Evaluating Top- N Queries in n -Dimensional Normed Spaces

Liang Zhu¹, Feifei Liu¹, Weiyi Meng², Qin Ma³, Yu Wang¹, Fang Yuan⁴

¹Intelligent Database Laboratory, School of Computer Science and Technology, Hebei University, Baoding, Hebei 071002, China

²Department of Computer Science, State University of New York at Binghamton, Binghamton, NY 13902, USA

³Department of Foreign Language Teaching and Research, Hebei University, Baoding, Hebei 071002, China

⁴School of Mathematics and Information Science, Hebei University, Baoding, Hebei 071002, China

E-mail addresses: zhu@hbu.edu.cn (L. Zhu), liufeifei9476@126.com (F. Liu), meng@cs.binghamton.edu (W. Meng), maqin@hbu.edu.cn (Q. Ma), wy@hbu.edu.cn (Y. Wang), yuanfang@hbu.edu.cn (F. Yuan)

ABSTRACT

For the problem of processing top- N queries, the threshold algorithm (TA) is an important method in many scenarios. The model of top- N queries with TA has three characteristics: (1) the ranking function is monotone, (2) the query point is fixed, and (3) TA scans the sorted index lists unidirectionally. In many database applications, however, there are opportunities for evaluating top- N queries with arbitrary query points and generic (not necessarily monotone) ranking functions. In this paper, we propose an approach for evaluating top- N queries in n -dimensional normed spaces. Given a query point $Q = (q_1, \dots, q_n)$ in the real vector space $\mathbb{R}^n$ and a generic norm distance d as ranking function, our method employs the norm equivalence theorem in Functional Analysis so that the candidate tuples of top- N query Q with d can be obtained using the Maximum distance d_∞ . This method projects each q_i on its corresponding axis and constructs an interval centered at q_i , and then enlarges each interval bidirectionally until the n -dimensional hyperrectangle contains enough candidate tuples so that the top- N tuples are retrieved according to the given norm distance d . Extensive experiments are conducted to measure the performance of our approach for both low-dimensional and high-dimensional data.

Keywords: top- N query, n -dimensional normed space, relational database, ranking function

1. INTRODUCTION

A top- N (also ranking, ranked, top- K or top- k) query against a relation is to find a sorted set of N (e.g., $N = 1, 20$, or 100) tuples that best, but not necessarily completely, satisfy the query condition. Top- N queries have been studied extensively since late 1990s [7, 11, 12], which are useful in many fields such as data mining, search engine, decision support system, multimedia database, information retrieval, Web database and so on. The threshold algorithm (TA) introduced by Fagin et al. [14] is an important method in many scenarios. For a relation $\mathbf{R}$ with schema $\mathbf{R}(A_1, A_2, \dots, A_n)$, we call $(\max(A_1), \max(A_2), \dots, \max(A_n))$ *Max-point*, and we call $(\min(A_1), \min(A_2), \dots, \min(A_n))$ *Min-point*. Then, the model of top- N queries with TA can be generalized as follows [14].

Assume that $\mathbf{R} \subset [0, 1]^n$ is a relation (or a finite dataset) where $[0, 1]^n$ is the Cartesian product of n closed intervals $[0, 1]$, and $f(\mathbf{x})$ is a monotonically increasing function defined on $[0, 1]^n$. A top- N query against $\mathbf{R}$ is to find N tuples $\langle t_1, t_2, \dots, t_N \rangle$ that have the highest values of $f(t)$ for all $t \in \mathbf{R}$. Denoting $\mathbf{x} = (x_1, x_2, \dots, x_n)$ and $\mathbf{y} = (y_1, y_2, \dots, y_n)$, a function $f(\mathbf{x})$ is monotonically increasing (or decreasing) if and only if $f(\mathbf{x}) \leq f(\mathbf{y})$ (or $f(\mathbf{x}) \geq f(\mathbf{y})$) whenever $x_i \leq y_i$ for every $1 \leq i \leq n$ [14]. The “*monotone*” here is called “*F-monotone*” (means *Fagin-monotone*) in this paper, in order to distinguish it from the terms “*p-monotone*” and “*x-monotonic*” in Section 3.1.

Now, let Q be the Max-point $(1, 1, \dots, 1)$ of $\mathbf{R} \subset [0, 1]^n$ and $f(\mathbf{x})$ be F -monotonically increasing. Then $d(Q, t) = f(Q) - f(t) = |f(Q) - f(t)|$ is an F -monotonically decreasing function with respect to t . Thus, the above query model with TA can become an equivalent model that finds N tuples $\langle t_1, t_2, \dots, t_N \rangle$ from $\mathbf{R}$ with the smallest values of $d(Q, t)$. The query model with TA has three characteristics: (1) the ranking function is F -monotone; (2) the query point Q is fixed, say, $Q = (1, 1, \dots, 1)$ or $(0, 0, \dots, 0)$, i.e., Max-point or Min-point; and (3) TA scans the sorted index lists unidirectionally (i.e., top down). However, if a query point is neither Max-point nor Min-point, the F -monotonicity of $d(Q, t) = |f(Q) - f(t)|$ cannot be guaranteed. For example, $f(\mathbf{x}) = x_1 + x_2$ is F -monotonically increasing for arbitrary $\mathbf{x} = (x_1, x_2)$ in $[0, 1]^2$, but $d(Q, t) = |f(Q) - f(t)|$ is not F -monotone when $Q = (q_1, q_2) = (0.5, 0.5)$. Similarly, the *Manhattan* distance function $d(Q, t)$

$= \sum_{i=1}^n |q_i - t_i|$ is not F -monotone for $Q = (q_1, q_2, \dots, q_n)$ and $t = (t_1, t_2, \dots, t_n)$ in $[0, 1]^n$ if Q is neither Max-point nor Min-point.

TA is a simple but very powerful algorithm with $O(n|\mathbf{R}|)$ space and instance optimality [14], where $|\mathbf{R}|$ indicates the number of tuples in the relation $\mathbf{R}$ (i.e., the size of $\mathbf{R}$). There are many variations of TA for different applications [20]. In general, the techniques of processing top- N queries are dependent on the properties of ranking functions. The F -monotonicity of ranking functions plays a central role in TA-like algorithms. Most proposed techniques take into account F -monotone ranking functions, and few proposals deal with general functions as using nonmonotone ranking functions in top- N queries is a challenge [20].

In many applications, there are opportunities for evaluating top- N queries with arbitrary query points, and the ranking function is not F -monotone, as demonstrated by the following example [39, 41].

Example 1. Consider a database system of used cars with schema *Usedcars*(*id#*, *make*, *model*, *year*, *price*, *mileage*), the top-50 query Q with (*price* = \$5000, *mileage* = 6000) and the ranking function being the Manhattan distance between a query point and a tuple t . The query point $Q = (5000, 6000)$ is not likely to be either Max-point or Min-point, when $\min(\text{price}) < 5000 < \max(\text{price})$ and $\min(\text{mileage}) < 6000 < \max(\text{mileage})$. $\square$

As shown above, the ranking function in Example 1 is not F -monotone when a query point is neither Max-point nor Min-point, thus, TA is not applicable for processing this query as described in [38].

The query model in Example 1 looks like the model of Nearest Neighbor Search (NNS) (or K -Nearest Neighbor (KNN) Query) in the n -dimensional ℓ_p spaces. NNS in the metric spaces, typically in Euclidean spaces, has been an important issue in Computational Geometry and has attracted much attention due to its applications in a wide range of domains [21, 36]. Generally, NNS involves sophisticated hierarchical data structures and methods, and it has been solved optimally in low dimensions (usually less than 10) [35, 36]. For instance, if points lie on a 2-dimensional plane, the nearest neighbor can be found with $O(\log|\mathbf{R}|)$ time per query, using only $O(|\mathbf{R}|)$ of storage space [17, 36]. Due to the “dimensionality curse”, however, it

is one of the most challenging open problems to perform exact NNS in high dimensions [36]. Thus, many methods are proposed to efficiently process NNS only for the approximate cases [17, 21, 33]. For applications that need to support nearest neighbor queries efficiently and exactly, it is crucial to find good data structures and design efficient algorithms that scale well with the database size and with the dimension [21].

From the database perspective, in this paper, we design algorithms inspired by the methods in [14] to evaluate the models of exact top- N queries as shown in Example 1. Our query models will be different from the models of NNS in metric spaces that require nothing else than the object distance function [34]. In general, the various complex index structures and access methods for NNS are not currently supported by many traditional relational database management systems [5]. Besides distance (or ranking) functions, however, our query models are based on “sorted and/or random accesses” and “no wild guesses”, which demand that the algorithms should be database-friendly, efficient and practical [6, 13, 14, 20]. For this purpose, we abstract a relation $\mathbf{R}$ in a database system to a finite dataset in a real n -dimensional normed space $\mathfrak{R}^n$, and then we develop a new approach using the principle of Functional Analysis to transform a generic norm distance to the *Maximum* distance. Thus, our processing techniques of top- N queries are independent of the ranking functions in n -dimensional normed spaces, and as a result, the ranking functions are not the key factors in designing the algorithms. The contributions of this paper are summarized below:

- We analyze the query model in a real n -dimensional normed space where: (1) the ranking function is the distance induced by an arbitrarily given norm, which is not necessarily F -monotone, and (2) the query point is an arbitrary point (i.e., a real vector). TA-like methods are not suitable for this query model. We propose database-friendly algorithms to answer the top- N queries of this model with the following three cases of data access methods: (1) both sorted and random accesses, (2) restricting sorted access, and (3) no random access [6, 14, 20].
- For real n -dimensional spaces with generic norms, we present three algorithms GTA, GTAz and GNRA for the above three cases of data access methods, respectively.

- For $\mathbf{x}$ -monotonic norm distances (say, p -norm distances) in real n -dimensional normed spaces, we provide three algorithms mTA, mTAz and mNRA for the above three cases of data access methods, respectively.
- For both low-dimensional (2, 3, and 4) and high-dimensional (25, 50, and 104) data, we conduct extensive experiments that demonstrate the good performance of our proposed algorithms.

The rest of the paper is organized as follows. In Section 2, we briefly review some related work. In Section 3, we introduce some notions, the query model and the main idea of our approach. Section 4 proposes the algorithms that process top- N queries. Section 5 discusses the buffer size and optimality of our algorithms, and compares the algorithms that are instance optimal in the TA family against optimal algorithms of NNS. In Section 6, we present the experimental results. Finally, Section 7 concludes the paper.

2. RELATED WORK

Numerous research works on the efficient evaluation of top- N queries have been proposed. A survey of relational top- N query processing can be found in [20], which classified the processing techniques into three categories based on the type of ranking functions, including (1) *F-monotone function*, (2) *generic function*, and (3) *no ranking function* (say, the skyline technique [4]). Here, we only review some of the related works involving *F-monotone* or *generic* ranking functions.

2.1. Monotone Ranking Functions

Most of the current techniques for processing top- N queries are based on *F-monotone* ranking functions (including linear functions) since they are applicable to many practical applications and have appealing properties making for handling top- N queries efficiently [20], for instance, the threshold algorithm (TA) [14] and its family (say, [1, 15, 19, etc.]).

Nepal and Ramakrishna [28] and Güntzer et al. [15] proposed their own algorithms independently that are equivalent to TA for processing queries over multimedia databases. The algorithms in [6, 37] are introduced in the context of Web-accessible sources. Using index structures, [19] proposed the MPro

algorithm that ensures every probe performed is necessary for evaluating the top- N tuples. Tracking the “best positions” in each sorted list and reducing the number of accesses, [1] presented techniques BPA and BPA2 to optimize the TA algorithm; the execution cost of BPA can be $(n-1)$ times lower than that of TA, and that of BPA2 can be about $(n-1)$ times lower than that of BPA for n sorted lists. [27] proposed the algorithm LARA to optimize the NRA algorithm in [14] by employing a lattice to reduce the computational cost of NRA; thus, LARA can access fewer objects and be orders of magnitude faster than NRA. Building a table $\mathbf{R}'$ to maintain the sorted access order in the lists such that the score computation of an object using sorted lists can be reduced by $\mathbf{R}'$, the STA method in [23] outperforms a few state-of-the-art TA-like algorithms by 1-2 order of magnitude. [18] proposed a layer-ordering method, PL-Index, by using the convex skyline and partitioning-merging technique. Top- N queries involving joins have also been studied (e.g., [25, 16, 29]). The RankSQL in [25] discusses the order of binary rank joins at query-planning time. Han et al. [16] presented the TJJE algorithm which is suitable for handling massive data by using pre-computed information. [29] introduced a model for determining the expected depths $depth_{thres}$ and $depth_{result}$ by employing Bloom filters to group the sorted lists into buckets.

Obviously, our query models are different from those of the TA family since the ranking functions in our models are not necessarily F -monotone.

2.2. Nonmonotone Ranking Functions

Efficient processing of top- N queries using nonmonotone ranking functions is a challenging task [20]. Few methods address general functions, since they cannot benefit from special properties of F -monotone functions that facilitate early termination [20].

The foundation of the methods in [38] and [40] is to create recursively a space of joint states by exploiting existing B-tree and R-tree indices, i.e., for each joint state S , its child states are the Cartesian products of child nodes of every index node in S . The search algorithm OPT* proposed in [40] supports nonmonotone ranking functions by modeling top- N query as an N -constrained optimization problem. Xin et al. [38] extended TA to ad hoc ranking functions with lower-bounded; they sorted the joint state space

by $f(S)$ that is the lower bound of the function $f(\cdot)$ over a joint state S , and presented the methods BL, PE and PE+SIG. [31] introduced the model of SD-Query involving a kind of non-monotonic scoring function, which is a mixture of similarity on attractive dimensions and distance on repulsive dimensions, i.e., the difference of two weighted Manhattan distances. SD-Index methods in [31] develop index structures based on the isolines of the scoring functions.

The algorithms in [38, 40] extend TA to support ad hoc ranking functions that are not F -monotone; however, they are not suitable to arbitrary query points, since the form of their ranking functions is $f(t)$ with one variable (i.e., tuple t). Furthermore, for high dimensions (say, larger than 12), the performance of the algorithms rapidly decreases because of R-tree index or the joint state space [2]. In contrast, the form of our ranking functions is $d(Q, t)$ with two variables (query point Q and tuple t). Based on isoline, the methods in [31] are the special techniques for particular ranking functions; however, our methods can be used for a wider range of ranking functions.

Based on space partitioning and data partitioning techniques, there are many sophisticated data structures and methods for nearest neighbor search (NNS) in high dimensional spaces [33]. For example, locality sensitive hashing (LSH) [17, 21] and iDistance [22] are two state-of-the-art methods. For approximate NNS, LSH is efficient and scalable due to its probabilistic nature, and Multi-Probe LSH is proposed in [26] to reduce the space cost of LSH by probing more data. For exact NNS, iDistance indexes all points by their distances to the reference point in each partition and it works well for clustered datasets of dimensions as high as 30; however, iDistance will incur expensive query costs for higher dimensional datasets (say, ≥ 32). Inspired by iDistance, the SIMP technique proposed in [35] consists of an index structure and a two-step pruning method in high (between 32 and 256) dimensional Euclidean space, which is significantly higher than the dimensions that can be handled by iDistance. SIMP computes multiple 2-dimensional projections of the high dimensional data by using polar grids of multiple viewpoints chosen randomly from the dataset in order to reduce query cost, while incurring higher space cost than iDistance [35]. The methods with sophisticated data structures (e.g., various complex trees) are fixed to distance functions that are available to the system in advance; however, they do not support

adaptable distance functions [34]. Based on X-tree, [34] presented a multi-step algorithm for processing deterministic KNN queries (i.e., the answer contains at least K objects, and all objects that have the same distance as the K th ranked objects are returned); the key idea of the multi-step algorithm is its first step that gets the optimal value of a filter distance function by evaluating some selected objects in the dataset. In this paper, we design algorithms with a simple data structure (i.e., a dynamic array, or a B^+ -tree) from the database perspective.

Moving one of two pointers (one up and the other down) is a useful mechanism when searching a database, and there are various Tools or Controls/Widgets with up-down or previous-next pointers for developing database applications. By maintaining and using two pointers for each sorted list, [8] and [13] proposed the methods to process approximate nearest neighbors in the Hamming spaces [8] and in the Euclidean spaces [13], respectively. Using the geometric lemma (Lemma 2.1) in [24] and $m = \alpha \varepsilon^{-2} \log(|\mathbf{R}|)$ attributes selected randomly from n attributes of $\mathbf{R}$, [13] proposed three database-friendly algorithms MEDRANK, OMEDRANK and L2TA. Using the method of “no random access”, the algorithm MEDRANK [13] is derived from the two algorithms FA [12, 14] and NRA [14], and OMEDRANK is a variation of MEDRANK. Differing from the NRA [14], the MEDRANK and OMEDRANK do not employ any threshold values; they obtain an approximate nearest neighbor that is the seen element with the largest frequency f_c satisfying $f_c > \text{MINFREQ} * m$ for the parameter MINFREQ between 0.5 and 0.9. In the spirit of the TA [14], the algorithm L2TA in [13] uses threshold values via both sorted and random accesses. [13] did not discuss the method of restricting sorted accesses, which is important in Web-accessible databases [6]. Fagin et al. showed that the algorithm MEDRANK is instance optimal in [13], where they ignored the cost of locating the query. Similar to the NRA and TA [14], for m sorted lists, the execution cost (or called middleware cost in [14]) of MEDRANK is between $O(m)$ and $O(m|R|)$, whereas the location cost of a query Q is $O(m \log|R|)$ by binary search, and will be a significant factor in the search time for processing Q when the execution cost of Q is $O(m)$. Unlike exact NNS, approximate NNS does not suffer from the dimensionality curse so it is easier to process [3]. In our work, we evaluate exact top- N queries (or exact nearest neighbors) in n -dimensional normed spaces. Using the fundamental principle

of Functional Analysis and threshold values, our database-friendly algorithms will also employ the mechanism with up-down pointers. We will introduce “pseudo instance optimal” that involves the location cost of a query, and will define two algorithms GTAz and mTAz that use the method of restricting sorted accesses.

The query model in [5, 9, 41] is most related to the model in this paper. In [5, 9, 41], the ranking functions are p -norm distances for $p = 1, 2$, and ∞ , and the queries may be arbitrary points in the real vector space $\Re^n$. Given a top- N query, the basic idea of the strategies in [5, 9, 41] is to find a small n -dimensional square centered at the query point with side length $2r$ such that all of the top N tuples but very few undesired ones are contained in the n -dimensional square, and then the strategies are used to map the top- N query over a relational database into a traditional *range selection query*. The key of techniques in [5, 9, 41] is how to estimate the search distance r , i.e., half of the side length $2r$. The main weakness of the methods in [5, 9, 41] is that an incorrect estimation of the search distance r may lead to the number of retrieved tuples to be either less than or much larger than N .

The histogram-based approaches in [5] deal with exact top- N queries and guarantee the retrieval of all top- N tuples, but they are not suitable for high-dimensional datasets, since their performance deteriorates rapidly when the number of dimensions of the data exceeds 3 [5, 9]. In contrast, the sampling-based methods in [9] for approximate top- N queries do not suffer from the “dimensionality curse” as they are suitable for high-dimensional data, but they do not guarantee the retrieval of exact top- N tuples (for 2 to 104 dimensional data, the target recall is set to 90% and 95% in the experiments [9]). The learning-based method in [41] for exact top- N queries guarantees the retrieval of all top- N tuples for high-dimensional datasets.

We would like to point out that the methods in [5, 9, 41] are different from our algorithms; in fact, they are not the members of the TA family, but of the Filter-Restart category [20]. The SELECT-FROM-WHERE SQL statements are the component parts of the methods in [5, 9, 41]; however, no such SQL statements are used in the TA-like algorithms.

3. PROBLEM DEFINITION

In this section, we first present the preliminaries, and then introduce the query model and the main idea of our approach.

3.1. Preliminaries

In this subsection, we present normed spaces, related terminologies and some basic properties (e.g., theorems) that are known in the literature of Functional Analysis (say, [10]). They are included here (some are re-worded) in order to keep the discussion self-contained.

Definition 3.1 (Normed Space). Let X be a linear space (or a vector space) over the field $\mathfrak{R}$ of real scalars. A *norm* on X is a function $\|\cdot\|: X \rightarrow [0, \infty)$ such that

$$(1^\circ) \|\mathbf{x}\| \geq 0 \text{ for all } \mathbf{x} \in X, \text{ and } \|\mathbf{x}\| = 0 \text{ if and only if } \mathbf{x} = \mathbf{0};$$

$$(2^\circ) \|c\mathbf{x}\| = |c| \|\mathbf{x}\| \text{ for all } \mathbf{x} \in X \text{ and } c \in \mathfrak{R};$$

$$(3^\circ) \text{ (Triangle Inequality): } \|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|.$$

A linear space X together with a norm $\|\cdot\|$ is called a *normed linear space*, a *normed vector space*, or simply a *normed space*, denoted by $(X, \|\cdot\|)$ or X for short.

Let $(X, \|\cdot\|)$ be a normed space. The norm $\|\cdot\|$ can induce a *distance* on X : $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$ for arbitrary $\mathbf{x}, \mathbf{y} \in X$. In the following discussion, we assume that a distance $d(\mathbf{x}, \mathbf{y})$ on a linear space X can be induced by a norm $\|\cdot\|$, and then we do not distinguish between $d(\mathbf{x}, \mathbf{y})$ and $\|\mathbf{x} - \mathbf{y}\|$.

Definition 3.2. If $\|\cdot\|$ and $\|\cdot\|'$ are two norms on a linear space X , they are called to be *equivalent norms* if there exist positive constants c_1 and c_2 such that

$$c_1 \|\mathbf{x}\| \leq \|\mathbf{x}\|' \leq c_2 \|\mathbf{x}\|, \text{ for all } \mathbf{x} \in X.$$

Theorem 3.1 (Norm Equivalence Theorem). All norms on a finite dimensional linear space are equivalent.

For the real vector space $\mathfrak{R}^n$, as an example, elliptic norm is an important one on $\mathfrak{R}^n$, and is defined by $\|\mathbf{x}\| = (\mathbf{x} \mathbf{A} \mathbf{x}^T)^{1/2}$, where $\mathbf{A}$ is a symmetric positive definite matrix, $\mathbf{x} = (x_1, x_2, \dots, x_n) \in \mathfrak{R}^n$ is a row vector,

and $\mathbf{x}^T$ denotes its transpose. In particular, p -norm (or ℓ_p -norm) $\|\cdot\|_p$ plays an important role on $\mathfrak{R}^n$, for $\mathbf{x} = (x_1, x_2, \dots, x_n) \in \mathfrak{R}^n$

$$\|\mathbf{x}\|_p = (\sum_{i=1}^n |x_i|^p)^{1/p}, \text{ for } 1 \leq p < \infty,$$

$$\|\mathbf{x}\|_\infty = \max_{1 \leq i \leq n} (|x_i|), \text{ for } p = \infty$$

Moreover, $\|\mathbf{x}\|_p \rightarrow \|\mathbf{x}\|_\infty$ as $p \rightarrow \infty$. In $(\mathfrak{R}^n, \|\cdot\|_p)$, p -norm $\|\cdot\|_p$ will induce distance function $d_p(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_p$. When $p=1, 2$, and ∞ , $d_p(\mathbf{x}, \mathbf{y})$ will be the *Manhattan distance* $d_1(\mathbf{x}, \mathbf{y})$, *Euclidean distance* $d_2(\mathbf{x}, \mathbf{y})$, and *Maximum distance* $d_\infty(\mathbf{x}, \mathbf{y})$ respectively, which are useful in many applications [5, 9, 39, 41].

One of the important basic properties of $\|\cdot\|_p$ is that it is monotone decreasing in terms of p , and is called “ p -monotone” in our paper, which is stated as the following Theorem 3.2.

Theorem 3.2. $\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_q \leq \|\mathbf{x}\|_p \leq \|\mathbf{x}\|_1$, for any $1 < p < q < \infty$, and arbitrary fixed vector $\mathbf{x} \in \mathfrak{R}^n$.

Moreover, $\|\mathbf{x}\|_p \leq n^{(1/p-1/q)} \|\mathbf{x}\|_q$, if $1 \leq p < q$.

Given a normed space $(X, \|\cdot\|)$, the set $\{\mathbf{x} \in X: \|\mathbf{x}\| \leq 1\}$ is called the (closed) unit ball in $(X, \|\cdot\|)$. The sets are depicted in Figure 1 for several norms on $\mathfrak{R}^2$.

By Theorem 3.2, for $\mathfrak{R}^n$, as shown in Figure 1(a), we have

Corollary 3.1. Let B_p denote the unit ball $\{\mathbf{x} \in \mathfrak{R}^n: \|\mathbf{x}\|_p \leq 1\}$. Then $B_p \subset B_q$ for $1 \leq p < q \leq \infty$. In particular, $B_1 \subset B_{3/2} \subset B_2 \subset B_4 \subset B_\infty$.

(a) For p -norm $\|\cdot\|_p$

(b) For elliptic norm $\|\mathbf{x}\| = (\mathbf{x}A\mathbf{x}^T)^{1/2}$

Figure 1. The unit balls of several norms on $\mathfrak{R}^2$. (a) For p -norm $\|\cdot\|_p$, they are described as diamond, superellipse, circle, squircle, and square from inner to outer when $p = 1, 3/2, 2, 4$, and ∞ , respectively. (b) The ball is an ellipse for elliptic norm $\|\mathbf{x}\| = (\mathbf{x}A\mathbf{x}^T)^{1/2}$.

By Theorem 3.1, all norms on $\mathfrak{R}^n$ are equivalent, we have

Theorem 3.3. Given a norm $\|\cdot\|$ on $\Re^n$, there are positive constants α and β such that

$$\alpha\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\| \leq \beta\|\mathbf{x}\|_\infty \text{ for all } \mathbf{x} \in X. \quad (3.1)$$

In particular, we have

$$(1^\circ) \quad \|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_1 \leq n\|\mathbf{x}\|_\infty$$

$$(2^\circ) \quad \|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_2 \leq \sqrt{n} \|\mathbf{x}\|_\infty$$

$$(3^\circ) \quad \|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_p \leq n^{1/p} \|\mathbf{x}\|_\infty$$

Theorems 3.1, 3.2 and 3.3 lay the foundation of our algorithms for evaluating top- N queries in $(\Re^n, \|\cdot\|)$.

Theoretically speaking, we can obtain candidate tuples for a top- N query by transforming a distance $\|\mathbf{x}-\mathbf{y}\|$ into the Maximum distance $\|\mathbf{x}-\mathbf{y}\|_\infty$. Moreover, the following concept will be used in our algorithms.

A norm $\|\cdot\|$ on $\Re^n$ is monotonic (with respect to $\mathbf{x}$) if and only if $|x_i| \leq |y_i|, i = 1, \dots, n$, implies that $\|\mathbf{x}\| \leq \|\mathbf{y}\|$. Here, the term “monotonic” is said to be “ $\mathbf{x}$ -monotonic” in order to distinguish it from “ p -monotone” in Theorem 3.2 and “ F -monotone” in Section 1. Similarly, a function $f(\cdot)$ is $\mathbf{x}$ -monotonic if and only if $|x_i| \leq |y_i|, i = 1, \dots, n$, implies that $f(\mathbf{x}) \leq f(\mathbf{y})$. Recall that $f(\cdot)$ is F -monotone if and only if $f(\mathbf{x}) \leq f(\mathbf{y})$ whenever $x_i \leq y_i$ for every i (see Section 1).

If p is a fixed real number with $1 \leq p \leq \infty$, then p -norm $\|\mathbf{x}\|_p$ is $\mathbf{x}$ -monotonic. However, the elliptic norms are not $\mathbf{x}$ -monotonic in general. In fact, the elliptic norm $\|\mathbf{x}\| = (\mathbf{x} \mathbf{A} \mathbf{x}^T)^{1/2}$ is $\mathbf{x}$ -monotonic if and only if matrix $\mathbf{A}$ is diagonal.

If a query point $Q = (q_1, \dots, q_n)$ is neither Max-point nor Min-point as defined in Section 1, the F -monotonicity of p -norm distance function $d_p(Q, \mathbf{t})$ cannot be guaranteed for $\mathbf{t} = (t_1, \dots, t_n)$. For example, we consider the Manhattan distance function $d_1(Q, \mathbf{t}) = \sum_{i=1}^2 |q_i - t_i|$ in the square $[0, 1]^2$, and $Q = (q_1, q_2) = (0.5, 0.5)$. If $\mathbf{s} = (s_1, s_2) = (0.1, 0.2)$ and $\mathbf{t} = (t_1, t_2) = (0.6, 0.7)$, then $s_i \leq t_i$ for every $1 \leq i \leq 2$, and $d_1(Q, \mathbf{s}) = 0.7 > 0.3 = d_1(Q, \mathbf{t})$. On the other hand, if $\mathbf{x} = (x_1, x_2) = (0.4, 0.4)$ and $\mathbf{y} = (y_1, y_2) = (0.7, 0.8)$, then $x_i \leq y_i$ for every i , $d_1(Q, \mathbf{x}) = 0.2 < 0.5 = d_1(Q, \mathbf{y})$. Thus, $d_1(Q, \mathbf{t})$ is not F -monotone.

In mathematics, the monotonicity (i.e., F -monotonicity in this paper) of a ranking function $f(\cdot)$ in TA-like model involves the component-wise inequality, which is an expression of the form: $\mathbf{x} \leq \mathbf{y}$ if and only if $x_i \leq y_i, i = 1, 2, \dots, n$. We will extend TA-like model to the model with $|x_i| \leq |y_i|, i = 1, \dots, n$, using $\mathbf{x}$ -monotonic norms (say, $\|\cdot\| = a\|\cdot\|_2 + b\|\cdot\|_\infty$ for $a, b > 0$ and p -norms are $\mathbf{x}$ -monotonic, but not F -monotonic). Without loss of generality, we only consider $\mathfrak{R}^n$ in the following discussion.

3.2. The Model of Top- N Queries

Let $(\mathfrak{R}^n, \|\cdot\|)$ be an n -dimensional normed space and $d(\cdot, \cdot)$ be the distance function induced by $\|\cdot\|$. Suppose $\mathbf{R} \subset \mathfrak{R}^n$ is a relation (or a finite dataset). Consider a query point $Q = (q_1, \dots, q_n) \in \mathfrak{R}^n$ and an integer $N > 0$. A top- N selection query (Q, N) , or top- N query for short, is to find a sorted set of N tuples in $\mathbf{R}$ that are closest to Q according to the distance $d(\cdot, \cdot)$. The results of top- N query are called top- N tuples. We assume $N < |\mathbf{R}|$ (the size of $\mathbf{R}$); otherwise, we just retrieve all tuples in $\mathbf{R}$. For simplicity, a top- N query (Q, N) is usually denoted by Q .

Equivalently, a sorted set $\mathbf{Y} = \langle t_1, t_2, \dots, t_N \rangle \subset \mathbf{R}$ is one of the answers to (Q, N) according to $d(\cdot, \cdot)$ if and only if $d(t_1, Q) \leq d(t_2, Q) \leq \dots \leq d(t_N, Q)$ (ties are broken arbitrarily) and $d(t_N, Q) \leq d(t, Q)$ for arbitrary $t \in \mathbf{R} - \mathbf{Y}$. Obviously, the result set of (Q, N) is non-deterministic when more than N tuples satisfy the above condition.

In general, the schema of the relation $\mathbf{R}$ is $\mathbf{R}(tid, A_1, \dots, A_n)$ with n attributes $(A_1, \dots, A_n)$ corresponding to $\mathfrak{R}^n = \mathfrak{R}_1 \times \dots \times \mathfrak{R}_n$ where the i th axis $\mathfrak{R}_i = \mathfrak{R}$ for every i , each tuple in $\mathbf{R}$ is associated with a *tid* (tuple identifier), and $\mathbf{R}$ is stored as a base table in a relational database system. We do not distinguish between $\mathbf{R}(tid, A_1, \dots, A_n)$ and $\mathbf{R}(A_1, \dots, A_n)$ if there is no need to refer to *tid*.

Definition 3.3. Given a point $Q \in \mathfrak{R}^n$ and $r > 0$, $B(Q, r, d) \subset \mathbf{R}$ denotes the (closed) ball centered at Q with radius r for the distance $d(\cdot, \cdot)$, i.e., $B(Q, r, d) = \{t \in \mathbf{R}: d(t, Q) \leq r\}$. $|B(Q, r, d)|$ means the size of $B(Q, r, d)$, i.e., the number of tuples in $B(Q, r, d)$.

Note: (1) As shown in Figure 1, generally, $B(Q, r, d) = \{x \in \mathfrak{R}^n: d(x, Q) \leq r\}$ is the subset of space $\mathfrak{R}^n$ in the literature of Functional Analysis, particularly, $B(Q, r, d_\infty) = \prod_{i=1}^n [q_i - r, q_i + r]$ is the n -dimensional square (n -square for short) centered at Q with side length $2r$, i.e., the Cartesian product of the n intervals $[q_i - r, q_i + r], i = 1, \dots, n$; while our ball $B(Q, r, d)$ is the subset of the relation $\mathbf{R}$, in fact, our ball $B(Q, r, d) = \{x \in \mathfrak{R}^n: d(x, Q) \leq r\} \cap \mathbf{R}$. A tuple $t = (t_1, \dots, t_n) \in B(Q, r, d_\infty)$ means $q_i - r \leq t_i \leq q_i + r$ for every i , then it is easy to determine if t is in $B(Q, r, d_\infty)$. (2) In the following discussion, an *element* (or a *vector*) will be called a *point* if it is in $\mathfrak{R}^n$, and called a *tuple* if we need to emphasize that it is in $\mathbf{R}$. (3) Our approach can apply with impunity to any weighted norm, thus, “circles” and “squares” with p -norm in Figure 1(a) will become “ellipses” and “rectangles” respectively.

Using the positive constants α and β in Theorem 3.3, we have

Theorem 3.4. Let $(\mathfrak{R}^n, \|\cdot\|)$ be a normed space, and $\mathbf{R} \subset \mathfrak{R}^n$ be a relation. Then,

$$B(Q, r/\beta, d_\infty) \subset B(Q, r, d) \subset B(Q, r/\alpha, d_\infty) \quad (3.2)$$

Moreover, $|B(Q, r, d)| \geq N$ if $|B(Q, r/\beta, d_\infty)| \geq N$ and only if $|B(Q, r/\alpha, d_\infty)| \geq N$.

Proof. First, (3.2) follows from (3.1) in Theorem 3.3.

For any tuple $t \in B(Q, r/\beta, d_\infty)$, by the definition of $B(Q, r/\beta, d_\infty)$, we have $d_\infty(t, Q) \leq r/\beta$. Since $\|t - Q\|_\infty = d_\infty(t, Q)$, $\|t - Q\|_\infty \leq r/\beta$, that is, $\beta\|t - Q\|_\infty \leq r$. By the right part of (3.1) in Theorem 3.3, we obtain $\|t - Q\| \leq \beta\|t - Q\|_\infty \leq r$; therefore, $t \in B(Q, r, d)$, thus $B(Q, r/\beta, d_\infty) \subset B(Q, r, d)$. Similarly, we can show that the right part of (3.2) is true, i.e., $B(Q, r, d) \subset B(Q, r/\alpha, d_\infty)$.

Next, by (3.2), the second part of Theorem 3.4 holds. $\square$

Corollary 3.2. $B(Q, \alpha r, d) \subset B(Q, r, d_\infty) \subset B(Q, \beta r, d)$.

By Theorem 3.4, we also have

$$B(Q, r, d) = \{t \in \mathbf{R}: t \in B(Q, r/\alpha, d_\infty) \text{ and } d(t, Q) \leq r\} \quad (3.3)$$

and then we have

Theorem 3.5. Let $(\mathfrak{R}^n, \|\cdot\|)$ be a normed space, and $\mathbf{R} \subset \mathfrak{R}^n$ be a relation. Then,

$$B(Q, \alpha r, d) = \{t \in \mathbf{R}: t \in B(Q, r, d_\infty) \text{ and } d(t, Q) \leq \alpha r\} \quad (3.4)$$

Moreover, $|B(Q, \alpha r, d)| \geq N$ if and only if $|\{t \in \mathbf{R}: t \in B(Q, r, d_\infty) \text{ and } d(t, Q) \leq \alpha r\}| \geq N$.

Proof. Replacing r in (3.3) by αr , we have (3.4). $\square$

Figure 2(a) illustrates an example of Theorem 3.4 for $d(\cdot, \cdot) = d_2(\cdot, \cdot)$ in $\mathfrak{R}^2$. If $B(Q, r/\sqrt{2}, d_\infty)$ contains N tuples in $\mathbf{R} \subset \mathfrak{R}^2$, $B(Q, r, d_2)$ have at least N tuples in $\mathbf{R}$. On the other hand, if $B(Q, r, d_\infty)$ contains N tuples, $B(Q, r, d_2)$ may or may not contain N tuples. Figure 2(b) shows an example of Theorem 3.5 in $\mathfrak{R}^2$, that is, $B(Q, r, d_2) = \{t \in \mathbf{R}: t \in B(Q, r, d_\infty) \text{ and } d_2(t, Q) \leq r\}$, which means that “for a tuple $t \in \mathbf{R}$, if $t \in B(Q, r, d_\infty)$ then it is a candidate tuple; moreover, if $d_2(t, Q) \leq r$ then t is one of the top- K tuples where $K = |B(Q, r, d_2)|$ ”.

(a) An example of Theorem 3.4

(b) An example of Theorem 3.5

Figure 2. Let $Q = (0, 0)$ in $\mathfrak{R}^2$. The symbol “x” indicates Q , and “•” a tuple. (a) $B(Q, r, d_2)$ is a circle centered at Q with radius $r > 0$, it is the circumcircle of the smaller square $B(Q, r/\sqrt{2}, d_\infty)$, and the incircle of the larger square $B(Q, r, d_\infty)$. (b) Tuple t is in the circle $B(Q, r, d_2)$ if and only if t is in the square $B(Q, r, d_\infty)$ and $d_2(t, Q) \leq r$.

Theorem 3.5 describes the main idea of our approach to obtain the top- N tuples for a query Q in $(\mathfrak{R}^n, \|\cdot\|)$. That is, we enlarge the n -square $B(Q, r, d_\infty)$ step by step; if $d(t, Q) \leq \alpha r$ for $t \in B(Q, r, d_\infty)$, then t is a top- K tuple of Q with $K = |B(Q, r, d)|$. We stop the process until we find the top- N tuples. Figure 2(b) shows an example for $d(\cdot, \cdot) = d_2(\cdot, \cdot)$ in $\mathfrak{R}^2$; when the square is enlarged again and again, the circle will be enlarged and will contain more and more tuples. When there are at least N tuples in the circle, we rank these tuples according to $d_2(\cdot, \cdot)$, then output the top- N tuples.

4. ALGORITHMS

Motivated by the spirit of the algorithms in the TA family, “simplicity is power”, we design algorithms that are also simple and powerful in the case of no wild guesses [14], that is, a tuple must be “discovered” under sorted access before it can be probed using random access. For the three cases of data access methods: (1) *both sorted and random access*, (2) *restricting sorted access*, and (3) *no random access* [14, 20], we first discuss general norms on $\mathfrak{R}^n$ and give three algorithms GTA, GTAz and GNRA respectively, and then define algorithms mTA, mTAz and mNRA respectively in $\mathfrak{R}^n$ with $\mathbf{x}$ -monotonic norms. In the names of our algorithms, prefix “G” indicates “General norm”, prefix “m” means “(x-)monotonic norm”; while TA stands for the Threshold Algorithm, NRA means “no random access”, and suffix “z” indicates $Z = \{i_1, i_2, \dots, i_m\} \subset \{1, 2, \dots, n\}$ ($1 \leq m < n$), the set of indices i of those lists L_i that can be accessed under sorted access.

For each attribute/column A_i of a relation $\mathbf{R}(tid, A_1, \dots, A_n) \subset \mathfrak{R}^n$, its values are sorted in ascending order from $\min(A_i)$ to $\max(A_i)$, $1 \leq i \leq n$. In database applications, many existing techniques for evaluating top- N queries are based on tree-structured indices (e.g., B-tree, R-tree [9, 38] or B^+ -tree [1, 5, 40]) to index the attribute values.

For each attribute A_i , assume that there is a B^+ -tree in which all leaf nodes form a *doubly linked list* L_i in sorted A_i -values [30]. The B^+ -tree is used to locate q_i in L_i quickly for a given query $Q = (q_1, \dots, q_n) \in \mathfrak{R}^n$, $1 \leq i \leq n$. Alternately, we may use a dynamic array L_i in sorted A_i -values, and determine the position of q_i in L_i by binary search. Therefore, the cost of locating q_i is $O(\log|\mathbf{R}|)$ in the worst case.

Each entry of L_i is of the form (tid, a) , where $a = \mathbf{t}[A_i]$ is the value of $\mathbf{t}$ with tid under attribute A_i (i.e., the coordinate of $\mathbf{t}$ on the i th axis $\mathfrak{R}_i$). We use $a_{ij} = L_i.a_j$ to indicate L_i 's j th attribute-value.

Let $\{L_1, \dots, L_n\}$ be the set of n sorted lists of the relation $\mathbf{R} \subset \mathfrak{R}^n$. We say that an algorithm $\mathcal{A}$ locates $Q = (q_1, \dots, q_n)$ on $\mathbf{R}$ if $\mathcal{A}$ can locate Q over $\{L_1, \dots, L_n\}$, that is, can find a_{ij} and $a_{i,j+1}$ in L_i such that $a_{ij} \leq q_i \leq a_{i,j+1}$ for $i = 1, \dots, n$.

We consider two modes of access to data, sorted access and random access. The meaning of “random access” in our paper is the same as in [14], i.e., the value of a tuple’s attribute is obtained by providing the *tid*. “Sorted access” in [14] is “to obtain the attribute value of a tuple in one of the sorted lists by proceeding through the list sequentially from the top”. Here, we define the direction(s) of “sorted access” in this paper as follows.

Suppose that the query $Q=(q_1, \dots, q_n)$ has been located on $\mathbf{R}$, say, we have determined the position of q_i in L_i with $a_{ij} \leq q_i \leq a_{ij+1}$ for every i by a B^+ -tree or binary search.

If $q_i \leq \min(A_i)$, then $a_{ij+1} = \min(A_i)$, the direction is “ $q_i \rightarrow \max(A_i)$ ”, i.e., “sorted access” scans the list L_i sequentially from $\min(A_i)$ to $\max(A_i)$. If $\max(A_i) \leq q_i$, then $a_{ij} = \max(A_i)$, “ $\min(A_i) \leftarrow q_i$ ”, i.e., sorted access is from $\max(A_i)$ to $\min(A_i)$. For the two cases, sorted access has the same meaning as in [14]. If $\min(A_i) < q_i < \max(A_i)$, we use two directions “ $\min(A_i) \leftarrow q_i \rightarrow \max(A_i)$ ” for sorted access, i.e., from a_{ij} to $\min(A_i)$ for the interval $[\min(A_i), a_{ij}]$, and from a_{ij+1} to $\max(A_i)$ for $[a_{ij+1}, \max(A_i)]$.

We consider two types of costs, *location cost* and *execution cost*, for our algorithms.

Location cost is the cost incurred by locating $Q = (q_1, \dots, q_n)$ on $\mathbf{R}$. For example, it is $O(n \log |\mathbf{R}|)$ for n sorted lists $L_1, \dots, L_n$ by binary search.

Execution cost (called *middleware cost* in [14]) is the sum of sorted access cost and random access cost. Let c_s be the cost of a sorted access, and c_r be the cost of a random access. If an algorithm does s sorted accesses and r random accesses to find the top- N tuples, then its execution cost is $s \cdot c_s + r \cdot c_r$, for some positive constants c_s and c_r .

4.1. General Norms

Let $(\mathfrak{R}^n, \|\cdot\|)$ be a normed space, and $\|\cdot\|$ be a general norm. Recall that $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$ is the distance function induced by $\|\cdot\|$, and we do not distinguish between $d(\mathbf{x}, \mathbf{y})$ and $\|\mathbf{x} - \mathbf{y}\|$.

4.1.1. GTA for both sorted and random accesses

The basic idea of GTA is as follows: Firstly, we locate a query Q . Secondly, in order to get the candidate tuples of Q , we construct the smallest nonempty n -square $B(Q, r, d_\infty)$ with the Max-Min-Radius r_{maxmin} , by computing the maximum value of all the minimum distances between the coordinates of tuples and/or Q on each of the n axes respectively. Thirdly, for each sorted list L_i , we do sorted access and then do random access to the other lists to determine candidate tuples in $B(Q, r, d_\infty)$. Finally, if $B(Q, r, d_\infty)$ contains at least N candidate tuples with distances less than or equal to the *threshold distance* αr , then we obtain top- N tuples by Theorem 3.5; else we enlarge the $B(Q, r, d_\infty)$ with r_{maxmin} to find more candidate tuples. We provide the algorithm GTA below in a style similar to that in [14].

Algorithm GTA // for General norm $\|\cdot\|$, Threshold Algorithm
Input: (Q, N) // $Q = (q_1, \dots, q_n) \in \mathcal{R}^n$ is a query point; N is a positive integer
Output: Y // a sorted set of N tuples $Y = \langle (t_1, \|t_1 - Q\|), \dots, (t_N, \|t_N - Q\|) \rangle$ with $\|t_1 - Q\| \leq \dots \leq \|t_N - Q\|$

(0) $r = 0$; $r_{min} = 0$; $r_{maxmin} = 0$; // initialization

(1) **For** each $i = 1$ to n //locate Q by B^+ -tree or binary search

Find a_{ij} and a_{ik} in L_i such that $a_{ij} \leq q_i \leq a_{ik}$; //where $k = j + 1$, a_{ij} means L_i 's j th attribute-value

If $q_i \leq \min(A_i)$ then $r_{min} = a_{ik} - q_i$; //that is, $q_i \leq a_{ik}$

Else If $\max(A_i) \leq q_i$ then $r_{min} = q_i - a_{ij}$; //that is, $a_{ij} \leq q_i$

Else $r_{min} = \min(q_i - a_{ij}, a_{ik} - q_i)$;

$r_{maxmin} = \max(r_{maxmin}, r_{min})$;

End For

$r = r + r_{maxmin}$; $r_{maxmin} = 0$;

(2) **For** each $L_i, i=1$ to n

$r_{min} = 0$;

If ($q_i - r \leq a_{ij}$ and j is not exhausted) then do sorted access to find a_{ij} with decreasing j by 1 until
($a_{ij} < q_i - r$ or j is exhausted);

If ($a_{ik} \leq q_i + r$ and k is not exhausted) then do sorted access to find a_{ik} with increasing k by 1 until
($q_i + r < a_{ik}$ or k is exhausted);

As a tuple t is seen under sorted access in list L_i , do random access to the other lists to find the attribute value t_i of tuple t in every list. If $q_i - r \leq t_i \leq q_i + r$ for every i , then compute distance $d(t, Q)$. If $d(t, Q)$ is one of the N smallest distances computed so far, then remember tuple t and its distance $d(t, Q)$ (ties are broken arbitrarily, so that only N tuples and their distances need to be remembered at any time);

$$r_{min} = \max(0, \min((q_i - r) - a_{ij}, a_{ik} - (q_i + r)));$$

$$r_{maxmin} = \max(r_{maxmin}, r_{min});$$

End For

(3) **If** there are at least N seen tuples in n -square $B(Q, r, d_\infty)$ with $d(t, Q) \leq \alpha r$, then halt;

Else $\{r = r + r_{maxmin}; r_{maxmin} = 0; \text{goto (2)}\}$.

(4) Let Y be a set containing the N seen tuples in $B(Q, r, d_\infty)$ with the smallest $d(t, Q)$'s. Output them with their ranked distances.

The following theorem shows the correctness of the above algorithm GTA.

Theorem 4.1. If $(\mathfrak{R}^n, \|\cdot\|)$ is a normed space, then algorithm GTA will correctly find the top- N tuples.

Proof. Let Y be as in Step (4) of GTA and $B(Q, \alpha r, d)$ be the ball at the moment when GTA halts. Then $Y \subset B(Q, \alpha r, d)$. It is sufficient to show that $d(t, Q) \leq d(z, Q)$ for every $t \in Y$ and every $z = (z_1, z_2, \dots, z_n) \notin Y$. By the definition of Y , it is true for each tuple $z \in B(Q, r, d_\infty) = \prod_{i=1}^n [q_i - r, q_i + r]$ that has been seen in running GTA. Now, assume that (1) z has been seen and $z \notin B(Q, r, d_\infty)$ or (2) z was unseen. For unseen z , $z_i \notin [q_i - r, q_i + r]$ for every i , thus $z \notin B(Q, r, d_\infty)$. By the first part in Corollary 3.2 in Section 3.2, $z \notin B(Q, \alpha r, d)$ for the cases (1) and (2). Since $t \in Y \subset B(Q, \alpha r, d)$, by the definition of $B(Q, \alpha r, d)$, we have $d(t, Q) \leq \alpha r < d(z, Q)$. $\square$

The difference between $B(Q, \alpha r, d)$ and Y is that $|B(Q, \alpha r, d)| \geq N$ and $B(Q, \alpha r, d)$ contains all top- N tuples of Q , but $|Y| = N$ and $Y \subset B(Q, \alpha r, d)$ (ties are broken arbitrarily).

Example 2. To illustrate algorithm GTA, consider the relation R and query point Q in $(\mathfrak{R}^2, \|\cdot\|_1)$ as shown in Figure 3. We use GTA to handle the top-3 query $Q = (q_1, q_2) = (7, 6)$ in $\mathfrak{R}^2$ with Manhattan distance $d_1(Q, t) = |q_1 - t_1| + |q_2 - t_2| = |7 - t_1| + |6 - t_2|$ for $t = (t_1, t_2)$ in R .

First, locating $Q = (7, 6)$ in L_1 and L_2 , we have $j_1 = 4$ and $k_1 = 5$ for L_1 ; $j_2 = 6$ and $k_2 = 7$ for L_2 , which are indicated by the four black arrows as shown in Figure 3(b), and then $r = 1$; $\alpha r = 1$ since $\alpha = 1$ from Theorem 3.3.

In Round 1: for $r = 1$ and $\alpha r = 1$, we have seen $\{\mathbf{t1}\} \cup \{\mathbf{t11}, \mathbf{t3}, \mathbf{t4}\} \cup \{\mathbf{t7}, \mathbf{t8}, \mathbf{t1}\} \cup \{\mathbf{t9}\}$ respectively by starting from $\{j_1 = 4; k_1 = 5; j_2 = 6; k_2 = 7;\}$ and moving with $\{j_1--; k_1++; j_2--; k_2++;\}$, and the stop

positions of j_1, k_1, j_2 and k_2 are pointed by the four red arrows in Figure 3(b). In this process, when $\{j_1--\}$ makes $L_1.j = j_1 = 3$, we neither see $L_1.tid = \mathbf{t9}$ nor do random access to it since $L_1.A_1 = 3 < q_1 - r = 6$, as a result, we have seen $\{\mathbf{t1}\}$ with $\{j_1 = 4\}$ and $\{j_1--\}$; the others are similar and thus omitted. From all seen tuples, hence, we have obtained $Y = \{(\mathbf{t1}; 2)\}$ where $(\mathbf{t1}; 2)$ means $d_1(Q, \mathbf{t1}) = 2$ (that is, $\mathbf{t1}$ is in the smallest square as shown in Figure 3(a)). Thus, $count = 0 < N = 3$ since no Manhattan distance in Y is less than or equal to $\alpha r = 1$; that is, there is no tuple in the smallest diamond as shown in Figure 3(a). Then we compute and obtain $r = 3$ and $\alpha r = 3$.

(a) $Q = (q_1, q_2) = (7, 6)$, and $R = \{\mathbf{t1}, \dots, \mathbf{t12}\}$ in $\mathbb{R}^2$. The three blue diamonds and red squares from inner to outer correspond to the three Rounds respectively.

L_1			L_2		
j	tid	A_1	j	tid	A_2
1	t2	1	1	t11	1
2	t12	2	2	t10	3
3	t9	3	3	t12	3
4	t1	6	4	t1	5
5	t11	7	5	t8	5
6	t3	8	6	t7	5
7	t4	8	7	t9	7
8	t8	9	8	t2	9
9	t6	9	9	t3	9
10	t10	10	10	t6	10
11	t7	12	11	t5	11
12	t5	13	12	t4	12

(b) List L_1 for A_1 and List L_2 for A_2

Figure 3. A top- N query Q over a relation $R(tid, A_1, A_2) \subset \mathbb{R}^2$. (a) R has 12 tuples; (b) Sorted Lists L_1 and L_2 .

In Round 2: for $r = 3$ and $\alpha r = 3$, we have seen $\{\} \cup \{\mathbf{t8}, \mathbf{t6}, \mathbf{t10}\} \cup \{\mathbf{t12}, \mathbf{t10}\} \cup \{\mathbf{t2}, \mathbf{t3}\}$ by starting from $\{j_1 = 3; k_1 = 8; j_2 = 3; k_2 = 8\}$ and moving with $\{j_1--; k_1++; j_2--; k_2++\}$ respectively, where $\{\}$

means empty set, and we have obtained $Y = \{(\mathbf{t1}; 2), (\mathbf{t8}; 3), (\mathbf{t3}; 4)\}$, $\mathbf{t10} \notin Y$ due to $d_1(Q, \mathbf{t10}) > d_1(Q, \mathbf{t3})$. In this process, the starting points of j_1, k_1, j_2 and k_2 are their stop positions in Round 1, respectively. Now, the stop positions of j_1, k_1, j_2 and k_2 are pointed by the four blue arrows in Figure 3(b). We have $count = 2 < N = 3$ since there are two Manhattan distances in Y that are not larger than $\alpha r = 3$; that is, the second largest diamond contains two tuples $\mathbf{t1}$ and $\mathbf{t8}$ as shown in Figure 3(a). Then we compute and obtain $r = 4$ and $\alpha r = 4$.

In Round 3: for $r = 4$ and $\alpha r = 4$ we have seen $\{\mathbf{t9}\} \cup \{\} \cup \{\} \cup \{\mathbf{t6}\}$ by starting from $\{j_1 = 3; k_1 = 11; j_2 = 1; k_2 = 10\}$ and moving with $\{j_1--; k_1++; j_2--; k_2++\}$ respectively, and we have obtained $Y = \{(\mathbf{t1}; 2), (\mathbf{t8}; 3), (\mathbf{t3}; 4)\}$, which is the same as that in Round 2. Now, all three Manhattan distances in $Y \leq \alpha r = 4$; $count = 3 = N$; then the process is terminated. The top-3 tuples $\mathbf{t1}$, $\mathbf{t8}$ and $\mathbf{t3}$ are found in the largest diamond. In this Round, we omit the arrows pointing the stop positions in Figure 3(b). $\square$

By Theorem 3.2, for p -norm $\|\cdot\|_p$, we give a variant of GTA with stopping rule “halt when $|B(Q, r, d_\infty)| \geq N$ ” that stops sooner than GTA, as stated in the following Theorem 4.2.

Theorem 4.2. Let GTA compute distance $d_p(\mathbf{t}, Q)$ for all seen tuples and Y be a set containing the N tuples that have been seen with the N smallest $d_p(\mathbf{t}, Q)$'s. Then GTA will correctly find the top- N tuples for any $\|\cdot\|_p$ ($1 \leq p \leq \infty$) if and only if $|B(Q, r, d_\infty)| \geq N$.

Proof. The necessity is clear by the definition of GTA.

To prove sufficiency, let $|B(Q, r, d_\infty)| \geq N$ and $SEEN = \{\mathbf{t} \in \mathbf{R}: \mathbf{t} \in \bigcup_{i=1}^n ([q_i - r, q_i + r] \times \prod_{j \neq i} \mathfrak{R}_j)\}$ (i.e., all tuples have been seen; for a 2-dimensional example, see all “•” in the gray area in Figure 4) where $\mathfrak{R}_j = \mathfrak{R} = (-\infty, \infty)$ for every j , when GTA returns. By the definition of $B(Q, r, d)$ in Section 3.2 and p -monotonicity in Theorem 3.2 in Section 3.1, we have $B(Q, r, d_\infty) \subset B(Q, n^{(1/p)}r, d_p) \subset B(Q, nr, d_1)$; thus, it is sufficient to show that

$$B(Q, nr, d_1) \subset \bigcup_{i=1}^n ([q_i - r, q_i + r] \times \prod_{j \neq i} \mathfrak{R}_j).$$

In fact, for every $\mathbf{t} = (t_1, \dots, t_n) \in B(Q, nr, d_1)$, we have $d_1(Q, \mathbf{t}) \leq nr$, i.e., $\sum_i |q_i - t_i| \leq nr$; hence, there is $1 \leq k \leq n$ such that $|q_k - t_k| \leq r$. Thus, $t_k \in [q_k - r, q_k + r]$, i.e., $\mathbf{t} \in \bigcup_{i=1}^n ([q_i - r, q_i + r] \times \prod_{j \neq i} \mathcal{R}_j)$. $\square$

Figure 4 illustrates the sufficiency of Theorem 4.2 for $(\mathcal{R}^2, \|\cdot\|_p)$. When the square contains N tuples, all the tuples in the gray area (i.e., the set $SEEN$) have been seen by GTA. Additionally, using the sufficiency of Theorem 4.2 to above Example 2, Round 1 can obtain top-1 tuple, and Round 2 can obtain top-3 and top-4 tuples for distance d_1 . Thus, this variant stops sooner than GTA.

Figure 4. GTA for $(\mathcal{R}^2, \|\cdot\|_p)$; $SEEN = \{\text{all “}\bullet\text{” in the gray area}\}$

4.1.2. GTAz for restricting sorted access

We discuss a situation where it is not possible to access certain parts of the lists under sorted access. In this situation, top- N processing techniques assume the availability of at least one sorted access source.

Let Z be a nonempty proper subset of $\{1, 2, \dots, n\}$, i.e., the set of indices i of those lists L_i that can be accessed under sorted access. Then at least one of the lists can be accessed under sorted access. Assume that $Z = \{i_1, i_2, \dots, i_m\}$, and $1 \leq m = |Z| < n$. Without loss of generality, we assume that $Z = \{1, 2, \dots, m\}$, $1 \leq m < n$. Define GTAz to be the following natural modification of GTA, which handles the restriction on sorted access.

Let the number n of the For-loop “For each $(L_i, i=1 \text{ to } n)$ ” in the steps (1) and (2) of algorithm GTA be replaced by the number m , then we obtain GTAz. Obviously, GTAz will be GTA if $m = n$.

Theorem 4.3. If $(\mathcal{R}^n, \|\cdot\|)$ is a normed space, then algorithm GTAz will correctly find the top- N tuples.

Proof. It follows the proof of Theorem 4.1 by a small change. $\square$

4.1.3. GNRA for no random access

Now, we discuss the situations where random accesses are impossible, and modify GTA to obtain our algorithm GNRA that does no random accesses. In contrast to NRA [14] that finds the top- N tuples without their grades, GNRA can get the top- N tuples with their distances to query Q . We present GNRA as follows.

Algorithm **GNRA** // for General norm $\|\cdot\|$, the algorithm for No Random Access

Input: (Q, N) // $Q = (q_1, \dots, q_n) \in \mathcal{R}^n$ is a query point; N is a positive integer

Output: Y // a sorted set of N tuples $Y = \langle (t_1, \|t_1 - Q\|), \dots, (t_N, \|t_N - Q\|) \rangle$ with $\|t_1 - Q\| \leq \dots \leq \|t_N - Q\|$

(0) $r = 0$; $r_{min} = 0$; $r_{maxmin} = 0$; //initialization

(1) **For** each $i = 1$ to n //locate Q by B^+ -tree or binary search

Find a_{ij} and a_{ik} in L_i such that $a_{ij} \leq q_i \leq a_{ik}$; //where $k = j + 1$, a_{ij} means L_i 's j th attribute-value

If $q_i \leq \min(A_i)$ then $r_{min} = a_{ik} - q_i$; //that is, $q_i \leq a_{ik}$

Else If $\max(A_i) \leq q_i$ then $r_{min} = q_i - a_{ij}$; //that is, $a_{ij} \leq q_i$

Else $r_{min} = \min(q_i - a_{ij}, a_{ik} - q_i)$;

$r_{maxmin} = \max(r_{maxmin}, r_{min})$;

End For

$r = r + r_{maxmin}$; $r_{maxmin} = 0$;

(2) **For** each L_i , $i=1$ to n

$r_{min} = 0$;

If $(q_i - r \leq a_{ij}$ and j is not exhausted) then do sorted access to find a_{ij} with decreasing j by 1 until $(a_{ij} < q_i - r$ or j is exhausted);

If $(a_{ik} \leq q_i + r$ and k is not exhausted) then do sorted access to find a_{ik} with increasing k by 1 until $(q_i + r < a_{ik}$ or k is exhausted);

If all attribute values of tuple $t = (t_1, t_2, \dots, t_n)$ have been seen, then compute distance $d(t, Q)$. If this $d(t, Q)$ is one of the N smallest distances computed so far, then remember tuple t and the distance $d(t, Q)$ (ties are broken arbitrarily);

$r_{min} = \max(0, \min((q_i - r) - a_{ij}, a_{ik} - (q_i + r)))$;

$r_{maxmin} = \max(r_{maxmin}, r_{min})$;

End For

(3) **If** there are at least N tuples with all seen attribute values and $d(t, Q) \leq \alpha r$, then halt;

Else $\{r = r + r_{maxmin}$; $r_{maxmin} = 0$; goto (2) $\}$.

(4) Let Y be a set containing the N tuples with all seen attribute values and the smallest $d(t, Q)$'s. Output them with their ranked distances.

Theorem 4.4. If $(\mathfrak{R}^n, \|\cdot\|)$ is a normed space, then algorithm GNRA will correctly find the top- N tuples.

Proof. Let Y be as in Step (4) of GNRA and $B(Q, \alpha r, d)$ be the ball at the moment when GNRA halts.

Then $Y \subset B(Q, \alpha r, d)$. We need to show that $d(t, Q) \leq d(z, Q)$ for every $t \in Y$ and every $z \notin Y$.

There are three cases for $z \notin Y$. (1) All attribute values of z have been seen in running GNRA. By the definition of Y , $d(t, Q) \leq d(z, Q)$. (2) Some attribute values of z have been seen, but the others of z have not been seen in running GNRA. Assume that $z = (z_1, z_2, \dots, z_n)$ and z_k ($1 \leq k \leq n$) has not been seen in running GNRA. Then $z_k \notin [q_k - r, q_k + r]$, and $d_\infty(z, Q) = \max_{1 \leq i \leq n} (|z_i - q_i|) \geq |z_k - q_k| > r$. (3) None of the attribute values of z has been seen in running GNRA. Then, $z_i \notin [q_i - r, q_i + r]$ for every i , thus $d_\infty(z, Q) = \max_{1 \leq i \leq n} (|z_i - q_i|) > r$. For cases (2) and (3), we have $d_\infty(z, Q) > r$, and then $\alpha d_\infty(z, Q) > \alpha r$. By (3.1) in Theorem 3.3 in Section 3.1, $d(z, Q) \geq \alpha d_\infty(z, Q)$. Since $t \in Y \subset B(Q, \alpha r, d)$, by the definition of $B(Q, \alpha r, d)$, $d(t, Q) \leq \alpha r$. Hence, $d(t, Q) \leq \alpha r < \alpha d_\infty(z, Q) \leq d(z, Q)$. $\square$

4.2. x -Monotonic Norms

Let $(\mathfrak{R}^n, \|\cdot\|)$ be a normed space with an x -monotonic norm $\|\cdot\|$. Now, we present the algorithms mTA, mTAz and mNRA.

4.2.1. mTA for both sorted and random accesses

Finding the smaller one of the two distances between the coordinates on each of the n axes respectively, mTA determines the direction (up or down) of doing sorted access to each of the lists. Thus, for each sorted list, mTA sees one tuple per round/loop, then mTA defines a *threshold point* $p = (p_1, p_2, \dots, p_n)$ in $\mathfrak{R}^n$, and computes the *threshold distance* $\tau = d(p, Q)$ to find top- N tuples. The algorithm mTA is given below.

Algorithm mTA // for x -monotonic norm $\|\cdot\|$, Threshold Algorithm
Input: (Q, N) // $Q = (q_1, \dots, q_n) \in \mathfrak{R}^n$ is a query point; N is a positive integer
Output: Y // a sorted set of N tuples $Y = \langle (t_1, \|t_1 - Q\|), \dots, (t_N, \|t_N - Q\|) \rangle$ with $\|t_1 - Q\| \leq \dots \leq \|t_N - Q\|$

- (1) **For** each $i = 1$ to n // locate Q by B^+ -tree or binary search
 Find a_{ij} and a_{ik} in L_i such that $a_{ij} \leq q_i \leq a_{ik}$; //where $k = j + 1$
 If $q_i \leq \min(A_i)$ then $a_{ik} = \min(A_i)$; //we have only $q_i \leq a_{ik}$
 If $\max(A_i) \leq q_i$ then $a_{ij} = \max(A_i)$; //we have only $a_{ij} \leq q_i$
 End for
- (2) **For** each $L_i, i=1$ to n
 If $|q_i - a_{ij}| \leq |a_{ik} - q_i|$ {do sorted access to find $a_{ij}; p_i = a_{ij}; j = j + 1$;}
 Else {do sorted access to find $a_{ik}; p_i = a_{ik}; k = k + 1$;}
 As a tuple t is seen under sorted access in list L_i , do random access to the other lists to find the attribute value t_i of tuple t in every list. Then compute distance $d(t, Q)$ between t and Q . If this distance is one of the N smallest we have seen, then remember tuple t and its distance $d(t, Q)$ (ties are broken arbitrarily, so that only N tuples and their distances need to be remembered at any time);
 End For
- (3) Define the *threshold point* $p = (p_1, p_2, \dots, p_n)$, compute the *threshold distance* $\tau = d(p, Q)$;
 If there are at least N tuples that have been seen with $d(t, Q) \leq \tau$, then halt;
 Else goto (2).
- (4) Let Y be a set containing the N tuples that have been seen with the smallest $d(t, Q)$'s. Output them with their ranked distances.
-

4.2.2. mTAz for restricting sorted access

Without loss of generality, let $Z = \{1, 2, \dots, m\}$, $1 \leq m < n$ (see Section 4.1.2). We define mTAz to be the following natural modification of mTA.

Let the number n of the For-loop “For each (L_i ,) $i=1$ to n ” in the steps (1) and (2) of algorithm mTA be replaced by the number m , and let the step (3) of mTA be replaced by

- (3) Define the *threshold point* $p = (p_1, \dots, p_m, q_{m+1}, \dots, q_n)$, compute the *threshold distance* $\tau = d(p, Q)$. If there are at least N tuples that have been seen with $d(t, Q) \leq \tau$, then halt. Else goto (2). /* Note: $Q = (q_1, \dots, q_m, q_{m+1}, \dots, q_n)$, the last $(n - m)$ coordinates $q_{m+1}, \dots, q_n$ of the threshold point p are the same as those of the query point Q */

Then we obtain mTAz, which will become mTA if $m = n$.

4.2.3. mNRA for no random access

The algorithm mNRA uses the same strategy of doing sorted accesses as that of mTA (i.e., sees one tuple per round for each sorted list), and maintains two threshold points p and p' to compute respectively a lower bound and an upper bound of distances as the threshold values for determining candidate tuples.

The algorithm mNRA is shown below:

Algorithm **mNRA** // for x -monotonic norm $\|\cdot\|$, the algorithm for No Random Access
Input: (Q, N) // $Q = (q_1, \dots, q_n) \in \mathcal{R}^n$ is a query point; N is a positive integer
Output: Y // a set of N tuples $Y = \langle t_1, \dots, t_N \rangle$

- (1) **For** each $i = 1$ to n // Let $pmax_i$ be the Tail or Head of L_i
 - If** $|q_i - a_{i1}| \leq |a_{i|R} - q_i|$, $pmax_i = a_{i|R}$; **Else** $pmax_i = a_{i1}$.
 - End For** // a_{i1} is the Head, and $a_{i|R}$ is the Tail of L_i
- (2) **For** each $i = 1$ to n // locate Q by B^+ -tree or binary search
 - Find a_{ij} and a_{ik} in L_i such that $a_{ij} \leq q_i \leq a_{ik}$; // where $k = j + 1$
 - If** $q_i \leq \min(A_i)$ then $a_{ik} = \min(A_i)$; // we have only $q_i \leq a_{ik}$
 - If** $\max(A_i) \leq q_i$ then $a_{ij} = \max(A_i)$; // we have only $a_{ij} \leq q_i$
 - End for**
- (3) **For** each L_i , $i=1$ to n
 - If** $|q_i - a_{ij}| \leq |a_{ik} - q_i|$ { do sorted access to find a_{ij} ; $pmin_i = a_{ij}$; $j = j - 1$; }
 - Else** { do sorted access to find a_{ik} ; $pmin_i = a_{ik}$; $k = k + 1$; }
 - End for**
- (4) **For** every tuple $t = (t_1, \dots, t_n)$ with some unknown attributes, compute an upper bound for $d(t, Q)$, denoted $Dupp(t, Q)$, by substituting each unknown attribute value t_i with $pmax_i$, and compute a lower bound for $d(t, Q)$, denoted $Dlow(t, Q)$, by substituting each unknown attribute value t_i with $pmin_i$. If all attribute values of tuple $t = (t_1, \dots, t_n)$ have been seen, then $d(t, Q) = Dupp(t, Q) = Dlow(t, Q)$. If no attribute value of tuple t has been seen, then $Dlow(t, Q) = d(p, Q)$, and $Dupp(t, Q) = d(p', Q)$, where $p = (pmin_1, \dots, pmin_n)$ and $p' = (pmax_1, \dots, pmax_n)$.
- (5) Let Y be the set of N tuples with the smallest $Dupp(t, Q)$ values seen so far. If two tuples have the same upper bound, then ties are broken using their lower bound $Dlow(t, Q)$ such that the tuple with the smallest $Dlow$ value wins, and arbitrarily among tuples that additionally tie in lower bound. Let $Dupp_{max}$ be the maximal value of all $Dupp(t, Q)$ in Y (i.e., $Dupp(t, Q) \leq Dupp_{max}$ for every t in Y).

- (6) Call a tuple t viable if $Dlow(t) < Dupp_{max}$. Halt when (a) at least N distinct tuples have been seen, and (b) there are no viable tuples outside Y (i.e., if lower bound $Dlow(t) \geq Dupp_{max}$ for all $t \notin Y$), return Y (with or without their distances). Else goto step (3).
-

One of the main differences between our G-algorithms (GTA, GTAz, and GNRA) and our m-algorithms (mTA, mTAz, and mNRA) is that m-algorithms do sorted access to each of the lists “in lockstep” as the algorithms TA, TAz, and NRA [14] by a rate of movement of one tuple per round, whereas G-algorithms “in batches with variable sizes, not a constant size” as shown in Example 2. For n lists, each m-algorithm sees at most n distinct tuples per round; nevertheless, G-algorithms may see at most $\lceil R \rceil$ distinct tuples per round. The following example demonstrates the movements of mTA.

L_1			L_2																														
	j	<table><tr><th>tid</th><th>A_1</th></tr><tr><td>t9</td><td>3</td></tr><tr><td>t1</td><td>6</td></tr><tr><td>t11</td><td>7</td></tr><tr><td>t3</td><td>8</td></tr><tr><td>t4</td><td>8</td></tr><tr><td>t8</td><td>9</td></tr></table>	tid	A_1	t9	3	t1	6	t11	7	t3	8	t4	8	t8	9		j	<table><tr><th>tid</th><th>A_2</th></tr><tr><td>t1</td><td>3</td></tr><tr><td>t1</td><td>5</td></tr><tr><td>t8</td><td>5</td></tr><tr><td>t7</td><td>5</td></tr><tr><td>t9</td><td>7</td></tr><tr><td>t2</td><td>9</td></tr></table>	tid	A_2	t1	3	t1	5	t8	5	t7	5	t9	7	t2	9
tid	A_1																																
t9	3																																
t1	6																																
t11	7																																
t3	8																																
t4	8																																
t8	9																																
tid	A_2																																
t1	3																																
t1	5																																
t8	5																																
t7	5																																
t9	7																																
t2	9																																
	3		⑤	3																													
②	4		③	4																													
①	5		②	5																													
③	6		①	6																													
④	7		④	7																													
⑤	8			8																													

Figure 5. Parts of the Lists in Figure 3(b), $j = 3, \dots, 8$.

Example 3. To illustrate our algorithm mTA, consider the same situation as in Example 2, we use mTA to evaluate the top-3 query $Q = (7, 6)$. As shown in Figure 5, the numbers with ①, ②, ③, ④ and ⑤ indicate the movement orders of the pointers (j_1, k_1, j_2 , and k_2) with decreasing j_1 by 1 (j_1--), increasing k_1 by 1 (k_1++), decreasing j_2 by 1 (j_2--), and increasing k_2 by 1 (k_2++), where j_1, k_1, j_2 , and k_2 are the same as in Example 2. The algorithm mTA does sorted access to each of the n sorted lists in lockstep. That is, for each list L_i , mTA first determines the direction of access, next sees only one tuple in L_i , and does the random access(es) and internal computations for the seen tuple; then mTA performs for list L_{i+1} until $i = n$. To deal with the top-3 query Q , mTA needs five rounds corresponding to the numbers ①, ..., ⑤. Comparing with Example 2, the number of sorted accesses in running mTA is at most that in running

GTA, and the execution cost for mTA is always no more than that of GTA in the case of $\mathbf{x}$ -monotonic norms. $\square$

In order to show that mTA, mTAz, and mNRA are correct, we will convert them into the algorithms TA, TAz, and NRA [14] respectively, as the following Theorem 4.5 shows.

Theorem 4.5. If $(\mathfrak{R}^n, \|\cdot\|)$ is a normed space and $\|\cdot\|$ is $\mathbf{x}$ -monotonic, then each of algorithms mTA, mTAz and mNRA will correctly find the top- N tuples.

Proof. Given a query $Q = (q_1, \dots, q_n) \in \mathfrak{R}^n$, we define $s_i = |q_i - t_i|$ for every $\mathbf{t} = (t_1, \dots, t_n) \in \mathbf{R} \subset \mathfrak{R}^n$, and call s_i the score of $\mathbf{t}$ with respect to Q for the i th attribute A_i , $1 \leq i \leq n$. Let $\mathbf{s} = (s_1, \dots, s_n)$. By $\mathbf{x}$ -monotonicity of $\|\cdot\|$, aggregation function $f(\mathbf{s}) = \|\mathbf{s}\|$ is F -monotone. Thus, the top- N query Q becomes an equivalent query that finds N tuples $\langle \mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_N \rangle$ with the smallest values of $f(\mathbf{s})$. For each i , we define a new sorted list S_i by the s_i value in increasing order. Let $Q_0 = Q - Q = (0, \dots, 0)$, then Q_0 is Min-point. Thus, our algorithms mTA, mTAz and mNRA are converted into the TA, TAz and NRA [14] respectively with Q_0 , $\{S_i: i = 1, \dots, n\}$ and the aggregation function $f(\mathbf{s})$; hence, they are correct by correctness of TA, TAz and NRA, respectively. $\square$

For an $\mathbf{x}$ -monotonic function $f(\cdot)$ described in Section 3.1, given a query $Q = (q_1, \dots, q_n) \in \mathfrak{R}^n$, our m-algorithms mTA, mTAz and mNRA hold true. When we define a scoring function $s(Q, \mathbf{t}) = f(Q - \mathbf{t})$ for a tuple $\mathbf{t} \in \mathfrak{R}^n$, from the proof of Theorem 4.5, mTA, mTAz and mNRA will become TA, TAz and NTA [14], respectively, for Min-point $Q_0 = Q - Q = (0, \dots, 0)$ and the scoring function $s(Q, \mathbf{t})$.

Due to the cost of obtaining n sorted lists S_i ($i = 1, \dots, n$), however, our m-algorithms mTA, mTAz and mNRA cannot be substituted by the approach in the proof of Theorem 4.5 with the algorithms TA, TAz and NTA [14], respectively. For example, the cost of the approach in the proof of Theorem 4.5 with TA will be larger than that of the naïve algorithm [14] for $n \geq 2$ and $|\mathbf{R}| \geq 8$. Thus, the approach in the proof of Theorem 4.5 is only employed to prove the correctness of our algorithms mTA, mTAz and mNRA, but it is not a viable method to handle top- N queries.

5. BUFFER SIZE AND OPTIMALITY

In this section, we discuss the buffer size and optimality of our algorithms, and compare the time costs of TA-like algorithms with that of the algorithms of nearest neighbor search.

5.1. Buffer Size

From [14], TA and TAz require bounded buffers, but NRA may require unbounded buffers. Similarly, we have Theorem 5.1.

Theorem 5.1. Algorithms GTA, GTAz, mTA, and mTAz require only bounded buffers; the buffer size for each of them is independent of the size of the database.

Proof. By the definitions of the four algorithms, in addition to a little bit of bookkeeping, all that the algorithms must contain are the current top- N tuples and their distances, and pointers to the one or two last tuples seen in doing sorted access to each list. $\square$

Unlike the algorithms in Theorem 5.1, which need small constant-size buffer, GNRA and mNRA may require unbounded buffers for a lot of bookkeeping at each step to maintain the update for every tuple seen so far. In fact, the buffer size for GNRA or mNRA may be $O(n|\mathbf{R}|)$ in the worst case.

5.2. Optimality

Fagin et al. [14] defined “instance optimality”, and showed that their algorithms TA, TAz and NRA are instance optimal. Instance optimality corresponds to optimality in every instance, as opposed to just the worst case or the average case. As an example, binary search is guaranteed to require no more than $\log M$ probes for a sorted list with M data items in the worst case, but it is not instance optimal [14]. Obviously, our algorithms are not instance optimal, since they use a B⁺-tree or binary search to locate query point in each sorted list. By the definition of the *execution cost* of an algorithm in Section 4, usually, the cost of binary search is smaller than the execution cost of a top- N algorithm. To locate a real number q in a sorted list with 2^{30} items, say, binary search requires no more than $30 = \log(2^{30})$ probes; however, to find the top-100 items of the q (i.e., 100 nearest items) needs at least 100 sorted accesses after q is located. If we consider only the execution costs of our algorithms and disregard their location costs (defined in Section 4), are they “instance optimal”? To answer this question, we introduce the notion of “pseudo instance

optimal” slightly adapted from “instance optimal” in [14]. Note that the instance optimality in [14] involves only execution costs (i.e., the sum of the sorted access cost and the random access cost, see Section 4) and ignores other costs (say, the costs of computing and comparing the values of a ranking function).

Definition 5.1 (*locationally equivalent*). Let $\mathbf{D}$ be a class of relations ($\subseteq \mathfrak{R}^n$). Two algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ are said to be locationally equivalent if they incur almost the same cost for locating any query point $Q \in \mathfrak{R}^n$ on every relation $D \in \mathbf{D}$, i.e., if there is a constant $\delta > 0$ such that $|cost(\mathcal{A}_1 \text{ locates } Q \text{ on } D) - cost(\mathcal{A}_2 \text{ locates } Q \text{ on } D)| < \delta$ for every $Q \in \mathfrak{R}^n$, every $\mathcal{A}_1, \mathcal{A}_2 \in \mathbf{A}$ and every $D \in \mathbf{D}$.

Definition 5.2 (*pseudo instance optimality*). Let $\mathbf{D}$ be a class of relations ($\subseteq \mathfrak{R}^n$), and let $\mathbf{A}$ be a class of locationally equivalent algorithms. An algorithm $\mathcal{B} \in \mathbf{A}$ is *pseudo instance optimal* over $\mathbf{A}$ and $\mathbf{D}$ if there are constants c and c' such that for every $\mathcal{A} \in \mathbf{A}$ and every $D \in \mathbf{D}$ we have $cost(\mathcal{B}, D) \leq c \cdot cost(\mathcal{A}, D) + c'$, where $cost(a, D)$ is the execution cost incurred by algorithm a over D . The constant c is called the *optimality ratio*.

Theorem 5.2. Let $(\mathfrak{R}^n, \|\cdot\|)$ be a normed space and $\|\cdot\|$ be $\mathbf{x}$ -monotonic. Let $\mathbf{D}$ be the class of all relations, and $\mathbf{A}$ be the class of the top- N query processing algorithms that are locationally equivalent to mTA and that do not make wild guesses. Then mTA, mTAz and mNRA are pseudo instance optimal over $\mathbf{D}$ and $\mathbf{A}$.

Proof. From Definition 5.1, $|cost(\mathcal{A} \text{ locates } Q \text{ on } \mathbf{R}) - cost(\text{mTA locates } Q \text{ on } \mathbf{R})| < \delta$ for any algorithm $\mathcal{A}$ in $\mathbf{A}$, any query point $Q = (q_1, \dots, q_n)$ in $\mathfrak{R}^n$ and any relation $\mathbf{R}$ in $\mathbf{D}$. Assume that $\mathcal{A}$ sees λ distinct tuples running over the relation $\mathbf{R}$ and makes μ sorted accesses and ν random accesses when $\mathcal{A}$ halts. Since $\mathcal{A}$ makes no wild guesses, we have $\mu \geq \lambda$ and the execution cost of $\mathcal{A}$ is $cost(\mathcal{A}) = \mu \cdot c_s + \nu \cdot c_r$, where positive constants c_s and c_r are the cost of a sorted access and the cost of a random access for each of the n sorted lists $L_1, \dots, L_n$. Let the algorithm mTA do s sorted accesses and r random accesses to find the top- N

tuples and see m distinct tuples. Then $s \geq m$ and the execution cost of mTA is $cost(mTA) = s \cdot c_s + r \cdot c_r$ for the n sorted lists $L_1, \dots, L_n$.

We define $s_i = |q_i - t_i|$ for every $\mathbf{t} = (t_1, \dots, t_n) \in \mathbf{R}$, and call s_i the score of $\mathbf{t}$ in terms of Q for the i th attribute A_i , $1 \leq i \leq n$. Let $\mathbf{s} = (s_1, \dots, s_n)$. By $\mathbf{x}$ -monotonicity of $\|\cdot\|$, the aggregation function $f(\mathbf{s}) = \|\mathbf{s}\|$ is F -monotone. Thus, the top- N query Q becomes an equivalent query that finds top N tuples with the smallest values of $f(\mathbf{s})$. For each i , we define a new sorted list Γ_i by the s_i value in increasing order. Let $Q_0 = Q - Q = (0, \dots, 0)$. Then Q_0 is the Min-point. Thus, our mTA is reduced to the algorithm TA [14] with $Q_0, \{\Gamma_i: i = 1, \dots, n\}$ and the aggregation function $f(\mathbf{s})$. Therefore, the TA does the same numbers of sorted accesses and random accesses and sees the same m distinct tuples as the mTA does. Thus, the execution cost of TA is $cost(TA) = s \cdot c'_s + r \cdot c'_r$, where positive constants c'_s and c'_r are the cost of a sorted access and the cost of a random access for each of the n sorted lists $\Gamma_i, i = 1, \dots, n$. Assume that $\mathcal{A}'$ is the reduction of the algorithm $\mathcal{A}$ for $Q_0, \{\Gamma_i: i = 1, \dots, n\}$ and the aggregation function $f(\mathbf{s})$. Then $\mathcal{A}'$ sees the same λ distinct tuples and makes the same μ sorted accesses and ν random accesses as the algorithm $\mathcal{A}$ does. Therefore, the execution cost of $\mathcal{A}'$ is $cost(\mathcal{A}') = \mu \cdot c'_s + \nu \cdot c'_r$. By the instance optimality of TA in [14], we obtain $cost(TA) \leq \omega cost(\mathcal{A}') + \omega_0$, where the constants ω and ω_0 satisfy $\omega \leq n + n(n-1)c'_r/c'_s$, $\omega_0 \leq nNc'_s + n(n-1)Nc'_r$ and ω is the optimality ratio of TA. Thus, $s \cdot c'_s + r \cdot c'_r \leq \omega(\mu \cdot c'_s + \nu \cdot c'_r) + \omega_0$. By replacing c'_s and c'_r by c_s and c_r respectively, we obtain $cost(mTA) \leq \omega cost(\mathcal{A}) + \omega_0$, where ω and ω_0 satisfy $\omega \leq n + n(n-1)c_r/c_s$ and $\omega_0 \leq nNc_s + n(n-1)Nc_r$, that is, mTA is pseudo instance optimal over $\mathbf{D}$ and $\mathbf{A}$. Similarly, we can prove the pseudo instance optimality of mTAz and mNRA by the instance optimality of TAz and NRA [14], respectively. $\square$

Let $cost_{sum}(\mathcal{A})$ denote the sum of the location cost and execution cost of an algorithm $\mathcal{A}$. Form the proof of Theorem 5.2, we have

$$\begin{aligned} cost_{sum}(mTA) &= cost(mTA \text{ locates } Q \text{ on } \mathbf{R}) + cost(mTA) \\ &\leq cost(\mathcal{A} \text{ locates } Q \text{ on } \mathbf{R}) + \delta + \omega cost(\mathcal{A}) + \omega_0 \end{aligned}$$

$$\leq \theta \text{cost}_{\text{sum}}(\mathcal{A}) + \tau, \text{ where } \theta = \max(1, \omega) \text{ and } \tau = \delta + \omega_0.$$

We define a new algorithm mTA' that is a variation of mTA. The mTA' uses the sequential scan to locate Q in the step (1) of mTA, and finds the top- N tuples of Q by using the other steps (2), (3) and (4) of mTA, then the location cost of mTA' is $O(n|\mathbf{R}|)$ for n sorted lists $L_1, \dots, L_n$. For an algorithm $\mathcal{A} \in \mathbf{A}$, the class in Theorem 5.2, the location cost of $\mathcal{A}$ is $O(n \log |\mathbf{R}|)$. Thus, the above inequation, $\text{cost}_{\text{sum}}(\text{mTA}') \leq \theta \text{cost}_{\text{sum}}(\mathcal{A}) + \tau$, may not be true in general; for the sake of this fact we introduce “pseudo instance optimal” involving the location cost of a query.

From Theorem 5.2, our algorithms mTA, mTAz and mNRA are instance optimal if we ignore their location costs. Note that the instance optimality involves only execution costs, thus, it is similar to the situation of the algorithm NRA in [14] that our mNRA may need a lot of other costs (e.g., internal computation cost) for processing a top- N query (see our experimental results in Section 6).

Figure 6. GTA, GTAz and GNRA are not pseudo instance optimal over a relation.

Obviously, our algorithms GTA, GTAz and GNRA are not pseudo instance optimal, since they do sorted access to each of the sorted lists in batches with variable sizes related to the step-length $r_{\max}$, and there may be many (even all) entries in each closed interval constructed by the step-length. With both sorted and random accesses, for example, Figure 6(a) illustrates a relation $\mathbf{R} \subset \mathbb{R}^2$, $\mathbf{R} = \{\mathbf{t1}(1, 1), \mathbf{t2}(3, 3)\} \cup \{\mathbf{t}(t_1, t_2): 1 < t_1 < 3, t_2 < 1 \text{ or } t_2 > 3\}$, for top-2 query $Q = (2, 2)$ with $\|\cdot\|_\infty$, GTA does $|\mathbf{R}|$ sorted accesses to L_1 ; however, mTAz or GTAz will do 2 sorted accesses to L_2 when $Z = \{2\}$ and 2 random accesses to L_1 . That is, $\text{cost}(\text{GTA}) = O(|\mathbf{R}|)$, but $\text{cost}(\text{mTAz}) = \text{cost}(\text{GTaz}) = O(1)$ with $Z = \{2\}$. For GTAz and GNRA, Figure 6(b) depicts another state of $\mathbf{R}$ in which there is no tuple with $1 < t_1 < 3$ or $1 < t_2 < 3$ except for $\mathbf{t3}$ and

$\mathbf{t}_4$, and there are many (say, $|\mathbf{R}|/5$) tuples with $t_1 = 3$. For top-2 query $Q = (2, 2)$ with $\|\cdot\|_\infty$ as shown in Figure 6(b), GTAz with $Z = \{1\}$ or GNRA does at least $|\mathbf{R}|/5$ sorted accesses to L_1 to see those tuples with $t_1 = 3$, while mTAz with $Z = \{1\}$ will do 3 sorted access to L_1 and 3 random accesses to L_2 , and mNRA will do at most 4 sorted access to both L_1 and L_2 . That is, $\text{cost}(\text{GTaz with } Z = \{1\}) = \text{cost}(\text{GNRA}) = O(|\mathbf{R}|)$, while $\text{cost}(\text{mTAz with } Z = \{1\}) = \text{cost}(\text{mNRA}) = O(1)$.

5.3. Comparison with Optimal Algorithms of NNS

As described in Sections 1 and 2, Nearest Neighbor Search (NNS) has been solved optimally in low dimensions (usually less than 10) [35, 36]. Considering points of a dataset $\mathbf{R} \subset \mathbb{R}^2$, the nearest neighbor can be found with $O(\log|\mathbf{R}|)$ query time and $O(|\mathbf{R}|)$ storage space [17, 36], for example, by using persistent search trees [32, 36]. The models of NNS (or KNN query) in metric spaces require no restriction other than the object distance function, and the methods with sophisticated data structures (e.g., various complex trees) are fixed to distance functions that are available to the system in advance [34].

Based on sorted and/or random accesses, the algorithms with no wild guesses in the TA family are employed to process top- N queries in many scenarios from the database perspective. For instance, [6] discussed an application in Web-accessible databases where a user wants to search information about restaurants; the Zagat-Review website gives the *Rating* attribute of restaurants (sorted access and random access), the NYT-Review website provides the *Price* attribute (random access), and the *Address* attribute is handled by the MapQuest website (random access). Our algorithms in this paper are the new members in the TA family; therefore, it is sufficient to compare the cost of TA [14] with that of optimal algorithms of NNS.

Obviously, the storage space of TA is also $O(|\mathbf{R}|)$ on a 2-dimensional plane; however, the time cost of TA cannot be guaranteed to be $O(\log|\mathbf{R}|)$ because of no wild guesses. As an example shown in Figure 7, a relation $\mathbf{R}$ with two attributes A_1 and A_2 has $2m+1$ tuples, and there are two lists L_1 and L_2 which are sorted in descending order by A_1 -value and A_2 -value, respectively. For the tuple with $\text{tid} = t_{m+1}$, its A_1 value and A_2 value are both $m+1.5$; for a tuple with $\text{tid} = t_k$ ($k \neq m+1$), its A_1 value is $(2m+2) - k$ in L_1 and

its A_2 value is k in L_2 . The top-1 answer is the tuple with $t_{id} = t_{m+1}$ by TA with the ranking function $f(t) = t[A_1] + t[A_2]$. In order to obtain t_{m+1} , the number of sorted accesses to L_1 and L_2 is $2(m+1)$, so is the number of random accesses; that is, the execution cost of TA is $s \cdot c_s + r \cdot c_r = 2(m+1)c_s + 2(m+1)c_r$ for some positive constants c_s and c_r as defined in Section 4. Thus, the execution cost of TA is $O(|R|)$.

L_1		L_2	
t_{id}	A_1	t_{id}	A_2
t_1	$2m+1$	t_{2m+1}	$2m+1$
t_2	$2m$	t_{2m}	$2m$
...	...	...	...
t_m	$m+2$	t_{m+2}	$m+2$
t_{m+1}	$m+1.5$	t_{m+1}	$m+1.5$
t_{m+2}	m	t_m	m
...	...	...	...
t_{2m}	2	t_2	2
t_{2m+1}	1	t_1	1

Figure 7. A relation with two attributes A_1 and A_2 .

In fact, TA is instance optimal [14], and so it is very powerful generally. As an example, for a tuple t_0 in a relation $R(A_1, A_2) \subset \mathcal{R}^2$, if $t_0[A_1]$ is the largest in all values of A_1 and $t_0[A_2]$ is the second largest in the A_2 values, then the top-1 tuple (may or may not be t_0) can be found with $O(1)$ query time by TA. For a top- N query, moreover, an algorithm of the TA family will return N tuples exactly (ties are broken arbitrarily); this way may cause non-deterministic result. For processing deterministic NNS or KNN query as defined in [34], however, the algorithms will return all tuples that have the same distance as the first or K th ranked tuples; obviously, the result set is unique and deterministic, but there may be more than 1 or K tuples returned.

In $\mathcal{R}^2$, similarly, our algorithms in this paper can find the top-1 tuple with $O(1)$ time cost in many scenarios; however, they may use $O(|R|)$ query time in the worst case.

From the viewpoint of database, hence, the algorithms of the TA family with no wild guesses are different from that of NNS, and so are their time costs.

6. EXPERIMENTAL RESULTS

In this section, we report our experimental results. In Section 6.1, we describe the datasets and measures used in our experiments. In Section 6.2, we report the elapsed time needed to obtain the top- N tuples from the respective dataset for a query. In Section 6.3, we report the number of sorted accesses to the sorted lists by running our algorithms with different datasets and distance functions. In Section 6.4, we study the effect of different result size N on the performance of our approach. Finally, in Section 6.5, we compare our algorithms GTA and mTA against the some exist methods.

6.1. Data Sets and Preparations

As mentioned in Section 2, the query model in [5, 9, 41] is most related to our model with mTA. To facilitate comparison, the eight datasets we used include data of both low-dimensionality (2, 3, and 4 dimensions) and high-dimensionality (25, 50, and 104 dimensions) in [41]. For low-dimensional datasets, both synthetic and real datasets in [5] are used: the real datasets include *Census2D(Age, Income)* and *Census3D(Age, Income, WeeksWorkedPerYear)* (both with 210,138 tuples), and *Cover4D(Elevation, Aspect, Slope, DistanceToRoadways)* (581,010 tuples); the synthetic datasets are *Gauss3D* (500,000 tuples) with Gaussian distribution, and *Array3D* (507,701 tuples) with Zipfian distribution. For high-dimensional datasets, real datasets derived from Telcordia LSI (Latent Semantic Indexing) Engine [9] are used in our experiments. The LSI datasets have 20,000 tuples and the same 25, 50 and 104 attributes in [9] are used to create the datasets of 25, 50 and 104 dimensions, denoted by *Lsi25D*, *Lsi50D* and *Lsi104D*, respectively. In the names of all datasets, suffix “ nD ” indicates that the dataset has n dimensions. The attribute values are all integers for low-dimensional datasets, and are double precision floating point numbers in the domain $[-3.3991 \times 10^{38}, -8.01543 \times 10^{-43}] \cup [1.03108 \times 10^{-41}, 3.40237 \times 10^{38}]$ for high-dimensional datasets. We report results based on the following default setting: (1) each dataset uses a workload with 1000 queries that are the tuples randomly selected from respective dataset; (2) we use the three distance functions [5, 9, 41] (see Section 3.1): *Manhattan distance*, *Euclidean distance* and *Maximum distance*; (3) $N = 100$ (i.e., retrieve top 100 tuples for each query) for low-dimensional datasets

[5, 9, 41]; $N = 20$ for high-dimensional datasets [9, 41]. When a different setting is used, it will be explicitly specified.

For each dataset, the following measures are used in our experiments:

- *The elapsed time (millisecond, ms) used to obtain the top- N tuples:* The time needed to find the top- N tuples from the respective dataset for a query. We will present the average time for all queries in the workload with respect to the dataset.
- *The number of sorted accesses:* The average number of sorted accesses for all queries in the workload.

From Section 6.2 to 6.4, the experiments are carried out using Microsoft's VC++6.0 on a PC with Windows XP, Intel(R) Core(TM) i5-2400 CPU @ 3.10 GHz 3.09GHz, and 2.98GB memory. In Section 6.5, we compare our algorithms with existing techniques. We assume that all sorted lists are in main memory, and sorted access cost equals random access cost, i.e., $c_s = c_r$. Thus, we only report the number of sorted accesses, since GTA, GTAz, mTA and mTAz will do $n-1$ random accesses for each sorted access, and no random access is performed for GNRA and mNRA. For GTAz and mTAz, $Z = \{1\}$, i.e., there is only one sorted list L_1 for the first attribute A_1 of each dataset.

6.2. The Elapsed Time

For the eight datasets, six algorithms and three ranking functions, Figure 8 illustrates the average elapsed time for 1000 queries in each workload. We can see that Figure 8(a), (b) and (c) have almost the same trend. All six algorithms are efficient for 2- and 3-dimensional datasets; mTA and mTAz are more efficient than the others for the eight datasets since they are pseudo instance optimal. Larger execution costs (i.e., the sorted and random access costs) will lead to larger elapsed times; comparing with Figure 9, the algorithms GTA, GNRA and mNRA have the larger number of sorted accesses in Figure 9 for datasets Cover4D, Lsi25D, Lsi50D and Lsi104D, thus, having larger elapsed times.

Moreover, the elapsed time contains internal computation costs, and is related to the inherent property of each algorithm; thus, the elapsed times are different, especially for high-dimensional datasets. For example, comparing Figure 8 with Figure 9, it can be seen that GTAz has larger times in Figure 8 but

smaller sorted access numbers in Figure 9 for datasets Lsi25D, Lsi50D and Lsi104D by reason of the property of GTAz and the data distributions. The three high-dimensional datasets have the same values of attribute A_1 , all the 20000 values are distinct, and the difference $a_{1j+1} - a_{1j}$ in the list L_1 is too small for many j 's; thus, the n -square $B(Q, r, d_\infty)$ and the n -ball $B(Q, r, d)$ are enlarged too slowly, which leads to that GTAz sees tuples in lockstep and computes and compares the distances of tuples in the buffer Y again and again. Therefore, the elapsed time of GTAz contains lots of internal computation costs.

Figure 8. The elapsed time for the eight datasets and six algorithms. (a) $d_1(\cdot, \cdot)$; (b) $d_2(\cdot, \cdot)$; (c) $d_\infty(\cdot, \cdot)$.

As shown in Section 5.2, mNRA is pseudo instance optimal involving only execution costs. Being similar to GTAz, mNRA contains more internal computation costs than GNRA, thus the elapsed time of

mNRA is larger than that of GNRA for datasets Lsi25D, Lsi50D and Lsi104D, though they have almost the same number of sorted accesses in Figure 9.

6.3. The Number of Sorted Accesses

Figure 9 illustrates the average number of sorted accesses for each workload. Comparing Figure 9 with Figure 8, we can see that the number of sorted accesses has a direct and proportional impact on the elapsed time in Figure 8 for evaluation of top- N tuples in majority of cases except for the algorithms GTAz and mNRA with high-dimensional datasets. The reasons for the exceptions are the data distributions of datasets and the inherent properties of the two algorithms as explained in Section 6.2.

Figure 9. The number of sorted accesses for the eight datasets and six algorithms. (a) $d_1(\cdot, \cdot)$; (b) $d_2(\cdot, \cdot)$; (c) $d_\infty(\cdot, \cdot)$.

Figure 9(a), (b) and (c) have the same trend. Referring to the experimental results in [5, 9, 41], the number of candidate tuples satisfies $|CAND(d_1)| \geq |CAND(d_2)| \geq |CAND(d_\infty)|$ for an algorithm over a certain dataset [5, 9, 41], where $CAND(d)$ with distance function $d(\cdot, \cdot)$ indicates the set of candidate tuples in the n -square $B(Q, r, d_\infty)$. Intuitively, the numbers of sorted accesses for different functions should satisfy the similar inequalities. The results in Figure 9 confirm the assertion in almost every case, meanwhile the differences are small for many cases. For example, GTA has the same number of sorted accesses for $d_1(\cdot, \cdot)$, $d_2(\cdot, \cdot)$ and $d_\infty(\cdot, \cdot)$ over a certain dataset except for dataset Lsi25D with $d_2(\cdot, \cdot)$. The other exceptions include: (1) mTA for datasets Gauss3D and Array3D with the three distances, and (2) mNRA for datasets Cover4D, Lsi25D, Lsi50D and Lsi104D with the three distances. The primary reasons for the exceptions are: (1) different functions may lead to different sets of the top- N tuples, and (2) the number of sorted accesses involves some seen tuples counted repeatedly.

Consequently, the trends in Figures 8 and 9 illustrate that ranking functions are not the key factors in designing our algorithms.

6.4. Effect of Different Result Size N

In this subsection, we discuss the effect of N on the performance of our algorithms. As an example, we only report the average number of sorted accesses for our representative algorithms GTA and mTA with the Maximum distance $d_\infty(\cdot, \cdot)$. Let the increasing rate of N_1 and N_2 be $Rate(N_1, N_2) = (N_2 - N_1)/N_1$ where N_1 and N_2 are neighbored with $N_1 < N_2$, and let the increasing rate of sorted access numbers for N_1 and N_2 be $Rate(san(N_1), san(N_2)) = (san(N_2) - san(N_1))/san(N_1)$, where $san(N)$ denotes the sorted access number at N . We define that the *ratio* of the two rates as: $ratio(N_1, N_2) = Rate(san(N_1), san(N_2))/Rate(N_1, N_2)$, which is used to measure the increasing *ratio* of the number of sorted accesses.

Figure 10 depicts the number of sorted accesses for the five low-dimensional datasets. With increasing N from 10 to 500, the sorted access numbers of the algorithms GTA and mTA increase with N because more tuples in a dataset need to be seen in order to obtain more top- N tuples. However, the increases are small, all $ratio(N_1, N_2)$'s are in the interval $[0.06, 0.4]$ for GTA, and are in $[0.2, 0.4]$ for mTA. The reason

is that GTA enlarges the n -square to find the candidates of top- N tuples, and then it sees a lot of tuples before halting its execution. Thus, when GTA obtains the top- N tuples for a query, there is a high probability that the top- $(N+1)$ tuple of the query has already been included in the n -square. For mTA, the reason is that mTA sees tuples in each sorted list in lockstep and mTA is pseudo instance optimal.

Figure 10. The number of sorted accesses vs. N over low-dimensional datasets. (a) GTA; (b) mTA.

Figure 11. For GTA and mTA, the average number of sorted accesses vs. N over high-dimensional datasets.

Figure 11 shows the results with N from 2 to 50 for the three high-dimensional datasets. We can see that the sorted access numbers of GTA stay almost the same and all of $ratio(N_1, N_2)$'s are in the interval $[0.002, 0.007]$. The sorted access numbers of mTA are much smaller than those of GTA; however, the increases of mTA are larger than those of GTA, and all $ratio(N_1, N_2)$'s for mTA are in $[0.28, 1.0]$.

6.5. Performance Comparison

In this subsection, firstly, we compare our algorithms GTA and mTA against the some exist methods reported in [5, 9, 41] since the query model in [5, 9, 41] is most related to that with mTA as described in Section 2.

Zhu et al. [41] compared learning-based (LB) method against histogram-based method in [5] and sampling-based technique in [9], and the experimental results in [41] demonstrated that LB is highly competitive with the histogram-based and sampling-based techniques for both low-dimensional and high-dimensional data. Therefore, it is sufficient to compare our algorithms with LB method and Optimum method (OPT) in [5, 41].

In order to compare our algorithms against the methods reported in [41], we use the same setting as that in [41]: Microsoft's VC++6.0 and Windows XP on a PC of a Pentium 4 CPU with 2.8GHz and 768MB memory. A workload has 100 queries, the *Maximum distance* for low-dimensional datasets, and the *Euclidean distance* for high-dimensional datasets.

Table 1. The average elapsed time (millisecond, ms)

Datasets	GTA	mTA	OPT	LB	SCAN
Census2D	371	133	173	320	13017
Census3D	706	149	118	252	13651
Array3D	485	375	707	853	34389
Gauss3D	452	367	492	688	32948
Cover4D	2500	555	756	902	39816
Lsi25D	365	63	240	341	2961
Lsi50D	619	125	447	528	4245
Lsi104D	1093	328	724	850	6031

Table 1 lists the average elapsed time of the queries in the workload for five kinds of techniques with both sorted and random accesses, i.e., GTA and mTA in this paper, while Optimum (OPT), Learning-based (LB) and sequential scan (SCAN) techniques reported in [41]. As a baseline, OPT [5, 41] is the theoretical best technique that uses the smallest n -square containing the actual top- N tuples for a given query and it cannot be implemented in practice by the strategies in [5, 9, 41], where the smallest n -square

is obtained by using the SCAN technique in advance. SCAN [5, 41] is the alias of the Naïve Algorithm in [14].

As shown in [14, 15], TA will never access more distinct objects than Fagin's Algorithm (FA) [12, 14], but TA may perform more random accesses than FA; thus, TA is usually (but not always) more efficient than FA. Our mTA comes from TA and it is pseudo instance optimal; while OPT is a variant of FA with the smallest n -square like the variant of GTA in Theorem 4.2, and thus OPT is not pseudo instance optimal. From Table 1, it can be seen that mTA outperforms OPT except for Census3D (149ms vs. 118ms), while outperforms LB significantly for all eight datasets. Generally, OPT needs to do more executions than mTA. As an example, for top-3 query $Q = (7, 6)$ with $d_1(\cdot, \cdot)$ in Figure 3, the number of sorted accesses with mTA is 10 as shown in Example 3, whereas, that with OPT is 15. Therefore, our mTA is highly competitive with OPT [5, 41] and LB [41]. Note that mTA is designed for x -monotonic norms/functions while the techniques in [5, 9, 41] are for p -norms. Thus, mTA is able to handle a wider range of functions than the techniques in [5, 9, 41].

GTA outperforms LB for Array3D and Gauss3D, but underperforms LB for the other datasets; while it underperforms OPT for all eight datasets. Particularly, GTA takes 2500 and 1093 milliseconds for Cover4D and Lsi104D, respectively. The reason is that it is not pseudo instance optimal. GTA is designed for generic nonmonotone norms, for each list it does sorted accesses in a small interval that may include many (even all) entries for the dataset. Moreover, GTA outperforms SCAN significantly for all eight datasets; thus, it is always a better alternative than SCAN.

As described in Section 5.3, the algorithms in the TA family with no wild guesses are different from methods of NNS; therefore, we cannot compare their performance directly.

Recently, [35] presented the SIMP method for NNS in high-dimensional spaces, and showed that SIMP significantly outperforms four existing state-of-the-art methods: iDistance, p-stable LSH, Multi-Probe LSH and LSB tree. For nearest neighbor (i.e., top-1) queries with query range 200, using quad-core Intel(R) Xeon(R) CPU 5,140@2.33GHz in [35], the query elapsed times of SIMP with *Euclidean distance* are larger than 75ms and 100ms on 128-dimensional SIFT dataset and 32-dimensional Aerial

dataset, respectively.

From Figure 8(b) with *Euclidean distance*, all of the query elapsed times of our mTA and mTAz algorithms are between 13ms and 77ms for top-100 or top-20 queries; thus, mTA and mTAz are highly competitive with the above five existing methods of NNS in [35] although different experimental environments and application scenarios are involved.

7. CONCLUSIONS

We analyzed the query models with arbitrary query points and generic (not necessarily F -monotone) ranking functions in n -dimensional normed spaces, and then proposed an approach for evaluating top- N queries. Based on the three cases of data access methods, we presented three algorithms GTA, GTAz and GNRA for generic norm distances, and three algorithms mTA, mTAz and mNRA for x -monotonic norm distances, respectively. We showed that GTA, GTAz, mTA, and mTAz require only bounded buffers, whereas GNRA and mNRA may require unbounded buffers. In addition, we showed that mTA, mTAz and mNRA are pseudo instance optimal, but GTA, GTAz and GNRA are not. We carried out extensive experiments to measure the performance of our approach using a variety of datasets of different dimensions (including 2, 3, 4; 25, 50, and 104 dimensions). The results demonstrated that GTA outperforms SCAN [41] significantly, and mTA is highly competitive with OPT, the ideal technique in [5, 41]; moreover, mTA based on x -monotonic norms is able to handle a wider range of functions than the methods with p -norms [5, 9, 41]. Furthermore, ranking functions are not the key factors in designing our algorithms.

Many variations of the three algorithms TA, TAz and NRA [14] have been proposed for several applications as mentioned in Section 2.1. The algorithms proposed in this paper are motivated by TA, TAz and NRA [14]. In the future, we plan to further optimize our algorithms, especially the algorithms GTAz and mNRA with high-dimensional datasets.

Acknowledgments: This work is supported in part by NSFC (61170039) and the NSF of Hebei Province (F2012201006). The authors would also like to thank Dr. Zhibin Hou for providing many helpful suggestions.

REFERENCES

- [1] R. Akbarinia, E. Pacitti, P. Valduriez, Best position algorithms for top-k queries, in: Proc. 33rd Int. Conf. Very Large Data Bases (VLDB), Vienna, Austria, 2007, pp. 495-506.
- [2] S. Berchtold, C. Böhm, D.A. Keim, H.P. Kriegel, X. Xu, Optimal multidimensional query processing using tree striping, in: Proc. 2nd Int. Conf. Data Warehousing and knowledge discovery (DaWaK), London, UK, 2000, pp. 244-257.
- [3] A. Borodin, R. Ostrovsky, Y. Rabani, Lower bounds for high dimensional nearest neighbor search and related problems, in: Proc. 31st Annual ACM Symposium on Theory of Computing (STOC), Atlanta, Georgia, USA, 1999: 312-321
- [4] S. Börzsönyi, D. Kossmann, K. Stocker, The skyline operator, in: Proc. 17th Int. Conf. on Data Eng. (ICDE), Heidelberg, 2001, pp. 421-430.
- [5] N. Bruno, S. Chaudhuri, L. Gravano, Top-k selection queries over relational databases: mapping strategies and performance evaluation, ACM Trans. Database Syst. 27(2) (2002) 153-187.
- [6] N. Bruno, L. Gravano, A. Marian, Evaluating top-k queries over web-accessible databases, in: Proc. 18th Int. Conf. on Data Eng. (ICDE), IEEE Computer Society, Silver Spring, MD, 2002. pp 369-380.
- [7] M. Carey, D. Kossmann, On saying “enough already!” in SQL, in: Proc. ACM Int. Conf. Management of Data (SIGMOD), Tucson, Arizona, USA, 1997, pp. 219-230.
- [8] M. Charikar, Similarity estimation techniques from rounding algorithms. In Proc. 34th Annual ACM Symposium on Theory of Computing (STOC), Montréal, Québec, Canada, 2002, pp 380-388.
- [9] C. Chen, Y. Ling, A sampling-based estimator for top-k selection query, in: Proc. 18th Int. Conf. Data Eng. (ICDE), San Jose, California, USA, 2002, pp. 617-627.
- [10] J.B. Conway, A course in functional analysis. Springer-Verlag, New York, 1985.
- [11] D. Donjerkovic, R. Ramakrishnan, Probabilistic optimization of top N queries, in: Proc. 25th Int. Conf. Very Large Data Bases (VLDB), Edinburgh, Scotland, UK, 1999, pp. 411-422.
- [12] R. Fagin, Combining fuzzy information from multiple systems, J. Comput. System Sci. 58(1) (1999) 83–99.

- [13] R. Fagin, R. Kumar, D. Sivakumar, Efficient similarity search and classification via rank aggregation, in: Proc. ACM Int. Conf. Management of Data (SIGMOD), San Diego, California, USA, 2003, 301-312. //A. Andoni, R. Fagin, R. Kumar, M. Patrascu, D. Sivakumar, Corrigendum to "efficient similarity search and classification via rank aggregation" by Ronald Fagin, Ravi Kumar and D. Sivakumar (proc. SIGMOD'03), in: Proc. ACM Int. Conf. Management of Data (SIGMOD), Vancouver, BC, Canada, 2008, pp1375-1376.
- [14] R. Fagin, A. Lotem, M. Naor, Optimal aggregation algorithms for middleware, J. Comput. Syst. Sci. 66(4) (2003) 614-656.
- [15] U. Güntzer, W.-T. Balke, W. Kießling, Optimizing multi-feature queries for image databases, in: Proc. 26th Int. Conf. Very Large Databases (VLDB), Cairo, Egypt, 2000, pp. 419-428.
- [16] X. Han, J. Li, J. Wang, D. Yang, TJJE: An efficient algorithm for top-k join on massive data, Information Sciences 222 (2013) 362-383
- [17] S. Har-Peled, P. Indyk, R. Motwani, Approximate nearest neighbor: towards removing the curse of dimensionality. Theory of Comput. 8(1) (2012) 321-350.
- [18] J.-S. Heo, K.-Y. Whang, M.-S. Kim, Y.-R. Kim, I.-Y. Song, The partitioned-layer index: Answering monotone top-k queries using the convex skyline and partitioning-merging technique. Information Sciences 179 (2009) 3286-3308.
- [19] S.-W. Hwang, K.C.-C. Chang, Optimizing top-k queries for middleware access: a unified cost-based approach, ACM Trans. Database Syst. 32(1) (2007) NO.5.
- [20] I. Ilyas, G. Beskales, M.A. Soliman, A survey of top-k query processing techniques in relational database systems, ACM Comput. Surv. 40(4) (2008) No.11.
- [21] P. Indyk, Nearest neighbors in high-dimensional spaces, in: Handbook of Discrete and Computational Geometry, Chapter 39, CRC Press LLC, 2nd edition, 2004.
- [22] H. Jagadish, B. Ooi, K.-L. Tan, C. Yu, R. Zhang, iDistance: an adaptive B+-tree based indexing method for nearest neighbor search, ACM Trans. Database Syst. 30(2) (2005) 364-397.
- [23] W. Jin, J. Patel, Efficient and generic evaluation of ranked queries, in: Proc. ACM Int. Conf. Management of Data (SIGMOD), Athens, Greece, 2011, pp. 601-612.
- [24] J. Kleinberg, Two algorithms for nearest-neighbor search in high dimensions, in: Proc. 29th Annual ACM Symposium on Theory of Computing (STOC), El Paso, Texas, USA, 1997, pap 599-608.

- [25] C. Li, K. Chang, I. Ilyas, S. Song, RankSQL: query algebra and optimization for relational top-k queries, in: Proc. ACM Int. Conf. Management of Data (SIGMOD), Baltimore, Maryland, USA, 2005, pp. 131-142.
- [26] Q. Lv, W. Josephson, Z. Wang, M. Charikar, K. Li, Multi-probe LSH: efficient indexing for high-dimensional similarity search, in: Proc. 33rd Int. Conf. Very Large Data Bases (VLDB), Vienna, Austria, 2007, pp. 950-961.
- [27] N. Mamoulis, M.L. Yiu, K.H. Cheng, D.W. Cheung, Efficient top-k aggregation of ranked inputs, ACM Trans. Database Syst. 32(3) (2007) NO.19.
- [28] S. Nepal, M.V. Ramakrishna, Query processing issues in image (multimedia) databases, in: Proc. 15th Int. Conf. Data Eng. (ICDE), Sydney, 1999, pp. 22-29.
- [29] H. Pang, X. Ding, B. Zheng, Efficient processing of exact top-k queries over disk-resident sorted lists, VLDB J. 19(3) (2010) 437-456.
- [30] R. Ramakrishnan, Database Management Systems, WCB/McGraw-Hill, Boston, 1998.
- [31] S. Ranu, A.K. Singh, Answering top-k queries over a mixture of attractive and repulsive dimensions, in: Proc. 38th Int. Conf. Very Large Data Bases (VLDB), Istanbul, Turkey, 2012, pp. 169-180.
- [32] N. Sarnak, R.E. Tarjan, Planar point location using persistent search trees, Commun. ACM 29(7) (1986) 669-679.
- [33] M. Schäler, A. Grebhahn, R. Schröter, S. Schulze, V. Köppen, G. Saake, Queval: beyond high-dimensional indexing a la carte. Proc. VLDB Endowment 6(14) (2013) 1654-1665.
- [34] T. Seidl, H.-P. Kriegel, Optimal multi-step k-nearest neighbor search, in: Proc. ACM Int. Conf. Management of Data (SIGMOD), Seattle, Washington, 1998, pp. 154-165.
- [35] V. Singh, A.K. Singh, SIMP: accurate and efficient near neighbor search in high dimensional spaces. in: Proc. 15th Int. Conf. Extending Database Technol. (EDBT), Berlin, Germany, 2012, pp. 492-503
- [36] P. Tsaparas, Nearest neighbor search in multidimensional spaces. Depth oral report. Department of Computer Science, University of Toronto, 1999.
- [37] M. Wu, L. Berti-Equille, A. Marian, C. Procopiuc, D. Srivastava, Processing top-k join queries, Proc. VLDB Endowment 3(1) (2010) 860-870.
- [38] D. Xin, J. Han, K.C.-C. Chang, Progressive and selective merge: computing top-k with ad-hoc ranking functions, in: Proc. ACM Int. Conf. Management of Data (SIGMOD), Beijing, China, 2007, pp. 103-114.

- [39] C. Yu, G. Philip, W. Meng, Distributed top-N query processing with possibly uncooperative local systems. In: Proc. 29th Int. Conf. Very Large Data Bases (VLDB), Berlin, Germany, 2003, pp. 117-128.
- [40] Z. Zhang, S. Hwang, K.C.-C. Chang, M. Wang, C.A. Lang, Y. Chang, Boolean + ranking: querying a database by k -constrained optimization, in: Proc. ACM Int. Conf. Management of Data (SIGMOD), Chicago, Illinois, USA, 2006, pp. 359-370.
- [41] L. Zhu, W. Meng, C. Liu, W. Yang, D. Liu, Processing top-N relational queries by learning, J. Intell. Inf. Syst. 34(1) (2010) 21-55.