A Guide to Designing Top-k Indexes

Saladi Rahul
University of Illinois Urbana-Champaign
USA
saladi.rahul@gmail.com

Yufei Tao
Chinese University of Hong Kong
Hong Kong
taoyf@cse.cuhk.edu.hk

ABSTRACT

Top-k search, which reports the k elements of the highest importance from all the elements in an underlying dataset that satisfy a certain predicate, has attracted significant attention from the database community. The search efficiency crucially depends on the quality of an index structure that can be utilized to filter the underlying data by both the user-specified predicate and the ranking of importance. This article introduces the reader to a list of techniques for designing such indexes with strong performance guarantees. Several promising directions for future work are also discussed.

1. INTRODUCTION

Interactive exploration of a database system is often hampered by the fact that a query may return a set of records that is excessively large for a user to examine. On the other hand, a user would rarely be interested in all the records satisfying her/his query predicate q. In many situations, what matters most is the subset that contains only the k records — for a small integer k — in the result with the highest "importance", where importance is measured by an appropriate ranking function. For example, while the query "find all the creditcard transactions of today" can return millions of records, what a bank manager would actually like to do could be just to scrutinize the 100 transactions with the largest payments. Retrieving only the k best records is commonly known as *top-k search*, and becomes increasingly useful as database volumes continue to grow at a rapid pace.

Besides controlling the output size, top-k search also brings opportunities to boost the efficiency of query processing because returning only k records can potentially be significantly faster than fetching the full result. This requires a mechanism that can be deployed to avoid accessing the records that satisfy the search predicate q but do not have sufficiently high importance to be reported. Designing such mechanisms has been a major research topic in the database area during the past two decades (see, e.g., [1, 7–9, 19, 25–27, 30, 31] and the references therein). At the core of almost every mechanism is an index structure — henceforth referred to as a *top-k index* — which stores certain information pre-computed from the underlying data that can lead the query algorithm to finding the k target records efficiently. Unlike conventional access methods in a database system, a top-k index must allow a query to filter the data not only by the predicate q, but also by the ranking of importance.

In this article, we introduce the reader to a suite of representative techniques that have been proposed in the literature for designing top-k indexes. Our discussion will focus on obtaining indexes with strong theoretical guarantees, and therefore, will not be concerned with methods that are designed purely for empirical evaluation. We will also point out some directions that call for further research efforts on this fascinating topic.

The content of this article has little overlap with *Fagin's algorithm* [14] and the *threshold algorithm* by Fagin, Lotem, and Naor [15] which were developed for a class of problems on *distributed computation* which are sometimes referred to also under the name "top-k". As surveyed in [19], there have been multiple attempts to apply the algorithms of [14, 15] to answer top-k queries in centralized systems. However, none of those attempts has succeeded in attaining performance guarantees that are interesting through the lens of this article. The techniques we will describe all follow ideas different from those of [14, 15].

2. PROBLEM DEFINITIONS

In Section 2.1, we provide a problem formulation that encapsulates a broad class of top-k problems. Section 2.2 gives two representative problems that will be utilized to demonstrate the techniques to be discussed. Section 2.3 clarifies the computation model to be adopted, and the performance guarantees to be achieved.

2.1 Generic Query Formulation

Let $\mathbb{D}$ be an arbitrary set which serves as the data domain. Denote by $\mathbb{Q}$ a set of *predicates* that can be applied to the elements of $\mathbb{D}$. Specifically, given a predicate $q \in \mathbb{Q}$, we can evaluate q on every element $e \in \mathbb{D}$ to

obtain a boolean value 1 or 0; in the former case e is said to *satisfy* q, while in the latter e does not. We assume that $\mathbb{Q}$ has a special predicate `true`, which evaluates to 1 for all $e \in \mathbb{D}$.

The input dataset D is a subset of $\mathbb{D}$. For each predicate $q \in \mathbb{Q}$, define $q(D)$ to be the set of elements in D satisfying q. Given a predicate $q \in \mathbb{Q}$, a *reporting query* returns $q(D)$ in its entirety.

As mentioned earlier, a user is often interested only in the most "important" elements of $q(D)$. We formalize importance by resorting to a weight function $w : \mathbb{D} \to \mathbb{R}$, where $\mathbb{R}$ represents the set of real numbers. For each element $e \in \mathbb{D}$, we refer to the value $w(e)$ as the *weight* of e under w. Denote by $\mathbb{W}$ a non-empty set of such weight functions. Every reporting query has a top-k counterpart:

> Given a predicate $q \in \mathbb{Q}$, a weight function $w \in \mathbb{W}$, and an integer $k \geq 1$, a *top-k query* reports the k elements in $q(D)$ with the highest weights under w. Specially, if $|q(D)| < k$, then the entire $q(D)$ is reported.

If two elements have the same weight, we assume that the tie is broken by a consistent policy, e.g., regarding the element with a larger id to have a greater weight. Note that a top-1 query, which will be referred to as a *max query*, returns the element in $q(D)$ with the maximum weight.

Our exploration into the theory of top-k queries will encounter frequently another variant of reporting queries:

> Given a predicate $q \in \mathbb{Q}$, a weight function $w \in \mathbb{W}$, and a real value τ, a *prioritized query* reports all the elements $e \in q(D)$ with $w(e) \geq \tau$.

In general, the set $\mathbb{W}$ may contain an arbitrarily large number of functions. In the other extreme, $\mathbb{W}$ may include only a *single* function. In that case, obviously, all the top-k and prioritized queries must choose the same, unique, function w in $\mathbb{W}$, such that we can as well regard the weight $w(e)$ directly as an attribute associated with each element $e \in D$. When this happens, we will refer to D as a *weight-augmented* dataset.

2.2 Two Specialized Instances

Next, we specialize the above formulation into two concrete instances that are representative for several reasons. First, the top-k queries in both instances are important top-k problems that have been extensively studied. Second, their specialization is based on drastically different choices of $\mathbb{D}$, $\mathbb{Q}$, and $\mathbb{W}$, thereby demonstrating the generality of our query formulation. Third, they are among the simplest instances suitable for illustrating the techniques to be presented in later sections.

Problem 1: Linear Ranking. In this instance:

- $\mathbb{D} = \mathbb{R}^d$, where d is a positive constant integer;
- $\mathbb{Q} = \{\texttt{true}\}$, namely, $\mathbb{Q}$ has only a single predicate that always evaluates to 1;
- $\mathbb{W}$ is the set of linear functions which are defined by d real values $c_1, ..., c_d$, and map a point $p = (x_1, x_2, ..., x_d) \in \mathbb{D}$ to $\sum_{i=1}^{d} c_i x_i$.

Equivalently, a dataset D is a set of points in the d-dimensional space. Given the coefficients $c_1, ..., c_d$, a top-k query reports the k points $p = (x_1, ..., x_d) \in D$ that maximize $\sum_{i=1}^{d} c_i x_i$. Such queries constitute the top-k problem that has received by far the most attention from the *system* community (see [19] for a survey). As a classic application, consider that $d = 2$, and each point $e \in D$ captures the `price` and `rating` of a hotel as the x- and y-coordinates, respectively. A top-10 query finds the 10 best hotels maximizing $c_1 \cdot (-\texttt{price}) + c_2 \cdot \texttt{rating}$, where c_1 and c_2 are coefficients chosen by a user, reflecting her/his personal weighting on the two attributes.

Problem 2: One-Dimensional Range Searching. This is an instance on weight-augmented datasets:

- $\mathbb{D} = \mathbb{R}$;
- $\mathbb{Q}$ consists of all such predicates, each of which specifies an interval q in $\mathbb{R}$, and evaluates to 1 on every point $e \in \mathbb{D} \cap q$, and to 0 on every point $e \notin \mathbb{D} \cap q$.

Equivalently, a dataset D is a set of points in $\mathbb{R}$, each of which is associated with a real-valued weight. Given an interval q, a top-k query reports the k points in $D \cap q$ with the highest weights. Such queries constitute the most extensively studied top-k problem in the *theory* community [1, 7, 8, 25, 30, 31]. For an example, consider a `TRANSACTION` table with attributes `id`, `date` and `payment`, on which a top-100 query is "find the 100 tuples with the highest `payment` values among those with `date` in [01/2019, 03/2019]".

2.3 Computation Model and Design Goals

Our discussion will assume the standard word-RAM model. We further assume that every element in $\mathbb{D}$ can be stored in $O(1)$ words, and so can the encoding of each predicate in $\mathbb{Q}$ (e.g., in 1D range searching, each predicate is specified by an interval, which can be encoded in two words).

Define $n = |D|$, i.e., the number of elements in the input dataset. Our primary objective is to preprocess D into a top-k index that consumes near-linear space, and can be used to solve top-k queries efficiently. This means that the index should use $\tilde{O}(n)$ space, and answer a top-k query in $\mathcal{Q}_{top}(n) + \tilde{O}(k)$ time where notation

$\tilde{O}$ hides an $O(\text{polylog}\, n)$ factor, and $\mathcal{Q}_{top}$ is a slow-growing function of n. Ideally, we would also like the index to be *dynamic*, namely, updatable in $\tilde{O}(1)$ time per insertion and deletion.

Finally, it is worth mentioning that the result of a top-k query may return k elements in an arbitrary order. A sorted order can be generated by trivially sorting those elements in $O(k \log k)$ time. Remember that the value of k is supplied by a query as a parameter, instead of being fixed in advance.

3. TOP-K IMPLIES PRIORITIZED

Top-k and prioritized queries represent two similar ways to trim the result of a reporting query. Recall that, given a predicate $q \in \mathbb{Q}$, a reporting query returns $|q(D)|$ elements. The corresponding top-k query limits the output size to at most k explicitly. The corresponding prioritized query, on the other hand, filters out the elements of $q(D)$ with weights less than τ, after which the number t of elements reported can be anywhere from 0 to $|q(D)|$.

There is a subtle but important difference between top-k and prioritized queries. For a top-k query, whether an element $e \in q(D)$ should be reported does *not* depend solely on e, because it is also affected by the weights of the other elements in $q(D)$. In contrast, for a prioritized query, whether e should be reported can be decided by looking at e *itself*. Intuitively, this suggests that top-k queries ought to be at least as hard as prioritized queries.

This intuition turns out to be correct, namely, if we can find a top-k index with a certain space-query tradeoff, we must be able to achieve the same tradeoff asymptotically for prioritized queries:

THEOREM 1. *Fix D, $\mathbb{Q}$, and $\mathbb{W}$, and set $n = |D|$. Suppose that there is a top-k index on D that consumes $\mathcal{S}_{top}(n)$ space, and answers a top-k query in $\mathcal{Q}_{top}(n) + O(k)$ time, where $\mathcal{Q}_{top}(n) > 0$ for all n. Then, there is a data structure on D that uses $\mathcal{S}_{top}(n)$ space, and answers a prioritized query in $O(\mathcal{Q}_{top}(n)+t)$ time, where t is the number of reported elements.*

PROOF. Let $\mathcal{T}$ be the top-k index on D as described in the theorem. Given a prioritized query with parameters $q \in \mathbb{Q}, w \in \mathbb{W}$, and $\tau \in \mathbb{R}$, we use $\mathcal{T}$ to answer it by executing multiple rounds as follows. In round j (starting with $j = 1$), we issue a top-k_j query on $\mathcal{T}$ with the same q and w by setting $k_j = 2^{j-1} \cdot \mathcal{Q}_{top}(n)$. Two cases may arise:

- If the top-k_j query returns exactly k_j elements, we scan them to find the element e with the smallest weight. If $w(e) < \tau$, we do not go to the next round; otherwise, round $j + 1$ is launched.
- If the top-k_j query returns less than k_j elements, no more rounds are performed.

Let i be the number of rounds executed. Among the elements reported by the top-k_i query, we remove those with weights less than τ, and return the rest of the elements as the output of the prioritized query.

The space consumption of $\mathcal{T}$ is clearly $\mathcal{S}_{top}(n)$. Now we analyze the query time. If only one round is executed, the time is bounded by $\mathcal{Q}_{top}(n)+O(k_1) = O(\mathcal{Q}_{top}(n))$, noticing that $k_1 = \mathcal{Q}_{top}(n)$. If $i \geq 2$ rounds are performed, the time is bounded by

$$O\Big(\sum_{j=1}^{i} \mathcal{Q}_{top}(n) + 2^{j-1} \cdot \mathcal{Q}_{top}(n)\Big) = O(2^i \cdot \mathcal{Q}_{top}(n)).$$

By the fact that the execution entered i-th round, we know that $t \geq 2^{i-2} \cdot \mathcal{Q}_{top}(n)$, which gives $2^i \cdot \mathcal{Q}_{top}(n) \leq 4t$. It thus follows that the total query time is $O(t)$. □

The above result, which was independently observed by the authors of [24, 25], has an interesting implication: attempts to design an effective top-k index should be carried out *after* one has succeeded in obtaining a structure capable of resolving a prioritized query with the desired space-query tradeoff. Indeed, the prioritized query serves as a good starting point to approach a top-k problem, which is a pattern that will show up repeatedly in the rest of this article.

4. A FRAMEWORK FOR DESIGNING TOP-K INDEXES

In this section, we will establish a powerful framework for obtaining top-k indexes that enjoy strong theoretical guarantees in expectation. This framework is remarkably easy to apply, and works for all top-k queries captured by the formulation in Section 2.1.

4.1 Equivalence between Top-k and the Combination of Prioritized and Max

We have seen in Section 3 that, to design a top-k index with a certain space-query tradeoff, one must be able to obtain a structure with the same tradeoff for the corresponding prioritized query. Another similar but more obvious fact is that any top-k index guaranteeing $\mathcal{Q}_{top}(n) + O(k)$ time must be able to answer the corresponding max (a.k.a. top-1) query in $O(\mathcal{Q}_{top}(n))$ time. In other words, the top-k query is at least as hard as *both* of its corresponding prioritized and max queries.

It turns out that, in terms of expected efficiency, the opposite is also true: the top-k query is *no harder* than solving both of the prioritized and max queries. To state this formally, let us first define a *geometrically converging function* to be a function $f : \mathbb{R}^+ \to \mathbb{R}^+$ (where $\mathbb{R}^+$ is the set of positive real numbers) satisfying two conditions:

- When $x \leq 2$, $f(x) = O(1)$;

- When $x > 2$,

$$\sum_{i=0}^{h} f\left(\frac{x}{c^i}\right) \quad = \quad O(f(x))$$

holds for any $c \geq 2$, where h is the largest integer i satisfying $x/c^i \geq 2$.

Note that all polynomial functions are geometrically converging. We are now ready to present the theorem, which is due to Rahul and Tao [27], for reducing a top-k problem to its prioritized and max counterparts:

THEOREM 2. *Fix D, $\mathbb{Q}$, and $\mathbb{W}$, and set $n = |D|$. Suppose that*

- *there is a structure on D that uses $\mathcal{S}_{pri}(n) = O(n^2)$ space, and answers a prioritized query in $\mathcal{Q}_{pri}(n) + O(t)$ time, where t is the number of elements reported;*
- *there is a structure on D that uses $\mathcal{S}_{max}(n)$ space, and answers a max query in $\mathcal{Q}_{max}(n)$ time, where function $\mathcal{S}_{max}(n)$ is geometrically converging.*

Then, there is a structure on D that uses $\mathcal{S}_{top}(n)$ space in expectation, and answers a top-k query in $\mathcal{Q}_{top}(n) + O(k)$ expected time, where

$$\mathcal{S}_{top}(n) = O\left(\mathcal{S}_{pri}(n) + \mathcal{S}_{max}\left(\frac{6n}{\mathcal{Q}_{pri}(n)}\right)\right) \quad (1)$$

$$\mathcal{Q}_{top}(n) = O\left(\mathcal{Q}_{pri}(n) + \mathcal{Q}_{max}(n)\right). \quad (2)$$

Furthermore, if the prioritized and max structures support an update in $\mathcal{U}_{pri}(n)$ and $\mathcal{U}_{max}(n)$ time respectively, then the top-k structure supports an update in $O(\mathcal{U}_{pri}(n) + \mathcal{U}_{max}(n))$ expected time. If any of $\mathcal{U}_{pri}(n)$ and $\mathcal{U}_{max}(n)$ is amortized, the update cost of the top-k structure is amortized expected.

Several remarks are in order:

- The above reduction is optimal in the sense that there is no performance degradation (in expectation): the space, query, and update costs of the top-k structure are all determined by the worse between the prioritized and max structures. Theorems 1 and 2 together establish the equivalence (again, in terms of expected performance) between answering top-k queries and settling the combination of prioritized queries and max queries.
- Somewhat surprisingly, $\mathcal{S}_{top}(n)$ may even be smaller than $O(\mathcal{S}_{max}(n))$. For instance, plugging in $\mathcal{S}_{pri}(n) = O(n)$, $\mathcal{S}_{max}(n) = O(n \log n)$, and any $\mathcal{Q}_{pri}(n) \geq \log n$, we obtain from Theorem 2 that $\mathcal{S}_{top}(n) = O(n)$. Indeed, the theorem achieves a "bootstrapping" effect such that one does not need to try very hard to minimize the space of the max structure.
- The condition $\mathcal{S}_{max}(n) = O(n^2)$ essentially captures all the max structures useful in practice. The power 2 is not compulsory, and can be replaced by any constant.

We will prove Theorem 2 in Section 4.3.

4.2 Applications of Theorem 2

Theorem 2 provides a clear direction for designing top-k indexes with strong performance guarantees. Next, we demonstrate this on the two problems listed in Section 2.2.

Linear Ranking. Let D be the input set of n points in $\mathbb{R}^d$. Given real-valued coefficients $c_1, ..., c_k$ and a real value τ, a prioritized query returns all the points $p = (p_1, ..., p_d)$ in D such that $\sum_{i=1}^{d} c_i \cdot p_i \geq \tau$. Given c_1, ..., c_k, a max query returns the point $p = (p_1, ..., p_d) \in D$ that maximizes $\sum_{i=1}^{d} c_i \cdot p_i$.

Both types of queries have been well studied in computational geometry. The prioritized query is known as *halfspace reporting*, while the max query as the *extreme-point query*. When $d \leq 3$:

- Afshani and Chan [2] described a structure of $O(n)$ space that can answer a halfspace reporting query in $O(\log n + t)$ time, where t is the number of points reported;
- A technique of Dobkin and Kirkpatrick [12] yields a structure of $O(n)$ space that can answer an extreme-point query in $O(\log n)$ time.

Immediately, Theorem 2 guarantees a top-k index that uses $O(n)$ space and answers a top-k query in $O(\log n + k)$ time, both in expectation.

When $d \geq 4$, no known structure is able to answer a halfspace reporting query in $O(\text{polylog}\, n + t)$ expected time under the space budget of $\tilde{O}(n)$ expected. In fact, a lower bound of [13] even rules out the possibility of such structures for $d \geq 5$. Together with Theorem 1, these facts indicate that it is unrealistic to hope for a top-k index of near-linear space that can answer a top-k query in $O(\text{polylog}\, n + k)$ time. We will continue this discussion in Section 6, where a more suitable technique will be applied to design top-k indexes for $d \geq 4$.

1D Range Searching. Let D be the input set of n points in $\mathbb{R}$. Each point $e \in D$ is associated with a weight $w(e)$. Given an interval $q = [x, y]$ in $\mathbb{R}$ and a real value τ, a prioritized query returns all the points $e \in D$ satisfying $x \leq e \leq y$ and $w(e) \geq \tau$. Given an interval $q = [x, y]$, a max query reports the point of the maximum weight among the points $e \in D$ satisfying $x \leq e \leq y$.

It is rudimentary (see, e.g., [11]) to design a structure of $O(n)$ space which answers a max query in $O(\log n)$ time, and can be updated in $O(\log n)$ time. The prioritized query is what is called *3-sided range reporting*

in computational geometry. Specifically, let us create a set of 2D points $P = \{(e, w(e)) \mid e \in D\}$. A prioritized query with parameters $q = [x, y]$ and τ essentially returns all the points in P that fall in the 3-sided rectangle $[x, y] \times [\tau, \infty)$. By creating a *priority search tree* (PST) [21] on P, we can answer the query in $O(\log n + t)$ time, where t is the number of points reported. The PST occupies $O(n)$ space, and supports each update in $O(\log n)$ time.

Immediately, Theorem 2 yields a dynamic top-k index of $O(n)$ space that answers a top-k query in $O(\log n + k)$ time, and can be updated in $O(\log n)$ time, where all complexities hold in expectation.

4.3 Proof of Theorem 2

This subsection serves as a proof of Theorem 2. We consider that $\mathcal{Q}_{max}(n) = O(n)$ because a max query can be trivially answered by scanning D once.

Rank Sampling. Given a set S of real values, and a real value $0 < p \leq 1$, we define a *p-sample set* of S to be a set R obtained by the following random process. At the beginning, $R = \emptyset$; then, each element of S is added to R with probability p independently. Furthermore, we say that an element $e \in S$ has *rank* i if e is the i-th greatest in S. The following is a technical lemma that will be useful later.

LEMMA 1. *Let S be a set of n elements, and K a real value satisfying $2 \leq K \leq n/4$. For a $(1/K)$-sample set R of S, the following hold simultaneously with probability at least 0.09:*

- $|R| \geq 1$
- *The largest element in R has rank in S greater than K but at most $4K$.*

PROOF. The first bullet fails only if none of the elements in S was sampled, which occurs with probability

$$(1 - 1/K)^n \leq (1 - 1/K)^{4K} \leq 1/e^4$$

where the last inequality used the fact that $(1 - x)^{1/x} < 1/e$ for all $x > 0$.

Let x be the largest element in R, and denote by $\hat{K}$ the rank of x in S. Next, we bound the probability of the event $\hat{K} > 4K$, which occurs only if none of the $4K$ largest elements in D were sampled. Hence:

$$\mathbf{Pr}[\hat{K} > 4K] = (1 - 1/K)^{4K} \leq 1/e^4.$$

Finally, we bound the probability of the event $\hat{K} \leq K$, which occurs only if at least one of the K largest elements in D was sampled. Hence:

$$\mathbf{Pr}[\hat{K} \leq K] = 1 - (1 - 1/K)^K.$$

Applying the fact that $(1 - 1/x)^x \geq 1/e^2$ for all $x \geq 2$, we know:

$$\mathbf{Pr}[\hat{K} \leq K] \leq 1 - 1/e^2.$$

The union bound now shows that the probability of violating at least one bullet of Lemma 1 is at most

$$2/e^4 + (1 - 1/e^2) < 0.91$$

which completes the proof. □

Structure. We now describe how to design a top-k index using the given prioritized and max structures as black boxes. First, build a prioritized structure on the input dataset D. Then, fix a constant $\sigma = 1/20$, and define for each integer $i \geq 1$:

$$K_i \quad = \quad \mathcal{Q}_{max}(n) \cdot (1 + \sigma)^{i-1}.$$

Let h be the largest i such that $K_i \leq n/4$; clearly, $h = O(\log n)$. For each $i \in [1, h]$, we take a $(1/K_i)$-sample set R_i of D, and create a max structure on R_i. The top-k index consists of the prioritized structure and the h max structures constructed.

Query. Suppose that we need to answer a top-k query that chooses a predicate $q \in \mathbb{Q}$ and a weight function $w \in \mathbb{W}$. If $k < \mathcal{Q}_{max}(n)$, we obtain the result S of a top-$\mathcal{Q}_{max}(n)$ query with the same q and w, and extract the k elements in S with the largest weights using the k-selection algorithm of [6] in $O(|S|) = O(\mathcal{Q}_{max}(n))$ time. The total cost is therefore the time of the top-$\mathcal{Q}_{max}(n)$ query, which we will prove later is $O(\mathcal{Q}_{pri}(n) + \mathcal{Q}_{max}(n))$ in expectation, plus $O(\mathcal{Q}_{max}(n))$.

Next, we consider $k \geq \mathcal{Q}_{max}(n)$. If $k \leq K_h$, set j^* to the smallest integer i satisfying $K_i \geq k$; note that $K_j = \Theta(k)$. Starting with $j = j^*$, we perform a *round* as follows:

1. Determine whether $|q(D)| < 4K_j$. This can be done in $\mathcal{Q}_{pri}(n) + O(K_j)$ time by performing a prioritized query in a *cost-monitoring* manner. Specifically, run a prioritized query with the parameters q, w, and $\tau = -\infty$, but terminate the query *manually* as soon as $4K_j$ elements have been reported. If manual terminate occurs, $|q(D)| \geq 4K_j$, and we proceed to the next step. Otherwise, the prioritized query finishes normally and must have returned the entire $q(D)$, implying $|q(D)| < 4K_j$, in which case we declare the round *successful* and terminate the whole algorithm by returning $q(D)$ as the result of the original top-k query.

2. Identify the element e in $q(R_j)$ with the maximum weight by issuing a max query on R_j with the parameters q and w which takes $\mathcal{Q}_{max}(n)$ time. In the special case where $q(R_j)$ is empty, treat e as a dummy element with $w(e) = -\infty$.

3. Perform a prioritized query on D with q, w, and $\tau = w(e)$ in a cost-monitoring manner:

(a) Either the query terminates *by itself*, outputting a set S of elements,

(b) Or we terminate it as soon as $4K_j+1$ elements have been reported.

In both cases, the cost is $\mathcal{Q}_{pri}(n) + O(K_j)$.

4. Declare this round *failed* if either of the following is true:

   - Case 3(a) occurred, but $|S| \leq K_j$.
   - Case 3(b) occurred.

   Otherwise, declare this round *successful*.

5. If the round is successful, perform k-selection on S to produce the k elements in $q(D)$ with the largest weights, and terminate the algorithm by returning them as the result of the top-k query.

6. Otherwise (i.e., failed), increase j by 1.

   (a) If $j \leq h$, perform the next round from Step 1.

   (b) Else (i.e., $j = h + 1$), answer the top-k query naively by reading the whole D in $O(n) = O(K_j)$ time. The algorithm then terminates. This is the only scenario where termination can happen in a failed round.

To analyze the cost of the algorithm, notice that a round fails only if $|q(|D|)| > 4K_j$ (otherwise, Line 1 terminates the algorithm), and one of the two bullets in Step 4 is true. Thus, Lemma 1 tells us that failure happens with probability at most 0.91, noticing that $q(R_j)$ is a $(1/K_j)$-sample set of $q(D)$. This implies that round j, for any $j \geq j^*$, is executed only with probability 0.91^{j-j^*}, namely, only when all the preceding rounds have failed. Also observe that round j, regardless of whether it fails, takes $\mathcal{Q}_{pri}(n)+\mathcal{Q}_{max}(n)+O(K_j)$ time. Thus, the expected cost of the algorithm is bounded by

$$\sum_{j=j^*}^{h} O\Big(\Big(\mathcal{Q}_{pri}(n) + \mathcal{Q}_{max}(n) + K_j\Big) \cdot 0.91^{j-j^*}\Big)$$

$$= O\Big(\mathcal{Q}_{pri}(n) + \mathcal{Q}_{max}(n) + \sum_{j=j^*}^{h} K_j \cdot 0.91^{j-j^*}\Big) \quad (3)$$

Note that $K_j = K_{j^*} \cdot (1+\sigma)^{j-j^*} = O(k) \cdot (1+\sigma)^{j-j^*}$. Plugging this into (3) shows that the expected cost is bounded by

$$O\left(\mathcal{Q}_{pri}(n) + \mathcal{Q}_{max}(n) + k \sum_{j=j^*}^{h} ((1+\sigma) \cdot 0.91)^{j-j^*}\right)$$

which is $O(\mathcal{Q}_{pri}(n) + \mathcal{Q}_{max}(n) + k)$ because $(1+\sigma) \cdot 0.91 < 1$.

Space. The prioritized structure on D obviously takes up $\mathcal{S}_{pri}(n)$ space. We claim that all the max structures occupy $o(n) + O(\mathcal{S}_{max}(\frac{6n}{\mathcal{Q}_{max}(n)}))$ expected space in total, which implies the space result in Theorem 2 because $\mathcal{S}_{pri}(n) = \Omega(n)$.

The claim is fairly intuitive because $\mathbf{E}[|R_i|] = n/K_i$ geometrically decreases as i increases, which, together with the fact that $\mathcal{S}_{max}(n)$ is geometrically converging, seems to yield the claim immediately. The complication, however, is that $\mathcal{S}_{max}(n)$ may be a convex function, such that $\mathbf{E}[\mathcal{S}_{max}(|R_i|)]$ is *not* necessarily $O(\mathcal{S}_{max}(\mathbf{E}[|R_i|]))$. Next, we show how to circumvent this obstacle.

We will prove that all the max structures occupy $o(n) + O(\mathcal{S}_{max}(\frac{6n}{\mathcal{Q}_{max}(n)}))$ space in total with probability at least $1 - 1/n^2$. Combining this with the fact that all those structures obviously demand no more than $O(\mathcal{S}_{max}(n) \cdot h) = O(n^2 \cdot \log n)$ space gives the target claim.

Let i^* be the largest i such that $K_i \leq n/(3 \ln n)$. Consider an $i \in [1, i^*]$. Since $|R_i|$ is the sum of n independent Bernoulli variables each of which equals 1 with probability $1/K_i$, a standard application of Chernoff bounds[1] gives:

$$\begin{aligned} \mathbf{Pr}[|R_i| \geq 6 \cdot \mathbf{E}[|R_i|]] &\leq \exp(-\mathbf{E}[|R_i|]) \\ &= \exp(-n/K_i) \leq 1/n^3. \end{aligned}$$

Therefore, with probability at least $1 - h/n^3$, the max structures on $R_1, R_2, ..., R_{i^*}$ use at most

$$\begin{aligned} &\sum_{i=1}^{i^*} O\left(\mathcal{S}_{max}\left(\frac{6n}{\mathcal{Q}_{max}(n) \cdot (1+\sigma)^{i-1}}\right)\right) \\ = \; &O\left(h + \mathcal{S}_{max}\left(\frac{6n}{\mathcal{Q}_{max}(n)}\right)\right) \end{aligned}$$

space overall.

Let us now concentrate on $i \in [i^* + 1, h]$. Notice that there are only $O(\log \log n)$ such values of i. Also, by definition of i^*, we know that $\mathbf{E}[|R_i|] = n/K_i$ is in the range from 4 to $O(\log n)$. Another application of Chernoff bounds gives:

$$\begin{aligned} &\mathbf{Pr}[|R_i| \geq (\ln n^4) \cdot \mathbf{E}[|R_i|]] \\ \leq \; &\exp(-(\ln n^4) \cdot \mathbf{E}[|R_i|]/6) \\ \leq \; &\exp(-\ln n^{4 \cdot 2/3}) \\ = \; &1/n^{8/3}. \end{aligned}$$

Hence, with probability at least $1 - O(\log \log n)/n^{8/3}$, it holds that for all $i \in [i^* + 1, h]$:

$$|R_i| \leq 4 \ln n \cdot \mathbf{E}[|R_i|] = O(\log^2 n).$$

[1] Let $X_1, ..., X_n$ be independent Bernoulli variables such that $\mathbf{Pr}[X_i = 1] = p_i$. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = \mathbf{E}[X] = \sum_{i=1}^{n} p_i$. For any $\alpha \in (0, 1)$, $\mathbf{Pr}[X \leq (1-\alpha)\mu] \leq e^{-\alpha^2 \mu/3}$, while for any $\alpha \geq 2$, $\mathbf{Pr}[X \geq \alpha\mu] \leq e^{-\alpha\mu/6}$.

By the fact that $\mathcal{S}_{max}(n) = O(1 + n^2)$, the max structures on R_{i^*}, R_{i^*+1}, ..., R_h together consume no more than $O(h + \log\log n \cdot \log^4 n) = o(n)$ space. We thus conclude that, with probability at least $1 - h/n^3 - O(\log\log n)/n^{8/3} > 1 - 1/n^2$, all the max structures use $o(n) + O(\mathcal{S}_{max}(\frac{6n}{\mathcal{Q}_{max}(n)}))$ space.

Update. It remains to discuss how to support insertions and deletions on the input set D. A crucial observation is that, in expectation, each element of D appears only in a constant number of max structures, noticing that the element belongs to R_i with probability $1/K_i$, which geometrically decreases as i increases. We can record using in expectation $O(1)$ words which max structures include e. It thus becomes obvious that the insertion or deletion of an element can be supported in $O(\mathcal{U}_{pri} + \mathcal{U}_{max})$ expected time. The above argument still works even if one or both of $\mathcal{U}_{pri}$ and $\mathcal{U}_{max}$ are amortized. This completes the whole proof of Theorem 2.

5. TECHNIQUES FOR PROBLEMS ON WEIGHT-AUGMENTED DATASETS

This section will focus on top-k problems where $|\mathbb{W}| = 1$, and hence, the weight $w(e)$ of each element $e \in D$ can be taken directly as an extra attribute of e. We will introduce several techniques that are effective on such problems, and at the same time amenable to practical implementation. Our description will be based on top-k range searching in 1D (see Section 2.2) whose simplicity will facilitate the understanding of the core ideas underneath. Unlike the solutions in Section 4 that are efficient in expectation, we will aim to obtain top-k indexes whose guarantees hold deterministically.

5.1 High-Level Ideas behind Sections 5.2-5.4

In 1D top-k range searching, the input dataset D is a set of n points in $\mathbb{R}$. Given an interval $q = [x, y]$, a top-k query returns the k points $e \in q(D)$ with the highest $w(e)$, where $q(D)$ is the set of points in D covered by q. Conceptually, we will answer the query in three steps:

1. *Size checking:* Decide whether $|q(D)| < k$. If so, simply retrieve the entire $q(D)$, and return it as the final result. Proceed to Step 2 only if $|q(D)| \geq k$.

2. *Thresholding:* Find a real-valued threshold τ such that at least k but at most $O(k)$ points $e \in q(D)$ satisfy $w(e) \geq \tau$.

3. *Prioritized reporting:* Find the set S of points in $q(D)$ whose weights are at least τ. The definition of τ makes sure that $k \leq |S| = O(k)$. Collect the k points in S with the highest weights, and return them as the result of the top-k query.

It is rudimentary to implement Step 1 in $O(\log n + k)$ time by resorting to a binary search tree (BST) on D. In Step 3, S can be found using a PST (priority search tree) on D in $O(\log n + |S|) = O(\log n + k)$ time, as we explained in Section 4.2. Finding the k points of the largest weights in S can be done with the k-selection algorithm [6], which takes $O(|S|) = O(k)$ time. Note that all the data structures needed in Steps 1 and 3 can be updated in $O(\log n)$ time per insertion and deletion.

The challenge is to design a data structure for Step 2, a phenomenon that is very typical in attacking a top-k problem through the above 3-step approach. In fact, Step 2 itself makes an interesting stand-alone problem, which was named the *approximate k-threshold problem* in [30]. In Sections 5.2-5.4, we will present several techniques to tackle the challenge

5.2 Technique 1: Binary Search

Let us start with a simple approach to find the target threshold τ — as in Step 2 of the algorithm in Section 5.1 — in $O(\log^2 n + k \log n)$ time using linear space. In fact, for a top-k query with search interval $q = [x, y]$, this approach returns a value of τ such that *precisely* k elements $e \in q(D)$ satisfy $w(e) \geq \tau$; furthermore, the τ returned is guaranteed to be the weight of some element in $q(D)$.[2]

Imagine that the weights of the points in D have been sorted in descending order into a list L. We can find the target τ by performing binary search on L. Specifically, the search starts by setting z to the median of L. Define c as the number of points $e \in q(D)$ with $w(e) \geq z$. We then determine which of the following is true: $c < k$, $c = k$, or $c > k$. If $c = k$, the algorithm finishes by returning $\tau = z$; otherwise, the search continues by focusing on the first or second half of L recursively.

The comparison between c and k can be resolved in $O(\log n + k)$ time by searching a PST in a cost-monitoring manner. Specifically, issue a prioritized query in the way explained in Section 4.2 to find all the points in $q(D)$ whose weights are at least z, with the difference that we manually terminate the prioritized query as soon as $k+1$ points have been reported. If manual termination occurs, c must be greater than k. Otherwise, c must be at most k, and all those c points must have been returned by the prioritized query. Overall, the binary search attempts $O(\log n)$ values of z, and therefore, finishes in $O(\log^2 n + k \log n)$ time.

The above strategy actually has a deeper implication regarding any top-k problem on weight-augmented datasets. Suppose that there is a structure of $\mathcal{S}_{pri}(n)$ space that can answer the corresponding prioritized query in $\mathcal{Q}_{pri}(n) + O(t)$ time (where t is the number of el-

[2]The operation finding such a τ is known as the *range quantile query* [17]

ements reported). Then, there must exist a top-k index of $O(\mathcal{S}_{pri}(n))$ space that answers a top-k query in $O(\mathcal{Q}_{pri}(n) \cdot \log n + k \log n)$ time. In Section 6, we will see a stronger result that has a better query bound, and eliminates the requirement of $|\mathbb{W}| = 1$.

5.3 Technique 2: Resorting to Counting

The query efficiency of the strategy in Section 5.2 can usually be improved, provided that there is a specialized structure for finding the number c faster. This is indeed the case for 1D range searching. Recall that c equals the number of points $e \in D$ satisfying $x \leq e \leq y$ and $w(e) \geq z$. If we introduce a 2D dataset $P = \{(e, w(e)) \mid e \in D\}$, c equals precisely the number of points in P that are covered by the rectangle $[x, y] \times [z, \infty)$. Computing this number is known as *orthogonal range counting*, which has been very well understood. We can preprocess P into a structure of Chazelle [10] which uses $O(n)$ space, and finds $|P \cap r|$ for any axis-parallel rectangle r in $O(\log n)$ time. With this, the query time of the solution in Section 5.2 is improved to $O(\log^2 n + k)$.

For a general top-k problem, a counting structure for finding c efficiently may not be readily available. The merit of the technique in Section 5.2 is to assure a reasonably good bound on the query cost using *only* a prioritized structure, which *must* be available for the reason explained in Section 3.

5.4 Technique 3: Dyadic Intervals

Assume, for simplicity, that n is a power of 2, and set $\Delta = \log_2 n$. Let us partition $\mathbb{R}$ into n/Δ disjoint intervals — referred to as *slabs* henceforth — such that each slab has exactly Δ points. Given an arbitrary interval $[x, y]$, we call it *aligned* if x and y are both slab boundaries, and define its *span* as the number of slabs that are fully contained in $[x, y]$. A *dyadic interval* is an aligned interval whose span is a power of 2. Note that the total number of dyadic intervals is $O((n/\Delta) \log n)$.

LEMMA 2. *For any aligned interval q, there exist two possibly overlapping dyadic intervals I_1 and I_2 that satisfy $I_1 \cup I_2 = q$.*

The proof is simple and omitted from this article.

Structure. For every dyadic interval I, store a *sketch* which consists of the 2^i-th largest weight of the points in $D \cap I$, for each $i \in [0, \log_2 n]$. If $|D \cap I| < 2^i$, then the 2^i-th largest weight is defined to be $-\infty$. All the $O((n/\Delta) \log n)$ sketches constitute our structure, whose space is $O((n/\Delta) \log^2 n)$.

Query. Given a top-k query with interval $q = [x, y]$, we now explain how to find a value τ that satisfies the requirements in Step 2 of the algorithm in Section 5.1. Assume, without loss of generality, that k is a power of 2 (otherwise, bump k up to the nearest power of 2). The base case happens when q is completely *within* a certain slab σ. In that case, we retrieve the set S' of points in $q \cap \sigma$, which takes $O(\log n + \Delta)$ time by searching a BST on D. Then, τ can be simply set to the k-th largest weight of the points in S', which can be found by performing k-selection in $O(\Delta)$ time.

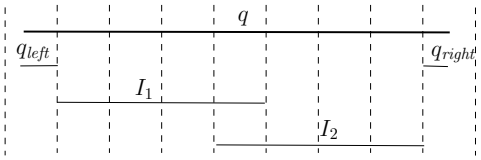

Figure 1: Partitioning a query interval

Let us now suppose that q intersects at least two slabs. Define $q_{mid} = [x', y']$ to be the longest *aligned* interval inside q, which gives rise to $q_{left} = [x, x']$ and $q_{right} = [y', y]$. Lemma 2 guarantees the existence of dyadic intervals I_1 and I_2 such that $I_1 \cup I_2 = q_{mid}$. See Figure 1 for an illustration, where the dashed lines represent slab boundaries.

Next, we obtain four values:

- τ_{left}: the k-th largest weight in $D \cap q_{left}$, or $-\infty$ if $|D \cap q_{left}| < k$;
- τ_1 (or τ_2): the k-th largest weight of the points in $D \cap I_1$ (or $D \cap I_2$, resp.);
- τ_{right}: the k-th largest weight in $D \cap q_{right}$, or $-\infty$ if $|D \cap q_{right}| < k$.

The values τ_{left} and τ_{right} can be obtained in $O(\Delta)$ time using the strategy illustrated earlier for the base case, while τ_1 and τ_2 can be fetched directly from the sketches of I_1 and I_2.

The τ returned is the *maximum* of the four values. It is easy to prove that at least k but at most $4k$ points in $q(D)$ can have weights at least τ.

Remark. Setting $\Delta = \log_2^2 n$ yields a linear space structure with $O(\log^2 n)$ query time, while $\Delta = \log_2 n$ gives a structure of $O(n \log n)$ space but $O(\log n)$ query time. It is possible to achieve linear space and $O(\log n)$ query time by recursively applying the same idea in each slab, but we will not delve into those details because competing for efficiency is not the purpose of this section.

5.5 Technique 4: Heap Selection

Let us define a *max-heap* H to be a tree where

- each internal node has a constant number of children, and
- each node u stores a real-valued *key* that is greater than all the keys stored in the proper subtree of u.

Note that H does not need to be balanced in any way. Given any max-heap H, Frederikson [16] described an algorithm to extract the k largest keys from H in $O(k)$ time, for any k ranging from 1 to the number of elements in H.

The above algorithm is useful for designing top-k indexes. Given a top-k query with predicate q, let us imagine that $q(D)$ has been divided into a number of sets $S_1, ..., S_s$ for some $s \geq 1$, and that the elements in each S_i ($1 \leq i \leq s$) have been stored in a max-heap H_i, using their weights as the keys. In such a scenario, we can find the top-k result in $O(s + k)$ time as follows. First coalesce $H_1, ..., H_s$ into a single max heap H on $S_1 \cup ... \cup S_s$. This can be done in $O(s)$ time, noticing that we only need to take the root of each H_i, and build a max-heap on those s roots. Once this is done, Frederikson's algorithm can be directly applied to find the k elements with the largest weights from H in $O(k)$ time.

Next, we apply the above idea to obtain an elegant top-k index for 1D range searching that consumes $O(n)$ space, guarantees $O(\log n + k)$ query time, and can be updated in $O(\log n)$ time per insertion and deletion.

Structure. Let us consider once again the set of n 2D points $P = \{(e, w(e) \mid e \in D\}$ constructed from D. It suffices to build a PST $\mathcal{T}$ on P, which can be defined recursively as follows (this is the first time in this article that we need to be concerned with the details of a PST):

- If $P = \emptyset$, $\mathcal{T}$ is an empty tree.
- If P contains only a single point $p = (p_x, p_y)$, $\mathcal{T}$ has only one node u which stores p, an *x-key* equal to p_x, and a *y-key* equal to p_y.
- Otherwise, let x_{med} be the median of the x-coordinates of the points in P, and $p^* = (p_x^*, p_y^*)$ be the highest point in P, i.e., having the greatest y-coordinate. Create the root u of $\mathcal{T}$, which stores p^*, an *x-key* equal to x_{med}, and a *y-key* equal to p_y^*.

  Define P_1 to be the set of points $p = (p_x, p_y)$ in $P \setminus \{p^*\}$ satisfying $p_x < x_{med}$, and P_2 symmetrically to be the set of points $p = (p_x, p_y)$ in $P \setminus \{p^*\}$ satisfying $p_x \geq x_{med}$. Recursively construct BSTs $\mathcal{T}_1$ and $\mathcal{T}_2$ on P_1 and P_2, respectively. Then, $\mathcal{T}$ is the tree obtained by making the root of $\mathcal{T}_1$ the left child of u, and that of $\mathcal{T}_2$ the right child of u.

Observe that $\mathcal{T}$ is a BST on the x-keys of the nodes, and simultaneously also a max-heap on the y-keys. It is clear that $\mathcal{T}$ has height $O(\log n)$, and occupies $O(n)$ space.

Query. Suppose that we are given a top-k (1D range searching) query with the search interval $q = [x_1, x_2]$. Without loss of generality, let us assume that both x_1 and x_2 are x-keys in $\mathcal{T}$. Denote by Π_1 (or Π_2) the path from the root of $\mathcal{T}$ to the node with x-key x_1 (or x_2, resp.).

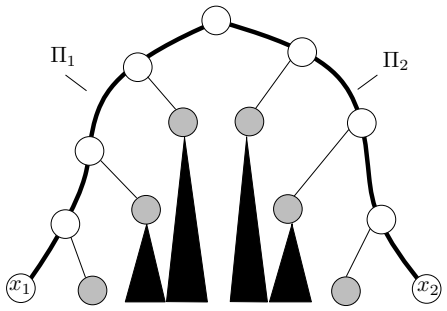

Figure 2: Searching a PST to perform top-k range searching in 1D space

From Π_1 and Π_2, we can obtain $s = O(\log n)$ nodes $v_1, ..., v_s$ with three properties:

- *Property 1*: The parent of each v_i ($1 \leq i \leq s$) is on Π_1 or Π_2.
- *Property 2*: The subtrees of $v_1, ..., v_s$, which are called the *canonical subtrees*, are mutually disjoint.
- *Property 3*: Every node, whose x-key is contained in q, must be either on $\Pi_1 \cup \Pi_2$ or in a canonical subtree.

Figure 2 shows an example where $s = 6$, and $v_1, ..., v_6$ are the nodes colored in gray.

Define $q(P)$ as the set of points $p = (p_x, p_y) \in P$ such that $x_1 \leq p_x \leq x_2$. Answering the top-k query is equivalent to finding the k highest points in $q(P)$. Property 3 ensures that every point in $q(P)$ must be stored at a node on $\Pi_1 \cup \Pi_2$, or a node in one of the canonical subtrees. Let us divide $q(P)$ into (i) P_1 (or P_2), which is the set of points in $q(P)$ stored on Π_1 (or P_2, resp.), and (ii) P_3, which the set of points stored in the canonical subtrees.

Let S_i ($1 \leq i \leq s$) be the set of y-keys in the canonical subtree rooted at v_i. Note that the canonical subtree is a max-heap on S_i. The k largest y-keys in $S_1 \cup ... \cup S_s$ can therefore be extracted in $O(k)$ time using Frederikson's algorithm. The points corresponding to those y-keys constitute the set S of k highest points in P_3. The final result of the top-k query is the k highest points in $S \cup P_1 \cup P_2$, which can be found using k-selection in $O(|S \cup P_1 \cup P_2|) = O(\log n + k)$ time.

Update. In [21], McCreight described a slightly different PST by allowing x_{med} to be an "approximate median". The benefit is that the resulting PST also supports an update in $O(\log n)$ time. The same query algorithm applies to that PST as well.

6. PRIORITIZED AS HARD AS TOP-K?

We now turn our attention back to all the top-k problems capured by our formulation in Section 6, i.e., no

matter whether $|\mathbb{W}| = 1$. We already know from Theorem 1 that top-k queries are no easier than prioritized queries, that is, a top-k index implies a prioritized structure with the same space-query tradeoff.

In this section, we will discuss the question opposite to the one resolved by Theorem 1. Let us fix D, $\mathbb{Q}$, and $\mathbb{W}$. Suppose that there is a structure on D that uses $\mathcal{S}_{pri}(n)$ space (recall that $n = |D|$), and answers any prioritized query in $\mathcal{Q}_{pri}(n) + O(t)$ time (where t is the number of elements reported). We want to use the structure as a black box to design a top-k structure. Let $\mathcal{S}_{top}(n)$ be the space consumption of the top-k structure, and $\mathcal{Q}_{top}(n) + O(k)$ its query cost. How good can the functions $\mathcal{S}_{top}(n)$ and $\mathcal{Q}_{top}(n)$ be?

Ideally, we would like to show $\mathcal{S}_{top}(n) = O(\mathcal{S}_{pri}(n))$ and $\mathcal{Q}_{top}(n) = O(\mathcal{Q}_{pri}(n))$. This would imply that the top-k query was no *harder* than the corresponding prioritized query which, in turn, would conclude that top-k and prioritized queries in fact had the same computational hardness! Unfortunately, whether this is true still remains elusive today.

Nevertheless, decent progress has been made towards settling this open question. We now know that, when $\mathcal{Q}_{pri}(n) = \Omega(n^\epsilon)$ for any constant $\epsilon > 0$, it indeed holds that $\mathcal{S}_{top}(n) = O(\mathcal{S}_{pri}(n))$ and $\mathcal{Q}_{top}(n) = O(\mathcal{Q}_{pri}(n))$. In other words, for *hard* problems whose prioritized queries demand a polynomial $\mathcal{Q}_{pri}(n)$, we can turn a prioritized structure into a top-k index with no efficiency loss! For easier problems with $\mathcal{Q}_{pri}(n) = \Omega(\log n)$, on the other hand, it is possible to show $\mathcal{S}_{top}(n) = O(\mathcal{S}_{pri}(n))$ and $\mathcal{Q}_{top}(n) = O(\mathcal{Q}_{pri}(n) \cdot \log n)$. In other words, we can still obtain a top-k index from a prioritized structure by entailing only an $O(\log n)$ deterioration factor in query time.

Next, we introduce the theorem behind the above claims. Let us start with the notion of *polynomial boundedness*. Fix an integer $k \in [1, n]$. Remember that a top-k query selects a predicate $q \in \mathbb{Q}$ and a weight function $w \in \mathbb{W}$. In other words, the query result is a function of q and w. Since $|\mathbb{Q}|$ or $|\mathbb{W}|$ may be very large, the number of possible queries can be unbounded, but even so, it is possible that the number of *distinct* top-k results may be much smaller. In particular, if that number is always bounded by $n^{O(1)}$ for any input dataset $D \subseteq \mathbb{D}$ of size n (regardless of k), we say that the triplet $(\mathbb{D}, \mathbb{Q}, \mathbb{W})$ is *polynomially bounded*.

Rahul and Tao established[3] the following in [27]:

THEOREM 3. *Fix a polynomially bounded triplet of $(\mathbb{D}, \mathbb{Q}, \mathbb{W})$, and a set $D \subseteq \mathbb{D}$ of size n. Suppose that there is a structure on D that uses $\mathcal{S}_{pri}(n)$ space, and answers a prioritized query in $\mathcal{Q}_{pri}(n) + O(t)$ time, where t is the number of reported elements, such that*

- *$\mathcal{S}_{pri}(n)$ is geometrically converging, and*
- *$\mathcal{Q}_{pri}(n) = \Omega(\log n)$.*

Then, there is a top-k index of space $\mathcal{S}_{top}(n)$ and query time $\mathcal{Q}_{top}(n) + O(k)$ with

$$\begin{aligned} \mathcal{S}_{top}(n) &= O(\mathcal{S}_{pri}(n)) \\ \mathcal{Q}_{top}(n) &= O\left(\mathcal{Q}_{pri}(n) \cdot \frac{\log n}{\log \frac{\mathcal{Q}_{pri}(n)}{\log n}}\right). \end{aligned}$$

The proof in [27], which is technically involved and omitted from this article, shows how to construct a top-k index in the theorem using $n^{O(1)}$ expected time, where the constant power depends on the underlying problem.

Note that all complexities in Theorem 3 hold in the worst case. For $\mathcal{Q}_{pri}(n) = \omega(\log n)$, $\mathcal{Q}_{top}(n)$ is actually $o(\mathcal{Q}_{pri}(n) \cdot \log n)$, namely, the deterioration factor with respect to $\mathcal{Q}_{pri}(n)$ is $o(\log n)$. As an example, if $\mathcal{Q}_{pri}(n) = \Omega(\log^{1+\epsilon} n)$ for any positive constant $\epsilon > 0$, it holds that $\mathcal{Q}_{top}(n) = O(\mathcal{Q}_{pri}(n) \cdot \frac{\log n}{\log \log n})$. For $\mathcal{Q}_{pri}(n) = \Omega(n^\epsilon)$, the deterioration factor is $O(1)$, as mentioned earlier.

Polynomial boundedness is a property of many top-k problems. One example is the linear ranking problem defined in Section 2.2 under any constant dimensionality d. As explained in Section 4.2, the corresponding prioritized query of this problem is known as *halfspace reporting*. For $d \geq 4$, Afshani and Chan [2] described a structure of $O(n)$ space that answers any halfspace reporting query in $\tilde{O}(n^{1-1/\lfloor d/2 \rfloor}) + O(t)$ time where t is the number of points reported. Immediately, Theorem 3 guarantees a top-k index of $O(n)$ space that answers a top-k query in $\tilde{O}(n^{1-1/\lfloor d/2 \rfloor}) + O(k)$ time.

7. BEYOND THIS ARTICLE

We have reviewed only a small portion of the existing work on the class of top-k problems formulated in Section 2.1. Efficient indexes have been developed for the top-k versions of many traditional reporting problems, e.g., orthogonal range reporting [1, 7, 8, 25, 26, 30, 31], halfspace reporting [25, 27], rectangle stabbing [9, 27], and so on. The design of those indexes harbors numerous inspiring ideas which unfortunately cannot be included in this article.

Another non-trivial direction that has received significant development is the theory of top-k indexes in the *external memory* (EM) model [1,7,26,27,30,31]. Closely relevant to database systems, this model is widely used to study the behavior of *I/O-oriented* algorithms, whose performance bottlenecks lie in the data exchanges between different levels of the memory hierarchy — e.g.,

[3]Strictly speaking, [27] proved the theorem only for problems with $|\mathbb{W}| = 1$. However, the proof can be adapted to cover all polynomially bounded problems under the formulation in Section 2.1.

between the main memory and the disk — rather than in CPU computation. Specifically, in the EM model, a machine is equipped with M words of memory, and a disk that has been formatted into *blocks* of B words each. An *I/O* either reads a disk block into memory, or writes B words of memory into a disk block. CPU operations can be performed only on the data in memory. The *time* of an algorithm is measured in the number of I/Os performed (CPU computation is for free), while the *space* of a structure is measured in the number of disk blocks occupied. A "good" top-k index on an input dataset of size n should consume $\tilde{O}((n/B))$ space, and answer a query in $\mathcal{Q}_{top}(n, B) + O(k/B)$ I/Os, where $\mathcal{Q}_{top}(n, B)$ is a slow-growing function of n and B. Many of the techniques discussed in this article can be adapted to work in EM. In particular, see [26] for the counterpart of Theorem 1, and [27] for the counterparts of Theorems 2 and 3.

Finally, it is worth pointing out that the theory community has studied other top-1 or top-k problems that do not fit directly into the formulation in Section 2.1, e.g., problems on text retrieval [5, 18, 20, 22, 23, 29], uncertain data [3, 4, 32], colored reporting [28], etc.

8. FUTURE WORK DIRECTIONS

We conclude this article by mentioning four directions for future research:

- **Direction 1:** *Resolve the conjecture that the prioritized query is as hard as the corresponding top-k query.* Currently, there is an $O(\log n)$ gap in the query cost between the two (see Theorem 3). If this gap could be closed, we would have the surprising fact that every top-k problem in the class formulated in Section 2.1 is essentially the same as its prioritized version in terms of space-query tradeoff.
- **Direction 2:** *Obtain a high-probability version of Theorem 2.* The guarantees in that theorem currently hold in expectation only. Can we make them hold with a high probability (e.g., at least $1 - 1/n^2$)?
- **Direction 3:** *Fast construction of a top-k index in Theorem 3.* As mentioned in Section 6, currently it takes $n^{O(1)}$ expected time to build a top-k index with the guarantees stated in the theorem, which limits the theorem's applicability in practice. Can we reduce the cost to $\tilde{O}(n)$, provided that the given prioritized structure can be built in $\tilde{O}(n)$ time?
- **Direction 4:** *Study individual top-k problems with significant importance in practice.* The top-k perspective in Section 2.1 offers motivation for studying prioritized queries some of which otherwise would not appear sufficiently important to justify serious research efforts. On good example is *top-k halfspace reporting*, whose prioritized query is the following problem. We are given a set D of points in $\mathbb{R}^d$, each of which is associated with a real-valued weight. Given a halfspace q in $\mathbb{R}^d$ and a real value τ, a query returns all the points in $D \cap q$ whose weights are at least τ. The challenge is to preprocess D into a structure that can answer any such query efficiently. As a particularly interesting question, for $d = 2$, can we obtain a structure of $O(n)$ space that answers a query in $O(\log n + t)$ time, where t is the number of points reported?

9. REFERENCES

[1] Peyman Afshani, Gerth Stolting Brodal, and Norbert Zeh. Ordered and unordered top-k range reporting in large data sets. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 390–400, 2011.

[2] Peyman Afshani and Timothy M. Chan. Optimal halfspace range reporting in three dimensions. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 180–186, 2009.

[3] Pankaj K. Agarwal, Boris Aronov, Sariel Har-Peled, Jeff M. Phillips, Ke Yi, and Wuzhou Zhang. Nearest-neighbor searching under uncertainty II. *ACM Transactions on Algorithms*, 13(1):3:1–3:25, 2016.

[4] Pankaj K. Agarwal, Nirman Kumar, Stavros Sintos, and Subhash Suri. Range-max queries on uncertain data. *Journal of Computer and System Sciences (JCSS)*, 94:118–134, 2018.

[5] Iwona Bialynicka-Birula and Roberto Grossi. Rank-sensitive data structures. In *String Processing and Information Retrieval (SPIRE)*, pages 79–90, 2005.

[6] Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Time bounds for selection. *Journal of Computer and System Sciences (JCSS)*, 7(4):448–461, 1973.

[7] Gerth Stolting Brodal. External memory three-sided range reporting and top-k queries with sublogarithmic updates. In *Proceedings of Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 23:1–23:14, 2016.

[8] Gerth Stolting Brodal, Rolf Fagerberg, Mark Greve, and Alejandro Lopez-Ortiz. Online sorted range reporting. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 173–182, 2009.

[9] Timothy Chan, Yakov Nekrich, Saladi Rahul, and Konstantinos Tsakalidis. Orthogonal point location

and rectangle stabbing queries in 3-d. In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, pages 31:1–31:14, 2018.

[10] Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal of Computing*, 17(3):427–462, 1988.

[11] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.

[12] David P. Dobkin and David G. Kirkpatrick. A linear algorithm for determining the separation of convex polyhedra. *J. Algorithms*, 6(3):381–392, 1985.

[13] Jeff Erickson. Better lower bounds for halfspace emptiness. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 472–481, 1996.

[14] Ronald Fagin. Combining fuzzy information from multiple systems. *Journal of Computer and System Sciences (JCSS)*, 58(1):83–99, 1999.

[15] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[16] Greg N. Frederickson. An optimal algorithm for selection in a min-heap. *Information and Computation*, 104(2):197–214, 1993.

[17] Travis Gagie, Simon J. Puglisi, and Andrew Turpin. Range quantile queries: Another virtue of wavelet trees. In *String Processing and Information Retrieval (SPIRE)*, pages 1–6, 2009.

[18] Wing-Kai Hon, Rahul Shah, Sharma V. Thankachan, and Jeffrey Scott Vitter. Space-efficient frameworks for top-k string retrieval. *Journal of the ACM (JACM)*, 61(2):9:1–9:36, 2014.

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

[20] Marek Karpinski and Yakov Nekrich. Top-k color queries for document retrieval. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 401–411, 2011.

[21] Edward M. McCreight. Priority search trees. *SIAM Journal of Computing*, 14(2):257–276, 1985.

[22] S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 657–666, 2002.

[23] Gonzalo Navarro and Yakov Nekrich. Time-optimal top-k document retrieval. *SIAM Journal of Computing*, 46(1):80–113, 2017.

[24] Manish Patil, Sharma V. Thankachan, Rahul Shah, Yakov Nekrich, and Jeffrey Scott Vitter. Categorical range maxima queries. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 266–277, 2014.

[25] Saladi Rahul and Ravi Janardan. A general technique for top-k geometric intersection query problems. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 26(12):2859–2871, 2014.

[26] Saladi Rahul and Yufei Tao. On top-k range reporting in 2d space. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 265–275, 2015.

[27] Saladi Rahul and Yufei Tao. Efficient top-k indexing via general reductions. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 277–288, 2016.

[28] Biswajit Sanyal, Prosenjit Gupta, and Subhashis Majumder. Colored top-k range-aggregate queries. *Information Processing Letters (IPL)*, 113(19-21):777–784, 2013.

[29] Rahul Shah, Cheng Sheng, Sharma V. Thankachan, and Jeffrey Scott Vitter. Top-k document retrieval in external memory. In *Proceedings of European Symposium on Algorithms (ESA)*, pages 803–814, 2013.

[30] Cheng Sheng and Yufei Tao. Dynamic top-k range reporting in external memory. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 121–130, 2012.

[31] Yufei Tao. A dynamic I/O-efficient structure for one-dimensional top-k range reporting. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 256–265, 2014.

[32] Ke Yi, Feifei Li, George Kollios, and Divesh Srivastava. Efficient processing of top-k queries in uncertain databases with x-relations. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 20(12):1669–1682, 2008.