How to Select the Top k Elements from Evolving Data?

Qin Huang[1], Xingwu Liu[1,2(✉)], Xiaoming Sun[1], and Jialin Zhang[1]

[1] Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
{huangqin,liuxingwu,sunxiaoming,zhangjialin}@ict.ac.cn
[2] State Key Laboratory of Software Development Environment, Beihang University, Beijing, China

Abstract. In this paper we investigate the top-k-selection problem, i.e. to determine and sort the top k elements, in the dynamic data model. Here dynamic means that the underlying total order evolves over time, and that the order can only be probed by pair-wise comparisons. It is assumed that at each time step, only one pair of elements can be compared. This assumption of restricted access is reasonable in the dynamic model, especially for massive data set where it is impossible to access all the data before the next change occurs. Previously only two special cases were studied [1] in this model: selecting the element of a given rank, and sorting all elements. This paper systematically deals with $k \in [n]$. Specifically, we identify the critical point k^* such that the top-k-selection problem can be solved error-free with probability $1 - o(1)$ if and only if $k = o(k^*)$. A lower bound of the error when $k = \Omega(k^*)$ is also determined, which actually is tight under some conditions. In contrast, we show that the top-k-set problem, which means finding the top k elements without sorting them, can be solved error-free with probability $1 - o(1)$ for all $1 \leq k \leq n$. Additionally, we consider some extensions of the dynamic data model and show that most of these results still hold.

1 Introduction

Sorting, a fundamental primitive in algorithms, has been an active research topic in computer science for decades. In the era of big data, it is the cornerstone of numerous vital applications – Web search, online ads, and recommendation systems to name but a few. While sorting has been extensively studied, little is known when the data is dynamic. Actually, dynamic data is common in practical applications: the linking topology of Web pages, the friendship network of Facebook, the daily sales of Amazon, and so on, all keep changing. The basic challenge in dealing with dynamic, massive data is that the access to the data is too restricted to catch the changes.

The work is partially supported by National Natural Science Foundation of China (61173009, 61170062, 61222202, 61433014, 61502449), State Key Laboratory of Software Development Environment Open Fund (SKLSDE-2014KF-01), and the China National Program for support of Top-notch Young Professionals.

K. Elbassioni and K. Makino (Eds.): ISAAC 2015, LNCS 9472, pp. 60–70, 2015.
DOI: 10.1007/978-3-662-48971-0_6

For example, it is impossible to get an exact snapshot of Web, and a third-party vendor can query the Facebook network only via a rate-limited API. As a result, this paper is devoted to studying the sorting problem on dynamic, access-restricted data.

In the seminal paper [1], Anagnostopoulos et al. formulated a model for dynamic data as follows. Given a set U of n elements, at every discrete time t, there is an underlying total order π^t on U. For every $t \geq 1$, π^t is obtained from π^{t-1} by sequentially swapping α random pairs of consecutive elements, where α is a constant number. The only way to probe π^t is querying the relative rank of ONE pair of elements in U at every time step. The goal is to learn about the true order π^t. Obviously, it is impossible to always exactly find out the orders, so our objective is that at any time t, the algorithm estimates the correct answer (or an approximate answer) with high probability. In this paper, "with high probability" and "with probability $1 - o(1)$" are used interchangeably.

Anagnostopoulos et al. [1] proved that the Kendall tau distance between π^t and $\tilde{\pi}^t$, defined in Sect. 2 and denoted by $\mathrm{KT}(\pi^t, \tilde{\pi}^t)$, is lower-bounded by $\Omega(n)$ with high probability at every t, where $\tilde{\pi}^t$ is the order estimated by any algorithm. This lower bound is nearly tight, since they proposed an algorithm with $\mathrm{KT}(\pi^t, \tilde{\pi}^t) = O(n \ln \ln n)$. Furthermore, they designed an algorithm that with high probability, exactly identifies the element of a given rank.

Though elegant, this model is too restricted: the evolution is extremely slow since α is constant, and is extremely local since only consecutive elements are swapped. Hence, it is extended in this paper by allowing α to be a function of n, and is called the consecutive-swapping model. We further generalize it to the Gaussian-swapping model by relaxing the locality condition.

Inspired by [1], we study the general top-k-selection problem: at every time t, figure out the top k elements and sort them, where $k \in \{1, 2, ...n\}$. Its two extreme cases where $k = n$ and $k = 1$ correspond to the sorting problem and the selection problem in [1], respectively. The error-free solvability of the selection problem suggests that the error in solving the top-k-selection problem may vanish as k decreases, so it is natural to investigate the critical point where the error vanishes and to find the optimal solution beyond the critical point. Another motivation lies in the wide application of top-k-selection, also known as partial sorting. It has been used in a variety of areas such as Web and multimedia search systems and distributed systems, where massive data has to be dealt with efficiently [2].

Additionally, we consider a closely related top-k-set problem: at every time t, identify the set of the top k elements. The top-k-set problem is weaker in that it does not require to sort the elements. In the static data setting, when a selection algorithm identifies the kth element, it automatically determines the set of the top k elements (see for example Knuth's book [3]). However, this is not apparent in the dynamic data model.

Our Contributions. The main results of this paper lie in two aspects in the consecutive-swapping model. First, it is shown that the top-k-set problem can be solved error-free with high probability for any $1 \leq k \leq n$. Second and more

important, $k^* = \Theta(\sqrt{\frac{n}{\alpha}})$ is proven to be the critical point of k for the top-k-selection problem, which means that this problem can be solved error-free with high probability if and only if $k = o(k^*)$.

In addition, for k beyond k^*, we obtain tight lower bounds of $\mathrm{KT}(\tilde{\pi}_k^t, \pi_k^t)$, the Kendall tau distance between the true order π_k^t and the algorithmically estimated order $\tilde{\pi}_k^t$ of the top k elements. Specifically, if $k = \Omega(\sqrt{\frac{n}{\alpha}})$, then for any algorithm, $\mathrm{KT}(\tilde{\pi}_k^t, \pi_k^t) \neq 0$ with constant probability. When $k = \omega(\sqrt{n})$ and $\alpha = O(1)$, for any algorithm, $\mathrm{KT}(\tilde{\pi}_k^t, \pi_k^t) = \Omega(\frac{k^2}{n})$ with high probability at every t. These lower bounds can be reached by ONE algorithm with parameter k, (see Algorithm 2), hence being tight.

The results of the top-k-selection problem in the consecutive-swapping model are summarized in Table 1. Most of the results are also generalized to the Gaussian-swapping model with constant α, as summarized in Table 2.

Table 1. Results in the consecutive-swapping model

k	$X \triangleq \mathrm{KT}(\tilde{\pi}_k^t, \pi_k^t)$
$o(\sqrt{\frac{n}{\alpha}})$	$\Pr(X = 0) = 1 - o(1)$
$\Theta(\sqrt{\frac{n}{\alpha}})$	$\Pr(X = 0) = \Theta(1) = \Pr(X > 0)$
$\omega(\sqrt{\frac{n}{\alpha}})$	$\Pr(X = O(\frac{k^2\alpha}{n})) = 1 - o(1)$[a]

In [a] case, this upper bound of X is tight for constant α. See Sect. 3

Table 2. Results in the Gaussian-swapping model

k	$X \triangleq \mathrm{KT}(\tilde{\pi}_k^t, \pi_k^t)$
$o(\frac{\sqrt{n}}{\ln^{0.25} n})$	$\Pr(X = 0) = 1 - o(1)$
$\Theta(\frac{\sqrt{n}}{\ln^{0.25} n})$	$\Pr(X = 0) = \Theta(1)$
$\omega(\frac{\sqrt{n}}{\ln^{0.25} n})$	$\Pr(X = O(\frac{k^2 \ln n}{n})) = 1 - o(1)$

Related Work. The sorting/selection problem has been actively investigated for decades [2,4–6], but the study of this problem in dynamic data setting was initiated very recently [1]. In [1], Anagnostopoulos et al. considered two special cases of the top-k-selection problem, namely $k = n$ and $k = 1$, in the consecutive-swapping model with constant α. Their work has inspired the problem and the data model in this paper. The theoretical results in [1] were experimentally verified by Moreland [7] in 2014.

Dynamic data is also studied in the graph setting. [8] considered two classical graph connectivity problems (path connectivity and minimum spanning trees) where the graph keeps changing over time and the algorithm, unaware of the changes, probes the graph to maintain a path or spanning tree. Bahmani et al. [9] designed an algorithm to approximately compute the PageRank of

evolving graphs, and Zhuang et al. [10] considered the influence maximization problem in dynamic social networks. On the other hand, Labouseur et al. [11] and Ren [12] dealt with the data structure and management issues, respectively, enabling efficient query processing for dynamic graphs.

It is worth noting that our dynamic data model is essentially different from noisy information model [13,14]. In computing with noisy information, the main difficulty is brought about by misleading information. On the contrary, in our model, the query results are correct, while the difficulty comes from the restricted access to the dynamic data. The ground truth can be probed only by local observation, so it is impossible to capture all changes in the data. The key issue is to choose query strategies in order to approximate the real data with high probability.

In the algorithm community, there are many other models dealing with dynamic and uncertain data, from various points of view. However, none of them captures the two crucial aspects of our dynamic data model: the underlying data keeps changing, and the data exposes limited information to the algorithm by probing. For example, data stream algorithms [15] deal with a stream of data, typically with limited space, but the algorithms can observe the entire data that has arrived; local algorithms on graphs [16,17] probe the underlying graphs by a limited number of query, but typically the graphs are static; in online algorithms [18], though the data comes over time and is processed without knowledge of the future data, the algorithms know all the data up to now; the multi-armed-bandit model [19] tends to optimize the total gain in a finite exploration-exploitation process, while our framework concerns the performance of the algorithm at every time step in an infinite process.

The rest of the paper is organized as follows. In Sect. 2, we provide the formal definition of the models and formulate the problems. Section 3 is devoted to solving the top-k-set problem and the top-k-selection problem in the consecutive-swapping model. In Sect. 4, the problems are studied in the Gaussian-swapping model. Section 5 concludes the paper. Due to the limitation of space, all proofs of the theorems will be omitted.

2 Preliminaries

We now formalize our dynamic data model.

Let $U = \{u_1, ..., u_n\}$ be a set with n elements, and $\mathcal{U}$ be the set of all total orders over U, that is, $\mathcal{U} = \{\pi : U \to [n] \mid \forall i \neq j, \pi(u_i) \neq \pi(u_j)\}$, where $[n] \triangleq \{1, 2, ...n\}$. For any $\pi \in \mathcal{U}$ and $k \in [n]$, we define $\pi^{-1}(k)$ to be the kth element and $\pi(u)$ to be the rank of u relative to π. If $\pi(u) < \pi(v)$, we say $u >_\pi v$ or simply by $u > v$ when π can be inferred from context.

In this paper, we consider the process where the order on U gradually changes over time. Time is discretized into steps sequentially numbered by nonnegative integers. At every time step t, there is an underlying total order π^t on U. For every $t \geq 1$, π^t is obtained from π^{t-1} by sequentially swapping α random pairs of

consecutive elements, where α is an integer function of n. This is our consecutive-swapping model.

Now we introduce the Gaussian-swapping model whose defining feature is that non-consecutive pairs can be swapped in the evolution. Specifically, for every $t \geq 1$, π^t is still obtained from π^{t-1} by sequentially swapping α pairs of elements. However, each pair (not necessarily consecutive) is selected as follows, rather than uniformly randomly. First, d is sampled from a truncated Gaussian distribution $\Pr(D = d) = \beta e^{\frac{-d^2}{2}}$ where β is the normalizing factor. Then, a pair of elements whose ranks differ by d is chosen uniformly randomly from all such pairs. Thus, the overall probability that a pair (u, v) gets swapped is $\frac{\beta e^{\frac{-d^2}{2}}}{n-d}$, where d is the difference between the ranks of u and v, related to π^{t-1}.

In either model, at any time step t, the changes of π^t are unknown by the algorithms running on the data. The only way to probe the underlying order is by comparative queries. At any time t, given an arbitrary pair of elements $u, v \in U$, an algorithm can query whether $\pi^t(u) > \pi^t(v)$ or not. At most *one* pair of elements can be queried at each time step.

Now we define $\mathcal{I}$-sorting problem for any index set $\mathcal{I} \subseteq [n]$: at each time step t, find out all the elements whose ranks belong to $\mathcal{I}$, and sort them according to π^t. The concept of $\mathcal{I}$-sorting problem unifies both the sorting problem ($|\mathcal{I}| = n$) and the selection problem ($|\mathcal{I}| = 1$). This paper mainly studies the top-k-selection problem, a special case of the $\mathcal{I}$-sorting problem with $\mathcal{I} = [k]$ for $k \in [n]$. For convenience, in this paper we use notation π_k^t to represent the true order on the top k elements at time t. A closely-related problem, called the top-k-set problem, is also studied. It requires to find out $(\pi^t)^{-1}([k])$ at each time t, without sorting them.

We then define the performance metrics of the algorithms. In the top-k-set problem, we want to maximize the probability that the output set is exactly the same as the true set for sufficiently large t. In the top-k-selection problem, we try to minimize the Kendall tau distance between the output order and the true order on the top k elements, for sufficiently large t. Since an algorithm solving the top-k-selection problem may output an order on a wrong set, we extend the definition of Kendall tau distance to orders on different sets. Specifically, given total orders σ on set V and δ on set W with $|V| = |W|$, their Kendall tau distance is defined to be $\mathrm{KT}(\sigma, \delta) = |\{(x, y) \in V^2 : \sigma(x) < \sigma(y) \text{ and } (x \notin W \text{ or } y \notin W \text{ or } \delta(x) > \delta(y))\}|$. Intuitively, it is the number of pairs that either are not shared by W and V or are ordered inconsistently by the two total orders.

Throughout this paper, one building block of the algorithms is the randomized quick-sort algorithm. We describe the randomized quick-sort algorithm briefly. Given an array, it works as follows: (1) Uniformly randomly pick an element, called a pivot, from the array. (2) Compare all elements with the pivot, resulting in two sub-arrays: one consisting of all the elements smaller than the pivot, and the other consisting of the other elements except the pivot. (3) Recursively apply steps 1 and 2 to the two sub-arrays until all the sub-arrays are singletons.

3 Consecutive-Swapping Model

In this section, we consider the top-k-set problem and the top-k-selection problem in the consecutive-swapping model. For the top-k-set problem, Sect. 3.1 shows an algorithm which is error-free with probability $1 - o(1)$ for arbitrary k. Section 3.2 is devoted to the top-k-selection problem. It presents an algorithm that is optimal when α is constant or k is small.

3.1 An Algorithm for the Top-k-set Problem

The basic idea is to repeatedly run quick-sort over the data U, extract the set of the top k elements from the resulting order, and output this set during the next run. But an issue should be addressed: since the running time of quick-sort is $\Omega(n \ln n)$ with high probability, the set of the top k elements will change with high probability during the next run, leading to out-of-date outputs. Because the rank of every element does not change too much during the next run of quick-sort, a solution is to parallel sort a small subset of U that contains the top k elements with high probability.

Algorithm 1. Top-k-set

Input: A set U of n elements
Output: $\widetilde{T}$

1: Initialize $\tilde{\pi}, L, C, \tilde{\pi}_C,$ and $\widetilde{T}$ arbitrarily
2: **while** (true) **do**
3: **Execute in odd steps:** /*QS_1*/
4: $\tilde{\pi} \leftarrow \text{quick\_sort}(U)$
5: $L \leftarrow \tilde{\pi}^{-1}([k - c\alpha \ln n])$ and $C \leftarrow \tilde{\pi}^{-1}([k + c\alpha \ln n]) \setminus L$ /*The constant c will be determined in the proof of Theorem 1*/
6: **Execute in even steps:** /*QS_2*/
7: $\tilde{\pi}_C \leftarrow \text{quick\_sort}(C)$
8: $\widetilde{T} \leftarrow L \bigcup \tilde{\pi}_C^{-1}([c\alpha \ln n])$
9: **end while**

Specifically, the algorithm Top-k-set consists of two interleaving procedures (denoted by QS_1 and QS_2, respectively), each of which restarts once it terminates. In the odd steps, QS_1 calls quick-sort to sort U, preparing two sets L and C. The set L consists of the elements that will remain among top k during the next run of QS_1 with high probability, while C contains the uncertain elements that might be among top k in this period. Then, QS_2 will sort the set C computed by the last run of QS_1 to produce the estimated set of top k elements. At any time t, the output $\widetilde{T}_t$ of the algorithm is the set $\widetilde{T}$ computed by the previous run of QS_2.

Theorem 1 shows that Algorithm 1 is error-free with high probability.

Theorem 1. *Assume that* $\alpha = o(\frac{\sqrt{n}}{\ln n})$. *For any* $k \in [n]$, $\Pr(\widetilde{T}_t = (\pi^t)^{-1}([k])) = 1 - o(1)$, *where* $\widetilde{T}_t$ *is the output of Algorithm 1 at time* t, π^t *is the true order on* U *at time* t, *and* t *is sufficiently large.*

The basic idea of the proof lies in two aspects. First, with high probability, the estimated rank of every element with respect to $\tilde{\pi}$ is at most $O(\alpha \ln n)$ away from the true rank, implying that all the elements in L are among top k and all top k elements are in $L \bigcup C$. Second, with high probability, the kth element of U does not swap throughout sorting C, so the set of top k elements remains unchanged and is exactly contained in $\widetilde{T}$. The detailed proof will be omitted.

3.2 An Algorithm for the Top-k-selection Problem

Now we present an algorithm to solve the top-k-selection problem. The basic idea is to repeatedly run quick-sort over the data U, extracting a small subset that includes all the elements that can be among top k during the next run. To exactly identify the top k elements in order, the small set is sorted and the order of the top k elements is produced accordingly. Like in designing the top-k-set algorithm, there is also an issue to address: since sorting the small set takes time $\Omega(k \ln k)$, the order of the top k elements will soon become out of date. Again note that with high probability the rank of each element does not change too much during sorting the small set, so the order of the top k elements can be regulated locally and keeps updated.

Specifically, Algorithm 2 consists of four interleaving procedures (QS_1, QS_2, QS_3, and Local-sort), each of which restarts once it terminates. At the $(4t+1)$-th time steps, QS_1 invokes a quick-sort on U, preparing a set C of size $k+O(\alpha \ln n)$ which with high probability, contains all the elements among top k during the next run of QS_1. At the $(4t+2)$-th time steps, QS_2 calls another quick-sort on the latest C computed by QS_1, producing a set P of size k. With high probability, the set P exactly consists of the top k elements of U during the next run of QS_2. At the $(4t+3)$-th time steps, the other quick-sort is invoked by QS_3 on the latest P computed by QS_2, periodically updating the estimated order over P. The resulting order is actually close to the true order over P during the next run of QS_3. Finally, at the $(4t)$-th time steps, an algorithm Local-sort is executed on the total order over P that is produced by the last run of QS_3, so as to locally regulate the order. At any time t, the output $\tilde{\pi}_k^t$ of Algorithm 2 is the last $\tilde{\pi}_k$ computed by Local-sort.

The main idea of Algorithm 3 (Local-sort) is to regulate the order over P block by block. Since block-by-block processing takes linear time, the errors can be corrected in time and few new errors will emerge during one run of Algorithm 3. Considering that the elements may move across blocks, it is necessary to make the blocks overlap. Actually, for each j, the element of the lowest rank in the j-th block is found, regarded as the j-th element of the final order, and removed from the block. The rest elements of the j-th block, together with the lowest-ranked element in P (according to the latest order produced by QS_3) that has not yet been processed, forms the $(j+1)$-th block. The element of the

Algorithm 2. Top-k-selection

Input: A set U of n elements
Output: $\tilde{\pi}_k$

Let t be the time
Initialize $\tilde{\pi}, C, \tilde{\pi}_C, P, \tilde{\pi}_P,$ and $\tilde{\pi}_k$ arbitrarily
while (true) **do**
  Execute in $t \equiv 1 (\mathbf{mod}\ 4)$ **steps** /*QS_1*/
  $\tilde{\pi} \leftarrow$ quick_sort(U)
  $C \leftarrow \tilde{\pi}^{-1}([k + c'\alpha \ln n])$ /*The constant c' will be determined in the proof of Theorem 2*/
  Execute in $t \equiv 2 (\mathbf{mod}\ 4)$ **steps** /*QS_2*/
  $\tilde{\pi}_C \leftarrow$ quick_sort(C)
  $P \leftarrow \tilde{\pi}_C^{-1}([k])$
  Execute in $t \equiv 3 (\mathbf{mod}\ 4)$ **steps** /*QS_3*/
  $\tilde{\pi}_P \leftarrow$ quick_sort(P)
  Execute in $t \equiv 0 (\mathbf{mod}\ 4)$ **steps** /*Local-sort*/
  $\tilde{\pi}_k \leftarrow$ Local-sort($P, \tilde{\pi}_P, 4c+1$) /*The constant c will be determined in the proof of Theorem 2*/
end while

Algorithm 3. Local-sort

Input: A set P; an order π over P; an integer c
Output: $\tilde{\pi}$

$m \leftarrow |P|$
$B_1 \leftarrow \pi^{-1}([c])$ /* Define the first block */
$\tilde{\pi}^{-1}(1) \leftarrow$ Maximum-Find(B_1)
$j = 2$
while $(c + j - 1 \leq m)$ **do**
  $B_j \leftarrow (B_{j-1} \backslash \tilde{\pi}^{-1}(j-1)) \bigcup \pi^{-1}(c + j - 1)$ /* Define the j-th block */
  $\tilde{\pi}^{-1}(j) \leftarrow$ Maximum-Find(B_j)
  $j = j + 1$
end while
$B_e \leftarrow B_{j-1}$ /*Deal with the final block*/
while $|B_e| \geq 1$ **do**
  $\tilde{\pi}^{-1}(j) \leftarrow$ Maximum-Find(B_e)
  $B_e \leftarrow B_e \backslash \tilde{\pi}^{-1}(j)$
  $j = j + 1$
end while

lowest rank in each block is found by calling Algorithm 4, which repeatedly runs sequential comparison. Both Algorithms 3 and 4 are self-explained, so detailed explanation is omitted here.

Theorem 2. *Assume* $\alpha = o(\frac{\sqrt{n}}{\ln n})$ *and* $k = O((\frac{n}{\alpha \ln n})^{1-\epsilon})$, *where* $\epsilon > 0$. *Let* $\tilde{\pi}_k^t$ *be the output of Algorithm 2 and* π_k^t *be the true order over the top* k *elements at time* t. *For sufficiently large* t, *we have that:*

Algorithm 4. Maximum-Find

Input: B
Output: u_{max}

```
1: u_max ← B(1)
2: j = 2
3: while (j ≤ |B|) do
4:    if u_max < B(j) then
5:       u_max ← B(j)
6:    end if
7:    j = j + 1
8: end while
```

1. *If* $k^2\alpha = o(n)$, $\Pr(\mathrm{KT}(\tilde{\pi}_k^t, \pi_k^t) = 0) = 1 - o(1)$,
2. *If* $k^2\alpha = \Theta(n)$, $\Pr(\mathrm{KT}(\tilde{\pi}_k^t, \pi_k^t) = 0) = \Theta(1)$, *and*
3. *If* $k^2\alpha = \omega(n)$, $\Pr(\mathrm{KT}(\tilde{\pi}_k^t, \pi_k^t) = O(\frac{k^2\alpha}{n})) = 1 - o(1)$.

We sketch the basic idea of the proof. First, with high probability, the rank of every element with respect to $\tilde{\pi}$ is at most $O(\alpha \ln n)$ away from the true rank, implying that all the top k elements are contained in C. Second, with high probability, the kth element of U does not swap throughout sorting C, so P is exactly the set of top k elements and the resulting rank of every element deviates from the true rank by at most a constant. Third, due to the small rank deviation of every element, the ordering can be corrected locally by sorting blocks of constant length. The detailed proof will be omitted.

3.3 Lower Bounds for the Top-*k*-selection Problem

Now we analyze the lower bounds of the performance of any top-k-selection algorithm. The lower bounds hold for both randomized and deterministic algorithms.

Let A be an arbitrary algorithm which takes our dynamic data as input and outputs a total order $\tilde{\pi}_k^t$ on a subset of size k at every time step t. Let π_k^t be the true order on the top k elements. The following theorems characterize the difference between $\tilde{\pi}_k^t$ and π_k^t when k is large.

Theorem 3. *Given* $k = \Omega(\sqrt{\frac{n}{\alpha}})$ *and* $\alpha = o(n)$, $\Pr(\mathrm{KT}(\tilde{\pi}_k^t, \pi_k^t) > 0) = \Theta(1)$ *for every* $t > k$.

The main idea of the proof is that with a constant probability, in any period of $\Theta(\sqrt{\frac{n}{\alpha}})$, exactly one swap occurs among the top k elements and the swap is not observed. The detailed proof will be omitted.

Theorem 4. *Given* $k = \omega(\sqrt{n})$ *and* $\alpha = O(1)$, $\mathrm{KT}(\tilde{\pi}_k^t, \pi_k^t) = \Omega(\frac{k^2}{n})$ *in expectation and with probability* $1 - o(1)$ *for every* $t > k/8$.

The basic idea of the proof is that with high probability, in any period of $\Theta(k)$, $\Omega(\frac{k^2}{n})$ swaps occur among the top k elements and a majority of the swaps are not observed. The detailed proof will be omitted.

From Theorems 2 and 3, we know that $\Theta(\sqrt{n/\alpha})$ is the critical point of k, and it is impossible to generally improve Algorithm 2 even if $k = \omega(\sqrt{n/\alpha})$. The term *critical point* means the least upper bound of k such that top-k-selection problem can be solved error-free with probability $1 - o(1)$.

4 Gaussian-Swapping Model

This section is devoted to extending the algorithms for the consecutive-swapping model to the Gaussian-swapping model. We focus on the special case where α is a constant, and still assume that at each time step only one pair of elements can be compared.

Algorithms 1 and 2 can be slightly adapted to solve the top-k-set problem and the top-k-selection problem in this model, respectively. Specifically, replacing α in lines 5 and 8 of Algorithm 1 with $\ln^{0.5} n$, one gets Algorithm 5; likewise, in Algorithm 2, replacing α in line 6 with $\ln^{0.5} n$ and $4c + 1$ in lines 13 with $4c \ln^{0.5} n + 1$, we get Algorithm 6. The following theorems state the performance of these algorithms, and the proofs are omitted.

Theorem 5. *For any $k \in [n]$, we have $\Pr(\widetilde{T}_t = (\pi^t)^{-1}([k])) = 1 - o(1)$, where $\widetilde{T}_t$ is the output of Algorithm 5 at time t, π^t is the true order at time t, and t is sufficiently large.*

Theorem 6. *Assume that $k = O((\frac{n}{\ln n})^{1-\epsilon})$, where $\epsilon > 0$. Let $\tilde{\pi}_k^t$ be the output of Algorithm 6 and π_k^t be the true order over the top k elements at time t. For sufficiently large t, we have:*

1. *If $k = o(\frac{\sqrt{n}}{\ln^{0.25} n})$, $\Pr(\mathrm{KT}(\tilde{\pi}_k^t, \pi_k^t) = 0) = 1 - o(1)$,*
2. *If $k = \Theta(\frac{\sqrt{n}}{\ln^{0.25} n})$, $\Pr(\mathrm{KT}(\tilde{\pi}_k^t, \pi_k^t) = 0) = \Theta(1)$, and*
3. *If $k = \omega(\frac{\sqrt{n}}{\ln^{0.25} n})$, $\Pr(\mathrm{KT}(\tilde{\pi}_k^t, \pi_k^t) = O(\frac{k^2 \ln n}{n})) = 1 - o(1)$.*

Except for the Gaussian distribution, d can also be determined by other discrete distributions, for example, $p(d) = \frac{\beta}{d^\gamma}$, where γ is a constant and β is a normalizing factor. When γ is large enough (say, $\gamma > 10$), the results similar to those in the Gaussian-swapping model can be obtained.

5 Conclusions

In this paper we identify the critical point k^* such that the top-k-selection problem can be solved error-free with high probability if and only if $k = o(k^*)$. A lower bound of the error when $k = \Omega(k^*)$ is also determined, which actually is tight under some condition. On the contrary, it is shown that the top-k-set problem can be solved error-free with probability $1 - o(1)$, for all $k \in [n]$. These results hold in the consecutive-swapping model and most of them can be extended to the Gaussian-swapping model.

A number of problems remain open for the top-k-selection problem in the consecutive-swapping model. For $\alpha = \omega(1)$, we have not shown whether the

upper bound $O(\frac{k^2\alpha}{n})$ of error is tight when $k = \omega(\sqrt{\frac{n}{\alpha}})$. For $\alpha = O(1)$, there exists a gap between $k = n$ and $k = O((\frac{n}{\ln n})^{1-\epsilon})$, where the lower bound $\Omega(\frac{k^2}{n})$ of error has not yet shown to be tight. We conjecture that these bounds are tight.

References

1. Anagnostopoulos, A., Kumar, R., Mahdian, M., Upfal, E.: Sort me if you can: how to sort dynamic data. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part II. LNCS, vol. 5556, pp. 339–350. Springer, Heidelberg (2009)
2. Ilyas, I., Beskales, G., Soliman, M.: A survey of top-k query processing techniques in relational database systems. ACM Comput. Surv. 40(4) (2008). Article 11
3. Knuth, D.E.: The Art of Computer Programming, vol. 3. Addison-Wesley, Boston (1973)
4. Kislitsyn, S.S.: On the selection of the kth element of an ordered set by pairwise comparison. Sibirskii Mat. Zhurnal **5**, 557–564 (1964)
5. Blum, M., Floyd, R., Pratt, V., Rivest, R., Tarjan, R.: Time bounds for selection. J. Comput. Syst. Sci. **7**(4), 448–461 (1973)
6. Dor, D., Zwick, U.: Selecting the median. In: SODA 1995, pp. 28–37 (1995)
7. Moreland, A.: Dynamic Data: Model, Sorting, Selection. Technical report (2014)
8. Anagnostopoulos, A., Kumar, R., Mahdian, M., Upfal, E., Vandin, F.: Algorithms on evolving graphs. In: 3rd Innovations in Theoretical Computer Science Conference (ITCS), pp. 149–160. ACM, New York (2012)
9. Bahmani, B., Kumar, R., Mahdian, M., Upfal, E.: Pagerank on an evolving graph. In: 18th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), pp. 24–32. ACM (2012)
10. Zhuang, H., Sun, Y., Tang, J., Zhang J., Sun, X.: Influence maximization in dynamic social networks. In: 13th IEEE International Conference on Data Mining (ICDM), pp. 1313–1318. IEEE (2013)
11. Labouseur, A.G., Olsen, P.W., Hwang, J.H.: Scalable and robust management of dynamic graph data. In: 1st International Workshop on Big Dynamic Distributed Data (BD3@VLDB), pp. 43–48 (2013)
12. Ren, C.: Algorithms for evolving graph analysis. Doctoral dissertation. The University of Hong Kong (2014)
13. Ajtai, M., Feldman, V., Hassidim, A., Nelson, J.: Sorting and selection with imprecise comparisons. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 37–48. Springer, Heidelberg (2009)
14. Feige, U., Raghavan, P., Peleg, D., Upfal, E.: Computing with noisy information. SIAM J. Comput. **23**(5), 1001–1018 (1994)
15. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS), pp. 1–16. ACM (2002)
16. Bressan, M., Peserico, E., Pretto, L.: Approximating PageRank locally with sublinear query complexity. ArXiv preprint (2014). arXiv:1404.1864
17. Fujiwara, Y., Nakatsuji, M., Shiokawa, H., Mishima, T., Onizuka, M.: Fast and exact top-k algorithm for pagerank. In: 27th AAAI Conference on Artificial Intelligence, pp. 1106–1112 (2013)
18. Albers, S.: Online algorithms: a survey. Math. Prog. **97**(1–2), 3–26 (2003)
19. Kuleshov, V., Precup, D.: Algorithms for multi-armed bandit problems. ArXiv preprint (2014). arXiv:1402.6028