle E_P, E_Q\rangle$ from H
5: **if** $mindist(E_P, E_Q) > maxCPD_k$ **then**
6: **break** // early termination
7: IMA-PEP(E_P, E_Q, H)
8: **return** S_R

Function: IMA-PEP (E_P, E_Q, H)
9: **if** E_P and E_Q are intermediate entries pointing to non-leaf nodes **then**
10: PRUNE(E_P, E_Q, H) // see Algorithm 1
11: **else if** E_P and E_Q are intermediate entries pointing to leaf nodes **then**
12: PRUNE(E_P, E_Q, S_R)

4.2 Iterative MkCP algorithm

In order to avoid *recursion* and visiting *unnecessary* entry pairs (e.g., $\langle E_{P6}, E_{Q3}\rangle$ in Example 1 as depicted in Appendix A), we develop *Iterative MkCP Algorithm* (IMA), as presented in Algorithm 2, which follows the *best-first* paradigm. In the first place, IMA initializes $maxCPD_k$ to infinity, and a global min-heap H to empty which stores the entry pairs in ascending order of *mindist* (line 1). Then, it updates $maxCPD_k$ using Lemmas 1–2, and meanwhile inserts qualified root entry pairs that cannot be pruned by Rule 1 into H (line 2). Thereafter, IMA evaluates iteratively every head entry pair $\langle E_P, E_Q\rangle$ of H having the *smallest mindist* until $mindist(E_P, E_Q) > maxCPD_k$ or $H = \emptyset$ (lines 3–7). Specifically, for each top entry pair $\langle E_P, E_Q\rangle$ of H, it determines whether $mindist(E_P, E_Q)$ is larger than $maxCPD_k$. If yes, the algorithm stops, and returns the result set S_R. Otherwise, IMA invokes the function IMA-PEP (line 7) to insert qualified subentry pairs into H (lines 9–10) or update the result set S_R (lines 10–11). Finally, IMA returns the result set S_R (line 8).

We illustrate IMA using the running example depicted in Fig. 7, and please refer to Appendix B for details.

The correctness proof for IMA is similar as that for RMA and thus omitted for space saving. Next, we analyze its I/O efficiency as follows.

Lemma 4 *IMA only visit the intermediate entry pairs* $\langle E_P, E_Q\rangle$ *with* $mindist(E_P, E_Q) \leq CPD_k$ *at most once.*

Proof Assume, to the contrary, that IMA visit an intermediate entry pair $\langle E_P, E_Q\rangle$ with $mindist(E_P, E_Q) > CPD_k$. Consider all the intermediate entry pairs $\langle E'_P, E'_Q\rangle$ that contain all answer object pairs $\langle p, q\rangle$. We can get $mindist(E'_P, E'_Q) \leq d(p, q) \leq CPD_k$. Since IMA visits entry pairs in ascending order of *mindist*s, all $\langle E'_P, E'_Q\rangle$ are visited before $\langle E_P, E_Q\rangle$, and hence, $maxCPD_k$ is updated to CPD_k before accessing $\langle E_P, E_Q\rangle$. Therefore, $\langle E_P, E_Q\rangle$ can be pruned by Rule 1, which contradicts our assumption. In order to complete the proof, we need to show that an entry pair is not visited *multiple times*. This is straightforward because each entry pair is evaluated *a single once* and popped right after evaluation. □

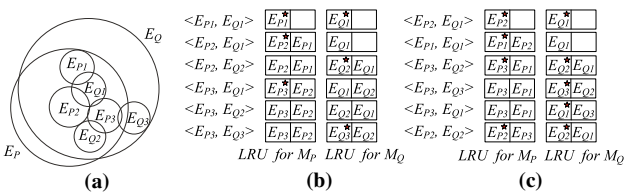

Fig. 8 Example of the I/O cost under LRU buffers. **a** E_P and E_Q. **b** Depth-first. **c** Best-first

4.3 Estimation-based hybrid MkCP algorithm

As pointed out in [18], recursion favors LRU buffer(s) in the backtracking phase. Thus, although RMA could visit unnecessary entry pairs, it might achieve better I/O performance than IMA in the presence of LRU. To further reduce I/O cost under LRU, we present a *hybrid* traversal paradigm, which combines the best-first (BF) and depth-first (DF) strategies. In particular, it utilizes a stack S to access the intermediate entry pairs $\langle E_P, E_Q\rangle$ with $mindist(E_P, E_Q) = 0$ in the DF fashion, and maintains min-heaps to visit other intermediate entry pairs in the BF manner. Take Fig. 8 as an example. Figure 8b, c depict the contents of LRU buffers, for visiting the entry pairs (shown in Fig. 8a) with *mindist* = 0 in DF and BF paradigms, respectively. In this case, we use two LRU buffers for M_P and M_Q, respectively, and red stars in each LRU buffer represent *page faults*. We can observe that it only needs 6 page faults for the DF paradigm, which is *smaller* than 9 for the BF paradigm. Note that, in real-life applications, it is likely that object sets P and Q overlap, resulting in lots of entry pairs having *mindist* = 0. Hence, the hybrid traversal paradigm might achieve the best I/O performance in most cases, which is also confirmed by our experiments (see Sect. 6.2).

Algorithm 3 Estimation based Hybrid MkCP Algorithm (EHM)

Input: M_P, M_Q, an estimated value $eCPD_k$ of CPD_k
Output: the result set S_R of an MkCP query
/* S: a stack storing $\langle E_P, E_Q\rangle$ with $mindist(E_P, E_Q) = 0$, H: a min-heap storing $\langle E_P, E_Q\rangle$ with $0 < mindist(E_P, E_Q) \leq \min\{eCPD_k, maxCPD_k\}$, EH: a min-heap storing $\langle E_P, E_Q\rangle$ with $emindist(E_P, E_Q) > eCPD_k$, CH: a min-heap storing $\langle E_P, E_Q\rangle$ with $mindist(E_P, E_Q) > eCPD_k$. */
1: $S = H = EH = CH = \emptyset$ and $maxCPD_k = \infty$
2: EHM-AP(S, H, EH, CH) // see Algorithm 4
3: EHM-C(EH, CH) // see Algorithm 5
4: **return** S_R

In addition, in RMA and IMA, $maxCPD_k$ is initially set to infinity, and becomes smaller as the algorithms proceed. Nonetheless, the adaptation of $maxCPD_k$ value has a crucial impact on the performance of MkCP retrieval, since $maxCPD_k$ is employed by Rules 1–2 to prune the search

space. If $maxCPD_k$ approaches to the real CPD_k slowly, which is true especially for larger k, the early stage of the algorithms cannot efficiently shrink the search space. To address this, our third algorithm, namely *Estimation-based Hybrid MkCP Algorithm* (EHM), whose pseudo-code is shown in Algorithm 3. EHM introduces a new pruning metric $eCPD_k$, an estimation value of CPD_k, for *aggressive pruning*, and we defer the detailed discussion of $eCPD_k$ computation later. Specifically, in the aggressive pruning step, (1) $maxCPD_k$, like RMA and IMA, is utilized to prune away unqualified entry pairs, and (2) $eCPD_k$ is used to further prune the search space. However, it will become too aggressive by choosing an *underestimated* value, i.e., $eCPD_k$ is smaller than CPD_k. To avoid any *false dismissal*, two min-heaps EH and CH are employed to store the entry pairs pruned by $eCPD_k$. After the aggressive pruning stage, EHM needs to search the final result set S_R in EH and CH for *compensation*.

Algorithm 4 Aggressive Prunnig Algorithm for EHM (EHM-AP)

Input: a stack S, min-heaps H, EH, and CH
1: update $maxCPD_k$ using Lemmas 1-2, and insert the root entry pairs $\langle E_P, E_Q\rangle$ not pruned by Rule 1 into S or H or CH according to $mindist(E_P, E_Q)$
2: **while** $S \neq \emptyset$ **do**
3: pop the top entry pair $\langle E_P, E_Q\rangle$ from S
4: EHM-PEP(E_P, E_Q, S, H, EH, CH)
5: **while** $H \neq \emptyset$ **do**
6: pop the top entry pair $\langle E_P, E_Q\rangle$ from H
7: **if** $mindist(E_P, E_Q) > maxCPD_k$ **then**
8: **return** // early termination
9: EHM-PEP(E_P, E_Q, S, H, EH, CH)

Function: EHM-PEP(E_P, E_Q, S, H, EH, CH)
10: **if** E_P and E_Q are intermediate entries pointing to non-leaf nodes **then**
11-16: these lines are identical to lines 14-19 of Algorithm 1
17: **for** each entry $e_{Pi} \in E_P$ not pruned **do**
18: **for** each entry $e_{Qj} \in E_Q$ not pruned **do**
19: **if** $emindist(e_{Pi}, e_{Qj}) \leq maxCPD_k$ **then** // Rule 2
20: **if** $emindist(e_{Pi}, e_{Qj}) > eCPD_k$ **then**
21: insert $\langle e_{Pi}, e_{Qj}\rangle$ into EH // for compensation later
22: **else if** $mindist(e_{Pi}, e_{Qj}) \leq maxCPD_k$ **then** // Rule 1
23: update $maxCPD_k$ using Lemmas 1-2
24: **if** $mindist(e_{Pi}, e_{Qj}) = 0$ **then**
25: push $\langle e_{Pi}, e_{Qj}\rangle$ into S // for DF traversal later
26: **else if** $mindist(e_{Pi}, e_{Qj}) > eCPD_k$ **then**
27: insert $\langle e_{Pi}, e_{Qj}\rangle$ into CH // for compensation later
28: **else**
29: insert $\langle e_{Pi}, e_{Qj}\rangle$ into H // for BF traversal later
30: **else if** E_P and E_Q are intermediate entries pointing to leaf nodes **then**
31-42: these lines are identical to lines 11-22
43: update S_R and $maxCPD_k$

Algorithm 5 Compensation Algorithm for EHM (EHM-C)

Input: min-heaps EH and CH
1: **while** $EH \neq \emptyset$ or $CH \neq \emptyset$ **do**
2: get top entries $\langle E_P, E_Q\rangle$ and $\langle E_P', E_Q'\rangle$ from EH and CH respectively
3: **if** $\min\{emindist(E_P, E_Q), mindist(E_P', E_Q')\} > maxCPD_k$ **then**
4: **return** //early termination
5: **if** $emindist(E_P, E_Q) < mindist(E_P', E_Q')$ **then**
6: pop the top entry pair $\langle E_P, E_Q\rangle$ from EH
7: **if** $mindist(E_P, E_Q) \leq maxCPD_k$ **then** // Rule 1
8: **if** E_P and E_Q are leaf entries **then**
9: update S_R and $maxCPD_k$
10: **else**
11: insert $\langle E_P, E_Q\rangle$ into CH
12: **else**
13: pop the top entry pair $\langle E_P', E_Q'\rangle$ from CH
14: IMA-PEP(E_P', E_Q', CH) // see Algorithm 2

Algorithm 4 depicts the pseudo-code of EHM-AP. Initially, EHM-AP updates $maxCPD_k$ using Lemmas 1–2, and adds the qualified root entry pairs $\langle E_P, E_Q\rangle$ that are not discarded by Rule 1 to S or H or CH according to $mindist(E_P, E_Q)$ (line 1). Then, it performs a *while-loop* (lines 2–4) to visit entry pairs in S until S is empty. Each time, the algorithm pops the head entry pair $\langle E_P, E_Q\rangle$ of S, and calls the function EHM-PEP to expand $\langle E_P, E_Q\rangle$. Next, EHM-AP runs another while-loop (lines 5–9) to visit entry pairs in H until H is empty. Specifically, it deheaps the top entry from H, and determines whether the early termination condition is satisfied. If yes, the algorithm stops, otherwise, it invokes EHM-PEP to expand $\langle E_P, E_Q\rangle$. In EHM-PEP, no matter whether the currently visited entry pair $\langle E_P, E_Q\rangle$ pointing to non-leaf nodes or leaf nodes, it first prunes $e_{Pi} \in E_P$ and $e_{Qj} \in E_Q$, respectively, similar as the function PRUNE (depicted in Algorithm 1), and then, for all the remaining entries e_{Pi} and e_{Qj} not pruned, it inserts all the entry pairs $\langle e_{Pi}, e_{Qj}\rangle$ with $eCPD_k < emindist(e_{Pi}, e_{Qj}) \leq maxCPD_k$ into EH (lines 11–21 and 31–42). Thereafter, if $mindist(e_{Pi}, e_{Qj}) \leq maxCPD_k$, (1) for non-leaf entry pairs $\langle e_{Pi}, e_{Qj}\rangle$, EHM-PEP updates $maxCPD_k$ using Lemmas 1–2, and adds $\langle e_{Pi}, e_{Qj}\rangle$ to S or CH or H based on $mindist(e_{Pi}, e_{Qj})$ (lines 22–29), and (2) for leaf entry pairs, it updates S_R and $maxCPD_k$ (line 43).

The pseudo-code of EHM-C is presented in Algorithm 5. It performs a while-loop until both EH and CH are empty. In every loop, EHM-C gets the top entry pairs $\langle E_P, E_Q\rangle$ and $\langle E_P', E_Q'\rangle$ from EH and CH, respectively. If the *minimal* value between $emindist(E_P, E_Q)$ and $mindist(E_P', E_Q')$ is *larger* than $maxCPD_k$, the algorithm terminates, since EH and CH can not contain any actual answer object pair (lines 3–4). Otherwise, EHM-C compares $emindist(E_P, E_Q)$ with $mindist(E_P', E_Q')$. If $emindist(E_P, E_Q) < mindist(E_P', E_Q')$, EHA-C pops the head entry pair $\langle E_P, E_Q\rangle$ from EH, and then checks whether $\langle E_P, E_Q\rangle$ can be pruned by Rule 1 (line 7). If not, (1) for the leaf entry pair, EHM-C updates S_R and $maxCPD_k$ (lines 8–9); and (2) for the non-leaf entry pair, it inserts $\langle E_P, E_Q\rangle$ into CH for evaluation later (lines 10–11). If $emindist(E_P, E_Q) \geq mindist(E_P', E_Q')$, EHM-C pops the top entry pair $\langle E_P', E_Q'\rangle$ from CH, and calls IMA-PEP (depicted in Algorithm 2) to expand $\langle E_P', E_Q'\rangle$.

We illustrate EHM using the running example shown in Fig. 7, and please refer to Appendix C for details.

Lemma 5 *EHM can return exactly the actual MkCP query result, i.e., the algorithm has no false negatives, no false positives, and the result set contains no duplicate objects.*

Proof First, no real answer object pairs are missed (i.e., *no false negatives*), as only unqualified (leaf and non-leaf) entry pairs are discarded by Rules 1–2. In particular, for EHM, the

entry pairs pruned using the aggressive pruning technique are preserved (not discarded), to be verified in the compensation phase. Second, all object entry pairs that cannot be discarded by Rules 1–2 are verified against other qualified entry pairs to ensure *no false positives*. Third, *no duplicate object pairs* are guaranteed because every qualified entry pair is pushed into only one corresponding min-heap according to its *mindist* and is popped right after evaluation. □

Lemma 6 *EHM only visit the intermediate entry pairs* $\langle E_P, E_Q\rangle$ *with* $mindist(E_P, E_Q) \le CPD_k$ *at most once.*

Proof Assume, to the contrary, that EHM visits an intermediate entry pair $\langle E_P, E_Q\rangle$ with $mindist(E_P, E_Q) > CPD_k$. For all the intermediate entry pairs $\langle E'_P, E'_Q\rangle$ that contain the answer object pairs $\langle p, q\rangle, mindist(E'_P, E'_Q) \le d(p, q) \le CPD_k$. EHM accesses the intermediate entry pairs whose *mindist* equals to 0 in DF paradigm, and other intermediate entry pairs not pruned in BF paradigm; thus, in general, EHM visits entry pairs in ascending order of *mindist*s. Hence, all $\langle E'_P, E'_Q\rangle$ are visited before $\langle E_P, E_Q\rangle$, and $maxCPD_k$ is updated to CPD_k before accessing $\langle E_P, E_Q\rangle$. Thus, $\langle E_P, E_Q\rangle$ can be pruned by Rule 1, which contradicts with our assumption. In order to complete the proof, we need to show that an entry pair is not visited *multiple times*. This is straightforward as each entry pair is evaluated *a single once* and is popped right after evaluation.

$eCPD_k$ computation A challenging issue for EHM is to obtain an appropriate value of $eCPD_k$. If the value of $eCPD_k$ is too *big*, it can not solve the *slow start*. Otherwise, if a *small* $eCPD_k$ is used, it needs *additional cost* to perform compensation. To estimate CPD_k accurately, we can utilize the distance distribution to obtain the $eCPD_k$ value. The overall distribution of distances over two metric datasets P and Q is defined as:

$$F(r) = \mathbf{Pr}\{d(p, q) \le r\} \tag{1}$$

where p is a random object in P, and q is a random object in Q. The distance distribution is the correct counterpart of data distribution used for Euclidean spaces, since it is the natural way to characterize metric datasets.

Based on the above definition of distance distribution F, CPD_k can be estimated as the minimal r that has at least k object pairs with their distances bounded by r:

$$eCPD_k = min\{r||P| \times |Q| \times F(r) \ge k\} \tag{2}$$

As an example, assume that $F(r)$ follows Uniform distribution in the range [0, 1] (i.e., $0 \le r \le 1$), then $eCPD_k$ can be set to $k/(|P| \times |Q|)$. Note that, $eCPD_k$ calculated by formula (2) can also be used in our cost model (see Sect. 4.4) to estimate CPD_k.

4.4 Cost model

Next, we drive a node-based cost model for MkCP retrieval because, (1) it can be utilized by databases to choose promising execution plans, and (2) it can find out the factors that may affect the cost of an MkCP query and thus facilitate better algorithm development.

In order to take a more general view and be able to deal with generic metric spaces, our cost model relies on *distance* (rather than *data*) distribution F, as F is the only information derivable from the analysis of metric datasets.

For an I/O optimal MkCP search algorithm, an entry pair $\langle E_P, E_Q\rangle$ has to be accessed iff $mindist(E_P, E_Q) \le CPD_k$ holds. Then, the probability of visiting $\langle E_P, E_Q\rangle$ can be denoted as:

$$\begin{aligned}&\mathbf{Pr}(\langle E_P, E_Q\rangle \text{ is accessed})\\ &= \mathbf{Pr}(mindist(E_P, E_Q) \le CPD_k)\\ &= \mathbf{Pr}(d(E_P.RO, E_Q.RO) \le E_P.r + E_Q.r + CPD_k)\\ &= F(E_P.r + E_Q.r + CPD_k)\end{aligned} \tag{3}$$

To determine the *expected I/O cost* (*EIC*) in terms of node (i.e., intermediate entry) accesses, it is sufficient to sum the above probabilities over all intermediate entry pairs between M_P and M_Q:

$$EIC = 2 \times \sum_{i=1}^{|M_P|\times|M_Q|} F(E_P.r + E_Q.r + CPD_k) \tag{4}$$

where CPD_k is set to the value calculated by Eq. (2).

Next, we estimate the CPU cost for MkCP retrieval, in terms of *selectivity* [14] which is the ratio of distance computations to the total number of object pairs, i.e., # of distance computations $/(|P| \times |Q|)$. As $|P| \times |Q|$ can be easily computed, we only need to obtain the number of distance computations during MkCP search.

When expanding an intermediate entry pair $\langle E_P, E_Q\rangle$, our proposed algorithms first prune away $e_{Pi} \in E_P$ and $e_{Qj} \in E_Q$, respectively, for minimizing the cost of checking the Cartesian product of $\langle e_{Pi}, e_{Qj}\rangle$, and then evaluate every qualified entry pair $\langle e_{Pi}, e_{Qj}\rangle$. Therefore, the *number of distance computations* (*NDC*) for processing every $\langle E_P, E_Q\rangle$ can be calculated as:

$$\begin{aligned}NDC(E_P, E_Q) = {} & E_P.num \times \mathbf{Pr}(mindist(e_{Pi}, E_Q) \text{ is computed})\\ & + E_Q.num \times \mathbf{Pr}(mindist(E_P, e_{Qj}) \text{ is computed}) + E_P.num\\ & \times E_Q.num \times \mathbf{Pr}(mindist(e_{Pi}, e_{Qj}) \text{ is computed})\end{aligned} \tag{5}$$

For ease of analysis, we approximate $maxCPD_k$ with CPD_k. Hence, a distance (i.e., *mindist*) is needed to calculate for the leaf or non-leaf entry pair $\langle E_P, E_Q\rangle$ iff $emindist(E_P, E_Q) \le CPD_k$. As $emindist(e_{Pi}, E_Q) =$

$|d(E_P.RO,\ E_Q.RO) - e_{Pi}.PD| - e_{Pi}.r - E_Q.r$, the probability that $mindist(e_{Pi},\ E_Q)$ needs to be computed can be denoted as:

$$\begin{aligned}
&\mathbf{Pr}(mindist(e_{Pi}, E_Q) \text{ is computed}) \\
&= \mathbf{Pr}(emindist(e_{Pi}, E_Q) \le CPD_k) \\
&= \mathbf{Pr}(|d(E_P.RO, E_Q.RO) - e_{Pi}.PD| \\
&\quad \le e_{Pi}.r + E_Q.r + CPD_k) \\
&= F(e_{Pi}.PD + e_{Pi}.r + E_Q.r + CPD_k) \\
&\quad - F(e_{Pi}.PD - e_{Pi}.r - E_Q.r - CPD_k - \delta)
\end{aligned} \tag{6}$$

In Eq. (6), $\delta = 0$ when F is continuous (e.g., L_P-*norm*), and $\delta = 1$ when F is discrete (e.g., *edit distance*). Similarly, since $emindist(E_P,\ e_{Qj}) = |d(E_P.RO,\ E_Q.RO) - e_{Qj}.PD| - E_P.r - e_{Qj}.r$, the probability that $mindist(E_P,\ e_{Qj})$ needs to be calculated can be denoted as:

$$\begin{aligned}
&\mathbf{Pr}(mindist(E_P, e_{Qj}) \text{ is computed}) \\
&= F(e_{Qj}.PD + E_P.r + e_{Qj}.r + CPD_k) \\
&\quad - F(e_{Qj}.PD - E_P.r - e_{Qj}.r - CPD_k - \delta)
\end{aligned} \tag{7}$$

For every entry pair $\langle e_{Pi},\ e_{Qj}\rangle$, $emindist(e_{Pi},\ e_{Qj}) = \max\{|d(e_{Pi}.RO,\ E_Q.RO) - e_{Qj}.PD|,\ |d(E_P.RO,\ e_{Qj}.RO) - e_{Pi}.PD|\} - e_{Pi}.r - e_{Qj}.r$. Let λ be $e_{Pi}.r + e_{Qj}.r + CPD_k$. Since $\langle e_{Pi}.RO,\ E_Q.RO\rangle$ and $\langle E_P.RO,\ e_{Qj}.RO\rangle$ are two independent object pairs, the probability of computing $mindist(e_{Pi},\ e_{Qj})$ is expressed as:

$$\begin{aligned}
&\mathbf{Pr}(mindist(e_{Pi}, e_{Qj}) \text{ is computed}) \\
&= \mathbf{Pr}(|d(e_{Pi}.RO, E_Q.RO) - e_{Qj}.PD| \le \lambda) \\
&\quad \times \mathbf{Pr}(|d(E_P.RO, e_{Qj}.RO) - e_{Pi}.PD| \le \lambda) \\
&= (F(e_{Qj}.PD + \lambda) - F(e_{Qj}.PD - \lambda - \delta)) \\
&\quad \times (F(e_{Pi}.PD + \lambda) - F(e_{Pi}.PD - \lambda - \delta))
\end{aligned} \tag{8}$$

To determine the *expected selectivity cost* (*ESC*), it is sufficient to sum the above *NDC* over all intermediate entry pairs between M_P and M_Q with respect to $|P| \times |Q|$:

$$ESC = \frac{\sum_{i=1}^{|M_P|\times|M_Q|} NDC(E_P, E_Q)}{|P| \times |Q|} \tag{9}$$

In summary, Eqs. (4) and (9) indicate that the query cost of MkCP search depends on several factors: (1) the cardinalities of object sets, (2) the M-tree/COM-tree structure, (3) the value of k, and (4) the distance distribution $F(r)$, which are verified one by one in Sect. 6.2. Note that, although the data distribution is an important factor that might affect query cost in the Euclidean space, the distance distribution is an counterpart used for the metric space, which can be obtained by the sampled dataset.

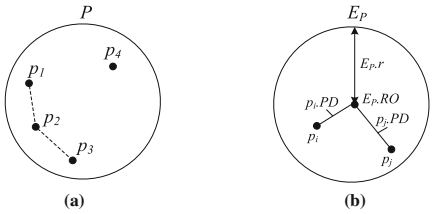

Fig. 9 Example of SMkCP search. **a** Illustration of a SM2CP query. **b** Illustration of Lemma 7

5 Extensions

In this section, we study two interesting variants of MkCP queries, i.e., *Self Metric kCP* (SMkCP) *search* and *Approximate Metric kCP* (AMkCP) *retrieval*, and present how our proposed algorithms and pruning rules can be adapted accordingly to solve these variations.

5.1 Self MkCP search

The MkCP query focuses on two different object sets, i.e., $P \neq Q$. However, in some real-life applications [6,18,30], two object sets may be identical (i.e., $P = Q$). As an example, clustering [30] and outlier detecting [6] algorithms aim at the same object set. Motivated by this, we introduce a natural variant of MkCP queries, namely Self Metric kCP (SMkCP) retrieval.

Definition 5 (*SMkCP Search*). Given an object set P in a metric space and an integer k ($1 \le k \le (|P|^2 - |P|)/2$), *a Self Metric* kCP (SMkCP) *query* finds k ordered different closest object pairs $\langle p_i,\ p_j\rangle$ with $p_i,\ p_j \in P$ and $p_i \neq p_j$.

An example of SMkCP search is depicted in Fig. 9, where $P = \{p_1,\ p_2,\ p_3,\ p_4\}$ and $k = 2$. The result set of the SM2CP query is $\{\langle p_1,\ p_2\rangle,\ \langle p_2,\ p_3\rangle\}$.

5.1.1 Estimation-based hybrid SMkCP algorithm

Our algorithms designed for MkCP queries (discussed in Sect. 4) can be flexible to support SMkCP retrieval. Since $\langle p_i,\ p_j\rangle$ and $\langle p_j,\ p_i\rangle$ are treated as the same object pair, and $\langle p_i,\ p_i\rangle$ cannot be contained in the final result set of SMkCP search, we need to perform the M($|P| + 2$k)CP query and filter out unqualified and duplicated object pairs. Nevertheless, it is very inefficient as $|P| + 2k$ is much bigger than k, especially for larger object set P. In order to further improve the SMkCP query performance, we develop *Estimation-based Hybrid SMkCP Algorithm* (EHS), which takes advantage of EHM algorithm (i.e., it performs the best as verified by our experiments in Sect. 6.2), and meanwhile integrates the characteristics of SMkCP retrieval.

Since SMkCP search performs on the same object set, i.e., $P = Q$, the intermediate entries of the entry pair $\langle E_P, E_Q \rangle$ to be processed can be identical. However, Lemmas 1–2, used to derive $maxCPD_k$ values, are only suitable when given two intermediate entries are different. Thus, we present a new lemma, to cover the situation when E_P and E_Q point to the same entry.

Lemma 7 *Given an intermediate entry* E_P, $maxCPD_k =$

$$\begin{cases} E_P.r & E_P.num > k \\ 2 \times E_P.r & E_P.num \leq k \leq (E_P.num^2 - E_P.num)/2 \end{cases}$$

Proof In order to prove this lemma, $maxCPD_k$ can be updated to r if having k different object pairs with their distances bounded by r.

If $E_P.num > k$, there exists k different object pairs $\langle p_i, E_P.RO \rangle (p_i \neq E_P.RO, p_i \in E_P)$ with their distances $d(p_i, E_P.RO)$ bounded by $E_P.r$, as $E_P.RO$ is a real object. Hence, $maxCPD_k$ can be set to $E_P.r$ when $E_P.num > k$.

Otherwise, i.e., $E_P.num \leq k$, as depicted in Fig. 9b, there exists $(E_P.num^2 - E_P.num)/2$ different object pairs $\langle p_i, p_j \rangle$ in ST_{E_P} with their distances $d(p_i, p_j) \leq p_i.PD + p_j.PD \leq 2 \times E_P.r$, due to the triangle inequality. Consequently, $maxCPD_k$ can be set as $2 \times E_P.r$ if $E_P.num \leq k \leq (E_P.num^2 - E_P.num)/2$. □

The framework of EHS is similar as EHM, the only difference is that, EHM calls EHM-AP for aggressive pruning, whereas EHS invokes EHS-AP, which is presented in Algorithm 6. Initially, it updates $maxCPD_k$ using Lemmas 1, 2, and 7, and inserts root entry pairs $\langle E_P, E_Q \rangle (E_P \in M_P, E_Q \in M_P)$ not pruned by Rule 1 into S (for the DF traversal) or H (for the BF traversal) or CH (for compensation), according to $mindist(E_P, E_Q)$ (line 1). Then, EHS-AP visits each entry pair $\langle E_P, E_Q \rangle$ of S until S is empty, and calls the EHS-PEP function to expand $\langle E_P, E_Q \rangle$ (lines 2–4). Next, EHS-AP visits every entry pair $\langle E_P, E_Q \rangle$ of H until H is empty or $mindist(E_P, E_Q) > maxCPD_k$, and invokes EHS-PEP to expand $\langle E_P, E_Q \rangle$ (lines 5–9).

In EHS-PEP, if $E_P \neq E_Q$, it calls EHM-PEP directly (line 33). Otherwise, i.e., $E_P = E_Q$, if E_P points to non-leaf nodes, for each sub entry e_{Pi} of E_P, it pushes $\langle e_{Pi}, e_{Pi} \rangle$ into S and updates $maxCPD_k$ using Lemma 7 (line 13). In order to avoid duplicated entry pair accesses, for every sub entry e_{Pj} of E_P stored after e_{Pi}, it inserts $\langle e_{Pi}, e_{Pj} \rangle$ into EH if $eCPD_k < emindist(e_{Pi}, e_{Pj}) \leq maxCPD_k$, or inserts $\langle e_{Pi}, e_{Pj} \rangle$ into S, or H, or CH based on $mindist(e_{Pi}, e_{Pj})$. If E_P points to leaf nodes, in order to avoid duplicated object pair accesses, for each sub entry e_{Pi} of E_P and e_{Pj} of E_P stored after e_{Pi}, it adds $\langle e_{Pi}, e_{Pj} \rangle$ to EH if $eCPD_k < emindist(e_{Pi}, e_{Pj}) \leq maxCPD_k$ (line 29), or updates S_R and $maxCPD_k$ if $mindist(e_{Pi}, e_{Pj}) \leq maxCPD_k$ (line 31).

Algorithm 6 Aggressive Prunning Algorithm for EHS (EHS-AP)

Input: a stack S, min-heaps H, EH, and CH
/* S: a stack storing $\langle E_P, E_Q \rangle$ with $mindist(E_P, E_Q) = 0$, H: a min-heap storing $\langle E_P, E_Q \rangle$ with $0 < mindist(E_P, E_Q) \leq \min\{eCPD_k, maxCPD_k\}$, EH: a min-heap storing $\langle E_P, E_Q \rangle$ with $emindist(E_P, E_Q) > eCPD_k$, CH: a min-heap storing $\langle E_P, E_Q \rangle$ with $mindist(E_P, E_Q) > eCPD_k$. */
1: update $maxCPD_k$ using Lemmas 1, 2, and 4, and insert the root entry pairs $\langle E_P, E_Q \rangle$ ($E_P \in M_P, E_Q \in M_P$) not pruned by Rule 1 into S or H or CH according to $mindist(E_P, E_Q)$
2: **while** $S \neq \emptyset$ **do**
3: pop the top entry pair $\langle E_P, E_Q \rangle$ from S
4: EHS-PEP(E_P, E_Q, S, H, EH, CH)
5: **while** $H \neq \emptyset$ **do**
6: pop the top entry pair $\langle E_P, E_Q \rangle$ from H
7: **if** $mindist(E_P, E_Q) > maxCPD_k$ **then**
8: **return** //early termination
9: EHS-PEP(E_P, E_Q, S, H, EH, CH)

Function: EHS-PEP(E_P, E_Q, S, H, EH, CH)
10: **if** $E_P = E_Q$ **then**
11: **if** E_P is an intermediate entry pointing to a non-leaf node **then**
12: **for** each entry $e_{Pi} \in E_P$ **do**
13: push $\langle e_{Pi}, e_{Pi} \rangle$ in S and update $maxCPD_k$ using Lemma 7
14: **for** each entry $e_{Pj} \in E_P$ stored after e_{Pi} **do**
15: **if** $emindist(e_{Pi}, e_{Pj}) \leq maxCPD_k$ **then** // Rule 2
16: **if** $emindist(e_{Pi}, e_{Pj}) > eCPD_k$ **then**
17: insert $\langle e_{Pi}, e_{Pj} \rangle$ into EH
18: **else if** $mindist(e_{Pi}, e_{Pj}) = 0$ **then**
19: insert $\langle e_{Pi}, e_{Pj} \rangle$ into S // for DF traversal later
20: **else if** $mindist(e_{Pi}, e_{Pj}) > eCPD_k$ **then**
21: insert $\langle e_{Pi}, e_{Pj} \rangle$ into CH
22: **else if** $mindist(e_{Pi}, e_{Pj}) \leq maxCPD_k$ **then** // Rule 1
23: insert $\langle e_{Pi}, e_{Pj} \rangle$ into H
24: **else if** E_P is an intermediate entry pointing to a leaf node **then**
25: **for** each entry $e_{Pi} \in E_P$ **do**
26: **for** each entry $e_{Pj} \in E_P$ stored after e_{Pi} **do**
27: **if** $emindist(e_{Pi}, e_{Pj}) \leq maxCPD_k$ **then** // Rule 2
28: **if** $emindist(e_{Pi}, e_{Pj}) > eCPD_k$ **then**
29: insert $\langle e_{Pi}, e_{Pj} \rangle$ into EH
30: **else if** $mindist(e_{Pi}, e_{Pj}) \leq maxCPD_k$ **then** // Rule 1
31: update S_R and $maxCPD_k$
32: **else** // $E_P \neq E_Q$
33: EHM-PEP(E_P, E_Q, S, H, EH, CH) // see Algorithm 4

We illustrate EHS using the SM2CP ($k = 2$) query on the object set O shown in Fig. 2a, and please refer to Appendix D for details.

5.1.2 COMdnn-tree-based SMkCP algorithm

Although EHS utilizes the characteristics of SMkCP search to boost query efficiency, it takes the framework of EHM, in which the performance degenerates as the overlapping between two object sets increases, as confirmed in Sect. 6.2. This is because, for two object sets with greater percentage of overlapping, EHM has to visit a lot of entry pairs with smaller *mindist*. However, SMkCP retrieval performs on one object set, i.e., the overlapping percentage is 100 %, resulting in poor query performance. To this end, we introduce a variant of COM-trees, namely COMdnn-tree, which integrates the nearest neighbor (NN) information into a COM-tree to improve the efficiency of SMkCP query processing. Note that, the COMdnn-tree can support efficient MkCP and SMkCP queries simultaneously.

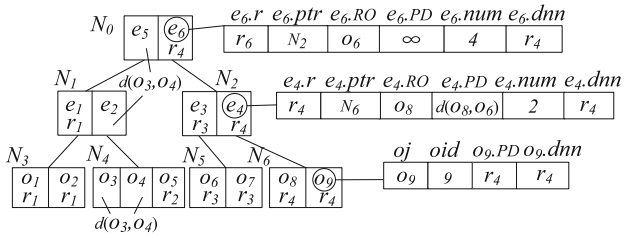

Fig. 10 Example of a COMdnn-tree

Figure 10 shows an example of COMdnn-tree on the object set depicted in Fig. 2a, which includes *e.dnn* in each leaf or non-leaf entry e to represent the NN distance of the object or the minimum NN distance for all the objects contained in ST_e. For instance, $o_9.dnn = r_4$, since the distance from o_9 to its NN (i.e., o_{10}) equals to r_4, and $e_6.dnn = r_4$, as r_4 is the minimum *dnn* for all objects o_6, o_7, o_8, and o_9 contained in ST_{e_6}. Note that, the number associated with every entry denotes *e.dnn*, which is obtained and stored during the construction of COMdnn-tree. In particular, when an object o is to be inserted into the COMdnn-tree, a NN query is performed to get its *dnn* value, and a reverse NN query is also conducted to find the objects whose *dnn* values could be affected by o.

Lemma 8 *Given an object set P in generic metric spaces, if* $\langle p_i, p_j \rangle$ *is contained in the SMkCP query result, then* $p_i.dnn$ *and* $p_j.dnn$ *are not larger than* $p_{2k}.dnn$*, where* $p_i.dnn$ *denotes* p_i*'s NN distance in P, and* $p_{2k}.dnn$ *represents the 2kth NN distance among all sorted objects in P.*

Proof To prove this lemma, we first prove that $maxCPD_k$ can be set to $p_{2k}.dnn$. Since the objects in P are sorted according to their *dnn* values, there exists $2k$ object pairs $\langle p_i, NN(p_i)\rangle$ $(1 \le i \le 2k)$ with their distances bounded by $p_{2k}.dnn$. Since p_i and $NN(p_i)$ might be mutual NN, i.e., $\langle p_i, NN(p_i)\rangle$ and $\langle NN(p_i), p_i\rangle$ representing the same object pair, there exists at least k different object pairs with their distances bounded by $p_{2k}.dnn$. Thus, $maxCPD_k$ can be set to $p_{2k}.dnn$. If $\langle p_i, p_j\rangle$ is contained in the SMkCP query result, then $d(p_i, p_j) \le CPD_k \le maxCPD_k = p_{2k}.dnn$. Due to the definition of NN, $p_i.dnn \le d(p_i, p_j)$ and $p_j.dnn \le d(p_i, p_j)$, and both $p_i.dnn$ and $p_j.dnn$ are not larger than $p_{2k}.dnn$, which completes the proof. □

According to Lemma 8, we present *COMdnn-tree-based SMkCP Algorithm* (MSA). In the first phase, by traversing COMdnn-Tree, MSA obtains the candidate object set CH, as the *dnn* of each object in CH is not larger than $p_{2k}.dnn$. In the second phase, the algorithm verifies the object pairs in the candidate object set in order, to get the final result set.

Algorithm 7 COMdnn-Tree based SMkCP Algorithm (MSA)

Input: a COMdnn-tree M_P
Output: the result set S_R of a SMkCP query
/* H: a min-heap storing the intermediate entries E_P of M_P with $E_P.dnn \le maxCPD_k$; CH: a min-heap storing the leaf entries E_P (i.e., the candidate objects) with $E_P.dnn \le maxCPD_k$. */
1: $maxCPD_k = \infty$ and $H = CH = \emptyset$
2: insert root entries E_P of M_P into H with $E_P.dnn \le maxCPD_k$
3: **while** $H \neq \emptyset$ **do**
4: pop the top entry E_P of H
5: **if** $E_P.dnn > maxCPD_k$ **then** break // early termination
6: **if** E_P points to a non-leaf node **then**
7: insert qualified subentries e of E_P into H with $e.dnn \le maxCPD_k$
8: **else** // E_P points to a leaf node
9: insert qualified subentries e of E_P into CH and update $maxCPD_k$
10: **for** each object p_i in CH **do**
11: **if** $p_i.dnn > maxCPD_k$ **then return** S_R // early termination
12: **for** each object p_j already visited **do**
13: **if** $d(p_i, p_j) \le maxCPD_k$ **then**
14: update S_R and $maxCPD_k$
15: **return** S_R

Algorithm 7 depicts the pseudo-code of MSA. It takes the COMdnn-tree M_P as an input, and outputs the result set S_R of a SMkCP query. First, it initializes $maxCPD_k$ to infinity, and two min-heaps H and CH (line 1). Then, MSA inserts root entries E_P of M_P into H with $E_P.dnn \le maxCPD_k$, and performs a while-loop until H is empty (lines 3–9). Each time, it pops the top entry E_P of H, and verifies whether the early termination is satisfied. If E_P points to the non-leaf node, MSA inserts all the qualified sub-entries of E_P into H. Otherwise, i.e., E_P points to the leaf node, MSA inserts all the qualified sub-entries of E_P into CH and updates $maxCPD_k$ using the $2k$th *dnn* in CH. Thereafter, for each object p_i in CH, if $p_i.dnn > maxCPD_k$, the algorithm terminates and returns S_R (line 11); otherwise, in order to avoid duplicated distance computations, for each object p_j already visited (i.e., stored before p_i in CH), if $d(p_i, p_j) \le maxCPD_k$, it updates S_R and $maxCPD_k$ (lines 13–14). Finally, the result set S_R is returned (line 15).

We illustrate MSA using the SM2CP ($k = 2$) query on the object set O shown in Fig. 2a, and please refer to Appendix E for details.

5.1.3 Discussion

In this section, we first present our method to compute the estimation value $eCPD_k$, which is needed for EHS, and then analyze the I/O complexities of two algorithms. Finally, we discuss how to further improve the efficiency of MSA.

For EHS developed for SMkCP search, a challenge issue is to estimate CPD_k accurately, because it might affect the efficiency of the algorithm. Similar as EHM, to obtain $eCPD_k$, we can utilize the distance distribution over the metric dataset P, which is defined as:

$$F(r) = \mathbf{Pr}\{d(p_i, p_j) \le r\} \tag{10}$$

where p_i and p_j are two random objects in P. Based on the distance distribution F, CPD_k can be estimated as the

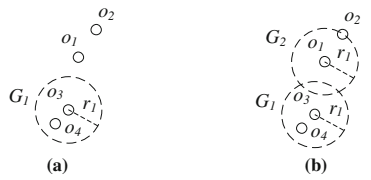

Fig. 11 Example of groups for MSA. **a** After processing o_3 and o_4. **b** After processing o_3, o_4, and o_1

minimal r that has at least k object pairs with their distances bounded by r, i.e.,

$$eCPD_k = min\{r|(|P|^2 - |P|)/2 \times F(r) \geq k\} \quad (11)$$

Next, we analyze the I/O costs of the algorithms designed for SMkCP retrieval. Let $|M_P|$ be the total number of nodes in M_P, $|M_P|_i$ be the number of nodes in the level i of M_P, and L be the height of M_P.

Lemma 9 *The I/O costs of EHS and MSA algorithms are* $O(\sum_{i=0}^{L-1} |M_P|_i^2)$ *and* $O(|M_P|)$*, respectively.*

Proof Since EHS follows the framework of EHM, which needs to visit intermediate entry pairs $\langle E_P, E_Q\rangle$ with *mindist* $(E_P, E_Q) \leq CPD_k$ according to Lemma 6, it has to access all intermediate entry pairs at the same level of M_P in the worst case. Hence, the I/O cost of EHS is $O(\sum_{i=0}^{L-1} |M_P|_i^2)$. However, according to Lemma 8, MSA only needs to traverse M_P once in worst case, to obtain all candidates with their *dnn*s within $p_{2k}.dnn$. Thus, the I/O cost of MSA is $O(|M_P|)$. □

According to Lemma 9, the I/O cost of MSA is much smaller than that of EHS, which is also verified in Sect. 6.5. In addition, it can also be demonstrated using Examples 4 and 5 (see Appendixes D and E), in which MSA only needs 4 node accesses, whereas EHS needs 13 node accesses.

For MSA, in the first phase, it traverses M_P to get $(2k+n)$ candidate objects, whose *dnn* are not larger than $p_{2k}.dnn$. Note that, n is needed when the distance ties occur, i.e., there exists more than one object whose *dnn* equals to $p_{2k}.dnn$. Then, in the second phase, MSA needs to verify all the object pairs among the candidate objects in order. Hence, the CPU cost (in terms of the number of distance computations) for MSA is O(k^2). Thus, for larger k, especially when k approaches the object set size $|P|$, MSA degenerates to the naive solution for SMkCP retrieval, which compares every object pairs to obtain the final result set.

In order to minimize the effect of quadratic cost of verification, we can partition the verified objects into disjoint groups. Similar as the intermediate entry defined in the M-tree, each group G is represented by a routing object $G.RO$ with a fixed radius $G.r$. Every visited object o is inserted into a group if $d(o, G.RO) \leq G.r$. Note that, for an object o, it may exist more than one group G_i satisfying the condition that $d(o, G_i.RO) \leq G_i.r$. Here, in order to obtain disjoint groups, i.e., $G_i \cap G_j = \emptyset (i \neq j)$, we choose the first group that satisfies the condition. If o cannot be inserted into any group, i.e., no group satisfying the condition, we create a new group G, with $G.RO = o$ and a fixed radius $G.r$. Consider the SMk2CP query shown in Fig. 2a, assuming that r_1 is chosen as the fixed radius. During the first phase, MSA obtains the candidate object set $\{o_3, o_4, o_1, o_2\}$. Figure 11a depicts the group G_1 after objects o_3 and o_4 are visited. Next, when processing object o_1, as $d(o_1, G_1) > r_1$, we create a new group G_2 with $G_2.RO = o_1$ and $G_2.r = r_1$, as illustrated in Fig. 11b.

By utilizing the grouping technique with a fix group radius r, MSA can be adapted to r-MSA, to reduce considerable distance computations during the verification. In particular, for an object to be verified, r-MSA first compares it with all the groups, instead of every object contained in each group. According to Rule 1 presented in Sect. 3.3, if $mindist(o, G) > maxCPD_k$, then $\langle o, G\rangle$ can be pruned. In other words, we can avoid evaluating all the objects contained in G for o. Consider the example shown in Fig. 11 again. For the object o_2 to be verified, $\langle o_2, G_1\rangle$ can be pruned away due to $mindist(o_2, G_1) > maxCPD_k (= r_1)$.

5.2 Approximate MkCP search

For MkCP retrieval, although we can utilize pruning heuristics and the aggressive pruning and compensation technique to accelerate query processing, efficiency still remains a problem since the query cost remains quadratic in worst case, especially for the high degree of overlapping between two object sets. Hence, it makes sense to study the *Approximate Metric kCP* (AMkCP) *query*, which trades the quality of the result for search time.

Definition 6 (*AMkCP Search*). Given two object sets P and Q in a generic metric space, and an integer $k(1 \leq k \leq |P| \times |Q|)$, an *Approximate* M$k$CP (AM$k$CP) *query finds* k ordered different object pairs from $P \times Q$, i.e., $AMkCP(P, Q) = \{\langle p_1, q_1\rangle, \langle p_2, q_2\rangle, \ldots, \langle p_k, q_k\rangle | p_1, p_2, \ldots, p_k \in P, q_1, q_2, \ldots, q_k \in Q, \langle p_i, q_i\rangle \neq \langle p_j, q_j\rangle, i \neq j, 1 \leq i, j \leq k$, and $\forall \langle p_i', q_i'\rangle \in MkCP(P, Q)$ s.t. $d(p_i, q_i) \geq d(p_i', q_i')\}$.

Consider the example depicted in Fig. 1 again. An AM2CP ($k = 2$) query may return the result set $\{\langle p_2, q_1\rangle, \langle p_2, q_3\rangle\}$, which is different from the result set $\{\langle p_2, q_1\rangle, \langle p_2, q_2\rangle\}$ returned by the M2CP query.

5.2.1 Estimation-based hybrid AMAkCP algorithm

In order to forsake some precision in exchange for improved efficiency, we can utilize the framework of EHM (which performs the best for MkCP search as verified in Sect. 6.2),

by integrating a popular approximate technique, i.e., ε-approximate technique [13,20,53]. Given a positive real ε (≥ 0) as the maximum distance relative error to be tolerated, for the ith answer object pair $\langle p_i, q_i \rangle$ contained in AMkCP(P, Q) and the ith answer object pair $\langle p'_i, q'_i \rangle$ included in MkCP(P, Q), ε-approximate technique makes that $(d(p_i, q_i) - d(p'_i, q'_i))/d(p'_i, q'_i) \leq \varepsilon$ holds.

However, since $\varepsilon(\geq 0)$ is unlimited, it is not easy for users to adjust the quality of the query result. Toward this, in this paper, we choose the α-allowance technique [20] with a bounded parameter ($0 < \alpha \leq 1$), which can be transferred to the ε-approximate technique with $\alpha = 1/(1+\varepsilon)$. Below, we propose an approximate pruning rule based on the α-allowance technique.

Rule 3 *Given two leaf or non-leaf entries E_P and E_Q, and a positive real $\alpha(0 < \alpha \leq 1)$, if emindist$(E_P, E_Q)$ or mindist (E_P, E_Q) is larger than maxCPD$_k \times \alpha$, then $\langle E_P, E_Q \rangle$ can be pruned away safely.*

Proof Given a relative distance error ε to be tolerated, if $eimindist(E_P, E_Q)$ or $mindist(E_P, E_Q)$ is larger than $maxCPD_k \times \alpha$, then, for any $\langle p, q \rangle$ ($p \in E_P$, $q \in E_Q$), $d(p,q) \times (1+\varepsilon) \geq mindist(E_P, E_Q)$ (or $emindist(E_P, E_Q)) \times (1+\varepsilon) > maxCPD_k \times \alpha \times (1+\varepsilon) > CPD'_k$ since $\alpha = 1/(1+\varepsilon)$ and CPD'_k denotes the accurate kth closest pair distance. Therefore, $\langle p, q \rangle$ cannot be an actual answer object pair due to $d(p,q) \times (1+\varepsilon) > CPD'_k$, and $\langle E_P, E_Q \rangle$ can be discarded safely accordingly. □

As depicted in Fig. 6, assume that $maxCPD_k = 40$ and $\alpha = 0.5$, then $\langle E_{P2}, E_{Q3} \rangle$ that cannot be pruned by Rules 1–2 can be discarded by Rule 3.

The pseudo-code of *Estimation-based Hybrid AMkCP algorithm* (EHA) is similar as EHM and thus omitted. It takes as inputs two COM-trees M_P and M_Q, an estimated value $eCPD_k$, and a real α ($0 < \alpha \leq 1$), and outputs the result set S_R of an AMkCP query. The only difference is that, EHA utilizes Rule 3 to prune leaf or non-leaf entry pairs, while EHM uses Rules 1 and 2.

5.2.2 GMdnn-tree-based AMkCP algorithm

As pointed out by [20,53], although the α-approximate (ε-approximate) technique utilized by EHA can achieve the high quality result set, the query efficiency does not improve much, which is also verified in Sect. 6.6. Motivated by this, we present *GMdnn-tree-based* AMkCP *Algorithm* (GMA), which employs the grouping and N-consider techniques to control the trade-off between query cost and accuracy of the query result.

GMdnn-tree is a variant of COMdnn-trees, which partitions the objects in each leaf node into disjoint groups. In particular, if two objects p and q are similar, i.e., the distance $d(p, q)$ between p and q is small, they can form a group. This is because, due to the triangle inequality, the difference between the distances from p and q to any other object o , i.e., $|d(p, o) - d(q, o)|$ is small if p and q are similar, as $|d(o, p) - d(o, q)| \leq d(p, q)$. Figure 12 shows two GMdnn-trees M_P and M_Q on the object sets P and Q (depicted in Fig. 7a), respectively. For instance, objects p_3, p_4, and p_5 contained in the leaf node pointed by E_{P5} are partitioned into two disjoint groups, i.e., $g_{P2} = \{p_3, p_5\}$, $g_{P3} = \{p_4\}$. Note that, g_{P3} only contains one object, since it does not exist any other object in this leaf node. As two objects in the same group are similar, one of them can be used to represent the whole group. For simplify, the first object in each group is chosen as the representative object. Also note that, the GMdnn-tree can support efficient MkCP, SMkCP, and AMkCP, queries simultaneously.

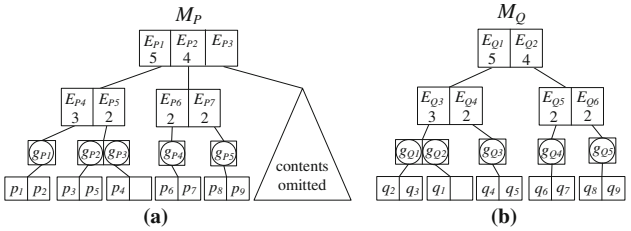

Fig. 12 Example of GMdnn-trees. **a** GMdnn-tree M_P on P. **b** GMdnn-tree M_Q on Q

With the help of GMdnn-tree, we can improve query efficiency significantly, as only the representative object of each group instead of the whole group is verified. Consider the example illustrated in Fig. 12 again, assume that p_1 and q_2 are representative objects of g_{P1} and g_{Q1}, respectively. All the object pairs between $g_{P1} = \{p_1, p_2\}$ and $g_{Q1} = \{q_2, q_3\}$ can be estimated using $\langle p_1, q_2 \rangle$, i.e., $mindist(g_{P1}, g_{Q1}) = d(p_1, q_2)$. In other words, if $\langle p_1, q_2 \rangle$ is pruned by Rule 1 or 2, other object pairs including $\langle p_1, q_3 \rangle$, $\langle p_2, q_2 \rangle$, and $\langle p_2, q_3 \rangle$ can also be pruned; otherwise, if $\langle p_1, q_2 \rangle$ cannot be discarded by Rules 1 and 2, other object pairs between two groups cannot be pruned as well.

A challenge issue for building the GMdnn-tree is how to group objects efficiently. A simple but efficient method is to choose a far away object o, and then partition the objects into disjoint groups in order of their distances to the chosen object o. Note that, a far away object o is needed, because the similarity of two objects p_i and p_j can be estimated well using the distance difference between $d(o, p_i)$ and $d(o, p_j)$ [7]. In this paper, we choose the far away object among all the routing objects stored in the GMdnn-tree. Take the example shown in Fig. 7 again. When grouping the objects in a leaf node E_{P5}, we can choose the routing object p_8 of E_{P7}, and sort the objects in E_{P5} in ascending order of their distances to p_8, i.e., p_3, p_5, and p_4. Hence,

E_{P5} can be partitioned into two groups $g_{P2} = \{p_3, p_5\}$ and $g_{P3} = \{p_4\}$.

Since GMdnn-trees are used to only achieve the approximation at the leaf node level, the N-consider technique [20] can be utilized in the intermediate node level, in order to further boost query efficiency. In particular, when visiting the entry pair $\langle E_P, E_Q\rangle$ which points to the intermediate nodes, we only consider the percentage N of all the sub entry pairs $\langle e_{Pi}, e_{Qj}\rangle$ $(e_{Pi} \in E_P, e_{Qj} \in E_Q)$.

Since the framework of GMA is similar as that of EHM, its pseudo-code is omitted here. The only difference between GMA and EHM is the processing for the intermediate entry pair $\langle E_P, E_Q\rangle$. If E_P and E_Q are intermediate entries pointing to non-leaf nodes, GMA uniformly chooses $\sqrt{N} \times E_P.num$ sub-entries of E_P and $\sqrt{N} \times E_Q.num$ sub-entries of E_Q for processing, in order to apply the N-consider technique. If E_P and E_Q are intermediate entries pointing to leaf nodes, it verifies and prunes groups instead of every object contained in groups.

5.2.3 Discussion

To quantify an approximate algorithm, it should consider not only improvement in performance efficiency, but also the quality of approximation. The quality of approximation can be measured by using the *precision*, i.e., the percentage of the k items of the approximate result that also contained in the exact result set.

Definition 7 (*Precision*). Given two object sets P and Q in a generic metric space, and an integer $k(1 \le k \le |P| \times |Q|)$, assume that $\langle p_i, q_i\rangle$ and $\langle p'_i, q'_i\rangle$ is the ith item contained in $AMkCP(P, Q)$ and $MkCP(P, Q)$, respectively.

$$precision = \frac{1}{k}\sum_{i=1}^{k}\begin{cases}1 & d(p_i, q_i) \le d(p_k^{'}, q_k^{'})\\ 0 & \text{otherwise}\end{cases}$$

Note that, we use the distance to determine whether $\langle p_i, q_i\rangle$ is contained in the exact result set $MkCP(P, Q)$, because, as analyzed in Sect. 3.1, $MkCP(P, Q)$ may be *not unique* due to the distance tie, and we randomly choose object pairs when the distance tie occurs. When *precision* = 1, the approximate result equals to the exact one. On the other hand, *precision* tends to 0 in worst case.

First, we derive the $precision$ for the approximate algorithm EHA, which only utilizes the α-approximate technique. In order to obtain $precision$(EHA), we need to determine under what conditions an object pair $(\in MkCP(P, Q))$ is certainly contained in $AMkCP(P, Q)$ returned by EHA, as stated as follows.

Lemma 10 *Given two object sets P and Q in a generic metric space, and a real value $\alpha(0 < \alpha \le 1)$, assume that $\langle p'_i, q'_i\rangle$ is the ith item contained in $MkCP(P, Q)$. If $d(p'_i, q'_i) \le \alpha \times d(p'_k, q'_k)$, then $\langle p'_i, q'_i\rangle$ should be also contained in $AMkCP(P, Q)$ returned by EHA.*

Proof Assume, to the contrary, that $\langle p'_i, q'_i\rangle$ is not included in $AMkCP(P, Q)$, i.e., $\langle p'_i, q'_i\rangle$ is pruned by Rule 3. For a leaf or non-leaf entry pair $\langle E_P, E_Q\rangle$ is or contain $\langle p'_i, q'_i\rangle$, $emindist(E_P, E_Q) \le mindist(E_P, E_Q) \le d(p'_i, q'_i) \le \alpha \times d(p'_k, q'_k) \le \alpha \times maxCPD_k$, which contradicts with the condition of Rule 3. Thus, the proof completes. □

According to Lemma 10, we can get $precision(EHA)$ bounded by:

$$\begin{aligned}\text{precison}(EHA) &\ge \frac{1}{k}\sum_{i=1}^{k}\begin{cases}1 & d(p_i^{'}, q_i^{'}) \le \alpha \times d(p_k^{'}, q_k^{'})\\ 0 & \text{otherwise}\end{cases}\\ &\ge \frac{F(\alpha \times d(p_k^{'}, q_k^{'}))}{F(d(p_k^{'}, q_k^{'}))}\end{aligned}$$

where $F(\ )$ denotes the distance distribution between two object sets P and Q. If $F(\ )$ follows the uniform distribution, then $precision(EHA)$ is bounded by α.

Next, we derive the lower bound of *precision* for GMA. For the grouping technique used in the leaf node level, since only one object pair is verified to represent four object pairs, and thus, the *precision* for the algorithm using grouping technique is bounded by 0.25. For each application of the N-consider technique in an intermediate node level, the probability that does not discard a real answer object pair is N. Hence, the *precision* for the algorithm using the N-consider technique equals to N^{L-2}, where L denotes the height of the tree index. As the two techniques are applied independently in GMA, $precision(GMA) \ge 0.25 \times N^{L-2}$.

6 Performance study

In this section, we experimentally evaluate the effectiveness of our developed pruning rules, the accuracy of cost models, and the performance of the algorithms for MkCP retrieval and its variants, using both real and synthetic data sets. The detailed experimental settings are described in Sect. 6.1. Five sets of experiments are conducted. The first set verifies the efficiency of our algorithms compared with the existing state-of-the-art (in-memory) MkCP search algorithms and kCP queries in Euclidean spaces, as presented in Sect. 6.2. The second set evaluates the effectiveness of pruning rules, as reported in Sect. 6.3. The third set demonstrates the accuracy of cost models developed for MkCP retrieval, as described in Sect. 6.4. Sections 6.5 and 6.6 present the last two sets of experiments, which evaluate the performance for two MkCP query variants, i.e., SMkCP search and AMkCP retrieval, respectively.

Table 2 Statistics of the datasets used

Dataset	Size	Dimensionality	Measurement
CA	62,173	2	L_1-norm
SF	87,328	2	L_1-norm
Color	112,544	4	L_2-norm
NASA	40,150	20	L_2-norm
Signature	50,000	64	Edit distance
Uniform	[0.1M, 10M]	2,16	L_2-norm
Gaussian	[0.1M, 10M]	2,16	L_2-norm

Table 3 Parameter ranges and default values

Parameter	Setting
k	1, 10, 100, **1000**, 10,000, 100,000
Overlap (%)	0, 25, **50**, 75, 100
Cardinality	**0.1M**, 1M, 10M
Ratio of $eCPD_k/CPD_k$	0.2, 0.4, 0.6, 0.8, 1, 3, 5, 7, 9
Buffer size (pages)	0, 32, 64, **128**, 256
r (% of maximum distance)	2, 4, **6**, 8, 10
α and N	0.9, 0.8, 0.7, 0.6, **0.5**

6.1 Experimental setup

We deploy four real datasets *CA*, *SF*, *Color*, and *NASA*. *CA* and *SF*[1] represent the locations in California and San Francisco, respectively. *Color*[2] contains the first four dimensions of color histograms extracted from an image database. *NASA*[3] is a set of feature vectors made by NASA. Following the experimental settings in [46], we generate a *Signature* dataset, in which each object is a string with 64 English letters. Since MkCP retrieval involves two object sets, we combine two GIS datasets *CA* and *SF*, and employ L_1-norm to simulate the shortest road network distance. However, for datasets *Color*, *NASA* and *Signature*, we divide them into two datasets with the same cardinality [18] , where L_2-norm and edit distance are utilized to measure the distances. Synthetic datasets following Uniform and Gaussian distributions, respectively, are also created, and L_2-norm is employed. Table 2 lists the statistics of the datasets used in our experiments. *Uniform* and *Gaussian* datasets with 16 dimensionality, and *NASA* are indexed using a page size of 10KB, whereas the other datasets are indexed using a page size of 4KB. The distance distribution F for every real or synthetic dataset is obtained by sampling, and is approximated by an equi-width histogram with 20,000 bins, separately storing values of $F(1)$, $F(2)$, and so on.

We investigate the performance of the proposed algorithms under various parameters, which are summarized in Table 3, where the bold denotes the defaults. Note that, in each experiment, only one factor varies, whereas the others are fixed to their default values. The main performance metrics include the total query cost (i.e., the sum of the I/O time and CPU time, where the I/O time is computed by charging 10ms for each page faults, as with [11]), the selectivity (defined in Sect. 4.4), and the number of node accesses (NA). All algorithms were implemented in C++, and all experiments were conducted on an Intel Core 2 Duo 2.93 GHz PC with 3 GB RAM.

[1] Available at http://www.census.gov/geo/www/tiger/.

[2] Available at http://www.dbs.informatik.uni-muenchen.de/~seidl.

[3] Available at http://www.sisap.org/metric_space_library.html.

6.2 Results on MkCP queries

The first set of experiments evaluates the performance of our presented algorithms (namely RMA, IMA, and EHM) in answering MkCP search, compared with existing state-of-the-art MkCP and Euclidean kCP algorithms. We study the influence of several parameters, including (1) the value of k, i.e., the number of closest pairs required, (2) the cardinalities of datasets, (3) the overlap between two datasets, (4) the ratio of $eCPD_k/CPD_k$, and (5) the buffer size.

Effect of k. First, we investigate the impact of k on the efficiency of the algorithms, using real and synthetic datasets. The results are depicted in Fig. 13, where abbreviations of algorithms (R for RMA, I for IMA, and E for EHM) are shown on the top of each column. The first observation is that the performance of the algorithms increases with the

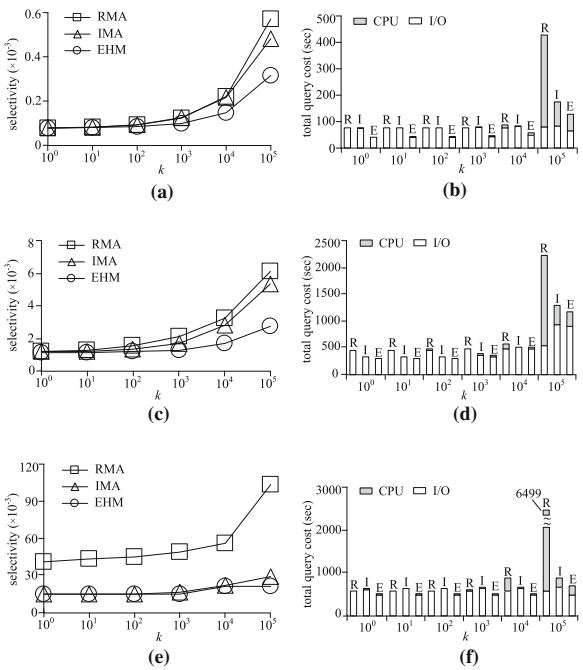

Fig. 13 MkCP search performance versus k. **a** *CA*, *SF*. **b** *CA*, *SF*. **c** *Color*, *Color*. **d** *Color*, *Color*. **e** *Signature*, *Signature*. **f** *Signature*, *Signature*

Table 4 Comparisons among EHM, PSI, and EPM on *Color*

k	Selectivity ($\times 10^{-3}$)			NA ($\times 10^3$)			CPU time (sec)		
	EHM	*PSI*	*EPM*	*EHM*	*PSI*	*EPM*	*EHM*	*PSI*	*EPM*
10^0	1.136	0.882	0.427	28.87	13.67	33.65	10.92	1.06	5.594
10^1	1.142	0.956	0.764	29.25	13.69	63.75	11.67	1.10	10.17
10^2	1.170	1.137	0.828	29.65	13.73	65.73	14.11	1.17	11.35
10^3	1.257	1.732	0.906	32.40	13.90	70.75	25.24	1.45	14.08
10^4	1.672	3.692	1.066	47.88	14.45	93.57	45.38	2.33	25.82
10^5	2.764	12.04	1.368	91.03	16.34	157.4	278.4	6.79	271.6

Table 5 Comparisons among EHM, AMP and LTC

	Selectivity			NA ($\times 10^4$)			CPU time (sec)		
	AMP	*LTC*	*EHM*	*AMP*	*LTC*	*EHM*	*AMP*	*LTC*	*EHM*
NASA	0.032	0.99	0.026	\	\	3.15	15.1	88	14.74
Signature	0.026	0.049	0.014	\	\	5.02	20.4	14	6.94
Uniform	0.52	0.99	0.23	\	\	26.3	3266	2140	1999
Gaussian	0.52	0.91	0.3	\	\	25.1	3559	2601	2142

growth of k. This is because, the more closest object pairs we need, the more entry pairs we need to evaluate. The second observation is that EHM performs the best, as it combines the depth-first and best-first traversal paradigms to reduce I/O cost, and utilizes the aggressive pruning and compensation technique to reduce computational cost. Note that, the CPU cost of RMA increases dramatically when k reaches 10^5. The reason is that, for larger k, it is more likely for RMA to access unnecessary branches.

Since EHM performs the best in most cases, it is utilized to compare against Euclidean kCP query algorithm PSI [18] and EHM based on PM-tree (EPM) [44], with the results depicted in Table 4 using *Color* dataset. It is observed that, PSI performs better than EHM and EPM, whereas it needs more distance computations especially for larger k values. This is because, PSI is designed particularly for the Euclidean space, where geometric properties can be employed to accelerate search; while EHM and EPM are applicable for any specific metric space, which aims to reduce the number of distance computations, since distance computation in the generic metric space is usually costly. The second observation is that, EPM outperforms EHM in terms of the CPU cost and selectivity, but it needs larger I/O cost. The reason is that, EPM utilizes the pivots with pre-computed distances to improve query efficiency, resulting in larger index storage and larger I/O overhead accordingly.

In addition, we compare our EHM algorithm with state-of-the-art MkCP query algorithms AMP [29] and LTC [32], using high dimensional real and synthetic datasets. The results are depicted in Table 5, where "\" denotes the NA of the corresponding algorithms is missing, because AMP and LTC are in-memory methods. The first observation is that EHM performs the best. The reason is that, our approach utilizes several punning rules based on COM-trees, and takes advantage of aggressive pruning and compensation, to improve query efficiency. Note that, the selectivity approaches to 1 on *Uniform* and *Gaussian* datasets in a 16 dimensional space. In other words, MkCP query algorithms degenerate to a brute-force algorithm, which needs to compare all the object pairs from two datasets. Hence, in the rest experiments of this paper, we employ 2 dimensional synthetic datasets.

Effect of cardinality Next, we show the scalability of our algorithms by comparing against existing MkCP search algorithms AMP [29] and LTC [32], using synthetic datasets. Tables 6 and 7 show the results as a function of $|P|(=|Q|)$, under *Uniform* and *Gaussian* datasets, respectively. Note that, in Tables 6 and 7, "\" represents the NA of the corresponding algorithms is missing, and "−" indicates that the corresponding algorithms cannot run due to memory overflow. This is because, both AMP and LTC are in-memory methods, whereas our algorithms are developed based on the disk-based COM-tree, and only load the data needed during MkCP query processing. As expected, our algorithms performs much better than AMP and LTC. In particular, LTC is several order magnitude worse than other algorithms. This is because, the efficiency of LTC degrades as the overlap percentage of datasets decreases. Here, the overlap percentage is set to 50% as the default for synthetic datasets. In addition, although EHM is the best in terms of selectivity, it has larger CPU cost than RMA and IMA. The reason is that, the additional CPU cost is needed for EHM in order to use the progressive pruning and compensation technique to further reduce the number of distance computations. Specifically,

Table 6 M*k*CP performance versus cardinality on *Uniform*

$\|P\|$	Selectivity ($\times 10^{-6}$)			NA ($\times 10^4$)			CPU time (sec)		
	0.1 M	1 M	10 M	0.1 M	1 M	10 M	0.1 M	1 M	10 M
AMP	85.4	8.1	–	\	\	–	49.8	3453	–
LTC	8.6E+5	8.2E+5	–	\	\	–	1736	1.8E+5	–
RMA	90.7	6	0.663	0.75	7.23	69.3	1.93	14.59	412.2
IMA	90.3	6	0.663	0.778	6.49	62.4	1.49	12.69	142.3
EHM	74.7	5.6	0.66	0.604	6.08	59.6	2.69	16.54	158.5

Table 7 M*k*CP performance versus cardinality on *Gaussian*

$\|P\|$	Selectivity ($\times 10^{-6}$)			NA ($\times 10^4$)			CPU time (sec)		
	0.1 M	1 M	10 M	0.1 M	1 M	10 M	0.1 M	1 M	10 M
AMP	92	8.44	–	\	\	–	26.9	1927	–
LTC	9E+5	9.9E+5	–	\	\	–	2601	1.6E+5	–
RMA	111	7.64	0.07	0.937	7.64	88.6	2.62	7.64	320.9
IMA	103	7.56	0.07	0.977	7.56	79.9	2.1	7.56	178.8
EHM	90	7.22	0.068	0.762	7.22	75.9	3.39	7.21	211.5

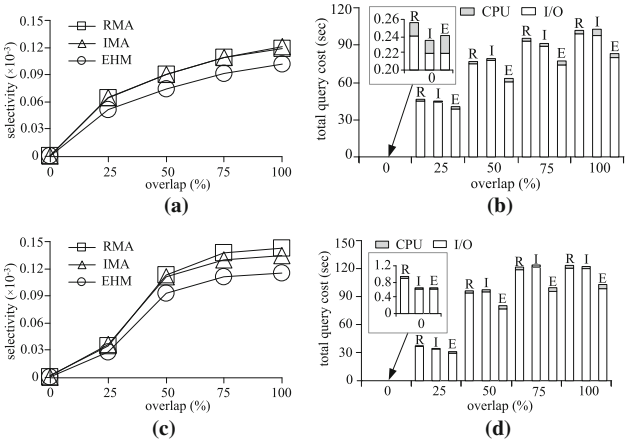

Fig. 14 M*k*CP search performance versus *overlap*. **a** *Uniform*, *Uniform*. **b** *Uniform*, *Uniform*. **c** *Gaussian*, *Gaussian*. **d** *Gaussian*, *Gaussian*

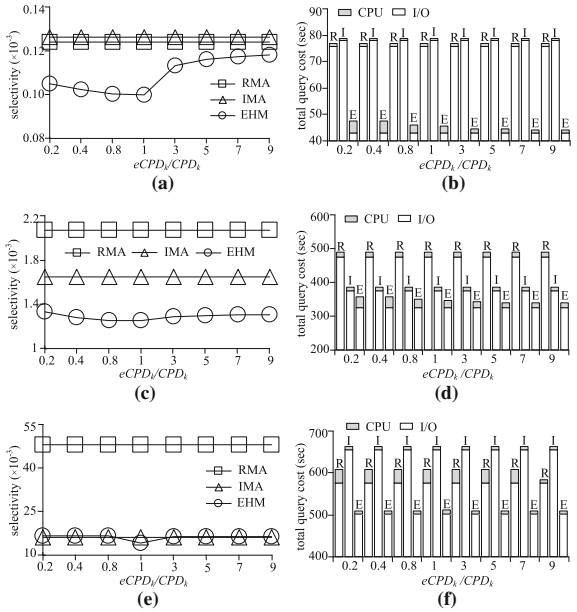

Fig. 15 M*k*CP search performance versus *ratio of* $eCPD_k/CPD_k$. **a** *CA*, *SF*. **b** *CA*, *SF*. **c** *Color*, *Color*. **d** *Color*, *Color*. **e** *Signature*, *Signature*. **f** *Signature*, *Signature*

EHM needs to insert object pairs into the min-heap for further verification even if they are pruned by using the aggressive pruning technique (line 41 of Algorithm 4), i.e., the insertion operation leads to additional CPU cost; while IMA and RMA compute immediately the distances and update the result set.

Effect of overlap Then, in order to explore the influence of *overlap* on the algorithms, we employ *Uniform* and *Gaussian* datasets. Figure 14 depicts the results under various overlap percentages. Notice that, in Fig. 14b, d, the total query cost for *overlap* = 0 is illustrated in the small sub figure. As expected, the selectivity and the total query cost of all the algorithms ascend with the growth of *overlap* percentage, because the M*k*CP query space grows as *overlap* increases. Consistent with the observation from previous experiments, EHM also performs the best.

Effect of ratio Next, we inspect the impact of the ratio of $eCPD_k/CPD_k$ on the efficiency of the algorithms. Figure 15 illustrates the results on real and synthetic datasets. The first observation is that, as $eCPD_k$ approaches to the real CPD_k value, the selectivity of EHM drops consistently. When $eCPD_k$ grows far beyond the real CPD_k value, the selectivity of EHM converges to that of IMA. Note that, for *CA* and *SF* datasets, the selectivity of EHM skips from the case when $eCPD_k/CPD_k = 1$ to the case when $eCPD_k/CPD_k = 3$, due to the distance distributions of the datasets. Specifically, there

exit almost 9000 object pairs with their distances in the range $[CPD_k, 3CPD_k]$, which is far more than those in other distance intervals (e.g., $[3CPD_k, 5CPD_k]$), resulting in a clear increase in terms of the number of distance computations. For *Signature*, the selectivity of EHM is almost the same as that of IMA, except for ratio = 1, because the distance function for *Signature* has *small* domain of *discrete* values. The second observation is that, the CPU cost of EHM decreases as the ratio grows, and stays stable eventually. This is because additional CPU cost is needed for compensation, if $eCPD_k$ is too small. Nonetheless, EHM always outperforms RMA and IMA, with $eCPD_k$ in a wide range of estimated values. However, in Fig. 15b, f, IMA performs worse than RMA in terms of total query cost. The reason is that, the recursion used by RMA favors LRU buffers than the iteration used by IMA [18]. Therefore, the I/O cost of IMA is worse than that of RMA, incurring larger total query cost of IMA.

Effect of buffer size Finally, we explore the influence of buffer size with respect to the efficiency of the algorithms, using real and synthetic datasets. Figure 16 only shows the total query cost of the algorithms under various buffer sizes, since the selectivity of the algorithms stays unchanged under different buffer sizes. Note that, the I/O costs of IMA and EHM are larger than RMA when the buffer size equals to 256 pages under *Signature* dataset. The reason is that, recursion in depth-first traversal favors LRU buffers [18]. Although EHM visits entry pairs with $mindist = 0$ in the depth-first order, it iteratively accesses entry pairs with $mindist > 0$. Therefore, the I/O cost of EHM could be larger than that of RMA. More important, nevertheless, in most cases, the I/O cost of EHM is the lowest.

Table 8 Prunning rule effectiveness versus k

k	10^0	10^1	10^2	10^3	10^4	10^5
Number of times applied ($\times 10^3$) *on (CA, SF)*						
RMA-R1	117	133	185	332	741	1814
RMA-R2	12,863	12,865	12,870	12,893	13,008	13,531
IMA-R1	117	131	184	343	728	1532
IMA-R2	12,868	12,874	12,878	12,909	12,933	12,983
EHM-R1	116	123	145	194	388	931
EHM-R2	12,869	12,873	12,869	12,850	12,782	12,489
Number of times applied ($\times 10^3$) *on (Color, Color)*						
RMA-R1	2543	2873	3630.5	5140	8377	15,857
RMA-R2	32,984	33,316	34,172	36,033	40,340	50,421
IMA-R1	2494	2631	3036	3905	7125	13,666
IMA-R2	32,905	33,093	33,515	34,425	37,971	45,906
EHM-R1	2369	2403	2418	2509	3407	6428
EHM-R2	33,574	33,349	34,225	34,628	34,784	37,430
Number of times applied ($\times 10^3$) *on (Signature, Signature)*						
RMA-R1	22,581	23,661	24,822	26,929	31,186	57,191
RMA-R2	33,679	32,830	31,848	30,191	28,149	30,497
IMA-R1	7137	7163	7245	8218.2	11,255	14,985
IMA-R2	25,649	25,661	25,615	25,063	23,952	23,006
EHM-R1	7113	7113	7110	7067	10,682	10,036
EHM-R2	25,753	25,736	25,656	24,778	23,199	22,270

6.3 Effectiveness of rules

The second set of experiments aims to evaluate the effectiveness of our developed pruning rules. We measure the effectiveness of a rule by how often it is successfully applied in every algorithm. For Rules 1–2, a successful application is counted when they prune an intermediate entry or an object. Table 8 depicts the number of times that each rule is successfully applied as a function of k. Obviously, all pruning rules are applied multiple times during MkCP search, confirming their usefulness. Note that, the number of pruning rules applied increases with k in most cases. This is because, as k grows, CPD_k used for pruning ascends. Hence, it is more difficult to prune high level entries, resulting in a lot of pruning rule applications for the entries in the low level.

6.4 Accuracy of cost models

The third set of experiments verifies the accuracy of the cost models for MkCP retrieval. Figure 17 plots the I/O cost (in terms of NA) and the CPU cost (in terms of selectivity) with respect to k. In particular, each diagram contains: (1) the *actual* cost of EHM, (2) the *estimated* cost computed by our derived cost models, and (3) the *relative error* between actual and estimated values (i.e., $|actual - estimated|/actual$). Note that, for clarity, we only include the cost of EHM here

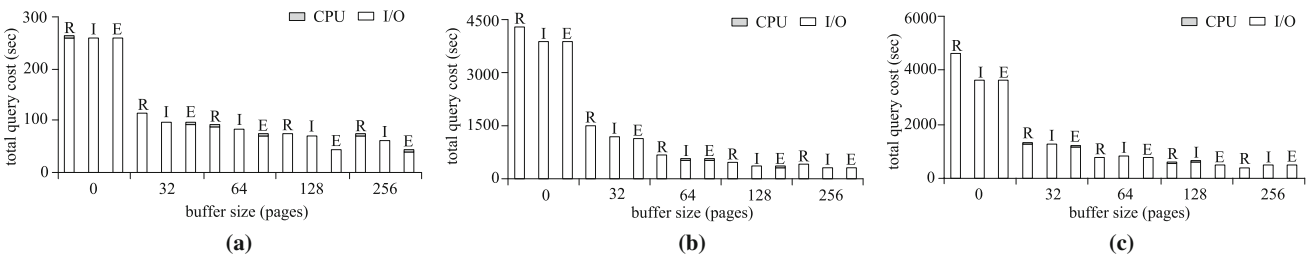

Fig. 16 MkCP search performance versus *buffer size*. **a** *CA*, *SF*. **b** *Color*, *Color*. **c** *Signature*, *Signature*

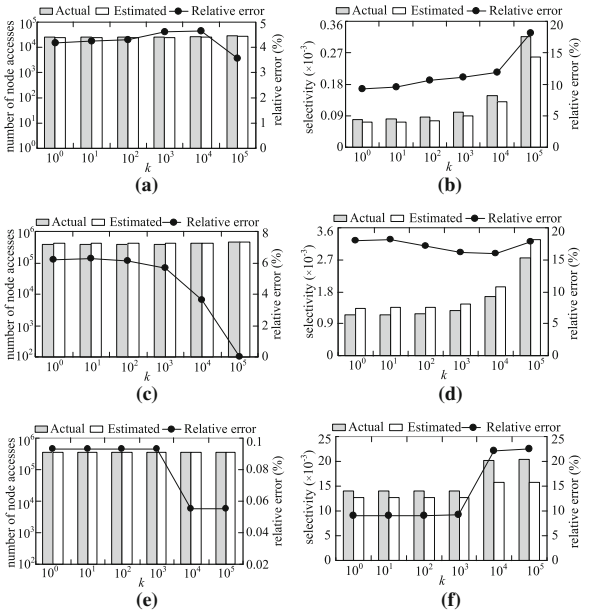

Fig. 17 Cost model versus k. **a** *CA*, *SF*. **b** *CA*, *SF*. **c** *Color*, *Color*. **d** *Color*, *Color*. **e** *Signature*, *Signature*. **f** *Signature*, *Signature*

because it performs the best, and its cost is the closest to the value derived by the cost model. It is observed that the cost model for I/O cost is very accurate, with the maximum relative error 6.2 %. However, the cost model derived for CPU cost is not accurate yet still good, with the maximal relative error 22.5 %.

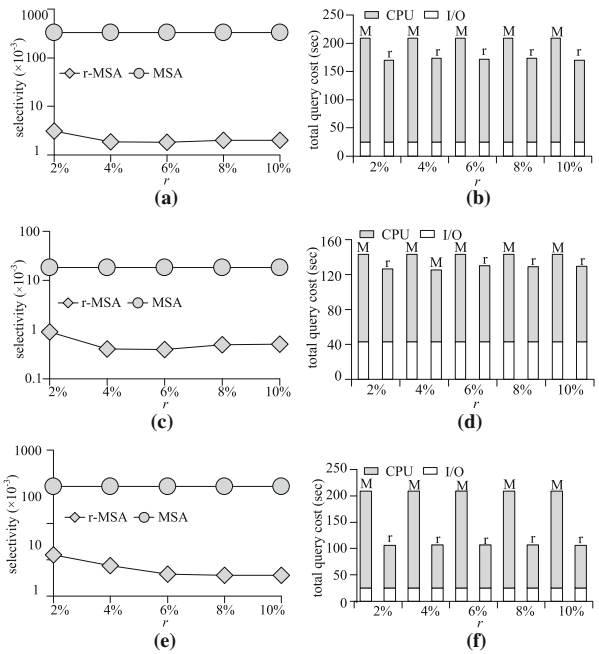

Fig. 18 SMkCP search performance versus r. **a** *SF*. **b** *SF*. **c** *Color*. **d** *Color*. **e** *Signature*. **f** *Signature*

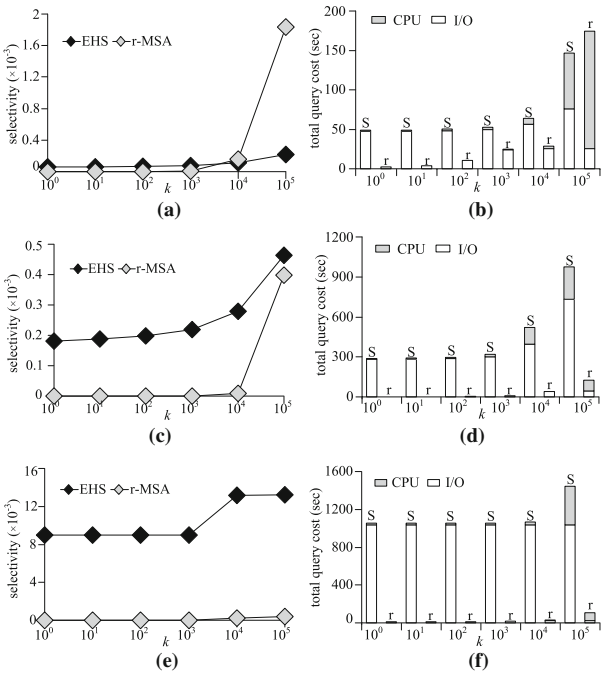

Fig. 19 SMkCP search performance versus k. **a** *SF*. **b** *SF*. **c** *Color*. **d** *Color*. **e** *Signature*. **f** *Signature*

6.5 Results on SMkCP queries

The fourth set of experiments evaluates the performance of our proposed algorithms (namely EHS and MSA) in answering SMkCP queries. We study the influence of various parameters, including (1) the grouping radius r, (i.e., the percentage with respect to the maximal distance in the metric space), and (2) the value of k, i.e., the number of closest pairs required.

Effect of r Figure 18 plots the performance of SMkCP search as a function of r, using real and synthetic datasets, where abbreviations of algorithms (M for MSA and r for r-MSA) are shown on the top of each column. The first observation is that, r-MSA performs much better than MSA. This is because r-MSA utilizes the grouping technique to minimize computational cost significantly, as discussed in Sect. 5.1.3. The second observation is that, r-MSA achieves the best performance when r approaches 6 %. The reason is that, for larger radius, the groups are not well clustered; in worst case, all the objects are partitioned into one group, resulting in poor query efficiency. Nevertheless, for smaller radius, it will result in too many groups and thus needs more additional cost to process these groups.

Effect of k Next, we inspect the impact of k on the efficiency of SMkCP search algorithms. Figure 19 illustrates the experimental results, based on real and synthetic datasets, where abbreviations of algorithms (S for EHS and r for r-MSA) are shown on the top of each column. The first observation is that, in most cases, r-MSA performs better

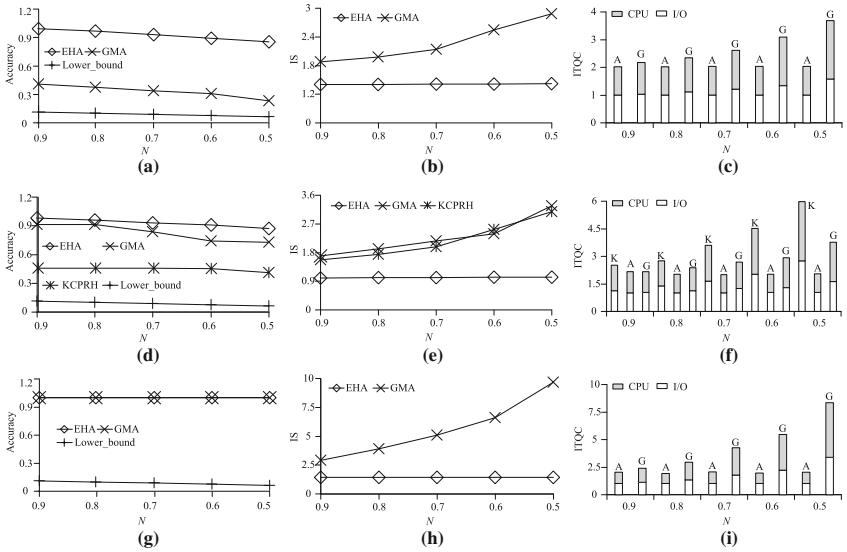

Fig. 20 AMkCP search performance versus α and N. **a** *CA*, *SF*. **b** CA, SF. **c** *CA*, *SF*. **d** *Color*, *Color*. **e** *Color*, *Color*. **f** *Color*, *Color*. **g** *Signature*, *Signature*. **h** *Signature*, *Signature*. **i** *Signature*, *Signature*

than EHS. This is because r-MSA utilizes the NN information to boost query performance. Note that, the efficiency of r-MSA degrades dramatically when k reaches 10^5, especially on *SF* and *Color* datasets. The reason is that, as discussed in Sect. 5.1.3, the time complexity of r-MSA is $O(k^2)$, and thus, the efficiency of r-MSA degrades as k increases, especially for larger k. Although r-MSA utilizes the grouping technique to minimize quadratic cost, the improvement degrades when k reaches 10^5, as the upper bound $maxCPD_k$ used to avoid unnecessary distance computations converges slowly. However, the efficiency degradation is not obvious on *Signature*, due to its polarized distance distribution.

6.6 Results on AMkCP queries

The last set of experiments verifies the performance of our proposed algorithms (namely EHA and GMA) in answering AMkCP queries. We study the influence of various parameters, including (1) the approximate parameters α and N, and (2) the value of k, i.e., the number of closest pairs required.

Effect of α and N First, we investigate the impact of approximate parameters α and N on the efficiency of the algorithms and the accuracy of the final result, compared with Euclidean AkCP algorithm KCPRH [20], using real and synthetic datasets. In particular, α is utilized for EHA, while N is used for GMA, to control the tradeoff between the quality of the query result and the query efficiency. The accuracy, the *improvement of selectivity* (IS for short), and the *improvement of total query cost* (ITQC for short) are depicted in Fig. 20, where abbreviations of algorithms (K for KCPRH, A for EHA, and G for GMA) are shown on the top of each column, and Lower_bound denotes the lower bound of *precision* for GMA algorithm as derived in Sect. 5.2.3. Note that, IS is measured as the ratio of the selectivity of AMkCP query algorithm and that of MkCP search algorithm, and IQTC is defined similarly. The first observation is that, the *precision* of the approximate algorithms drops with α and N, which is consistently with the *precision* derived in Sect. 5.2.3. The second observation is that, GMA can achieve better accuracy than KCPRH with similar query efficiency improvement. Moreover, the ITQC of KCPRH is more sensitive to approximate parameters. This is because, KCPRH utilizes the hybrid approximate technique; thus, its total query cost decreases with both parameters N and α; while GMA is only affected by N. In addition, although EHA can achieve the high accuracy of the final result, query performance does not improve much. Nevertheless, GMA provides much larger query performance improvement, which is sensitive to the approximate parameter, and meanwhile can get the tolerated *precision*.

Effect of k Figure 21 plots the *precision* and AMkCP search performance as a function of k, using both real and synthetic datasets. It is worth noting that, EHM is a MkCP search algorithm, which is used to compare against our proposed AMkCP search algorithms, to show the query performance improvement. As expected, GMA is better than EHA, since it can find a well-controlled trade-off between the query cost and the accuracy of the result. Note that, the peak of the accuracy (i.e., *precision* defined in Definition 7) for EHA occurs in Fig. 21a when $k = 10$ due to its randomness for smaller k values. In addition, the precision of GMA is zero when $k = 1$ in Fig. 21a, since its value can be either 1 or 0 according to Definition 7, and the precision is smaller than its lower bound when $k = 1$.

6.7 Conclusions from the experiments

From the previous exhaustive performance comparisons on both real and synthetic datasets, the most important conclusions are the following:

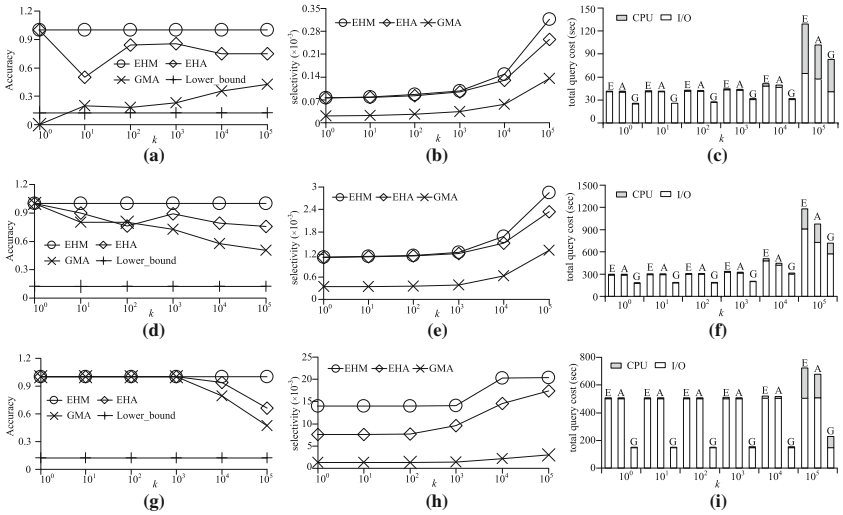

Fig. 21 AM*k*CP search performance versus k. **a** *CA*, *SF*. **b** *CA*, *SF*. **c** *CA*, *SF*. **d** *Color*, *Color*. **e** *Color*, *Color*. **f** *Color*, *Color*. **g** *Signature*, *Signature*. **h** *Signature*, *Signature*. **i** *Signature*, *Signature*

- For M*k*CP search, among the algorithms RMA, IMA, and EHM based on M-tree, EHM performs the best in most cases. In addition, our algorithms are flexible, i.e., they can be easily extended to other metric indexes (e.g., PM-tree [44]), in order to achieve better query performance in terms of selectivity and CPU time.
- Compared with the existing state-of-the-art M*k*CP search algorithms AMP [29] and LTC [32], our algorithms are more efficient and better scalable. In particular, the performance of LTC is poor for the case when the datasets with low overlap percentage. However, for the *k*CP query in the Euclidean space, the algorithms using R-trees [18] outperforms those using M-trees and PM-trees with respect to I/O cost and CPU time.
- For SM*k*CP retrieval, MSA based on the COMdnn-tree is several orders of magnitude better than EHS based on the COM-tree when k is much smaller than the cardinality of the dataset.
- For AM*k*CP search, the GMdnn-tree with the N-consider technique is more suitable for finding a good balance between query performance and query result accuracy, while the α-allowance technique is a good alternative when the user demands high quality of the result set regardless of necessary processing time.

7 Conclusions

In this paper, we explore the problem of M*k*CP search, which aims at *efficient* *k*CP query processing in *general metric spaces*. M*k*CP retrieval is not only interesting from a research point of view, but also useful in many real-life applications (e.g., GIS, data mining, recommender systems). We propose three algorithms (i.e., RMA, IMA, and EHM) that do not require the detailed representations of the objects and are applicable as long as the similarity between two objects can be evaluated and satisfies the triangle inequality. Our methods utilize *dynamic* metric indexes (i.e., COM-trees), employ a series of *pruning rules*, follow depth-first or/and best-first traversal paradigms, and make use of the *aggressive pruning and compensation* technique. In addition, we develop a cost model for M*k*CP search and study two interesting M*k*CP query variants. Extensive experiments using both real and synthetic data sets demonstrate the performance of the proposed algorithms, the effectiveness of the presented pruning rules, and the accuracy of the derived cost model.

In the future, we intend to further improve the performance of our presented algorithms, by developing more effective pruning rule(s) and more efficient approach of CPD_k estimation. Another promising direction for future work is to consider other interesting *k*CP query variants (e.g., exclusive *k*CP queries [48] and k farthest pair queries) in general metric spaces. Finally, it would be particularly interesting to investigate M*k*CP retrieval in the distribution environment.

Acknowledgments Yunjun Gao was supported in part by the National Key Basic Research and Development Program (i.e., 973 Program) No. 2015CB352502, NSFC Grant No. 61379033, the Cyber Innovation Joint Research Center of Zhejiang University, and the Key Project of Zhejiang University Excellent Young Teacher Fund (Zijin Plan). We would like to thank Prof. A. Corral and Prof. T. Skopal for their useful feedback on the source codes of their proposed algorithms in [18,44]. We also would like to express our gratitude to some anonymous reviewers for their giving valuable and helpful comments to improve the technical quality and presentation of this paper.

References

1. Achtert, E., Kriegel, H.P., Kroger, P., Renz, M., Zufle, A.: Reverse k-nearest neighbor search in dynamic and general metric databases. In: EDBT, pp. 886–897 (2009)
2. Alvarez, M., Pan, A., Raposo, J., Bellas, F., Cacheda, F.: Using clustering and edit distance techniques for automatic web data extraction. In: WISE, pp. 212–224 (2007)

3. Angiulli, F., Pizzuti, C.: An approximate algorithm for top-k closest pairs join query in large high dimensional data. Data Knowl. Eng. **53**(3), 263–281 (2005)
4. Arumugam, S., Jermaine, C.: Closest-point-of-approach join for moving object histories. In: ICDE, pp. 86–95 (2006)
5. Bohm, C.: A cost model for query processing in high dimensional data spaces. ACM Trans. Database Syst. **25**(2), 129–178 (2000)
6. Breunig, M.M., Kriegel, H.P., Ng, R.T., Sander, J.: LOF: identifying density-based local outliers. In: SIGMOD, pp. 93–104 (2000)
7. Bustos, B., Navarro, G., Chavez, E.: Pivot selection techniques for proximity searching in metric spaces. Pattern Recognit. Lett. **24**(14), 2357–2366 (2003)
8. Chavez, E., Navarro, G., Baeza-Yates, R., Marroquin, J.L.: Searching in metric spaces. ACM Comput. Surv. **33**(3), 273–321 (2001)
9. Cheema, M.A., Lin, X., Wang, H., Wang, J., Zhang, W.: A unified approach for computing top-k pairs in multi-dimensional space. In: ICDE, pp. 1031–1042 (2011)
10. Chen, C., Sun, W., Zheng, B., Mao, D., Liu, W.: An incremental approach to closest pair queries in spatial networks using best-first search. In: DEXA, pp. 136–143 (2011)
11. Chen, L., Lian, X.: Efficient processing of metric skyline queries. IEEE Trans. Knowl. Data Eng. **21**(3), 351–365 (2009)
12. Ciaccia, P., Nanni, A., Patella, M.: A query-sensitive cost model for similarity queries with M-tree. In: ADC, pp. 65–76 (1999)
13. Ciaccia, P., Patella, M.: PAC nearest neighbor queries: approximate and controlled search in high dimensional and metric spaces. In: ICDE, pp. 244–255 (2000)
14. Ciaccia, P., Patella, M., Zezula, P.: M-tree: an efficient access method for similarity search in metric spaces. In: VLDB, pp. 426–435 (1997)
15. Ciaccia, P., Patella, M., Zezula, P.: A cost model for similarity queries in metric spaces. In: PODS, pp. 59–68 (1998)
16. Corral, A., Almendros-Jimnez, J.: A performance comparison of distance-based query algorithms using R-trees in spatial databases. Inf. Sci. **177**(11), 2207–2237 (2007)
17. Corral, A., Manolopoulos, Y., Theodoridis, Y., Vassilakopoulos, M.: Closest pair queries in spatial databases. In: SIGMOD, pp. 189–200 (2000)
18. Corral, A., Manolopoulos, Y., Theodoridis, Y., Vassilakopoulos, M.: Algorithms for processing k-closest-pair queries in spatial databases. Data Knowl. Eng. **49**(1), 67–104 (2004)
19. Corral, A., Manolopoulos, Y., Theodoridis, Y., Vassilakopoulos, M.: Cost models for distance joins queries using R-trees. Data Knowl. Eng. **57**(1), 1–36 (2006)
20. Corral, A., Vassilakopoulos, M.: On approximate algorithms for distance-based queries using R-trees. Comput. J. **48**(2), 220–238 (2005)
21. Eppstein, D.: Fast hierarchical clustering and other applications of dynamic closest pairs. J. Exp. Algorithm. **5**, article 1 (2000)
22. Fredriksson, K., Braithwaite, B.: Quicker similarity joins in metric spaces. In: SISAP, pp. 127–140 (2013)
23. Fuhry, D., Jin, R., D.Zhang: Efficient skyline computation in metric space. In: EDBT, pp. 1042–1051 (2009)
24. Gutierrez, G., Saez, P.: The k closest pairs in spatial databases. GeoInformatica **17**(4), 543–565 (2013)
25. Hjaltason, G.R., Samet, H.: Incremental distance join algorithms for spatial databases. In: SIGMOD, pp. 237–248 (1998)
26. Hjaltason, G.R., Samet, H.: Index-driven similarity search in metric spaces. ACM Trans. Database Syst. **28**(4), 517–580 (2003)
27. Jacox, E.H., Samet, H.: Metric space similarity joins. ACM Trans. Database Syst. **33**(2), article 7 (2008)
28. Kim, Y.J., Patel, J.M.: Performance comparison of the R*-tree and the quadtree for $k$nn and distance join queries. IEEE Trans. Knowl. Data Eng. **22**(7), 1014–1027 (2010)
29. Kurasawa, H., Takasu, A., Adachi, J.: Finding the k-closest pairs in metric spaces. In: NTSS, pp. 8–13 (2011)
30. Nanopoulos, A., Theodoridis, Y., Manolopoulos, Y.: C2P: Clustering based on closest pairs. In: VLDB, pp. 331–340 (2001)
31. Papadopoulos, A.N., Nanopoulos, A., Manolopoulos, Y.: Processing distance join queries with constraints. Comput. J. **49**(3), 281–296 (2006)
32. Paredes, R., Reyes, N.: Solving similarity joins and range queries in metric spaces with the list of twin clusters. J. Discrete Algorithms **7**(1), 18–35 (2009)
33. Pearson, S.S., Silva, Y.N.: Index-based R-S similarity joins. In: SISAP, pp. 106–112 (2014)
34. Ristad, E.S., Yianilos, P.N.: Learning string-edit distance. IEEE Trans. Pattern Anal. Mach. Intell. **20**(5), 522–532 (1998)
35. Roumelis, G., Vassilakopoulos, M., Corral, A., Manolopoulos, Y.: A new plane-sweep algorithm for the k-closest-pairs query. In: SOFSEM, pp. 478–490 (2014)
36. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann, San Francisco (2006)
37. Sarma, A.D., He, Y., Chaudhuri, S.: Clusterjoin: a similarity joins framework using map-reduce. PVLDB **7**(12), 1059–1070 (2014)
38. Shan, J., Zhang, D., Salzberg, B.: On spatial-range closest-pair query. In: SSTD, pp. 252–269 (2003)
39. Shin, H., Moon, B., Lee, S.: Adaptive multi-stage distance join processing. In: SIGMOD, pp. 343–354 (2000)
40. Shin, H., Moon, B., Lee, S.: Adaptive and incremental processing for distance join queries. IEEE Trans. Knowl. Data Eng. **15**(6), 1561–1578 (2003)
41. Silva, Y.N., Pearson, S.: Exploiting database similarity joins for metric spaces. In: VLDB, pp. 1922–1925 (2012)
42. Silva, Y.N., Pearson, S., Cheney, J.A.: Database similarity join for metric spaces. In: SISAP, pp. 266–279 (2013)
43. Skopal, T., Lokoc, J.: Answering metric skyline queries by PM-tree. In: DATESO, pp. 22–37 (2010)
44. Skopal, T., Pokorny, J., Snasel, V.: PM-tree: pivoting metric tree for similarity search in multimedia databases. In: ADBIS, pp. 803–815 (2004)
45. Tao, Y., Yi, K., Sheng, C., Kalnis, P.: Efficient and accurate nearest neighbor and closest pair search in high-dimensional space. ACM Trans. Database Syst. **35**(3), article 20 (2010)
46. Tao, Y., Yiu, M.L., Mamoulis, N.: Reverse nearest neighbor search in metric spaces. IEEE Trans. Knowl. Data Eng. **18**(9), 1239–1252 (2006)
47. Tao, Y., Zhang, J., Papadias, D., Mamoulis, N.: An efficient cost model for optimization of nearest neighbor search in low and medium dimensional spaces. TKDE **16**(10), 1169–1184 (2004)
48. U, L.H., Mamoulis, N., Yiu, M.L.: Computation and monitoring of exclusive closest pairs. IEEE Trans. Knowl. Data Eng. **20**(12), 1641–1654 (2008)
49. Vlachou, A., Doulkeridis, C., Kotidis, Y.: Metric-based similarity search in unstructured peer-to-peer systems. Trans. Large Scale Data Knowl. Cent. Syst. **7100**, 28–48 (2012)
50. Wang, Y., Metwally, A., Parthasarathy, S.: Scalable all-pairs similarity search in metric spaces. In: KDD, pp. 829–837 (2013)
51. Xiao, C., Wang, W., Lin, X., Shang, H.: Top-k set similarity joins. In: ICDE, pp. 916–927 (2009)
52. Yang, C., Lin, K.I.: An index structure for improving closest pairs and related join queries in spatial databases. In: IDEAS, pp. 140–149 (2002)
53. Zezula, P., Savino, P., Amato, G., Rabitti, F.: Approximate similarity retrieval with M-trees. VLDB J. **7**(4), 275–293 (1998)
54. Zhou, P., Zhang, D., Salzberg, B., Cooperman, G., Kollios, G.: Close pair queries in moving object databases. In: GIS, pp. 2–11 (2005)