Sorting with adversarial comparators and application to density estimation

Jayadev Acharya
ECE, UCSD
jacharya@ucsd.edu

Ashkan Jafarpour
ECE, UCSD
ashkan@ucsd.edu

Alon Orlitsky
{ECE,CSE}, UCSD
alon@ucsd.edu

Ananda Theertha Suresh
ECE, UCSD
asuresh@ucsd.edu

***Abstract*—We consider the problems of sorting and maximum-selection of $\mathbf{n}$ elements using *adversarial comparators*. We derive a maximum-selection algorithm that uses $8\mathbf{n}$ comparisons in expectation, and a sorting algorithm that uses $4\mathbf{n}\log_2\mathbf{n}$ comparisons in expectation. Both are tight up to a constant factor. Our adversarial-comparator model was motivated by the practically important problem of *density-estimation*, where we observe samples from an unknown distribution, and try to determine which of $\mathbf{n}$ known distributions is closest to it. Existing algorithms run in $\Omega(\mathbf{n^2})$ time. Applying the adversarial comparator results, we derive a density-estimation algorithm that runs in only $\mathcal{O}(\mathbf{n})$ time.**

I. Introduction

Sorting and maximum selection are important tasks with applications in diverse fields ranging from biology to recommendation systems. For example, in tournaments, the objective is to select the best team with only a few matches. In recommendation systems, the goal is to select the most prominent items with the least number of prior explorations. Other applications include ranking players on online *first-person shooter* games based on pairwise matches and ranking cars according to popularity based on customer surveys. In many cases, comparison is the costliest operation.

In the above examples, the outputs of the comparators are noisy. For example, in a game, the better team may not win. Motivated by such scenarios, sorting and maximum selection have received wide attention in several communities. Our main motivation comes from density estimation, described in Section III.

The paper is organized as follows. In Section II we discuss related work. In section III we relate the problem of density estimation to ranking. In Section IV we discuss the problem in detail, and in Section V we present our algorithms.

II. Related work

While the motivation for noisy comparators is universal, the choice of noise model and the metric of error depends on the application. Unlike our adversarial model, most of the known noise models are probabilistic.

[1], [2] consider ranking with noisy comparators when the inputs have an underlying order and the outcome of each pairwise comparisons is errorneous with some probability strictly less than half independently. They propose optimal algorithms for ranking with and without *re-sampling*: namely, whether or not the same pair can be compared again. Under a similar noise model, [3] propose algorithms for maximum selection.

[4] consider the popular Bradley-Terry-Luce model for noise where each input i is associated with a score w_i. If two inputs i and j are compared, the winner is decided by a toss of a coin with bias $w_i/(w_i+w_j)$. They propose an algorithm to estimate the underlying scores w_i, based on pairwise comparisons.

[5] consider the trade-off between the number of comparisons and loss. They show that choosing the comparisons adaptively decreases the loss significantly.

Suppose that for three inputs a, b, and c, a comparator outputs $a > b$, $b > c$, and $c > a$, then these inputs cannot be sorted. If we change one of the outputs, say $c > a$ to $c < a$, then the inputs can be sorted. The problem of finding the minimum number of changes in comparator outputs to facilitate a ranking is known as *feedback arc set problem*. [6], [7], [8] show that solving it exactly is NP hard and propose efficient algorithms to approximate this number.

III. Adversarial noise model and density estimation

Unlike most of the above examples, our model is deterministic and adversarial. We first provide some motivation.

In density estimation, the objective is to approximate an unknown distribution based on its samples. More precisely, given independent samples from an unknown distribution P_0 we want to find a distribution P such that $|P - P_0| = \int_{x\in X} |P(x) - P_0(x)|dx$ is small.

One of the popular approaches to this problem [9] is to first use some process, for example Kernel estimation or prior information, to find a set $\mathcal{P}$ of *candidate* or *skeleton* distributions. Then find the candidate distribution that is closest to P_0 in ℓ_1 norm,

$$\arg\min_{P\in\mathcal{P}} |P - P_0|.$$

[10] considered this problem and proposed an algorithm called *Scheffe tournament*. We first discuss their results for the case $|\mathcal{P}| = 2$, where we need to choose between two possible distributions.

They constructed an algorithm C that takes k samples and

outputs a distribution $P' \in \mathcal{P}$ such that w.p. $1-\delta$,

$$|P' - P_0| \leq 3 \min_{P \in \mathcal{P}} |P - P_0| + \sqrt{\frac{10 \log \frac{1}{\delta}}{k}}.$$

Observe that this result is independent of the domain of these distributions, and that as the number of samples $k \to \infty$, C outputs a distribution that, while may not be necessarily closest, is at distance at most 3 times that of the closest.

It follows that given enough samples, if the distances between P_0 and the two distributions in $\mathcal{P}$ are within a factor of 3 from each other, the algorithm can output either distribution, but if the distances are more than a factor of 3 apart, the algorithm will output the closer one. It can therefore be viewed as a comparator C that takes two distributions P_i and P_j and outputs

$$C(P_i, P_j) = \begin{cases} P_i & \text{if } |P_i - P_0| < \frac{1}{3}|P_j - P_0|, \\ P_j & \text{if } |P_j - P_0| < \frac{1}{3}|P_i - P_0|, \\ \text{arbitary} & \text{otherwise.} \end{cases} \quad (1)$$

[10] extended this result for $|\mathcal{P}| > 2$, by running algorithm C between every pair of distributions in $\mathcal{P}$ and outputting the distribution that wins the most. They showed that this finds a distribution whose distance from P_0 is at most $9 \min_{P \in \mathcal{P}} |P - P_0|$. Note however that this algorithm runs in time $\mathcal{O}(|\mathcal{P}|^2)$ and faster algorithms are desired.

[11] improved the Scheffe tournament in two ways. They provided a modification of the minimum distance estimate that uses the same number, $\mathcal{O}(|\mathcal{P}|^2)$ of comparisons but outputs a distribution with distance $\leq 3 \min_{P \in \mathcal{P}} |P - P_0|$. They also derived an efficient minimum loss-weight estimate, that uses only a linear number of computations, assuming that $\mathcal{P}$ is exponentially preprocessed beforehand.

[12], [13], [14] used Scheffe tournament techniques for learning k-modal distributions, one dimension Gaussian mixtures, and d-dimensional Gaussian mixtures respectively. Furthermore, all of them proposed various modifications of the Scheffe algorithm for improving constants and time complexities. Our algorithm has smaller time complexity than all of the above mentioned algorithms.

In our model of this problem we relax the condition of arbitrary output in Equation (1) to adversarial output. Because adversarial output is more challenging, our results for this model can be applied to the original one. We compare our error guarantees to those of the best algorithm over all possible choices of arbitary cases in Equation (1). This model for error is also called as the minimax model where minimum is over all algorithms (strategies) and maximum is over all possible outcomes.

To simplify our model's description, we modify Equation (1) to obtain a comparator for real numbers. Let $x_i = -\log_3 |P_i - P_0|$, then for $x_i \in \mathbb{R}$, the comparator outputs

$$C(x_i, x_j) = \begin{cases} x_i & \text{if } x_i > x_j + 1, \\ x_j & \text{if } x_j > x_i + 1, \\ \text{arbitary} & \text{if } |x_i - x_j| \leq 1. \end{cases} \quad (2)$$

Let $n = |\mathcal{P}|$. After the above changes, the problem of finding the distribution closest to P_0 becomes finding the maximum element of the set $\{x_1, x_2, \ldots x_n\}$ under the above noisy comparator $C(\cdot, \cdot)$. Let $x^n \stackrel{\text{def}}{=} (x_1, \ldots, x_n)$.

IV. Problem statement

A. Noisy comparators and tournaments

Our objective is to minimize the total number of comparisons for (i) selecting the maximum and (ii) sorting the inputs in decreasing order, using the noisy comparator in Equation (2). Each comparison is referred to as a *query* and total number of queries are defined as *query complexity*.

Comparators can be either noisy or noiseless. Noiseless comparators are those whose outputs are the maximum of the inputs. The noisy comparators in our model are described in Equation (2).

In our model, a noisy comparator can be any comparator that satisfies the constraints in Equation (2) and is denoted by C. A set of all possible comparators are shown by $\mathcal{C}$. In the case of arbitrary output in Equation (2), we allow an adversary to choose the output. Note that this form of noise is stronger and more relevant to our problem than the probabilistic noisy comparators described in Section II.

A natural way to represent a comparator is by a complete oriented graph called as *tournament*, where if $C(x_i, x_j) = x_i$, then there is a directed edge from x_j to x_i. In the spirit of our example on games, we also refer this as x_i *wins* against x_j.

Because of the adversarial behavior of the noisy comparators, they can induce different tournaments based on our queries. Given x^n, the set of possible tournaments by any adversarial comparator are termed as *possible* tournaments.

Our results work for all the possible tournaments and hold for any probabilistic mix of possible tournaments.

B. Maximum selection

Given n real numbers x^n and a pairwise noisy comparator C, our goal is to find an algorithm $\mathcal{A}$, that compares the inputs in order to find the maximum or one close by. The output of $\mathcal{A}$ is denoted by $Y_{\mathcal{A}}$. Since algorithms can be randomized, $Y_{\mathcal{A}}$ is a random variable. The error of the output is

$$E_{\mathcal{A}} \stackrel{\text{def}}{=} E_{\mathcal{A}}(x^n) = \max(x^n) - Y_{\mathcal{A}}.$$

We measure the algorithm's performance by comparing $E_{\mathcal{A}}$ with some threshold t. For an algorithm $\mathcal{A}$ a t-approximation error is defined as

$$e_{\mathcal{A}}(t) \stackrel{\text{def}}{=} Pr(E_{\mathcal{A}} > t).$$

We are interested in designing algorithms whose $e_{\mathcal{A}}(t)$ is zero or small for a small constant t and all x^n and possible comparators.

C. Sorting

The inputs and comparators for sorting are same as those of maximum selection. The goal in sorting is to output a permutation of inputs that are sorted (or almost sorted) in decreasing order. We use the same notation as before and denote sorting algorithms also by $\mathcal{A}$. The usage becomes clear from the context. The output of $\mathcal{A}$ is denoted by $Y^n_{\mathcal{A}}$ which is a permutation of the inputs. As before, since algorithms are randomized, $Y^n_{\mathcal{A}}$ is a random variable. The error of sorting is defined by the maximum mis-sorting in the output.

$$E_{\mathcal{A}} \stackrel{\text{def}}{=} E_{\mathcal{A}}(x^n) \stackrel{\text{def}}{=} \max_{i \le j} Y_j - Y_i$$

Similar to maximum selection, the t-approximation error is

$$e_{\mathcal{A}}(t) \stackrel{\text{def}}{=} Pr(E_{\mathcal{A}} > t).$$

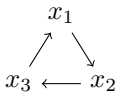
V. Algorithms

A. Lower bounds

Before proceeding to the algorithms, we provide simple lower bounds on $E_{\mathcal{A}}$ for both maximum selection and sorting. Since maximum selection is a subtask of sorting, for any algorithm sorting error is larger than maximum selection error.

Since the comparator cannot differentiate between 0 and 1, it is easy to see that for maximum selection, for any $\mathcal{A}$ and $t < 1$, $e_{\mathcal{A}}(t) \ge 1/2$. For example, if $x^2 = (0, 1)$, then no algorithm can conclusively find the maximum as the noisy comparator has error 1. An extension of this result shows that $e_{\mathcal{A}}(t)$ can be non-zero even for $t < 2$.

Example 1. $x^3 = (0, 1, 2)$



Since each of x_1, x_2, x_3 has won against another, no algorithm can conclusively determine the maximum and given all permutations of (x_1, x_2, x_3), every algorithm makes an error of 2 w.p. $\ge 1/3$ i.e., $e_{\mathcal{A}}(t) \ge 1/3$ for $t < 2$. The error $1/3$ can be made arbitrarily close to 1, by adding elements with values 0 and 1 in the above example.

We now consider a set of algorithms and their performance. Two of these algorithms Quick-select and quick-sort achieve the lower bounds for maximum selection and sorting respectively. Their query complexities are optimal up to a constant factor.

We consider the following algorithms for maximum selection and sorting:

- Scheffe tournament: Motivated by the Scheffe tournament in [10], we consider the same algorithm for maximum selection and sorting with noisy comparators. It has quadratic time complexity and achieves the best error bound.
- Sequential: A natural way of reducing the time complexity is to sequentially compare elements and keep the winner. We show that the error guarantee for this algorithm is unbounded.
- Knock-out: Motivated by sports tournaments, we consider Knock-out approach, which has linear query complexity. We show that with high probability $E < 3$.
- Quick-select: This algorithm is a variant of quick-sort and its query complexity is $8n$. It finds the maximum with performance similar to the lower bounds described in the previous subsection.
- Quick-sort: We show that the popular quick-sort has 2-approximation error guarantee and query complexity of less than $4n \log_2 n$ in expectation.

B. Scheffe tournament

Scheffe tournament compares all pair of inputs and sorts them depending on the number of wins for each input. Ties are broken randomly. Note that we are also finding the maximum element in the process. The algorithm is given in $\mathcal{A}^{\text{Scheffe}}$. This algorithm was previously studied in the context of density estimation [10].

Algorithm $\mathcal{A}^{\text{Scheffe}}$

input: x^n
let $\mathcal{Y}$ be an empty set
$\forall i, j$ where $i \neq j$ let $\mathcal{Y} = \mathcal{Y} \cup C(x_i, x_j)$
let m_i be the multiplicity of x_i in set $\mathcal{Y}$
output: sort inputs based on m_i

Lemma 2. *The query complexity of $\mathcal{A}^{\text{Scheffe}}$ is $\frac{n(n-1)}{2}$ and for noisy comparator C, $e_{\mathcal{A}^{\text{Scheffe}}}(2) = 0$ for both maximum selection and sorting.*

Proof: We show that for every two inputs x_i and x_j such that $x_i - x_j > 2$, $m_i > m_j$ in $\mathcal{A}^{\text{Scheffe}}$. Clearly, for all k if $C(x_j, x_k) = x_j$ then $C(x_i, x_k) = x_i$. Since $C(x_i, x_j) = x_i$, we have $m_i > m_j$. So x_i appears before x_j. ∎

The Scheffe tournament has query complexity of $\Theta(n^2)$. Note that for noiseless comparators we need only $\Theta(n)$ comparisons for selecting the maximum and $\Theta(n \log n)$ comparisons for sorting.

In the next two subsections we consider two algorithms that take $\Theta(n)$ comparisons and can select the maximum input in a noiseless model. Both has error worse than the Scheffe tournament but their query complexity is $\Omega(n)$.

C. Sequential Algorithm

To safeguard against possible adversaries, we derive a randomized algorithm.

Algorithm $\mathcal{A}^{\text{S}}$ compares first two numbers and keeps the larger number. The algorithm, then continues to compare the larger number with a new number till only one number remains and outputs this number as the maximum.

Observation 3. *$\mathcal{A}^{\text{S}}$ uses $n - 1$ comparisons and selects the maximum value without any error if the comparator is noiseless.*

Algorithm $\mathcal{A}^{\mathrm{S}}$

input: x^n
let $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$
choose a random $y \in \mathcal{X}$ and remove it from $\mathcal{X}$
while $|\mathcal{X}| > 0$
choose a random $x \in \mathcal{X}$ and remove it from $\mathcal{X}$
let $y = C(x, y)$
end while
output: y

For noisy comparators, we give an example where the error is unbounded with high probability. More precisely, $\mathcal{A}^{\mathrm{S}}$ outputs a number that is $m = \frac{\log n}{\log\log n}$ below the maximum w.p. close to 1; i.e., $e_{\mathcal{A}^{\mathrm{S}}}(t) \to 1$ for all $t < m$.

Example 4. *Suppose n is such that $\log n$, $\log\log n$ and $m = \frac{\log n}{\log\log n}$ are integers. Let*

$$x_i = \begin{cases} m & i = 1 \\ m-1 & i = 2, \ldots, \log n \\ m-2 & i = \log n + 1, \ldots, \log^2 n \\ \ldots & \\ 0 & i = \frac{n}{\log n} + 1, \ldots, n \end{cases}$$

Let C be as follows.

$$C(x_i, x_j) = \begin{cases} x_i & \text{if } x_i > x_j + 1 \\ x_j & \text{if } x_j > x_i + 1 \\ \min(x_i, x_j) & \text{otherwise} \end{cases}$$

The maximum is m, but $Pr(Y_{\mathcal{A}^{\mathrm{S}}} = 0) \geq 1 - \frac{1}{\log\log n}$.

Proof: Consider a random permutation of x^n and put them in a list. $\mathcal{A}^{\mathrm{S}}$ will compare this list sequentially. Let l_i be the last appearance of input with value i. One can show that $l_i > l_{i+1}$ w.p. $\geq 1 - \frac{1}{\log n}$. By the union bound, w.p. $1 - \frac{1}{\log\log n}$, $l_0 > l_1 > \ldots > l_m$ and because of the form of comparator, $\mathcal{A}^{\mathrm{S}}$ outputs 0 with probability $\geq 1 - \frac{1}{\log\log n}$. ■

Example 4 shows that sequential algorithms have bad performance compare to Scheffe tournament. A natural question is to ask if there is a linear query complexity algorithm with bounded error. We answer this question, by considering the popular format in games, Knock-out algorithm.

D. Knock-out algorithm

The knock-out algorithm or the knock-out tournament is divided into successive rounds. At each round, each player plays one game and the winner advances to the next round. After $\log n$ rounds we are left with exactly one winner or the maximum number in our case.

We modify this algorithm by keeping a fraction of inputs at each round in an extra set. The algorithm calls Scheffe tournament when the number of remaining inputs is square-root of original number of inputs. The algorithm is given in Algorithm $\mathcal{A}^{\mathrm{K}}$.

In the first stage of the algorithm $\mathcal{A}^{\mathrm{K}}$ runs the Knock-out tournament, but at each rounds keeps a set of $\sqrt{n}/\log n$ inputs. The first stage ends when $\mathcal{X}$ has at most $\sqrt{n}/\log n$ inputs.

Algorithm $\mathcal{A}^{\mathrm{K}}$

input: x^n
let $\mathcal{X}$ be the set of inputs and $\mathcal{Y}$ be an empty set
while $|\mathcal{X}| > \sqrt{n}/\log n$
choose $\sqrt{n}/\log n$ of inputs at random and add to $\mathcal{Y}$
pair the elements in $\mathcal{X}$ randomly and keep the winners
end while
run the $\mathcal{A}^{\text{Scheffe}}$ on set $\mathcal{X} \cup \mathcal{Y}$

The algorithm then runs Scheffe tournament $\mathcal{A}^{\text{Scheffe}}$ on the set $\mathcal{X} \cup \mathcal{Y}$. Note that $|\mathcal{Y}|$ is kept smaller than $\sqrt{n}$.

Now we discuss the performance of $\mathcal{A}^{\mathrm{K}}$.

Observation 5. *The number of comparison of the $\mathcal{A}^{\mathrm{K}}$ is $\Theta(n)$.*

Lemma 6. $e_{\mathcal{A}^{\mathrm{K}}}(3) \leq \log^3 n/\sqrt{n}$.

Proof: Without loss of generality assume $x_1 = \max(x^n)$. Note that $\mathcal{X}$ has $\frac{1}{2}\log n + \log\log n$ rounds while the algorithm is proceeding. At each step let $\mathcal{V}$ be the elements in $\mathcal{X}$ such that their values are $\geq x_1 - 1$. Let $\mathcal{W}$ be the elements in final $\mathcal{Y}$ such that their values are $\geq x_1 - 1$. Define $\alpha_i = |\mathcal{V}|/|\mathcal{X}|$ at step i. Let $\alpha = \max \alpha_i$. We have

$$Pr(|\mathcal{W}| = 0) \leq Pr(x_1 \notin \mathcal{Y}) < \alpha(\frac{1}{2}\log n + \log\log n).$$

On the other hand,

$$Pr(|\mathcal{W}| = 0) \leq (1-\alpha)^{\frac{\sqrt{n}}{\log n}}.$$

After maximizing the minimum of these two bounds over α we have $Pr(|\mathcal{W}| = 0) < \log^3 n/\sqrt{n}$. The rest of the proof is by using Lemma 2. ■

By comparing this algorithm with the Scheffe tournament, we observe that $\mathcal{A}^{\mathrm{K}}$ has smaller query complexity, but Scheffe tournament has better error guarantee.

For sorting problem, one can try algorithms such as merge-sort that is related to $\mathcal{A}^{\mathrm{K}}$ but we could not find a way for merging parts without loss of error guarantee.

In the next section we describe our main result which is an algorithm with optimal number of queries up to a constant factor for maximum selection problem. While this algorithm requires $8n$ expected queries, it has error guarantee same as Scheffe tournament.

E. Quick-select algorithm

So far in our algorithms, we were comparing the winners in order to select the maximum. Surprisingly, if we compare all the elements with a random pivot, the error guarantee gets better. The algorithm is given in Algorithm $\mathcal{A}^{\text{quick}}$.

The following lemma shows that $Y_{\mathcal{A}^{\text{quick}}}$ is close to $\max(x^n)$.

Lemma 7. $e_{\mathcal{A}^{\text{quick}}}(2) = 0$.

Proof: Without loss of generality assume x_1 is the maximum input. If x_1 remains in $\mathcal{X}$ then the lemma is true. Otherwise, x_1 is removed from $\mathcal{A}$ with some pivot x_r where $x_r \geq x_1 - 1$. At this stage, every remained element in $\mathcal{A}$ are $\geq x_r - 1 \geq x_1 - 2$. Hence the lemma is proved. ■

Algorithm $\mathcal{A}^{\text{quick}}$

input: x^n
let $\mathcal{X}$ be the set of inputs
while $|\mathcal{X}| > 1$
choose x_r, a random input $\in \mathcal{X}$
compare all the inputs in $\mathcal{X}$ with x_r and keep the winners
if $|\mathcal{X}| = 0$ add x_r to $\mathcal{X}$
end while
output: output the remained input in $\mathcal{X}$

Next we show that after comparing elements with a random pivot, the fraction of remaining elements is at most $\frac{3}{4}$ of the original elements w.p. $\frac{1}{2}$. This is a necessary condition to bound the running time of this algorithm. Note that for noiseless comparisons this condition holds.

Consider a complete oriented graph G on n nodes. Let d_v^{in} be the number of ingoing edges of node v.

Lemma 8. *For a uniformly randomly chosen node $v \in G$, $d_v^{in} \geq \frac{1}{4}(n-1)$ w.p. ≥ 0.5.*

Proof: The proof is omitted. ∎

By Lemma 8 we have shown that w.p. $\frac{1}{2}$, after comparing inputs in a set of size n with a random pivot, the number of remained input is $< \frac{3}{4}(n-1)$.

In the following we bound the expected number of queries to select the maximum using $\mathcal{A}^{\text{quick}}$. Let $f(n)$ be the expected number of queries that $\mathcal{A}^{\text{quick}}$ uses in order to select the maximum of n inputs. Using Lemma 8,

$$f(n) \leq n + \frac{1}{2}f(n) + \frac{1}{2}f(\frac{3}{4}n).$$

After solving this recursion we have $f(n) \leq 8n$. We did not put any effort to minimize the constant and a better constant by optimizing the numbers in Lemma 8 is possible.

F. Quick-sort algorithm

In this section we use quick-sort algorithm, $\mathcal{A}^{\text{quick}}$, with a noisy comparator to sort the inputs.

Similar to Lemma 7, quick-sort algorithm can sort the inputs with zero 2-approximation error.

Lemma 9. $e_{\mathcal{A}^{\text{quick}}}(2) = 0$.

The following lemma shows that w.p. $\frac{1}{2}$, a random pivot in quick-sort algorithm, divides the set of inputs into two sets where the size of each set is greater than a constant fraction of the primary set.

Lemma 10. *For a uniformly randomly chosen node $v \in G$, $\frac{1}{8}(n-1) \leq d_v^{in} \leq \frac{7}{8}(n-1)$ w.p. ≥ 0.5.*

Proof: The proof is omitted. ∎

Using Lemma 10 we can bound the query complexity of the quick-sort algorithm. let $f(n)$ be the expected number of comparisons that $\mathcal{A}^{\text{quick}}$ requires in order to sort n inputs. Note that query complexity of the quick-sort for this problem is different from original quick-sort since comparisons are noisy.

$$f(n) \leq n + \frac{1}{2}f(n) + \frac{1}{2}f(\frac{7}{8}n) + \frac{1}{2}f(\frac{1}{8}n).$$

After solving this recursion we have $f(n) \leq \frac{16}{24-7\log_2 7} n \log_2 n \leq 4n \log_2 n$.

[15] shows that the probability of quick-sort algorithm taking more than fraction c of its expected time, decreases exponentially with c. Same result for our model of noisy comparators holds with slightly changes.

VI. Density estimation

Recall the problem of density estimation, where given a set $\mathcal{P}$ of distributions and an unknown distribution P_0. The goal is to find a distribution $\mathcal{P}$ that is closest to P_0 in ℓ_1 distance using $i.i.d.$ samples from P_0. Quick-select along with the distribution comparator discussed in III can be used to obtain a linear time algorithm as opposed to the Scheffe tournament algorithm [10] that runs in quadratic time.

Theorem 11. *Quick-select algorithm uses $8|\mathcal{P}|$ comparisons in expectation and provides the same guarantee as Scheffe tournament.*

References

[1] R. M. Karp and R. Kleinberg, "Noisy binary search and its applications," in *SODA*, 2007, pp. 881–890.
[2] M. Braverman and E. Mossel, "Noisy sorting without resampling," in *SODA*, 2008, pp. 268–276.
[3] M. Adler, P. Gemmell, M. Harchol-Balter, R. M. Karp, and C. Kenyon, "Selection in the presence of noise: The design of playoff systems," in *SODA*, 1994, pp. 564–572.
[4] S. Negahban, S. Oh, and D. Shah, "Iterative ranking from pair-wise comparisons," in *NIPS*, 2012, pp. 2483–2491.
[5] N. Ailon, "An active learning algorithm for ranking from pairwise preferences with an almost optimal query complexity," *Journal of Machine Learning Research*, vol. 13, pp. 137–164, 2012.
[6] N. Alon, "Ranking tournaments," *SIAM J. Discrete Math.*, vol. 20, no. 1, pp. 137–142, 2006.
[7] N. Ailon, M. Charikar, and A. Newman, "Aggregating inconsistent information: Ranking and clustering," *J. ACM*, vol. 55, no. 5, 2008.
[8] D. Coppersmith, L. Fleischer, and A. Rudra, "Ordering by weighted number of wins gives a good ranking for weighted tournaments," *ACM Transactions on Algorithms*, vol. 6, no. 3, 2010.
[9] H. Scheffe, "A useful convergence theorem for probability distributions," in *The Annals of Mathematical Statistics*, vol. 18, 1947, pp. 434–438.
[10] L. Devroye and G. Lugosi, *Combinatorial Methods in Density Estimation*. New York: Springer - verlag, 2001.
[11] S. Mahalanabis and D. Stefankovic, "Density estimation in linear time," in *COLT*, 2008, pp. 503–512.
[12] C. Daskalakis, I. Diakonikolas, and R. A. Servedio, "Learning *k*-modal distributions via testing," in *SODA*, 2012, pp. 1371–1385.
[13] C. Daskalakis and G. Kamath, "Faster and sample near-optimal algorithms for proper learning mixtures of gaussians," *CoRR*, vol. abs/1312.1054, 2013.
[14] J. Acharya, A. Jafarpour, A. Orlitsky, and A. T. Suresh, "Near-optimal-sample estimators for spherical gaussian mixtures," *CoRR*, vol. abs/1402.4746, 2014.
[15] C. McDiarmid and R. Hayward, "Large deviations for quicksort," *J. Algorithms*, vol. 21, no. 3, pp. 476–507, 1996.