Maximum Selection and Ranking under Noisy Comparisons

Moein Falahatgar [1] **Alon Orlitsky** [1] **Venkatadheeraj Pichapati** [1] **Ananda Theertha Suresh** [2]

Abstract

We consider (ϵ, δ)-PAC maximum-selection and ranking using pairwise comparisons for general probabilistic models whose comparison probabilities satisfy strong stochastic transitivity and stochastic triangle inequality. Modifying the popular knockout tournament, we propose a simple maximum-selection algorithm that uses $\mathcal{O}\left(\frac{n}{\epsilon^2}\left(1+\log\frac{1}{\delta}\right)\right)$ comparisons, optimal up to a constant factor. We then derive a general framework that uses noisy binary search to speed up many ranking algorithms, and combine it with merge sort to obtain a ranking algorithm that uses $\mathcal{O}\left(\frac{n}{\epsilon^2}\log n(\log\log n)^3\right)$ comparisons for $\delta = \frac{1}{n}$, optimal up to a $(\log\log n)^3$ factor.

1. Introduction

1.1. Background

Maximum selection and sorting using pairwise comparisons are computer-science staples taught in most introductory classes and used in many applications. In fact, sorting, also known as *ranking*, was once claimed to utilize 25% of all computer cycles, *e.g.*, (Mukherjee, 2011).

In many applications, the pairwise comparisons produce only random outcomes. In sports, tournaments rank teams based on pairwise matches whose outcomes are probabilistic in nature. For example, Microsoft's *TrueSkill* (Herbrich et al., 2006) software matches and ranks thousands of Xbox gamers based on individual game results. And in online advertising, out of a myriad of possible ads, each web page may display only a few, and a user will typically select at most one. Based on these random comparisons, ad companies such as Google, Microsoft, or Yahoo, rank the ads' appeal (Radlinski & Joachims, 2007; Radlinski et al., 2008).

These and related applications have brought about a resurgence of interest in maximum selection and ranking using noisy comparisons. Several probabilistic models were considered, including the popular Bradley-Terry-Luce (Bradley & Terry, 1952) and its Plackett-Luce (PL) generalization (Plackett, 1975; Luce, 2005). Yet even for such specific models, the number of pairwise comparisons needed, or *sample complexity*, of maximum selection and ranking was known only to within a $\log n$ factor. We consider a significantly broader class of models and yet propose algorithms that are optimal up to a constant factor for maximum selection and up to $(\log\log n)^3$ for ranking.

1.2. Notation

Noiseless comparison assumes an unknown underlying ranking $r(1), \ldots, r(n)$ of the elements in $\{1, \ldots, n\}$ such that if two elements are compared, the higher-ranked one is selected. Similarly for noisy comparisons, we assume an unknown ranking of the elements, but now if two elements i and j are compared, i is chosen with some unknown probability $p(i,j)$ and j is chosen with probability $p(j,i) = 1 - p(i,j)$, where if i is higher-ranked, then $p(i,j) \geq \frac{1}{2}$. Repeated comparisons are independent of each other.

Let $\tilde{p}(i,j) = p(i,j) - \frac{1}{2}$ reflect the *additional probability* by which i is preferable to j. Note that $\tilde{p}(j,i) = -\tilde{p}(i,j)$ and $\tilde{p}(i,j) \geq 0$ if $r(i) > r(j)$. $|\tilde{p}(i,j)|$ can also be seen as a measure of dissimilarity between i and j. Following (Yue & Joachims, 2011), we assume that two natural properties, satisfied for example by the PL model, hold whenever $r(i) > r(j) > r(k)$: *Strong Stochastic Transitivity (SST)*, $\tilde{p}(i,k) \geq \max(\tilde{p}(i,j), \tilde{p}(j,k))$, and *Stochastic Triangle Inequality (STI)*, $\tilde{p}(i,k) \leq \tilde{p}(i,j) + \tilde{p}(j,k)$.

Two types of algorithms have been proposed for maximum selection and ranking under noisy comparisons: non-adaptive or offline (Rajkumar & Agarwal, 2014; Negahban et al., 2012; 2016; Jang et al., 2016) where the comparison pairs are chosen in advance, and *adaptive* or *online* where the comparison pairs are selected sequentially based on previous comparison results. We focus on the latter.

We specify the desired output via the (ϵ, δ)-PAC paradigm (Yue & Joachims, 2011; Szörényi et al., 2015) that requires the output to likely closely approximate the intended outcome. Specifically, given $\epsilon, \delta > 0$, with prob-

[1]University of California, San Diego [2]Google Research. Correspondence to: Venkatadheeraj Pichapati <dheerajpv7@ucsd.edu>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

ability $\geq 1-\delta$, maximum selection must output an ϵ-*maximum* element i such that for all j, $p(i,j) \geq \frac{1}{2}-\epsilon$. Similarly, with probability $\geq 1-\delta$, the ranking algorithm must output an ϵ-*ranking* $r'(1),\ldots,r'(n)$ such that whenever $r'(i) > r'(j)$, $p(i,j) \geq \frac{1}{2}-\epsilon$.

1.3. Outline

In Section 2 we review past work and summarize our contributions. In Section 3 we describe and analyze our maximum-selection algorithm. In Section 4 we propose and evaluate the ranking algorithm. In Section 5 we experimentally compare our algorithms with existing ones. In Section 6 we mention some future directions.

2. Old and new results

2.1. Related work

Several researchers studied algorithms that with probability $1-\delta$ find the exact maximum and ranking. (Feige et al., 1994) considered a simple model where the elements are ranked, and $\tilde{p}(i,j)=\epsilon$ whenever $r(i) > r(j)$. (Busa-Fekete et al., 2014a) considered comparison probabilities $p(i,j)$ satisfying the Mallows model (Mallows, 1957). And (Urvoy et al., 2013; Busa-Fekete et al., 2014b; Heckel et al., 2016) considered general comparison probabilities, without an underlying ranking assumption, and derived rankings based on Copeland- and Borda-counts, and random-walk procedures. As expected, when the comparison probabilities approach half, the above algorithms require arbitrarily many comparisons.

To achieve finite complexity even with near-half comparison probabilities, researchers adopted the PAC paradigm. For the PAC model with SST and STI constraints, (Yue & Joachims, 2011) derived a maximum-selection algorithm with sample complexity $\mathcal{O}\big(\frac{n}{\epsilon^2}\log\frac{n}{\epsilon\delta}\big)$ and used it to bound the regret of the problem's dueling-bandits variant. Related results appeared in (Syrgkanis et al., 2016). For the PL model, (Szörényi et al., 2015) derived a PAC ranking algorithm with sample complexity $\mathcal{O}(\frac{n}{\epsilon^2}\log n\log\frac{n}{\epsilon\delta})$.

Deterministic adversarial versions of the problem were considered by (Ajtai et al., 2015), and by (Acharya et al., 2014a; 2016) who were motivated by density estimation (Acharya et al., 2014b).

2.2. New results

We consider (ϵ,δ)-PAC adaptive maximum selection and ranking using pairwise comparisons under SST and STI constraints. Note that when $\epsilon \geq \frac{1}{2}$ or $\delta \geq 1-1/n$ for maximum selection and $\delta \geq 1-1/n^2$ for ranking, any output is correct. We show for $\epsilon < 1/4, \delta < \frac{1}{2}$ and any n:

- Maximum-selection algorithm with sample complexity $\mathcal{O}\big(\frac{n}{\epsilon^2}\big(1+\log\frac{1}{\delta}\big)\big)$, optimal up to a constant factor.
- Ranking algorithm with $\mathcal{O}\big(\frac{n}{\epsilon^2}(\log n)^3\log\frac{n}{\delta}\big)$ sample complexity.
- General framework that converts any ranking algorithm with sample complexity $\mathcal{O}\big(\frac{n}{\epsilon^2}(\log n)^x\log\frac{n}{\delta}\big)$ into a ranking algorithm that for $\delta \geq \frac{1}{n}$ has sample complexity $\mathcal{O}\big(\frac{n}{\epsilon^2}\log n(\log\log n)^x\big)$.
- Using the above framework, a ranking algorithm with sample complexity $\mathcal{O}\big(\frac{n}{\epsilon^2}\log n(\log\log n)^3\big)$ for $\delta = \frac{1}{n}$.
- An $\Omega\big(\frac{n}{\epsilon^2}\log\frac{n}{\delta}\big)$ lower bound on the sample complexity of any PAC ranking algorithm, matching our algorithm's sample complexity up to a $(\log\log n)^3$ factor.

3. Maximum selection

3.1. Algorithm outline

We propose a simple maximum-selection algorithm based on Knockout tournaments. Knockout tournaments are used to find a maximum element under non-noisy comparisons. Knockout tournament of n elements runs in $\lceil \log n \rceil$ rounds where in each round it randomly pairs the remaining elements and proceeds the winners to next round.

Our algorithm, given in KNOCKOUT uses $\mathcal{O}\big(\frac{n}{\epsilon^2}\big(1+\log\frac{1}{\delta}\big)\big)$ comparisons and $\mathcal{O}(n)$ memory to find an ϵ-maximum. (Yue & Joachims, 2011) uses $\mathcal{O}\big(\frac{n}{\epsilon^2}\log\frac{n}{\epsilon\delta}\big)$ comparisons and $\mathcal{O}(n^2)$ memory to find an ϵ-maximum. Hence we get $\log n$-factor improvement in the number of comparisons and also we use linear memory compared to quadratic memory. From (Zhou & Chen, 2014) it can be inferred that the best PAC maximum selection algorithm requires $\Omega\big(\frac{n}{\epsilon^2}\big(1+\log\frac{1}{\delta}\big)\big)$ comparisons, hence up to constant factor, KNOCKOUT is optimal.

(Yue & Joachims, 2011; Szörényi et al., 2015) eliminate elements one by one until only ϵ-maximums are remaining. Since they potentially need $n-1$ eliminations, in order to appply union bound they had to ensure that each eliminated element is not an ϵ-maximum w.p. $1-\delta/n$, requiring $\mathcal{O}(\log(n/\delta))$ comparisons for each eliminated element and hence a superlinear sample complexity $\mathcal{O}(n\log(n/\delta))$.

In contrast, KNOCKOUT eliminates elements in $\log n$ rounds. Since in Knockout tournaments, number of elements decrease exponentially with each round, we afford to endure more error in the initial rounds and less error in the latter rounds by repeating comparison between each pair more times in latter rounds. Specifically, let b_i be the highest-ranked element (according to the unobserved underlying ranking) at the beginning of round i. KNOCKOUT makes sure that w.p. $\geq 1-\frac{\delta}{2^i}$, $\tilde{p}(b_i,b_{i+1}) \leq \epsilon_i$ by repeating comparison between each pair in round i for $\mathcal{O}\left(\frac{1}{\epsilon_i^2}\log\frac{2^i}{\delta}\right)$ times. Choosing $\epsilon_i = \frac{c\epsilon}{2^{i/3}}$ with $c = 2^{1/3}-1$, we make sure that comparison complexity is $\mathcal{O}\left(\frac{n}{\epsilon^2}\left(1+\log\frac{1}{\delta}\right)\right)$ and by union bound and STI, w.p. $\geq 1-\delta$, $\tilde{p}(b_1, b_{\lceil\log n\rceil+1}) \leq \sum_{i=1}^{\lceil\log n\rceil+1}\frac{c\epsilon}{2^{i/3}} \leq \epsilon$.

For $\gamma \geq 1$, a relaxed notion of SST, called *γ-stochastic transitivity* (Yue & Joachims, 2011), requires that if $r(i) > r(j) > r(k)$, then $\max(\tilde{p}(i,j), \tilde{p}(j,k)) \leq \gamma \cdot \tilde{p}(i,k)$. Our results apply to this general notion of γ-stochastic transitivity and the analysis of KNOCKOUT is presented under this model. KNOCKOUT uses $\mathcal{O}\left(\frac{n\gamma^4}{\epsilon^2}\left(1+\log\frac{1}{\delta}\right)\right)$ comparisons.

Remark 1. *(Yue & Joachims, 2011) considered a different definition of ϵ-maximum as an element i that is at most ϵ dissimilar to true maximum i.e., for j with $r(j) = n$, $\tilde{p}(j,i) \leq \epsilon$. Note that this definition is less restrictive than ours, hence requires fewer comparisons. Under this definition, (Yue & Joachims, 2011) used $\mathcal{O}\left(\frac{n\gamma^6}{\epsilon^2}\log\frac{n}{\epsilon\delta}\right)$ comparisons to find an ϵ-maximum whereas a simple modification of* KNOCKOUT *shows that $\mathcal{O}\left(\frac{n\gamma^2}{\epsilon^2}\left(1+\log\frac{1}{\delta}\right)\right)$ comparisons suffice. Hence we also get a significant improvement in the exponent of γ.*

To simplify the analysis, we assume that n is a power of 2, otherwise we can add $2^{\lceil\log n\rceil} - n$ dummy elements that lose to every original element with probability 1. Note that all ϵ-maximums will still be from the original set.

3.2. Algorithm

We start with a subroutine COMPARE that compares two elements. It compares two elements i, j and maintains empirical probability $\hat{p}_i$, a proxy for $p(i,j)$. It also maintains a confidence value $\hat{c}$ s.t., w.h.p., $\hat{p}_i \in (p(i,j)-\hat{c}, p(i,j)+\hat{c})$. COMPARE stops if it is confident about the winner or if it reaches its comparison budget m. It outputs the element with more wins breaking ties randomly.

Algorithm 1 COMPRARE

Input: element i, element j, bias ϵ, confidence δ.
Initialize: $\hat{p}_i = \frac{1}{2}$, $\hat{c} = \frac{1}{2}$, $m = \frac{1}{2\epsilon^2}\log\frac{2}{\delta}$, $r = 0$, $w_i = 0$.

1. **while** ($|\hat{p}_i - \frac{1}{2}| \leq \hat{c} - \epsilon$ and $r \leq m$)
   (a) Compare i and j. **if** i wins $w_i = w_i + 1$.
   (b) $r = r+1$, $\hat{p}_i = \frac{w_i}{r}$, $\hat{c} = \sqrt{\frac{1}{2r}\log\frac{4r^2}{\delta}}$.

if $\hat{p}_i \leq \frac{1}{2}$ **Output:** j. **else Output:** i.

We show that COMPARE w.h.p., outputs the correct winner if the elements are well seperated.

Lemma 2. *If $\tilde{p}(i,j) \geq \epsilon$, then*

$$Pr(\text{COMPARE}(i,j,\epsilon,\delta) \neq i) \leq \delta.$$

Note that instead of using fixed number of comparisons, COMPARE stops the comparisons adaptively if it is confident about the winner. If $|\tilde{p}(i,j)| \gg \epsilon$, COMPARE stops much before comparison budget $\frac{1}{2\epsilon^2}\log\frac{2}{\delta}$ and hence works better in practice.

Now we present the subroutine KNOCKOUT-ROUND that we use in main algorithm KNOCKOUT.

3.2.1. KNOCKOUT-ROUND

KNOCKOUT-ROUND takes a set S and outputs a set of size $|S|/2$. It randomly pairs elements, compares each pair using COMPARE, and returns the set of winners. We will later show that maximum element in the output set will be comparable to maximum element in the input set.

Algorithm 2 KNOCKOUT-ROUND

Input: Set S, bias ϵ, confidence δ.
Initialize: Set $O = \emptyset$.

1. Pair elements in S randomly.
2. **for** every pair (i,j):
   Add COMPARE(i,j,ϵ,δ) to O.

Output: O

Note that comparisons between each pair can be handled by a different processor and hence this algorithm can be easily parallelized.

S can have several maximum elements. Comparison probabilities corresponding to all maximum elements will be essentially same because of STI. We define $\max(S)$ to be the maximum element with the least index, namely,

$$\max(S) \stackrel{\text{def}}{=} S\Big(\min\{i : \tilde{p}(S(i), S(j)) \geq 0 \quad \forall j\}\Big).$$

Lemma 3. KNOCKOUT-ROUND(S,ϵ,δ) *uses $\frac{|S|}{4\epsilon^2}\log\frac{2}{\delta}$ comparisons and with probability $\geq 1-\delta$,*

$$\tilde{p}\Bigg(\mathit{max}(S), \mathit{max}\Big(\text{KNOCKOUT-ROUND}(S,\epsilon,\delta)\Big)\Bigg) \leq \gamma\epsilon.$$

3.2.2. KNOCKOUT

Now we present the main algorithm KNOCKOUT. KNOCKOUT takes an input set S and runs $\log n$ rounds of KNOCKOUT-ROUND halving the size of S at the end of each round. Recall that KNOCKOUT-ROUND makes sure that maximum element in the output set is comparable to maximum element in the input set. Using this, KNOCKOUT makes sure that the output element is comparable to maximum element in the input set.

Since the size of S gets halved after each round, KNOCKOUT compares each pair more times in the latter rounds. Hence the bias between maximum element in input set and maximum element in output set is small in latter rounds.

Algorithm 3 KNOCKOUT

Input: Set S, bias ϵ, confidence δ, stochasticity γ.
Initialize: $i = 1$, S = set of all elements, $c = 2^{1/3} - 1$.
while $|S| > 1$

1. S = KNOCKOUT-ROUND$\left(S, \frac{c\epsilon}{\gamma^2 2^{i/3}}, \frac{\delta}{2^i}\right)$.
2. $i = i + 1$.

Output: the unique element in S.

Note that KNOCKOUT uses only memory of set S and hence $\mathcal{O}(n)$ memory suffices.

Theorem 4 shows that KNOCKOUT outputs an ϵ-maximum with probability $\geq 1 - \delta$. It also bounds the number of comparisons used by the algorithm.

Theorem 4. KNOCKOUT(S, ϵ, δ) *uses* $\mathcal{O}\left(\frac{\gamma^4|S|}{\epsilon^2}\left(1 + \log\frac{1}{\delta}\right)\right)$ *comparisons and with probability at least* $1 - \delta$*, outputs an* ϵ*-maximum.*

4. Ranking

We propose a ranking algorithm that with probability at least $1 - \frac{1}{n}$ uses $\mathcal{O}\left(\frac{n\log n(\log\log n)^3}{\epsilon^2}\right)$ comparisons and outputs an ϵ-ranking.

Notice that we use only $\tilde{\mathcal{O}}\left(\frac{n\log n}{\epsilon^2}\right)$ comparisons for $\delta = \frac{1}{n}$ where as (Szörényi et al., 2015) uses $\mathcal{O}\big(n(\log n)^2/\epsilon^2\big)$ comparisons even for constant error probability δ. Furthermore (Szörényi et al., 2015) provided these guarantees only under Plackett-Luce model which is more restrictive compared to ours. Also, their algorithm uses $\mathcal{O}(n^2)$ memory compared to $\mathcal{O}(n)$ memory requirement of ours.

Our main algorithm BINARY-SEARCH-RANKING assumes the existence of a ranking algorithm RANK-x that with probability at least $1 - \delta$ uses $\mathcal{O}\big(\frac{n}{\epsilon^2}(\log n)^x \log\frac{n}{\delta}\big)$ comparisons and outputs an ϵ-ranking for any $\delta > 0$, $\epsilon > 0$ and some $x > 1$. We also present a RANK-x algorithm with $x = 3$.

Observe that we need RANK-x algorithm to work for any model that satisfies SST and STI. (Szörényi et al., 2015) showed that their algorithm works for Plackett-Luce model but not for more general model. So we present a RANK-x algorithm that works for general model.

The main algorithm BINARY-SEARCH-RANKING randomly selects $\frac{n}{(\log n)^x}$ elements (anchors) and rank them using RANK-x . The algorithm has then effectively created $\frac{n}{(\log n)^x}$ bins, each between two successively ranked anchors. Then for each element, the algorithm identifies the bin it belongs to using a noisy binary search algorithm. The algorithm then ranks the elements within each bin using RANK-x .

We first present MERGE-RANK, a RANK-3 algorithm.

4.1. Merge Ranking

We present a simple ranking algorithm MERGE-RANK that uses $\mathcal{O}\left(\frac{n(\log n)^3}{\epsilon^2}\log\frac{n}{\delta}\right)$ comparisons, $O(n)$ memory and with probability $\geq 1 - \delta$ outputs an ϵ-ranking. Thus MERGE-RANK is a RANK-x algorithm for $x = 3$.

Similar to Merge Sort, MERGE-RANK divides the elements into two sets of equal size, ranks them separately and combines the sorted sets. Due to the noisy nature of comparisons, MERGE-RANK compares two elements i, j sufficient times, so that the comparison output is correct with high probability when $|\tilde{p}(i,j)| \geq \frac{\epsilon}{\log n}$. Put differently, MERGE-RANK is same as the typical Merge Sort, except it uses COMPARE as the comparison function. Due to lack of space, MERGE-RANK is presented in Appendix A.

Let's define the error of an ordered set S as the maximum distance between two wrongly ordered items in S, namely,

$$err(S) \stackrel{\text{def}}{=} \max_{1\leq i\leq j\leq|S|} \tilde{p}(S(i), S(j)).$$

We show that when we merge two ordered sets, the error of the resulting ordered set will be at most $\frac{\epsilon}{\log n}$ more than the maximum of errors of individual ordered sets.

Observe that MERGE-RANK is a recursive algorithm and the error of a singleton set is 0. Two singleton sets each containing a unique element from the input set merge to form a set with two elements with an error at most $\frac{2\epsilon}{\log n}$, then two sets with two elements merge to form a set with four elements with an error of at most $\frac{3\epsilon}{\log n}$ and henceforth. Thus the error of the output ordered set is bounded by ϵ.

Lemma 5 shows that MERGE-RANK can output an ϵ-ranking of S with probability $\geq 1 - \delta$. It also bounds the number of comparisons used by the algorithm.

Lemma 5. MERGE-RANK$\left(S, \frac{\epsilon}{\log|S|}, \frac{\delta}{|S|^2}\right)$ *takes* $\mathcal{O}\left(\frac{|S|(\log|S|)^3}{\epsilon^2}\log\frac{|S|}{\delta}\right)$ *comparisons and with probability* $\geq 1 - \delta$*, outputs an* ϵ*-ranking. Hence,* MERGE-RANK *is a* RANK-3 *algorithm.*

Now we present our main ranking algorithm.

4.2. BINARY-SEARCH-RANKING

We first sketch the algorithm outline below. We then provide a proof outline.

4.2.1. ALGORITHM OUTLINE

Our algorithm is stated in BINARY-SEARCH-RANKING. It can be summarized in three major parts.

Creating anchors: (Steps 1 to 3) BINARY-SEARCH-RANKING first selects a set S' of $\frac{n}{(\log n)^x}$ random elements (anchors) and ranks them using RANK-x . At the end of this part, there are $\frac{n}{(\log n)^x}$ ranked anchors. Equivalently, the algorithm creates $\frac{n}{(\log n)^x} - 1$ bins, each bin between two successively ranked anchors.

Coarse ranking: (Step 4) After forming the bins, the algorithm uses a random walk on a binary search tree, to find which bin each element belongs to. INTERVAL-BINARY-SEARCH is similar to the noisy binary search algorithm in (Feige et al., 1994). It builds a binary search tree with the bins as the leaves and it does a random walk over this tree. Due to lack of space the algorithm INTERVAL-BINARY-SEARCH is presented in Appendix B but more intuition is given later in this section.

Ranking within each bin: (Step 5) For each bin, we show that the number of elements far from both anchors is bounded. The algorithm checks elements inside a bin whether they are close to any of the bin's anchors. For the elements that are close to anchors, the algorithm ranks them close to the anchor. And for the elements that are away from both anchors the algorithm ranks them using RANK-x and outputs the resulting ranking.

Algorithm 4 BINARY-SEARCH-RANKING

Input: Set S, bias ϵ.
Initialize: $\epsilon' = \epsilon/16$, $\epsilon'' = \epsilon/15$, and $S^o = \emptyset$. $S_j = \emptyset$, $C_j = \emptyset$ and $B_j = \emptyset$, for $1 \le j \le \left\lfloor \frac{n}{(\log n)^x} \right\rfloor + 2$.

1. Form a set S' with $\left\lfloor \frac{n}{(\log n)^x} \right\rfloor$ random elements from S. Remove these elements from S.
2. Rank S' using RANK-x $\left(S', \epsilon', \frac{1}{n^6}\right)$.
3. Add dummy element a at the beginning of S' such that $p(a, e) = 0\ \forall e \in S \bigcup S'$. Add dummy element b at the end of S' such that $p(b, e) = 1\ \forall e \in S \bigcup S'$.
4. **for** $e \in S$:
   (a) k = INTERVAL-BINARY-SEARCH(S', e, ϵ'').
   (b) Insert e in S_k.
5. **for** $j = 1$ to $\left\lfloor \frac{n}{(\log n)^x} \right\rfloor + 2$:
   (a) **for** $e \in S_j$:
      i. **if** COMPARE2$(e, S'(j), 10\epsilon''^{-2} \log n) \in \left[\frac{1}{2} - 6\epsilon'', \frac{1}{2} + 6\epsilon''\right]$, insert e in C_j.
      ii. **else if** COMPARE2$(e, S'(j+1), 10\epsilon''^{-2} \log n) \in \left[\frac{1}{2} - 6\epsilon'', \frac{1}{2} + 6\epsilon''\right]$, then insert e in C_{j+1}.
      iii. **else** insert e in B_j.
   (b) Rank B_j using RANK-x $\left(B_j, \epsilon'', \frac{1}{n^4}\right)$.
   (c) Append $S'(j)$, C_j, B_j in order at the end of S^o.

Output: S^o

4.2.2. ANALYSIS OF BINARY-SEARCH-RANKING

Creating anchors In Step 1 of the algorithm we select $n/(\log n)^x$ random elements. Since these are chosen uniformly random, they lie nearly uniformly in the set S. This intuition is formalized in the next lemma.

Lemma 6. *Consider a set S of n elements. If we select $\frac{n}{(\log n)^x}$ elements uniformly randomly from S and build an ordered set S' s.t. $\tilde{p}(S'(i), S'(j)) \ge 0\ \forall i > j$, then with probability $\ge 1 - \frac{1}{n^4}$, for any $\epsilon > 0$ and all k,*

$$|\{e \in S : \tilde{p}(e, S'(k)) > \epsilon, \tilde{p}(S'(k+1), e) > \epsilon\}| \le 5(\log n)^{x+1}.$$

In Step 2, we use RANK-x to rank S'. Lemma 7 shows the guarantee of ranking S'.

Lemma 7. *After Step 2 of the* BINARY-SEARCH-RANKING *with probability $\ge 1 - \frac{1}{n^6}$, S' is ϵ'-ranked.*

At the end of Step 2, we have $\frac{n}{(\log n)^x} - 1$ bins, each between two successively ranked anchors. Each bin has a left anchor and a right anchor . We say that an element belongs to a bin if it wins over the bin's left anchor with probability $\ge \frac{1}{2}$ and wins over the bin's right anchor with probability $\le \frac{1}{2}$. Notice that some elements might win over $S'(1)$ with probability $< \frac{1}{2}$ and thus not belong to any bin. So in Step 3, we add a dummy element a at the beginning of S' where a loses to every element in $S \bigcup S'$ with probability 1. For similar reasons we add a dummy element b to the end of S' where every element in $S \bigcup S'$ loses to b with probability 1.

Coarse Ranking Note that $S'(i)$ and $S'(i+1)$ are respectively the left and right anchors of the bin S_i.

Algorithm 5 COMPARE2

Input: element i, element j, number of comparisons m.

1. Compare i and j for m times and return the fraction of times i wins over j.

Since S' is ϵ'-ranked and the comparisons are noisy, it is hard to find a bin S_i for an element e such that $p(e, S'(i)) \geq \frac{1}{2}$ and $p(S'(i+1), e) \geq \frac{1}{2}$. We call a bin S_i a $\epsilon''-$*nearly correct* bin for an element e if $p(e, S'(i)) \geq \frac{1}{2} - \epsilon''$ and $p(S'(i+1), e) \geq \frac{1}{2} - \epsilon''$ for some $\epsilon'' > \epsilon'$.

In Step 4, for each element we find an ϵ''-*nearly correct* bin using INTERVAL-BINARY-SEARCH . Next we describe an outline of INTERVAL-BINARY-SEARCH.

INTERVAL-BINARY-SEARCH first builds a binary search tree of intervals (see Appendix B) as follows: the root node is the entire interval between the first and the last elements in S'. Each non-leaf node interval I has two children corresponding to the left and right halves of I. The leaves of the tree are the bins between two successively ranked anchors.

To find an ϵ''-*nearly correct* bin for an element e, the algorithm starts at the root of the binary search tree and at every non-leaf node corresponding to interval I, it checks if e belongs to I or not by comparing e with I's left and right anchors. If e loses to left anchor or wins against the right anchor, the algorithm backtracks to current node's parent.

If e wins against I's left anchor and loses to its right one, the algorithm checks if e belongs to the left or right child by comparing e with the middle element of I and moves accordingly.

When at a leaf node, the algorithm checks if e belongs to the bin by maintaining a counter. If e wins against the bin's left anchor and loses to the bin's right anchor, it increases the counter by one or otherwise it decreases the counter by one. If the counter is less than 0 the algorithm backtracks to the bin's parent. By repeating each comparison several times, the algorithm makes a correct decision with probability $\geq \frac{19}{20}$.

Note that there could be several ϵ''-*nearly correct* bins for e and even though at each step the algorithm moves in the direction of one of them, it could end up moving in a loop and never reaching one of them. We thus run the algorithm for $30 \log n$ steps and terminate.

If the algorithm is at a leaf node by $30 \log n$ steps and the counter is more than $10 \log n$ we show that the leaf node bin is a ϵ''-*nearly correct* bin for e and the algorithm outputs the leaf node. If not, the algorithm puts in a set Q all the anchors visited so far and orders Q according to S'.

We select $30 \log n$ steps to ensure that if there is only one nearly correct bin, then the algorithm outputs that bin w.p. $\geq 1 - \frac{1}{n^6}$. Also we do not want too many steps so as to bound the size of Q.

By doing a simple binary search in Q using BINARY-SEARCH (see Appendix B) we find an anchor $f \in Q$ such that $|\tilde{p}(e, f)| \leq 4\epsilon''$. Since INTERVAL-BINARY-SEARCH ran for at most $30 \log n$ steps, Q can have at most $60 \log n$ elements and hence BINARY-SEARCH can search effectively by repeating each comparison $\mathcal{O}(\log n)$ times to maintain high confidence. Next paragraph explains how BINARY-SEARCH finds such an element f.

BINARY-SEARCH first compares e with the middle element m of Q for $\mathcal{O}(\log n)$ times. If the fraction of wins for e is between $\frac{1}{2} - 3\epsilon''$ and $\frac{1}{2} + 3\epsilon''$, then w.h.p. $|\tilde{p}(e, m)| \leq 4\epsilon''$ and hence BINARY-SEARCH outputs m. If the fraction of wins for e is less than $\frac{1}{2} - 3\epsilon''$, then w.h.p. $\tilde{p}(e, m) \leq -2\epsilon''$ and hence it eliminates all elements to the right of m in Q. If the fraction of wins for e is more than $\frac{1}{2} + 3\epsilon''$, then w.h.p. $\tilde{p}(e, m) \geq 2\epsilon''$ and hence it eliminates all elements to the left of m in Q. It continues this process until it finds an element f such that the fraction of wins for e is between $\frac{1}{2} - 3\epsilon''$ and $\frac{1}{2} + 3\epsilon''$.

In next Lemma, we show that INTERVAL-BINARY-SEARCH achieves to find a $5\epsilon''$-*nearly correct* bin for every element.

Lemma 8. *For any element $e \in S$, Step 4 of* BINARY-SEARCH-RANKING *places e in bin S_l such that $\tilde{p}(e, S'(l)) > -5\epsilon''$ and $\tilde{p}(S'(l+1), e) > -5\epsilon''$ with probability $\geq 1 - \frac{1}{n^5}$.*

Ranking within each bin Once we have identified the bins, we rank the elements inside each bin. By Lemma 6, inside each bin all elements are close to the bin's anchors except at most $5(\log n)^{x+1}$ of them.

The algorithm finds the elements close to anchors in Step $5a$ by comparing each element in the bin with the bin's anchors. If an element in bin S_j is close to bin's anchors $S'(j)$ or $S'(j+1)$, the algorithm moves it to the set C_j or C_{j+1} accordingly and if it is far away from both, the algorithm moves it to the set B_j. The following two lemmas state that this separating process happens accurately with high probability. The proofs of these results follow from the Chernoff bound and hence omitted.

Lemma 9. *At the end of Step 5a, for all j, $\forall e \in C_j$, $|\tilde{p}(e, S'(j))| < 7\epsilon''$ with probability $\geq 1 - \frac{1}{n^3}$.*

Lemma 10. *At the end of Step 5a, for all j, $\forall e \in B_j$, $\min(\tilde{p}(e, S'(j)), \tilde{p}(S'(j+1), e)) > 5\epsilon''$ with probability $\geq 1 - \frac{1}{n^3}$.*

Combining Lemmas 6, 7 and 10 next lemma shows that the size of B_j is bounded for all j.

Lemma 11. *At the end of Step 5a, $|B_j| \leq 5(\log n)^{x+1}$ for all j, with probability $\geq 1 - \frac{3}{n^3}$.*

Since all the elements in C_j are already close to an anchor, they need not be ranked. By Lemma 11 with probability $\geq 1 - \frac{3}{n^3}$ the number of elements in B_j is at most $5(\log n)^{x+1}$. We use RANK-x to rank each B_j and output the final ranking.

Lemma 12 shows that all B_j's are ϵ''-ranked at the end of Step 5b. Proof follows from properties of RANK-x and union bound.

Lemma 12. *At the end of Step 5b, all B_js are ϵ''-ranked with probability $\geq 1 - \frac{1}{n^3}$.*

Combining the above set of results yields our main result.

Theorem 13. *Given access to* RANK-x, BINARY-SEARCH-RANKING *with probability $\geq 1 - \frac{1}{n}$, uses $O\left(\frac{n \log n (\log \log n)^x}{\epsilon^2}\right)$ comparisons and outputs an ϵ-ranking.*

Using MERGE-RANK as a RANK-x algorithm with $x = 3$ leads to the following corollary.

Corollary 14. BINARY-SEARCH-RANKING *uses $O\left(\frac{n \log n (\log \log n)^3}{\epsilon^2}\right)$ comparisons and outputs an ϵ-ranking with probability $\geq 1 - \frac{1}{n}$.*

Using PALPAC-AMPRR (Szörényi et al., 2015) as a RANK-x algorithm with $x = 1$ leads to the following corollary over PL model.

Corollary 15. *Over PL model,* BINARY-SEARCH-RANKING *with probability $\geq 1 - \frac{1}{n}$ uses $O\left(\frac{n \log n \log \log n}{\epsilon^2}\right)$ comparisons and outputs an ϵ-ranking.*

It is well known that to rank a set of n values under the noiseless setting, $\Omega(n \log n)$ comparisons are necessary. We show that under the noisy model, $\Omega\left(\frac{n}{\epsilon^2} \log \frac{n}{\delta}\right)$ samples are necessary to output an ϵ-ranking and hence our algorithm is near-optimal.

Theorem 16. *For $\epsilon \leq \frac{1}{4}$, $\delta \leq \frac{1}{2}$, there exists a noisy model that satisfies SST and STI such that to output an ϵ-ranking with probability $\geq 1 - \delta$, $\Omega\left(\frac{n}{\epsilon^2} \log \frac{n}{\delta}\right)$ comparisons are necessary.*

5. Experiments

We compare the performance of our algorithms with that of others over simulated data. Similar to (Yue & Joachims, 2011), we consider the stochastic model where $p(i, j) = 0.6 \, \forall i < j$. Note that this model satisfies both SST and STI. We find 0.05-maximum with error probability $\delta = 0.1$. Observe that $i = 1$ is the only 0.05-maximum. We compare the sample complexity of KNOCKOUT with that of **BTM-PAC** (Yue & Joachims, 2011), **MallowsMPI** (Busa-Fekete et al., 2014a), and **AR** (Heckel et al., 2016). **BTM-PAC** is an (ϵ, δ)-PAC algorithm for the same model considered in this paper. **MallowsMPI** finds a Condorcet winner which exists under our general model. **AR** finds the maximum according to *Borda* scores. We also tried **PLPAC** (Szörényi et al., 2015), developed originally for PL model but the algorithm could not meet guarantees of $\delta = 0.1$ under this model and hence omitted. Note that in all the experiments the reported numbers are averaged over 100 runs.

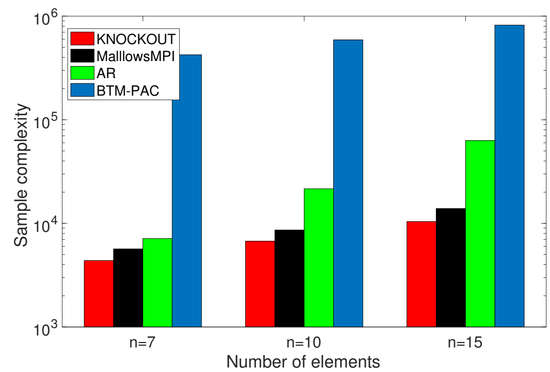

Figure 1. Comparison of sample complexity for small input sizes, with $\epsilon = 0.05$, and $\delta = 0.1$

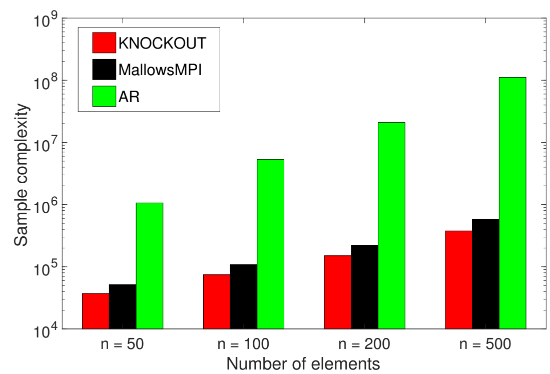

Figure 2. Comparison of sample complexity for large input size, with $\epsilon = 0.05$, and $\delta = 0.1$

In Figure 1, we compare the sample complexity of algorithms when there are 7, 10 and 15 elements. Our algorithm outperforms all the others. **BTM-PAC** performs much worse in comparison to others because of high constants in the algorithm. Further **BTM-PAC** allows comparing an element with itself since the main objective in (Yue & Joachims, 2011) is to reduce the regret. We exclude **BTM-PAC** for further experiments with higher number of elements.

In Figure 2, we compare the algorithms when there are 50, 100, 200 and 500 elements. Our algorithm outperforms others for higher number of elements too. Performance of **AR** gets worse as the number of elements increases since Borda scores of the elements get closer to each other and hence **AR** takes more comparisons to eliminate an element. Notice that number of comparisons is in logarithmic scale and hence the performance of **MallowsMPI** appears to be close to that of ours.

As noted in (Szörényi et al., 2015), sample complexity of **MallowsMPI** gets worse as $\tilde{p}(i, j)$ gets close to 0. To

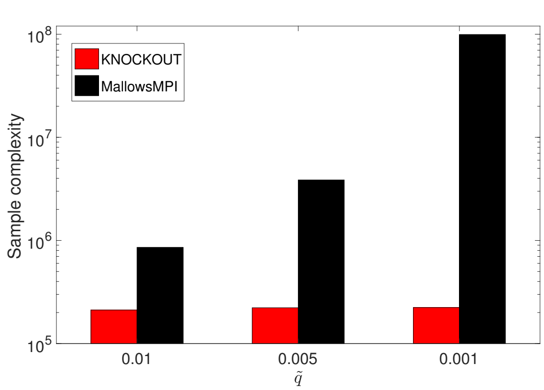

Figure 3. Sample complexity of KNOCKOUT and MallowsMPI for different values of $\tilde{q}$, with $\epsilon = 0.05$ and $\delta = 0.1$

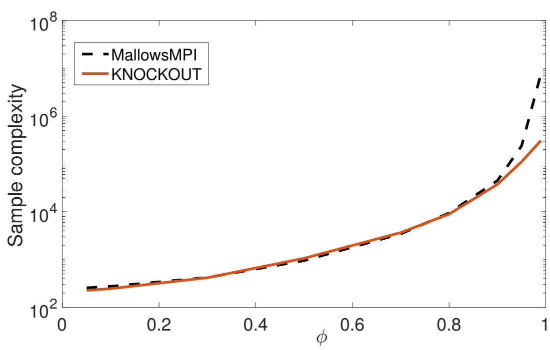

Figure 4. Sample complexity of KNOCKOUT and **MallowsMPI** under Mallows model for various values of ϕ

show the pronounced effect, we use the stochastic model $p(1, j) = 0.6\ \forall j > 1$, $p(i, j) = 0.5 + \tilde{q}\ \forall j > i, i > 1$ where $\tilde{q} < 0.1$, and the number of elements is 15. Here too we find 0.05-maximum with $\delta = 0.1$. Note that $i = 1$ is the only 0.05-maximum in this stochastic model. In Figure 3, we compare the algorithms for different values of $\tilde{q}$: 0.01, 0.005 and 0.001. As discussed above, the performance of **MallowsMPI** gets much worse whereas our algorithm's performance stays unchanged. The reason is that **MallowsMPI** finds the Condorcet winner using successive elimination technique and as $\tilde{q}$ gets closer to 0, **MallowsMPI** takes more comparisons for each elimination. Our algorithm tries to find an alternative which defeats Condorcet winner with probability $\geq 0.5 - 0.05$ and hence for alternatives that are very close to each other, our algorithm declares either one of them as winner after comparing them for certain number of times.

Next we evaluate KNOCKOUT on Mallows model which does not satisfy STI. Mallows is a parametric model which is specified by single parameter ϕ. As in (Busa-Fekete et al., 2014a), we consider $n = 10$ elements and various values for ϕ: 0.03, 0.1, 0.3, 0.5, 0.7, 0.8, 0.9, 0.95 and 0.99. Here again we seek to find 0.05-maximum with $\delta = 0.05$.

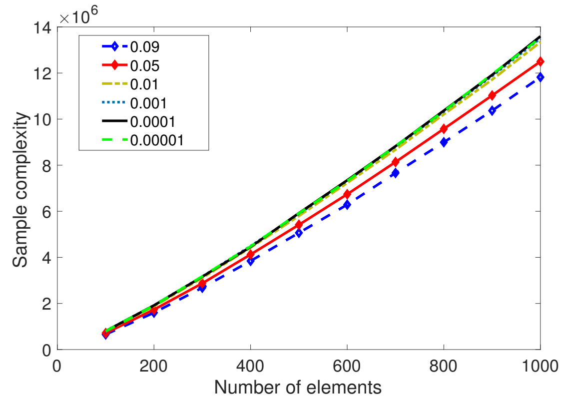

Figure 5. Sample complexity of MERGE-RANK for different ϵ

As we can see in Figure 4, sample complexity of KNOCKOUT and **MallowsMPI** is essentially same under small values of ϕ but KNOCKOUT outperforms **MallowsMPI** as ϕ gets close to 1 since comparison probabilities grow closer to $1/2$. Surprisingly, for all values of ϕ except for 0.99, KNOCKOUT returned Condorcet winner in all runs. For $\phi = 0.99$, KNOCKOUT returned second best element in 10 runs out of 100. Note that $\tilde{p}(1, 2) = 0.0025$ and hence KNOCKOUT still outputed a 0.05-maximum. Even though we could not show theoretical guarantees of KNOCKOUT under Mallows model, our simulations suggest that it can perform well even under this model.

For the stochastic model $p(i, j) = 0.6\ \forall i < j$, we run our MERGE-RANK algorithm to find an ϵ-ranking with $\delta = 0.1$. Figure 5 shows that sample complexity does not increase a lot with decreasing ϵ. We attribute this to the subroutine COMPARE that finds the winner faster when the elements are more dissimilar.

Some more experiments are provided in Appendix G.

6. Conclusion

We studied maximum selection and ranking using noisy comparisons for broad comparison models satisfying SST and STI. For maximum selection we presented a simple algorithm with linear, hence optimal, sample complexity. For ranking we presented a framework that improves the performance of many ranking algorithms and applied it to merge ranking to derive a near-optimal algorithm.

We conducted several experiments showing that our algorithms perform well and out-perform existing algorithms on simulated data.

The maximum-selection experiments suggest that our algorithm performs well even without STI. It would be of interest to extend our theoretical guarantees to this case. For ranking, it would be interesting to close the $(\log \log n)^3$ ratio between the upper- and lower- complexity bounds.

7. Acknowledgements

We thank Yi Hao and Vaishakh Ravindrakumar for very helpful discussions and suggestions, and NSF for supporting this work through grants CIF-1564355 and CIF-1619448.

References

Acharya, Jayadev, Jafarpour, Ashkan, Orlitsky, Alon, and Suresh, Ananda Theertha. Sorting with adversarial comparators and application to density estimation. In *ISIT*, pp. 1682–1686. IEEE, 2014a.

Acharya, Jayadev, Jafarpour, Ashkan, Orlitsky, Alon, and Suresh, Ananda Theertha. Near-optimal-sample estimators for spherical gaussian mixtures. *NIPS*, 2014b.

Acharya, Jayadev, Falahatgar, Moein, Jafarpour, Ashkan, Orlitsky, Alon, and Suresh, Ananda Theertha. Maximum selection and sorting with adversarial comparators and an application to density estimation. *arXiv preprint arXiv:1606.02786*, 2016.

Ajtai, Miklós, Feldman, Vitaly, Hassidim, Avinatan, and Nelson, Jelani. Sorting and selection with imprecise comparisons. *ACM Transactions on Algorithms (TALG)*, 12(2):19, 2015.

Bradley, Ralph Allan and Terry, Milton E. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

Busa-Fekete, Róbert, Hüllermeier, Eyke, and Szörényi, Balázs. Preference-based rank elicitation using statistical models: The case of mallows. In *Proc. of the ICML*, pp. 1071–1079, 2014a.

Busa-Fekete, Róbert, Szörényi, Balázs, and Hüllermeier, Eyke. Pac rank elicitation through adaptive sampling of stochastic pairwise preferences. In *AAAI*, 2014b.

Feige, Uriel, Raghavan, Prabhakar, Peleg, David, and Upfal, Eli. Computing with noisy information. *SIAM Journal on Computing*, 23(5):1001–1018, 1994.

Heckel, Reinhard, Shah, Nihar B, Ramchandran, Kannan, and Wainwright, Martin J. Active ranking from pairwise comparisons and when parametric assumptions don't help. *arXiv preprint arXiv:1606.08842*, 2016.

Herbrich, Ralf, Minka, Tom, and Graepel, Thore. Trueskill: a bayesian skill rating system. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, pp. 569–576. MIT Press, 2006.

Jang, Minje, Kim, Sunghyun, Suh, Changho, and Oh, Sewoong. Top-k ranking from pairwise comparisons: When spectral ranking is optimal. *arXiv preprint arXiv:1603.04153*, 2016.

Luce, R Duncan. *Individual choice behavior: A theoretical analysis*. Courier Corporation, 2005.

Mallows, Colin L. Non-null ranking models. i. *Biometrika*, 44 (1/2):114–130, 1957.

Mukherjee, Sudipta. *Data structures using C: 1000 problems and solutions*. McGraw Hill Education, 2011.

Negahban, Sahand, Oh, Sewoong, and Shah, Devavrat. Iterative ranking from pair-wise comparisons. In *NIPS*, pp. 2474–2482, 2012.

Negahban, Sahand, Oh, Sewoong, and Shah, Devavrat. Rank centrality: Ranking from pairwise comparisons. *Operations Research*, 2016.

Plackett, Robin L. The analysis of permutations. *Applied Statistics*, pp. 193–202, 1975.

Radlinski, Filip and Joachims, Thorsten. Active exploration for learning rankings from clickthrough data. In *Proceedings of the 13th ACM SIGKDD*, pp. 570–579. ACM, 2007.

Radlinski, Filip, Kurup, Madhu, and Joachims, Thorsten. How does clickthrough data reflect retrieval quality? In *Proceedings of the 17th ACM conference on Information and knowledge management*, pp. 43–52. ACM, 2008.

Rajkumar, Arun and Agarwal, Shivani. A statistical convergence perspective of algorithms for rank aggregation from pairwise data. In *Proc. of the ICML*, pp. 118–126, 2014.

Syrgkanis, Vasilis, Krishnamurthy, Akshay, and Schapire, Robert E. Efficient algorithms for adversarial contextual learning. *arXiv preprint arXiv:1602.02454*, 2016.

Szörényi, Balázs, Busa-Fekete, Róbert, Paul, Adil, and Hüllermeier, Eyke. Online rank elicitation for plackett-luce: A dueling bandits approach. In *NIPS*, pp. 604–612, 2015.

Urvoy, Tanguy, Clerot, Fabrice, Féraud, Raphael, and Naamane, Sami. Generic exploration and k-armed voting bandits. In *Proc. of the ICML*, pp. 91–99, 2013.

Yue, Yisong and Joachims, Thorsten. Beat the mean bandit. In *Proc. of the ICML*, pp. 241–248, 2011.

Zhou, Yuan and Chen, Xi. Optimal pac multiple arm identification with applications to crowdsourcing. 2014.

A. Merge Ranking

We first introduce a subroutine that is used by MERGE-RANK. It merges two ordered sets in the presence of noisy comparisons.

A.1. MERGE

MERGE takes two ordered sets S_1 and S_2 and outputs an ordered set Q by merging them. MERGE starts by comparing the first elements in each set S_1 and S_2 and places the loser in the first position of Q. It compares the two elements sufficient times to make sure that output is near-accurate. Then it compares the winner and the element right to loser in the corresponding set. It continues this process until we run out of one of the sets and then adds the remaining elements to the end of Q and outputs Q.

Algorithm 6 MERGE

Input: Sets S_1, S_2, bias ϵ, confidence δ.
Initialize: $i = 1$, $j = 1$ and $O = \emptyset$.

1. **while** $i \leq |S_1|$ and $j \leq |S_2|$.
   (a) **if** $S_1(i) = \text{COMPARE}(S_1(i), S_2(j), \epsilon, \delta)$, then append $S_2(j)$ at the end of O and $j = j + 1$.
   (b) **else** append $S_1(i)$ at the end of O and $i = i + 1$.
2. **if** $i \leq |S_1|$, then append $S_1(i : |S_1|)$ at the end of O.
3. **if** $j \leq |S_2|$, then append $S_2(j : |S_2|)$ at the end of O.

Output: O.

We show that when we merge two ordered sets using MERGE, the error of resulting ordered set is not high compared to the maximum of errors of individual ordered sets.

Lemma 17. *With probability $\geq 1 - (|S_1| + |S_2|)\delta$, error of $\text{MERGE}(S_1, S_2, \epsilon, \delta)$ is at most ϵ more than the maximum of errors of S_1 and S_2. Namely, with probability $\geq 1 - (|S_1| + |S_2|)\delta$,*

$$err(\text{MERGE}(S_1, S_2, \epsilon, \delta)) \leq \max(err(S_1), err(S_2)) + \epsilon.$$

A.2. MERGE-RANK

Now we present the algorithm MERGE-RANK. MERGE-RANK partitions the input set S into two sets S_1 and S_2 each of size $|S|/2$. It then orders S_1 and S_2 separately using MERGE-RANK and combines the ordered sets using MERGE. Notice that MERGE-RANK is a recursive algorithm. The singleton sets each containing an unique element in S are merged first. Two singleton sets are merged to form a set with two elements, then the sets with two elements are merged to form a set with four elements and henceforth. By Lemma 17, each merge with bound parameter ϵ' adds at most ϵ' to the error. Since error of singleton sets is 0 and each element takes part in $\log n$ merges, the error of the output set is at most $\epsilon' \log n$. Hence with bound parameter $\epsilon / \log n$, the error of the output set is less than ϵ.

Algorithm 7 MERGE-RANK

Input: Set S, bias ϵ, confidence δ.

1. $S_1 = \text{MERGE-RANK}(S(1 : \lfloor |S|/2 \rfloor), \epsilon, \delta)$.
2. $S_2 = \text{MERGE-RANK}(S(\lfloor |S|/2 \rfloor + 1 : |S|), \epsilon, \delta)$.

Output: $\text{MERGE}(S_1, S_2, \epsilon, \delta)$.

B. Algorithms for Ranking

Algorithm 8 INTERVAL-BINARY-SEARCH

Input: Ordered array S, search element e, bias ϵ

1. T = BUILD-BINARY-SEARCH-TREE($|S|$).
2. Initialize set $Q = \emptyset$, node $\alpha = \text{root}(T)$, and count $c = 0$.
3. **repeat** for $30 \log n$ times
   (a) **if** $\alpha_2 - \alpha_1 > 1$,
      i. Add α_1, α_2 and $\lceil \frac{\alpha_1 + \alpha_2}{2} \rceil$ to Q.
      ii. **if** $\text{COMPARE2}(S(\alpha_1), e, \frac{10}{\epsilon^2}) > 1/2$ or $\text{COMPARE2}(e, S(\alpha_2), \frac{10}{\epsilon^2}) > 1/2$ then go back to the parent, $\alpha = \text{parent}(\alpha)$.
      iii. **else**
         - **if** $\text{COMPARE2}(S(\lceil \frac{\alpha_1 + \alpha_2}{2} \rceil), e, \frac{10}{\epsilon^2}) > 1/2$ go to the left child,$\alpha = \text{left}(\alpha)$.
         - **else** go to the right child, $\alpha = \text{right}(\alpha)$.
   (b) **else**
      i. **if** $\text{COMPARE2}(e, S(\alpha_1), \frac{10}{\epsilon^2}) > 1/2$ and $\text{COMPARE2}(S(\alpha_2), e, \frac{10}{\epsilon^2}) > 1/2$,
      $$c = c + 1.$$
      ii. **else**
         A. **if** $c = 0$, $\alpha = \text{parent}(\alpha)$.
         B. **else** $c = c - 1$.
4. (a) **if** $c > 10 \log n$, **Output:** α_1.
   (b) **else**
      i. Sort Q.
      ii. **Output:** $\text{BINARY-SEARCH}(S, Q, e, \epsilon)$.

Algorithm 9 BUILD-BINARY -SEARCH-TREE

Input: size n.

// Recall that each node m in the tree is an interval between left end m_1 and right end m_2.

1. Initialize set $T' = \emptyset$.
2. Initialize the tree T with the root node $(1, n)$.
$$m = (1, n) \qquad \text{where } m_1 = 1 \text{ and } m_2 = n,$$
$$\text{root}(T) = m$$
3. Add m to T'.
4. **while** T' is not empty
   (a) Consider a node i in T'.
   (b) **if** $i_2 - i_1 > 1$, create a left child and right child to i and set their parents as i.
   $$\alpha = \left(i_1, \left\lceil \frac{i_1 + i_2}{2} \right\rceil\right), \qquad \beta = \left(\left\lceil \frac{i_1 + i_2}{2} \right\rceil, i_2\right),$$
   $$\text{left}(i) = \alpha, \qquad \text{right}(i) = \beta,$$
   $$\text{parent}(\alpha) = i, \qquad \text{parent}(\beta) = i.$$
   and add nodes α and β to T'.
   (c) Remove node i from T'.

Output: T.

Algorithm 10 BINARY-SEARCH

Input: Ordered array S, ordered array Q, search item e, bias ϵ.
Initialize: $l = 1, h = |Q|$.

1. **while** $h - l > 0$
   (a) $t = \text{COMPARE2}\Big(e, S(Q(\lceil \frac{l+h}{2} \rceil), \frac{10 \log n}{\epsilon^2}\Big)$.
   (b) **if** $t \in [\frac{1}{2} - 3\epsilon, \frac{1}{2} + 3\epsilon]$, then **Output:** $Q(\lceil \frac{l+h}{2} \rceil)$.
   (c) **else if** $t < \frac{1}{2} - 3\epsilon$, then move to the left.
   $$h = \left\lceil \frac{l + h}{2} \right\rceil.$$
   (d) **else** move to the right.
   $$l = \left\lceil \frac{l + h}{2} \right\rceil.$$

Output: $Q(h)$.

C. Some tools for proving lemmas

We first prove an auxilliary result that we use in the future analysis.

Lemma 18. *Let W = COMPARE(i, j, ϵ, δ) and L be the other element. Then with probability $\geq 1 - \delta$,*

$$p(W, L) \geq \frac{1}{2} - \epsilon.$$

Proof. Note that if $|\tilde{p}(i, j)| < \epsilon$, then $p(i, j) > \frac{1}{2} - \epsilon$ and $p(j, i) > \frac{1}{2} - \epsilon$. Hence, $p(W, L) \geq \frac{1}{2} - \epsilon$.

If $|\tilde{p}(i, j)| \geq \epsilon$, without loss of generality, assume that i is a better element i.e., $\tilde{p}(i, j) \geq \epsilon$. By Lemma 2, with probability atleast $1 - \delta$, $W = i$. Hence

$$Pr\left(p(W, L) \geq \frac{1}{2} - \epsilon\right) = Pr(W = i) \geq 1 - \delta. \qquad \square$$

We now prove a Lemma that follows from SST and STI that we will use in future analysis.

Lemma 19. *If $\tilde{p}(i, j) \leq \epsilon_1$, $\tilde{p}(j, k) \leq \epsilon_2$, then $\tilde{p}(i, k) \leq \epsilon_1 + \epsilon_2$.*

Proof. We will divide the proof into four cases based on whether $\tilde{p}(i, j) > 0$ and $\tilde{p}(j, k) > 0$.

If $\tilde{p}(i, j) \leq 0$ and $\tilde{p}(j, k) \leq 0$, then by SST, $\tilde{p}(i, k) \leq 0 \leq \epsilon_1 + \epsilon_2$.

If $0 < \tilde{p}(i, j) \leq \epsilon_1$ and $0 < \tilde{p}(j, k) \leq \epsilon_2$, then by STI, $\tilde{p}(i, k) \leq \epsilon_1 + \epsilon_2$.

If $\tilde{p}(i, j) < 0$ and $0 < \tilde{p}(j, k) \leq \epsilon_2$, then by SST, $\tilde{p}(i, k) \leq \epsilon_2 \leq \epsilon_1 + \epsilon_2$.

If $0 < \tilde{p}(i, j) \leq \epsilon_1$ and $\tilde{p}(j, k) < 0$, then by SST, $\tilde{p}(i, k) \leq \epsilon_1 \leq \epsilon_1 + \epsilon_2$. $\square$

D. Proofs of Section 3

Proof of Lemma 2

Proof. Let $\hat{p}_i^r$ and $\hat{c}^r$ denote $\hat{p}_i$ and $\hat{c}$ respectively after r number of comparisons. Output of COMPARE(i, j, ϵ, δ) will not be i only if $\hat{p}_i^r < \frac{1}{2} + \epsilon - \hat{c}^r$ for any $r < m = \frac{1}{2\epsilon^2} \log \frac{2}{\delta}$ or if $\hat{p}_i < \frac{1}{2}$ for $r = m$. We will show that the probability of each of these events happening is bounded by $\frac{\delta}{2}$. Hence by union bound, Lemma follows.

After r comparisons, by Hoeffding's inequality,

$$Pr(\hat{p}_i^r < \frac{1}{2} + \epsilon - \hat{c}^r) \leq e^{-2r(\hat{c}^r)^2} = e^{-\log \frac{4r^2}{\delta}} = \frac{\delta}{4r^2}.$$

Using union bound,

$$Pr(\exists r \text{ s.t. } \hat{p}_i^r \leq \frac{1}{2} + \epsilon - \hat{c}^r) \leq \frac{\delta}{2}$$

After $m = \frac{1}{2\epsilon^2} \log \frac{2}{\delta}$ rounds, by Hoeffding's inequality,

$$Pr(\hat{p}_i^m < \frac{1}{2}) \leq e^{-2m\epsilon^2} = \frac{\delta}{2}. \qquad \square$$

Proof of Lemma 3

Proof. Each of the $\frac{|S|}{2}$ pairs is compared at most $\frac{1}{2\epsilon^2}\log\frac{2}{\delta}$ times, hence the total comparisons is $\le \frac{|S|}{4\epsilon^2}\log\frac{2}{\delta}$. Let $k^* = \max(\text{KNOCKOUT-ROUND}(S,\epsilon,\delta))$ and $s^* = \max(S)$. Let a be the element paired with s^*. There are two cases: $\tilde{p}(s^*,a) \ge \epsilon$ and $\tilde{p}(s^*,a) < \epsilon$.

If $\tilde{p}(s^*,a) \ge \epsilon$, by Lemma 2 with probability $\ge 1-\delta$, s^* will win and hence by definitions of s^* and k^*, $\tilde{p}(s^*,k^*) = 0 \le \gamma\epsilon$. Alternatively, if $\tilde{p}(s^*,a) < \epsilon$, let winner$(s^*,a)$ denote the winner between s^* and a. Then,

$$r(a) \overset{(a)}{\le} r(\text{winner}(s^*,a)) \overset{(b)}{\le} r(k^*) \overset{(c)}{\le} r(s^*)$$

where (a) follows from $r(a) \le r(s^*)$, (b) and (c) follow from the definitions of s^* and k^* respectively. From stochastic tranisitivity on a, k^* and s^*, $\tilde{p}(s^*,k^*) \le \gamma\tilde{p}(s^*,a) \le \gamma\epsilon$. □

Proof of Theorem 4

Proof. We first bound the number of comparisons. Let $n_i = \frac{|S|}{2^{i-1}}$ be the number of elements in the set at the beginning of round i. The number of comparisons at round i is

$$\le \frac{n_i}{2}\cdot\frac{\gamma^4 2^{2i/3}}{2c^2\epsilon^2}\cdot\log\frac{2^{i+1}}{\delta}.$$

Hence the number of comparisons in all rounds is

$$\begin{aligned}\sum_{i=1}^{\log|S|}\frac{|S|}{2^i}\cdot\frac{\gamma^4 2^{2i/3}}{2c^2\epsilon^2}\cdot\log\frac{2^{i+1}}{\delta} &\le \frac{|S|\gamma^4}{2c^2\epsilon^2}\sum_{i=1}^{\infty}\frac{1}{2^{i/3}}\left(i+\log\frac{2}{\delta}\right)\\ &= \frac{|S|\gamma^4}{2c^2\epsilon^2}\left(\frac{2^{1/3}}{c^2}+\frac{1}{c}\log\frac{2}{\delta}\right)\\ &= \mathcal{O}\left(\frac{|S|\gamma^4}{\epsilon^2}\left(1+\log\frac{1}{\delta}\right)\right).\end{aligned}$$

We now show that with probability $\ge 1-\delta$, the output of KNOCKOUT is an ϵ-maximum. Let $\epsilon_i = c\epsilon/(\gamma 2^{i/3})$ and $\delta_i = \delta/2^i$. Note that ϵ_i and δ_i are bias and confidence values used in round i. Let b_i be a maximum element in the set S before round i. Then by Lemma 3, with probability $\ge 1-\delta_i$,

$$\tilde{p}(b_i,b_{i+1}) \le \epsilon_i \tag{1}$$

By union bound, the probability that Equation 1 does not hold for some round $1 \le i \le \log|S|$ is

$$\le \sum_{i=1}^{\log|S|}\delta_i = \sum_{i=1}^{\log|S|}\frac{\delta}{2^i} \le \delta.$$

With probability $\ge 1-\delta$, Equation 1 holds for all i and by stochastic triangle inequality,

$$\tilde{p}(b_1,b_{\log|S|+1}) \le \sum_{i=1}^{\log|S|}\tilde{p}(b_i,b_{i+1}) \le \sum_{i=1}^{\infty}\frac{c\epsilon}{\gamma 2^{i/3}} = \epsilon/\gamma.$$

We now show that if $\tilde{p}(b_1,e) \le \epsilon/\gamma$, e is an ϵ-maximum, namely $\tilde{p}(f,e) \le \epsilon\ \forall f \in S$. Note that b_1 is a maximum element in the original set S and hence $r(b_1) = n$. If $r(f) \ge r(e)$, then by γ-stochastic transitivity, $\tilde{p}(f,e) \le \gamma\tilde{p}(b_1,e) \le \epsilon$ and if $r(f) \le r(e)$, then $\tilde{p}(f,e) \le 0 \le \epsilon$. □

E. Proofs of Section 4.1

Proof of Lemma 17

Proof. Let $Q = \text{MERGE}(S_1,S_2,\epsilon,\delta)$. We will show that for every k, w.p. $\ge 1-\delta$, $\tilde{p}(Q(k),Q(l)) \le \max(err(S_1),err(S_2)) + \epsilon\ \forall l > k$. Note that if this property is true for every element then $err(Q) \le \max(err(S_1),err(S_2)) + \epsilon$. Since there are $|S_1|+|S_2|$ elements in the final merged set, the Lemma follows by union bound.

If $S_1(i)$ and $S_2(j)$ are compared in MERGE algorithm, without loss of generality, assume that $S_1(i)$ loses i.e., $S_1(i)$ appears before $S_2(j)$ in T. The elements that appear to the right of $S_1(i)$ in Q belong to set $Q_{\ge S_1(i)} = \{S_1(k): k > i\}\bigcup\{S_2(k): k \ge j\}$. We will show that w.p. $\ge 1-\delta$, $\forall e \in Q_{\ge S_1(i)}$, $\tilde{p}(S_1(i),e) \le \max(err(S_1),err(S_2)) + \epsilon$.

By definition of error of an ordered set,

$$\tilde{p}(S_1(i),S_1(k)) \le err(S_1) \quad \forall k > i \tag{2}$$

$$\tilde{p}(S_2(j),S_2(k)) \le err(S_2) \quad \forall k \ge j. \tag{3}$$

By Lemma 18, w.p. $\ge 1-\delta$,

$$\tilde{p}(S_1(i),S_2(j)) \le \epsilon. \tag{4}$$

Hence by Equations 3, 4 and Lemma 19, w.p. $\ge 1-\delta$, $\tilde{p}(S_1(i),S_2(k)) \le \epsilon + err(S_2)\ \forall k \ge j$. □

Proof of Lemma 5

Proof. We first bound the total comparisons. Let $C(Q,\epsilon',\delta')$ be the number of comparisons that the MERGE-RANK uses on a set Q. Since MERGE-RANK is a recursive algorithm,

$$\begin{aligned}C(Q,\epsilon',\delta') \le\ & C(Q[1:\lfloor|Q|/2\rfloor],\epsilon',\delta')\\ &+ C(Q[\lfloor|Q|/2\rfloor:|Q|],\epsilon',\delta') + \frac{|Q|}{2\epsilon'^2}\log\frac{2}{\delta'}.\end{aligned}$$

From this one can obtain that $C(S,\epsilon',\delta') = \mathcal{O}\left(\frac{|S|\log|S|}{\epsilon'^2}\log\frac{1}{\delta'}\right)$. Hence,

$$C\left(|S|,\frac{\epsilon}{\log|S|},\frac{\delta}{|S|^2}\right) = \mathcal{O}\left(\frac{|S|\log^3|S|}{\epsilon^2}\log\frac{|S|^2}{\delta}\right).$$

Now we bound the error. By Lemma 17, with probability $\ge 1-|Q|\delta$,

$$\begin{aligned}&err(\text{MERGE-RANK}(Q,\epsilon',\delta')) \le\\ &\max\{err\big(\text{MERGE-RANK}\big(Q[1:\lfloor|Q|/2\rfloor],\epsilon',\delta'\big)\big),\\ &err\big(\text{MERGE-RANK}\big(T[\lfloor|Q|/2\rfloor+1:|Q|],\epsilon',\delta'\big)\big)\} + \epsilon'.\end{aligned} \tag{5}$$

We can bound the total times MERGE is called in a single instance of MERGE-RANK(S,ϵ',δ'). MERGE combines the singleton sets and forms the sets with two elements, it combines the sets with two elements and forms the sets with four elements and henceforth. Hence the total times MERGE is called is $\sum_{i=1}^{\log|S|}\frac{|S|}{2^i} \le |S|$. Therefore, the probability that Equation 5 holds every time when two ordered sets are merged in MERGE-RANK(S,ϵ',δ') is $\le |S|\cdot|S|\delta' = |S|^2\delta'$.

If Equation 5 holds every time MERGE is called, then error of MERGE-RANK(S,ϵ',δ') is at most $\sum_{i=1}^{\log|S|}\epsilon' \le \epsilon'\log|S|$. This is because $err(S)$ is 0 if S has only one element. And a singleton set participates in $\log n$ merges before becoming the final output set.

Therefore, w.p. $\geq 1 - |S|^2\delta'$,

$$err(\text{MERGE-RANK}(S, \epsilon', \delta')) \leq \log |S| \epsilon'.$$

Hence with probability $\geq 1 - \delta$,

$$err\left(\text{MERGE-RANK}\left(S, \frac{\epsilon}{\log |S|}, \frac{\delta}{|S|^2}\right)\right) \leq \epsilon. \qquad \square$$

F. Proofs for Section 4.2

Proof of Lemma 6

Proof. Let set S be ordered s.t. $\tilde{p}(S(i), S(j)) \geq 0 \; \forall i > j$. Let $S''_k = \{S(l) : k \leq l \leq k + 5(\log n)^{x+1} - 1\}$. The probability that none of the elements in S''_k is selected for a given k is

$$\leq \left(1 - \frac{5(\log n)^{x+1}}{n}\right)^{n/(\log n)^x} < \frac{1}{n^5}.$$

Therefore by union bound, the probability that none of the elements in S''_k is selected for any k is

$$\leq n \cdot \frac{1}{n^5} = \frac{1}{n^4}. \qquad \square$$

Proof of Lemma 8

We prove Lemma 8 by dividing it into smaller lemmas. We refer to $|\tilde{p}(e, f)|$ as a measure of distance between elements e and f.

We divide all elements in S into two sets based on distance from anchors. First set contains all elements that are far away from all anchors and the second set contains all elements which are close to atleast one of the anchors. INTERVAL-BINARY-SEARCH acts differently on both sets.

We first show that for elements in the first set, INTERVAL-BINARY-SEARCH places them in between the right anchors by using just the random walk subroutine.

For elements in the second set, INTERVAL-BINARY-SEARCH might fail to find the right anchors just by using the random walk subroutine. But we show that INTERVAL-BINARY-SEARCH visits a close anchor during random walk and BINARY-SEARCH finds a close anchor from the set of visited anchors using simple binary search.

We first prove Lemma 8 for the elements of first set.

Lemma 20. *For $\epsilon'' > \epsilon'$, consider an ϵ'-ranked S'. If an element e is such that $|\tilde{p}(e, S'(j))| > \epsilon'' \; \forall j$, then with probability $\geq 1 - \frac{1}{n^6}$ step 4a of* INTERVAL-BINARY-SEARCH(S', e, ϵ'') *outputs the index y such that $\tilde{p}(e, S'(y)) > \epsilon''$ and $\tilde{p}(S'(y+1), e) > \epsilon''$.*

Proof. We first show that there is an unique y s.t. $\tilde{p}(e, S'(y)) > \epsilon''$ and $\tilde{p}(S'(y+1), e) > \epsilon''$.

Let i be the largest index such that $\tilde{p}(e, S'(i)) > \epsilon''$. By Lemma 19, $\tilde{p}(e, S'(j)) > \epsilon'' - \epsilon' > 0 \quad \forall j < i$. Hence by the assumption on e, $\tilde{p}(e, S'(j)) > \epsilon'' \quad \forall j < i$. Let k be the smallest index such that $\tilde{p}(S'(k), e) > \epsilon''$. By a similar argument as previously, we can show that $\tilde{p}(S'(j), e) > \epsilon'' \quad \forall j > k$.

Hence by the above arguments and the fact that $|\tilde{p}(e, S'(j))| > \epsilon'' \quad \forall j$, there exists only one y such that $\tilde{p}(e, S'(y)) > \epsilon''$ and $\tilde{p}(S'(y+1), e) > \epsilon''$.

Thus in the tree T, there is only one leaf node w such that $\tilde{p}(e, S'(w_1)) > \epsilon''$ and $\tilde{p}(S'(w_2), e) > \epsilon''$.

Consider some node m which is not an ancestor of w. Then either $\tilde{p}(S'(m_1), e) > \epsilon''$ or $\tilde{p}(S'(m_2), e) < -\epsilon''$. Since we compare e with $S'(m_1)$ and $S'(m_2)$ $\frac{10}{\epsilon''^2}$ times, we move to the parent of m with probability atleast $\frac{19}{20}$.

Consider some node m which is an ancestor of w. Then $\tilde{p}(S'(m_1), e) < -\epsilon''$, $\tilde{p}(S'(m_2), e) > \epsilon''$, and $|\tilde{p}(S'(\lceil \frac{m_1+m_2}{2} \rceil), e)| > \epsilon''$. Therefore we move in direction of w with probability atleast $\frac{19}{20}$.

Therefore if we are not at w, then we move towards w with probability atleast $\frac{19}{20}$ and if we are at w then the count c increases with probability atleast $\frac{19}{20}$.

Since we start at most $\log n$ away from w if we move towards w for $21 \log n$ then the algorithm will output y. The probability that we will move towards w less than $21 \log n$ times is $\leq e^{-30 \log n D(\frac{21}{30} || \frac{19}{20})} \leq \frac{1}{n^6}$. $\square$

To prove Lemma 8 for the elements of the second set, we first show that the random walk subroutine of algorithm INTERVAL-BINARY-SEARCH placing an element in wrong bin is highly unlikely.

Lemma 21. *For $\epsilon'' > \epsilon'$, consider an ϵ'-ranked set S'. Now consider an element e and y such that either $\tilde{p}(S'(y), e) > \epsilon''$ or $\tilde{p}(S'(y+1), e) < -\epsilon''$, then step 4a of* INTERVAL-BINARY-SEARCH(S', e, ϵ'') *will not output y with probability $\geq 1 - \frac{1}{n^7}$.*

Proof. Recall that step 4a of INTERVAL-BINARY-SEARCH outputs y if we are at the leaf node $(y, y+1)$ and the count c is atleast $10 \log n$.

Since either $\tilde{p}(S'(y), e) > \epsilon''$ or $\tilde{p}(S'(y+1), e) < -\epsilon''$, when we are at leaf node $(y, y+1)$, the count decreases with probability atleast $\frac{19}{20}$. Hence the probability that INTERVAL-BINARY-SEARCH is at $(y, y+1)$ and the count is greater than $10 \log n$ is at most $\sum_{i=10\log n}^{30 \log n} e^{-i \cdot D(\frac{i - 10 \log n}{2i} || \frac{19}{20})} < 20 \log n e^{-10 \log n D(\frac{1}{3} || \frac{19}{20})} \leq \frac{1}{n^7}$. $\square$

We now show that for an element of the second set, the random walk subroutine either places it in correct bin or visits a close anchor.

Lemma 22. *For $\epsilon'' > \epsilon'$, consider an ϵ'-ranked set S'. Now consider an element e that is close to an element in S' i.e., $\exists g : |\tilde{p}(S'(g), e)| < \epsilon''$. With probability $\geq 1 - \frac{1}{n^6}$, step 4a of* INTERVAL-BINARY-SEARCH(S', e, ϵ'') *will either output the right index y such that $\tilde{p}(S'(y), e) < \epsilon''$ and $\tilde{p}(S'(y+1), e) > -\epsilon''$ or* INTERVAL-BINARY-SEARCH *visits $S'(h)$ such that $|\tilde{p}(S'(h), e)| < 2\epsilon''$.*

Proof. By Lemma 21, step 4a of INTERVAL-BINARY-SEARCH does not output a wrong interval with probability $1 - \frac{1}{n^7}$. Hence we just need to show that w.h.p., e visits a close anchor.

Let i be the largest index such that $\tilde{p}(e, S'(i)) > 2\epsilon''$. Then $\forall j < i$, by Lemma 19, $\tilde{p}(e, S(j)) > 2\epsilon'' - \epsilon' > \epsilon''$.

Let k be the smallest index such that $\tilde{p}(S'(k), e) > 2\epsilon''$. Then $\forall j > k$, by Lemma 19, $\tilde{p}(S'(j), e) > \epsilon''$.

Therefore for $u < v$ such that $\min(|\tilde{p}(S'(u), e)|, |\tilde{p}(S'(v), e)|) \geq 2\epsilon''$ only one of three sets $\{x : x < u\}$,$\{x : u < x < v\}$ and $\{x : x > v\}$ contains an index z such that $|\tilde{p}(S'(z), e)| < \epsilon''$.

Let a node α be s.t. for some $c \in \{\alpha_1, \alpha_2, \lceil\frac{\alpha_1+\alpha_2}{2}\rceil\}$, $|\tilde{p}(S'(c), e)| \leq 2\epsilon''$. If INTERVAL-BINARY-SEARCH reaches such a node α then we are done.

So assume that INTERVAL-BINARY-SEARCH is at a node β s.t. $\forall c \in \{\beta_1, \beta_2, \lceil\frac{\beta_1+\beta_2}{2}\rceil\}$, $|\tilde{p}(S'(c), e)| > 2\epsilon''$. Note that only one of three sets $\{x : x < \beta_1 \text{ or } x > \beta_2\}$, $\{x : \beta_1 < x < \lceil\frac{\beta_1+\beta_2}{2}\rceil\}$ and $\{x : \lceil\frac{\beta_1+\beta_2}{2}\rceil < x < \beta_2\}$ contains an index z such that $|\tilde{p}(S'(z), e)| < \epsilon''$ and INTERVAL-BINARY-SEARCH moves towards that set with probability $\frac{19}{20}$. Hence the probability that we never visit an anchor that is less than $2\epsilon''$ away is at most $e^{-30\log n D(\frac{15.5}{30}||\frac{19}{20})} \leq \frac{1}{n^7}$. ∎

We now complete the proof by showing that for an element e from the second set, if Q contains an index y of an anchor that is close to e, BINARY-SEARCH will output one such index.

Lemma 23. *For $\epsilon'' > \epsilon'$, consider ordered sets S', Q s.t. $p(S'(Q(i)), S'(Q(j))) > \frac{1}{2} - \epsilon'\ \forall i > j$. For an element e s.t., $\exists g : |\tilde{p}(S'(Q(g)), e)| < 2\epsilon''$, BINARY-SEARCH$(S', Q, e, \epsilon'')$ will return y such that $|\tilde{p}(S'(Q(y)), e)| < 4\epsilon''$ with probability $\geq 1 - \frac{1}{n^6}$.*

Proof. At any stage of BINARY-SEARCH, there are three possibilities that can happen . Consider the case when we are comparing e with $S'(Q(i))$.

1. $|\tilde{p}(S'(Q(i)), e)| < 2\epsilon''$. Probability that the fraction of wins for e is not between $\frac{1}{2} - 3\epsilon''$ and $\frac{1}{2} + 3\epsilon''$ is less than $e^{-\frac{10\log n}{\epsilon''^2}\epsilon''^2} \leq \frac{1}{n^{10}}$. Hence BINARY-SEARCH outputs $Q(i)$.

2. $\tilde{p}(S'(Q(i)), e) > 2\epsilon''$. Probability that the fraction of wins for e is more than $\frac{1}{2}$ is less than $e^{-\frac{10\log n}{\epsilon''^2}\epsilon''^2} \leq \frac{1}{n^{10}}$. So BINARY-SEARCH will not move right. Also notice that $\tilde{p}(S'(Q(j)), e) > 2\epsilon'' - \epsilon' > \epsilon''\ \forall j > i$.

3. $\tilde{p}(S'(Q(i)), e) > 4\epsilon''$. Probability that the fraction of wins for e is more than $\frac{1}{2} - 3\epsilon''$ is less than $e^{-\frac{10\log n}{\epsilon''^2}\epsilon''^2} \leq \frac{1}{n^{10}}$. Hence BINARY-SEARCH will move left. Also notice that $\tilde{p}(S'(Q(j)), e) > 4\epsilon'' - \epsilon' > \epsilon''\ \forall j > i$.

We can show similar results for $\tilde{p}(S'(Q(i)), e) < -2\epsilon''$ and $\tilde{p}(S'(Q(i)), e) < -4\epsilon''$. Hence if $|\tilde{p}(S'(Q(i)), e)| < 2\epsilon''$ then BINARY-SEARCH outputs $Q(i)$, and if $2\epsilon'' < |\tilde{p}(S'(Q(i)), e)| < 4\epsilon''$ then either BINARY-SEARCH outputs $Q(i)$ or moves in the correct direction and if $|\tilde{p}(S'(Q(i)), e)| > 4\epsilon''$, then BINARY-SEARCH moves in the correct direction. ∎

Lemma 24. INTERVAL-BINARY-SEARCH(S, e, ϵ) *terminates in $\mathcal{O}(\frac{\log n \log\log n}{\epsilon^2})$ comparisons for any set S of size $O(n)$.*

Proof. Step 3 of INTERVAL-BINARY-SEARCH runs for $30\log n$ iterations. In each iteration, INTERVAL-BINARY-SEARCH compares e with at most 3 anchors and repeats each comparison for $10/\epsilon^2$. So total comparisons in step 3 is $\mathcal{O}(\log n/\epsilon^2)$. The size of Q is upper bounded by $90\log n$ and BINARY-SEARCH does a simple binary search over Q by repeating each comparison $10\log n/\epsilon^2$. Hence total comparisons used by BINARY-SEARCH is $\mathcal{O}(\log n \log\log n/\epsilon^2)$ ∎

Combining Lemmas 7, 21, 22, 23, 24 yields the result.

Proof of Lemma 11

Proof. Combining Lemmas 7, 10 and using union bound, at the end of step 5a ,w.p. $\geq 1 - \frac{2}{n^3}$, S' is ϵ'-ranked and $\forall j, e \in B_j$, $\min(\tilde{p}(e, S'(j)), \tilde{p}(S'(j+1), e)) > 5\epsilon''$. Hence by Lemma 19, $\forall j, k < j, e \in B_j$, $\tilde{p}(e, S'(k)) > 5\epsilon'' - \epsilon' > 4\epsilon''$. Similarly, $\forall j, k > j, e \in B_j$, $\tilde{p}(S'(k), e) > 5\epsilon'' - \epsilon' > 4\epsilon''$.

If $|B_j| > 0$, then $\tilde{p}(e, S'(k)) > 4\epsilon''$ for $e \in B_j, k \leq j$, $\tilde{p}(S'(l), e) > 4\epsilon''$ for $e \in B_j, l \geq j+1$. Hence by stochastic transitivity, $\tilde{p}(S'(l), S'(k)) > 4\epsilon''$ for $l > j \geq k$. Therefore there exists k, l s.t. $\tilde{p}(S'(l), f) > 0\ \forall f \in \{S'(y) : y \leq j\}$, $\tilde{p}(S'(k), S'(l)) > 0$ and $\tilde{p}(f, S'(k)) > 0\ \forall f \in \{S'(y) : y > j\}$. Now by Lemma 6, w.p. $\geq 1 - \frac{1}{n^4}$, size of all such sets B_j is less than $5(\log n)^{x+1}$.

Lemma follows by union bound. ∎

Proof of Theorem 13

We first bound the running time of BINARY-SEARCH-RANKING algorithm.

Theorem 25. BINARY-SEARCH-RANKING *terminates after $O(\frac{n(\log\log n)^x}{\epsilon^2}\log n)$ comparisons with probability $\geq 1 - \frac{1}{n^2}$.*

Proof. Step 2 RANK-$x(S', \epsilon', \frac{1}{n^6})$ terminates after $\mathcal{O}(\frac{n}{\epsilon^2}\log n)$ comparisons with probability $\geq 1 - \frac{1}{n^6}$.

By Lemma 8, for each element e, the step 4a INTERVAL-BINARY-SEARCH(S', e, ϵ'') terminates after $\mathcal{O}(\frac{\log n \log\log n}{\epsilon^2})$ comparisons. Hence step 4 takes at most $\mathcal{O}(\frac{n\log n \log\log n}{\epsilon^2})$ comparisons.

Comparing each element with the anchors in steps 5a takes at most $O(\frac{\log n}{\epsilon^2})$ comparisons.

With probability $\geq 1 - \frac{1}{n^4}$ step 5b RANK-X$(B_i, \epsilon'', \frac{1}{n^4})$ terminates after $\mathcal{O}(|B_i|\frac{(\log|B_i|)^x}{\epsilon^2}\log n)$ comparisons. By Lemma 11, $|B_i| \leq 5(\log n)^{x+1}$ for all i w.p. $\geq 1 - \frac{3}{n^3}$. Hence, w.p. $\geq 1 - \frac{3}{n^3}$, total comparisons to rank all B_is is at most $\sum_i \mathcal{O}(|B_i|\frac{(\log|B_i|)^x}{\epsilon^2}\log n) \leq \sum_i \mathcal{O}(\frac{|B_i|\log n(\log(5(\log n)^{x+1}))^x}{\epsilon^2}) = \mathcal{O}(\frac{n\log n(\log\log n)^x}{\epsilon^2})$.

Therefore, by summing comparisons over all steps, with probability $\geq 1 - \frac{1}{n^2}$ total comparisons is at most $O\left(\frac{n\log n(\log\log n)^x}{\epsilon^2}\right)$. ∎

Now we show that BINARY-SEARCH-RANKING outputs an ϵ-ranking with high probability.

Theorem 26. BINARY-SEARCH-RANKING *produces an ϵ-ranking with probability at least $1 - \frac{1}{n^2}$.*

Proof. By combining Lemmas 7, 9, 10, 12 and using union bound, w.p. $\geq 1 - \frac{1}{n^2}$, at the end of step 5b,

- S' is ϵ'-ranked.
- Each C_i has elements such that $|\tilde{p}(C_i(j), S(i))| < 7\epsilon''$ for all j.
- Each B_i has elements such that $\tilde{p}(S'(i), B_i(j)) < -5\epsilon''$ and $\tilde{p}(S'(i+1), B_i(j)) > 5\epsilon''$ for all j.
- All B_is are ϵ''-ranked.

For $j \ge i$, $e \in B_{i-1} \bigcup S'(i) \bigcup C_i$, $f \in S'(j) \bigcup C_j \bigcup B_j$, $\tilde{p}(e,f) \le \tilde{p}(e, S'(i)) + \tilde{p}(S'(i), S'(j)) + \tilde{p}(S'(j), f) \le 7\epsilon'' + \epsilon' + 7\epsilon'' < 15\epsilon'' = \epsilon$. Combining the above results proves the Theorem. □

Combining Theorems 25, 26 yields the result.

Proof Sketch for Theorem 16

Proof sketch. Consider a stochastic model where there is an inherent ranking r and for any two consecutive elements $p(i, i+1) = \frac{1}{2} - 2\epsilon$. Suppose there is a genie that knows the true ranking r up to the sets $\{r(2i-1), r(2i)\}$ for all i i.e., for each i, genie knows $\{r(2i-1), r(2i)\}$ but it does not know the ranking between these two elements. Since consecutive elements have $\epsilon(i, i+1) = 2\epsilon > \epsilon$, to find an ϵ-ranking, the genie has to correctly identify the ranking within all the $n/2$ pairs. Using Fano's inequality from information theory, it can be shown that the genie needs at least $\Omega\left(\frac{n}{\epsilon^2} \log \frac{n}{\delta}\right)$ comparisons to identify the ranking of the consecutive elements with probability $1 - \delta$. □

G. Additional Experiments

As we mentioned in Section 5, **BTM-PAC** allows comparison of an element with itself. It is not beneficial when the goal is to find ϵ-maximum. So we modify their algorithm by not allowing such comparisons. We refer to this restricted version as **R-BTM-PAC**.

As seen in figure, performance of **BTM-PAC** does not increase by much by restricting the comparisons.

We further reduce the constants in **R-BTM-PAC**. We change Equations (7) and (8) in (Yue & Joachims, 2011) to $c_\delta(t) = \sqrt{\frac{1}{t} \log \frac{n^3 N}{\delta}}$ and $N = \lceil \frac{1}{\epsilon^2} \log \frac{n^3 N}{\delta} \rceil$, respectively.

We believe the same guarantees hold even with the updated constants. We refer to this improved restricted version as **IR-BTM-PAC**. Here too we consider the stochastic model where $p(i,j) = 0.6 \forall\ i < j$ and we find 0.05-maximum with error probability $\delta = 0.1$.

In Figure 6 we compare the performance of KNOCKOUT and all variations of BTM-PAC. As the figure suggests, the performance of **IR-BTM-PAC** improves a lot but KNOCKOUT still outperforms it significantly.

In Figure 7, we consider the stochastic model where $p(i,j) = 0.6\ \forall i < j$ and find ϵ-maximum for different values of ϵ. Similar to previous experiments, we use $\delta = 0.1$. As we can see the number of comparisons increases almost linearly with n. Further the number of comparisons does not increase significantly even when ϵ decreases. Also the number of comparisons seem to be converging as ϵ goes to 0. KNOCKOUT outperforms **MallowsMPI** even for the very small ϵ values. We attribute this to the subroutine COMPARE that finds the winner faster when the distance between elements are much larger than ϵ.

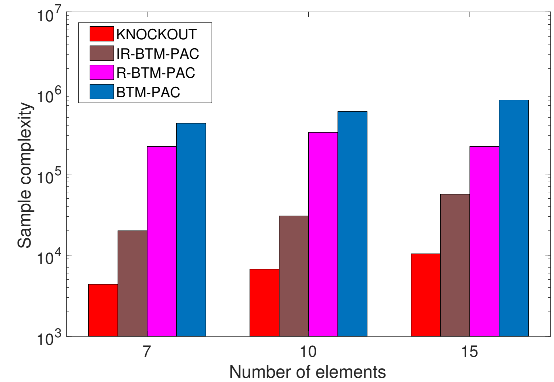

Figure 6. Sample complexity comparison of KNOCKOUT and variations of BTM-PAC for different input sizes, with $\epsilon = 0.05$ and $\delta = 0.1$

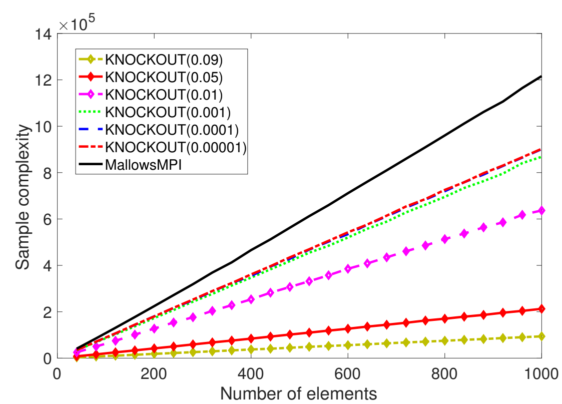

Figure 7. Sample complexity of KNOCKOUT for different values of n and ϵ