Active Ranking and Matchmaking, with Perfect Matchings

Anonymous Authors[1]

Abstract

We address the challenge of actively ranking a set of items/players with varying values/strengths. The comparison outcomes are random, with a greater noise the closer the values. A crucial requirement is that, at each iteration of the algorithm, all items must be compared once, i.e., an iteration is a perfect matching. Furthermore, we presume that comparing two players with closely matched strengths incurs no cost and, in contrast, a unit cost is associated with comparing players whose strength difference is more substantial. Our secondary objective is to determine an optimal matching between players based on this cost function: we propose and analyze an algorithm that draws on concepts from both AKS sorting networks and bandit theory. Our algorithm achieves both objectives with high probability, and the total cost is optimal (up to logarithmic terms).

1. Introduction

Background and motivation: Sorting through pairwise comparisons is a foundational challenge in computer science, extending its significance to diverse applications (Bengs et al., 2021). However, in many practical scenarios, these comparisons inherently possess a random nature. For instance, consider the assessment or ranking of players in games or sports, such as Chess, where Elo scores are employed (Elo, 1978). In such cases, when two players or teams engage repeatedly, the outcome is not consistently identical; the proximity of their values or strengths corresponds to a likelihood of winning converging toward 1/2.

Consequently, the task of ranking items or players based on these noisy, random comparisons has evolved into a critical objective (Minka et al., 2018; Bengs et al., 2021). Microsoft's TrueSkill software, for instance, is employed to match and rank thousands of Xbox gamers by leveraging their historical performance records, reflecting the outcomes of past matches (Herbrich et al., 2006; Minka et al., 2018). A noteworthy challenge arises from the dual nature of ranking and proposing engaging matches, both of which, though complementary, can be challenging to harmonize.

On one hand, the ranking process necessitates the exploration of potential comparisons. Merely pairing two players with each other, for instance, does not provide insights into how these players compare with others in the broader context. On the other hand, this exploration may result in matches between players with substantially different values. While informative for ranking purposes, such deterministic comparisons may lack interest for the involved players, as the outcome tends to be predictable, with one player consistently prevailing. Recognizing this challenge, platforms aim to rank players while concurrently pairing individuals of relatively similar skills (Minka et al., 2018). These principles align with the broader framework of "active ranking" (Falahatgar et al., 2017b; Zoghi et al., 2017; Szörényi et al., 2015; Saha & Gopalan, 2019; Ren et al., 2019).

The majority of active ranking algorithms hinge on determining the minimum number of comparisons (sample complexity) required to produce the accurate ranking with a predetermined confidence level (Falahatgar et al., 2017b;a; 2018; Ren et al., 2019).

However, a common trait of these algorithms is their tendency to leave certain players unpaired. Typically, the algorithm repetitively compares players who are challenging to discriminate, with these individuals being selected multiple times (Ren et al., 2019). Conversely, players with significantly greater (or lesser) strength than the majority are often left unpaired after a few games.

This characteristic proves undesirable in the context of video games, where a player left unattended for numerous rounds may disengage from the platform. To address this concern, we introduce an additional constraint on active ranking algorithms. Specifically, at each iteration (or round), these algorithms are required to pair all N players (equivalently, perform $N/2$ comparisons in parallel). In essence, the algorithm, at each round, selects a perfect matching of players based on the outcomes of previous rounds, collects the results of all pairs, and proceeds to the next round.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

Acknowledging that players generally prefer matchups against opponents of comparable skill levels, we assign a cost to each proposed pair, set to 0 if the difference in values between the players in the pair is small and 1 if the difference is substantial. Consequently, the cost of a matching is the sum of the costs associated with its pairs.

Model and notations: We consider a scenario involving N items or players (for simplicity, we assume that N is even), each associated with an unknown ranking denoted as $(r(1), r(2), \ldots, r(N))$. The primary objective is to discern this ranking through pairwise noisy comparisons. Specifically, when two items, i and j, are paired, the outcome of the comparison is "i beats j" with a probability of $p(i,j)$ and "j beats i" with a complementary probability of $p(j,i) = 1 - p(i,j)$. For the sake of simplicity, we assume the absence of ties, although this would not pose a significant challenge. An implicit, albeit crucial, assumption underlying this framework is the independence of results from queried comparisons (Falahatgar et al., 2017a;b; Heckel et al., 2019; Saha & Gopalan, 2019).

To reconstruct the ranking from the comparison results, we introduce a structure on the matrix $p(i,j)$. The initial assumption is that $p(i,j) > 1/2$ if i has a better rank than j, denoted as $r(i) < r(j)$. Consequently, the key parameters in noisy ranking are the $\varepsilon(i,j) = p(i,j) - 1/2$, the additional probability of i winning against j. Notably, $\varepsilon(j,i) = -\varepsilon(i,j)$ and $\varepsilon(i,j) \geq 0$ whenever $r(i) < r(j)$. The magnitude of $|\varepsilon(i,j)|$ is interpreted as the "skill gap" between players i and j. We shall consider the standard following assumptions on the $\varepsilon(i,j)$ matrix (Yue et al., 2012):

Assumption 1.1. Strong Stochastic Transitivity (SST) (Tversky & Russo, 1969) For any $i,j,k \in [N]$, if $r(i) < r(j) < r(k)$, then $\varepsilon(i,k) \geq \max(\varepsilon(i,j), \varepsilon(j,k))$.

Assumption 1.1 posits that if player i is stronger than j ($\varepsilon(i,j) \geq 0$) and player j is stronger than k ($\varepsilon(j,k) \geq 0$), then it is easier for player i to beat k than j. This assumption is commonplace and satisfied in various models, with additional details provided in Appendix A.

Assumption 1.2. Stochastic Triangle Inequality (STI) If $r(i) < r(j) < r(k)$, then $\varepsilon(i,k) \leq \varepsilon(i,j) + \varepsilon(j,k)$.

Assumption 1.2 reflects a "local skill ordering": when players i and j exhibit minimal differences in strength (i.e., $0 \leq \varepsilon(i,j), \varepsilon(j,k) \ll 1$), player i should not be significantly stronger than player k. Further discussion on this assumption is provided in Appendix A.

To learn the true ranking $r(\cdot)$, the algorithm sequentially selects, at each round t, a perfect matching M_t, i.e., a partition of the player set into pairs. In other words, $M_t = \{\{g_{t,1}, g_{t,2}\}, \{g_{t,3}, g_{t,4}\}, \ldots, \{g_{t,N-1}, g_{t,N}\}\}$, where $\{g_{t,1}, g_{t,2}, \ldots, g_{t,N}\} = \{1, 2, \ldots, N\}$. Henceforth, we refer to "perfect matchings" simply as "matchings".

Example. Bradley-Terry-Luce model The Bradley-Terry-Luce model (BTL) (Bradley & Terry, 1952) serves as a well-established parametric model for defining $p(i,j)$. In this model, each player possesses a corresponding strength denoted as θ_i, and the probability $p(i,j)$ is given by the formula $\frac{\exp(\theta_i)}{\exp(\theta_i)+\exp(\theta_j)}$. It is straightforward to verify that BTL adheres to both Assumption 1.1 and 1.2. More generally, the winning excess probabilities can be expressed as $\varepsilon(i,j) = F(\theta_i - \theta_j)$, where F, referred to as the "Model function," is any non-decreasing Lipschitz function satisfying $F(-x) = -F(x)$ and $F(+\infty) \leq \frac{1}{2}$.

Matching Cost: As previously mentioned, one of the motivating examples pertains to efficient matchmaking in video games (Minka et al., 2018; Herbrich et al., 2006). The objective is to devise a matchmaking system that selects a matching M_t at each round. For the sake of simplicity, we assume a fixed set of players, all eager to participate in each round against an opponent with a sufficiently similar skill level; otherwise, they express dissatisfaction (if the game is perceived as too easy or too hard). The platform's dual aim is to rank players while minimizing the number of dissatisfaction. To model this, we consider a fixed known threshold ε^*, such that players i and j are content with their pairing if, and only if, $|\varepsilon(i,j)| \leq \varepsilon^*$. Specifically, the cost of a matching M is the sum of the costs associated with the generated pairings, i.e., $C_M = \sum_{i,j \in M} \mathbb{1}\{|\varepsilon(i,j)| > \varepsilon^*\}$.

Problem statement and objectives: An algorithm, at each round t, selects a matching M_t and observes the outcomes of the induced random comparisons (and only those). The overarching objectives are to sequentially and adaptively choose matchings $M_1, M_2, \ldots, M_t$ so that, after some (random) number of rounds T, the algorithm provide two key outputs, correct with a probability of at least $1 - \delta$, where $\delta \in (0,1)$ is a predetermined confidence parameter. The first one is an estimated ranking $\hat{r}(\cdot)$ and the second one, a matching $\hat{M}^*$ minimizing the matching cost.

The algorithm's performance is evaluated based on either the sample complexity (the number of rounds T needed, or equivalently, the number of comparisons $NT/2$) or alternatively the cumulative regret, defined as:

$$R(\delta) = \sum_{t=1}^{T} C_{M_t} - TC^*, \tag{1}$$

where $C^* = \min_{M \in \mathcal{M}} C_M$. It is important to note that T is a random stopping time determined by the algorithm, not a fixed budget of comparisons.

Main Results Our main contributions are threefold:

The first contribution is an algorithm that outputs a weakened version $\hat{r}$ of the ranking, termed (ε, δ)-PAC ranking. In essence, any two items such that $\varepsilon(i,j) > \varepsilon$ satisfy $\hat{r}(i) < \hat{r}(j)$ with high probability. The sample complexity (number of comparisons) is $\mathcal{O}(\frac{N}{\varepsilon^2}\log^3 N \log(N/\delta))$, and the time complexity (number of rounds) is $\mathcal{O}(\frac{\log^3 N \log(N/\delta)}{\varepsilon^2})$.

The second contribution utilizes the aforementioned algorithm to obtain the exact ranking in $\mathcal{O}(\frac{\log^3(N)\log(N/\delta)}{\min_i \Delta_i^2})$ rounds where $\Delta_i = \varepsilon(\sigma(i), \sigma(i+1))$ where $\sigma(i)$ denotes the i-th ranked player, i.e., $\sigma = r^{-1}$.

The third contribution establishes that the latter algorithm can be employed to learn an optimal matching, with an additional regret term (hiding $\log\log$ terms) of order $\mathcal{O}\Big(\sum_i \frac{\log(N/\delta)}{\Delta_{\varepsilon^*,i}^2}\Big)$, where $\Delta_{\varepsilon^*,i} = \Delta_i - \varepsilon^*$.

1.1. Related works

Active ranking, exact and PAC: The first related algorithm (Feige et al., 1994) is tailored for the case of N totally-ordered elements, where comparisons between any two of them have the same known probability of error $\alpha < 1/2$. Its sample complexity is of order $\mathcal{O}(\frac{N\log(N/\delta)}{(1/2-\alpha)^2})$ to retrieve an exact ranking with a probability greater than $1-\delta$.

A major limitation of this algorithm lies in the assumption of a uniform probability of error on any comparison, known and bounded away from $1/2$. In many practical scenarios, comparisons are prone to errors with different probabilities, which can be arbitrarily close to $1/2$, and their values are unknown. These limitations were partially mitigated in (Falahatgar et al., 2017b;a; Saha & Gopalan, 2019; Ren et al., 2018) to develop algorithms that find (ε, δ)-PAC ranking in $\mathcal{O}(\frac{N\log(N/\delta)}{\varepsilon^2})$ comparisons, achieving optimal sample complexity (Falahatgar et al., 2017a). Subsequent improvements (Ren et al., 2019; Saha & Gopalan, 2019; 2020) led to reduced sample complexities, leveraging individual skill gaps, ultimately reaching a sample complexity of $\mathcal{O}(\sum \frac{\log\log(1/\Delta_i)+\log(N/\delta)}{\Delta_i^2})$. Unfortunately, these algorithms are sequential in nature, querying comparisons one by one without parallelization or matching constraints. Specifically, they construct a tree-like structure sequentially exploited for efficient ranking, either with Binary-Insertion-Sort (Ren et al., 2019) or Quick-Sort (Szörényi et al., 2015). In both cases, a form of "congestion" arises at the root of the tree, as every unsorted player must play a substantial number of times against the root's player. This is impractical in real-world applications, as in the video games example, as it leads to significant waiting times before getting paired.

Our algorithm addresses the problem of (ε, δ)-PAC ranking in the fully parallelized case with a sample complexity $\log^3(N)$ of the lower bound for sample complexity and time complexity (obtained in the uniform error case). The SST and STI assumptions we consider are relatively mild and commonly encountered (Falahatgar et al., 2017b;a).

Combinatorial bandits The matching problem can be reframed as a specific instance of combinatorial semi-bandit (Cesa-Bianchi & Lugosi, 2012), a domain that has recently garnered significant attention and witnessed improvements in regret minimization (Merlis & Mannor, 2021; Perrault et al., 2020; Wang & Chen, 2018). The combinatorial structure is evident, as the set of all matchings on a graph with N vertices has a cardinality of $\frac{(N)!}{2^{N/2}(N/2)!} \simeq \Omega\left(\left(\frac{N}{e}\right)^{N/2}\right)$. The main difference though is that the cost function is quite different. Nonetheless, the standard algorithms and arguments still hold (i.e., computing the number of times each action must be sampled would follow the same line of proof), computing the regret requires different – but straightforward – computations. In the ranking from matching problem, combinatorial bandit algorithms would incur a regret scaling, discarding $\log(N)$ terms, as $\mathcal{O}\left(\frac{N^2\log(1/\delta)}{\Delta_{\min}^2}\right)$, where $\Delta_{\min}$ denotes the expected gap between an optimal matching and the best sub-optimal matching (Merlis & Mannor, 2020; Kveton et al., 2015). We refer to Section 2.1 and Appendix E for more details. In contrast, our algorithm incurs a regret scaling as $\mathcal{O}\left(\frac{N\log(1/\delta)}{\Delta_0^2}\right)$, where $\Delta_0 := \min_i \min\{\Delta_i, \Delta_{\varepsilon^*,i}\}$ is another problem parameter that is typically much bigger than $\Delta_{\min}$. We refer to Section 2.1 for such examples.

Parallel Sorting algorithms Parallel comparison-based sorting algorithms, suitable for parallel computing, have been explored early on (Knuth et al., 1973; Batcher, 1968; Valiant, 1975). A specific class of parallel sorting algorithms is the "sorting networks" (Knuth et al., 1973; Ajtai et al., 1983), with the property that no element is involved in multiple comparisons at the same round. This property is crucial for relevance, as players to be ranked can only engage in one game at a time. For a comprehensive introduction to sorting networks, refer to (Knuth et al., 1973).

Various methods exist for constructing sorting networks, such as Batcher sorting networks (Batcher, 1968). The sample complexity of Batcher sorting networks is of order $\mathcal{O}\left(N\log^2(N)\right)$, implying a time complexity of at least $\mathcal{O}\left(\log^2(N)\right)$, as there are at most $N/2$ comparisons simultaneously. Another notable sorting network is the AKS-Paterson sorting network (Ajtai et al., 1983; Paterson, 1990), with a sample complexity of $\mathcal{O}\left(N\log(N)\right)$ and a time complexity of $\mathcal{O}\left(\log N\right)$. However, it comes with a hidden constant, denoted as D, which might be large (Natvig, 1990; Seiferas, 2009). Since the AKS sorting network has a time complexity of $\mathcal{O}(\log N)$, it enables the ranking of N elements using only $\mathcal{O}(\log N)$ matchings.

We shall leverage the AKS-Paterson algorithm as a black-

box to construct a selection scheme for sequentially choosing matchings and finding an $(\varepsilon, \delta)-$PAC ranking.

1.2. Organization of the paper

Section 2 is dedicated to the first objective: recovering the exact ranking. The necessity of this step is illustrated in Section 2.1, highlighting its role in obtaining an optimal matching. Subsequently, we delve into the description of the ranking algorithm in Section 2.2, where we first outline the relevant properties of AKS-Paterson (Ajtai et al., 1983; Paterson, 1990) for this problem. We then explain how to employ AKS-Paterson (or any other sorting network with $\mathcal{O}(\log N)$ time complexity) to obtain an (ε, δ)-PAC ranking, and then how to reach an exact ranking by refining ε further and further.

In Section 3, we demonstrate how to leverage an exact ranking to output the optimal matching with a small (almost minimax optimal) regret.

More general comments, proofs and most pseudocodes are postponed to the Appendix, due to space limit.

2. Ranking

In this section, we start by giving the general scheme to obtain an (ε, δ)-PAC ranking by carefully adapting the classical AKS-Paterson sorting network. Next, we show how to obtain the exact ranking using additional techniques. We give a general bound for the sample complexity (number of comparisons used to retrieve an (ε, δ)-PAC ranking and to retrieve the exact ranking) that depends on the instance.

2.1. Exact ranking is needed for optimal matching

As the cost function satisfies $c(i, j) = 0$ if $\varepsilon(i, j) \leq \varepsilon^*$, it might give the impression that the optimal matching could be recovered from any $(\varepsilon^*/2)$-correct ranking (or more generally, by some $\Omega(\varepsilon^*)$-correct ranking). Unfortunately, this is not the case, as proved by the following instances, illustrated in Figure 1. This proves that the exact ranking is (sometimes) necessary; of course, this is not always the case (for instance if $\varepsilon(i, j) < \varepsilon^*$ for all pairs (i, j)).

Given some arbitrarily small parameter $\eta > 0$, the instance is described by $\varepsilon(1, i) = \varepsilon^* + \eta/2$ for any $i > 2$, by $\varepsilon(1, 2) = \varepsilon^* - \eta/2$; the other gaps all being equal to η, i.e., $\varepsilon(i, j) = \eta$, for all $j > i > 1$.

In this instance, the optimal matching has a cost of 0, it matches player 1 to player 2. As a consequence, it requires ranking all $N - 1$ weakest players (or at least finding the exact best one) to pair 1 with 2. Notice that ranking the weakest $N - 1$ players requires an η-correct ranking.

We also illustrate the complexity of the problem with another simple instance with only 4 vertices (players are ranked in their index number, i.e., $r(i) = i$). The skill gaps are defined, for $\eta \ll \Delta \ll \varepsilon^*$ as

$$\varepsilon(1, 2) = \varepsilon(3, 4) = \varepsilon^* - \Delta, \text{ and } \varepsilon(2, 3) = \Delta + \eta$$
$$\varepsilon(1, 3) = \varepsilon(1, 4) = \varepsilon(2, 4) = \varepsilon^* + \eta.$$

The optimal matching is $\{1 \sim 2, 3 \sim 4\}$, with a cost of 0. It is not difficult to show that ranking the items and then finding this matching has a global cost (neglecting all log terms) of $\mathcal{O}(\frac{1}{\Delta^2})$. On the other hand, if the ranking is ignored, detecting that the matching $\{2 \sim 3, 1 \sim 4\}$ is suboptimal has a cost of $\Omega(\frac{1}{\eta^2})$, that can be arbitrarily larger. A reason behind this result is that the cost of matching is not monotonous with the cumulative skill gaps of its comparison: in this example, the optimal matching has a cumulative skill gap of $2(\varepsilon^* - \Delta) \simeq 2\varepsilon^*$, while the suboptimal one has a cumulative skill gap of $\varepsilon^* + \Delta + 2\eta \simeq \varepsilon^*$, hence twice smaller. This also explains why naïve combinatorial bandit algorithms would perform poorly.

Once again, we emphasize that it is not always the case that the exact ranking must be known before finding an optimal matching. There are many instances of problems where there are many optimal different matchings. The trivial example is when all the comparisons are costless, i.e., $|\varepsilon(i, j)| < \varepsilon^*$ for all $i, j \in [N]$. In that case, the true ranking is irrelevant. In Appendix D we also show that, if all $\varepsilon(\sigma(i), \sigma(i + 1)) \leqslant \varepsilon^*/2$, then any $(\varepsilon^*/2)$-correct ranking can be used to build an optimal matching. The interested reader can check that our algorithms for building optimal matchings can be adapted to avoid looking for perfect matchings in these favorable situations.

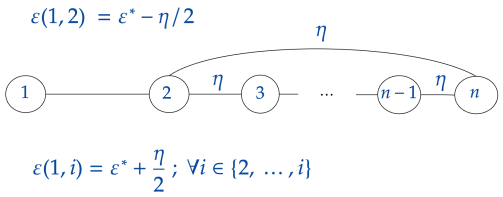

Figure 1. In this instance, finding the second ranked item is necessary to compute the optimal matching, irrespectively of η

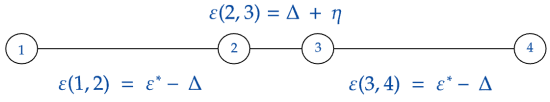

Figure 2. In this instance, checking that a matching is suboptimal (without using the ranking) can be arbitrarily complex for small η

2.2. PAC ranking

For a detailed presentation of AKS-Paterson sorting network, we refer to (Chvatal; Paterson, 1990). The main property of this sorting algorithm is that it has a time complexity of $\mathcal{O}(\log N)$. This implies that using $\mathcal{O}(\log N)$ matchings, with noiseless comparisons, the algorithm outputs the input list sorted. To handle noise, we will use the subroutine COMPARE acting as a substitution for the deterministic comparison procedure. COMPARE, whose simple pseudocode is postponed to Appendix F.1 takes as input two players i and j and two parameters ε and δ), used as a quality requirement for the performed comparison. Precisely, COMPARE returns the stronger player with probability at least $1-\delta$ with "confidence", whenever the gap $\varepsilon(i,j)$ is larger than the input threshold ε. Lemma 2.1 below shows that COMPARE indeed behaves as described.

Lemma 2.1 (Theoretical Performance of COMPARE)**.** COMPARE *terminates after* $b = O(\varepsilon^{-2}\log(1/\delta))$ *comparisons and if* $\varepsilon \leq |\varepsilon(i,j)|$*, it returns the strongest player and "Confident" with probability at least* $1-\delta$*. Conversely, if "confident", then* $|\varepsilon(i,j)| > \varepsilon$ *with probability at least* $1-\delta$*.*

In the other case, when $\varepsilon \geq |\varepsilon(i,j)|$*,* COMPARE *returns the best player with probability greater than* $1/2$*.*

Lemma 2.2 below is an implication of Lemma 2.1. The proof is relegated to Appendix C.1. Note that the constant D present in the following statement is independent of N and is a characteristic of the AKS-Paterson algorithm: AKS-Paterson time complexity, see Section 1.1.

Lemma 2.2. *Using* COMPARE *in the AKS-Paterson algorithm as a comparison procedure with the parameters* $\varepsilon/(2D^2\log(n))$ *and* $\delta' = \delta/(ND\log N)$ *outputs a ranking that is* (ε,δ)*-PAC in at most* $\mathcal{O}\left(\varepsilon^{-2}N\log^3(N)\log(N/\delta)\right)$ *comparisons, and in* $\mathcal{O}\left(\varepsilon^{-2}\log^3(N)\log(N/\delta)\right)$ *rounds. This algorithm will be referred to as AKS*(ε,δ)*.*

The sample complexity for retrieving an (ε,δ)-PAC ranking is bounded from below by $\Omega\left(\frac{N}{\varepsilon^2}\log(N/\delta)\right)$ (Ren et al., 2018). Thus, using AKS-Paterson algorithm with COMPARE as a comparison procedure is at most $\log^3(N)$ shy of the lower bound. Conversely, our method yields a time complexity of $O\left(\frac{\log^3(N)}{\varepsilon^2}\log(N/\delta)\right)$, which is much faster than the state of the art (Falahatgar et al., 2017b; Ren et al., 2019), requiring $O\left(\frac{N}{\varepsilon^2}\log(N/\delta)\right)$ time complexity.

2.3. Exact ranking

In this section, we leverage the confidence statement of COMPARE to design Algorithm CASCADINGAKS which retrieves an exact ranking with high probability (or stated otherwise a $(0,\delta)-$PAC ranking algorithm). The main intuition is to work by phases $s = 1, 2, \ldots$, where each phase ends with a ε_s-correct estimated ranking, for $\varepsilon_s = \frac{1}{2^s}$. More specifically, a phase ends with the construction of a clustering of similarly skilled players (two players in different clusters are correctly ranked with high enough probability), and the next phase will refine this clustering, until each cluster is a singleton.

The choices of parameters are $\varepsilon_s = \frac{1}{2^s}$, with the associated confidence $\delta_s = \frac{6\delta}{\pi^2 s^2}$. We shall denote by $\hat{r}_s$ the estimated ranking obtained at the end of phase s, that shall be ε_s-correct with probability at least $1 - \sum_{l=1}^{s}\delta_l$.

2.3.1. PRELIMINARY LEMMAS

This subsection is devoted to simple, easy-to-state, and to-prove lemmas that will be useful in the construction and the refinement of the clustering. Lemma 2.3 explains how comparisons that are "Confident" allow propagation of this confidence across the ranking.

Lemma 2.3. *Let* $i, j \in [N]$ *such that* $r(i) < r(j)$ *and* $\varepsilon(i,j) > \varepsilon$*. Let* $\hat{r}$ *be an* ε*-correct ranking. Then for all* $k \in [N]$ *such that* $\hat{r}(j) < \hat{r}(k)$*, it holds that* $r(i) < r(k)$*.*

Proof. Suppose for the sake of contradiction that there is an element k such that $r(j) < r(k)$ and $\hat{r}(k) < \hat{r}(i)$. Then, since $r(i) < r(j)$, Assumption 1.1 implies that $\varepsilon(i,k) \geq \varepsilon(i,j) > \varepsilon$. This contradicts the fact that $\hat{r}$ is ε-correct. □

Lemma 2.4 is a direct consequence of Lemma 2.3; it provides a simple way to implement a divide-and-conquer strategy for ranking.

Lemma 2.4. *Let* $j_1, j_2, j_3 \in [N]$ *such that* $r(j_1) < r(j_2) < r(j_3)$ *and* $\varepsilon(j_1,j_2) > \varepsilon$*,* $\varepsilon(j_2,j_3) > \varepsilon$*. Let* $\hat{r}$ *be an* ε*-correct ranking. Then for all* $i, k \in [N]$ *such that* $\hat{r}(i) < \hat{r}(j_1)$ *and* $\hat{r}(j_3) < \hat{r}(k)$*, it holds that* $r(i) < r(k)$*.*

Proof. Applying Lemma 2.3 on (j_1, j_2) and (j_2, j_3) gives the following: for all $i, k \in [N]$ such that $\hat{r}(i) < \hat{r}(j_1)$ and $\hat{r}(j_3) < \hat{r}(k)$, it holds that $r(i) < r(j_2) < r(k)$. □

In the situation described in Lemma 2.4, two elements u and v satisfying $\hat{r}(u) < \hat{r}(i)$ and $\hat{r}(k) < \hat{r}(v)$ do not need to be compared. This is the purpose of flagging comparisons as confident or not, and is a key property in the construction of the clustering detailed in the following section.

2.3.2. CONSTRUCTION OF A CLUSTERING

In this subsection, we illustrate how to cluster a set of players $[N]$. We call the sub-algorithm performing this task ANCHORING and its pseudo-code is given in Section F.2. We shall assume that there was no cluster created yet (this happens if the skill gaps are all very small). In the subsequent phases of the main algorithm, this clustering procedure will be performed independently on each cluster created (as in standard hierarchical clustering).

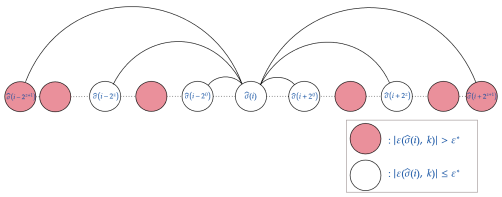

Figure 3. Illustration of comparisons involving $\hat{\sigma}(i)$ (Edges) queried by Anchoring.

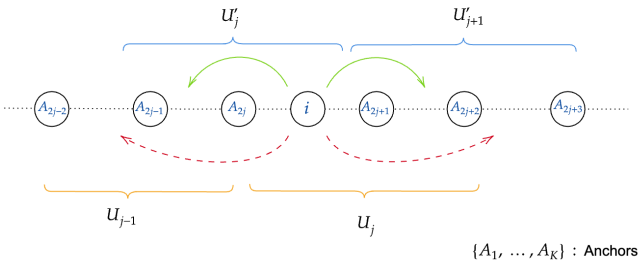

Figure 4. Illustration of the behavior CASCADINGAKS. i is confidently between A_{2j-1} and A_{2j+2}, but its rank with respect to A_{2j} and A_{2j+1} is uncertain.

The clustering relies on "anchor points", $A_1, A_2, \dots, A_K$ (chosen data-adaptively for some integer $K \in \mathbb{N}$ and of increasing rank in $\hat{r}_s$), such that COMPARE$(A_k, A_\ell, \varepsilon_s, \delta_{s+1})$ is confident, and $\hat{r}_s(A_k) - \hat{r}_s(A_\ell)$ is not too large, of order $\mathcal{O}(\max\{|\mathcal{N}_{\varepsilon_s}(A_k)|, |\mathcal{N}_{\varepsilon_s}(A_\ell)|\})$, where $\mathcal{N}_\varepsilon(i) = \{j \in [N], |\varepsilon(i,j)| \le \varepsilon\}$ is the ε-neighborhood of i. The clustering generated by anchor points is defined by

$$\mathcal{C}_k = \left\{ i \in [N], \hat{r}_s(A_{k-1}) < \hat{r}_s(i) < \hat{r}_s(A_k) \right\},$$

with the convention that $\hat{r}_s(A_0) = 0$ and $\hat{r}_s(A_{K+1}) = N+1$. The anchor points A_k are added to either $\mathcal{C}_k$ or $\mathcal{C}_{k+1}$ depending on their parity, see Algorithms 5 and 6 for details. A key quantity of a clustering is its maximal weight, defined as $W_{\hat{r}}(A_1, \dots, A_K) = \max_{k \in [K+1]} |\mathcal{C}_k|$.

To find anchor points, we introduce the following graph $G_{\hat{r}_s}$, whose vertices set is $[N]$ and $\{i, j\}$ is an edge of $G_{\hat{r}_s}$ if and only if $|\hat{r}_s(i) - \hat{r}_s(j)| = 2^m$ for any integer $m \in \mathbb{N}$. Algorithm 3 samples edges of $G_{\hat{r}_s}$ (following COMPARE procedure) and identify, for each player i, two different elements $m_s(i)$ and $M_s(i)$ such that $m(i)$ is confidently weaker than i and $M(i)$ is confidently stronger than i. Lemma 2.5 shows that $|\hat{r}_s(i) - \hat{r}_s(m_s(i))| = \mathcal{O}(\max\{|\mathcal{N}_{2\varepsilon_s}(i)|, |\mathcal{N}_{2\varepsilon_s}(m_s(i))|\}$ and $|\hat{r}_s(i) - \hat{r}_s(M_s(i))| = \mathcal{O}(\max\{|\mathcal{N}_{2\varepsilon_s}(i)|, |\mathcal{N}_{2\varepsilon_s}(M_s(i))|\}$.

We illustrate Algorithm 3 behavior in Figure 3. The objective is to identify elements confidently ranked below/above $\hat{\sigma}(i)$. To achieve this, it is compulsory to seek items, in red, not in $\mathcal{N}_\varepsilon(\hat{\sigma}(i))$, whose elements are in white. Since $\hat{r}(i)$ is an approximate ranking, red and white items can be intertwined, thus to identify elements confidently ranked below/above $\hat{\sigma}(i)$, one must seek far from $\hat{\sigma}(i)$.

Lemma 2.5. *Let $\hat{r}$ be an ε-correct ranking. Then Anchoring$(N, \hat{r}, \varepsilon, \delta/(N \log N))$ has a time complexity of order $\mathcal{O}\left(\frac{\log(N/\delta)}{\varepsilon^2} \max_{i \in [N]} \log\left(|\mathcal{N}_{2\varepsilon}(i)|\right)\right)$.*

It returns a set of anchor points $A_1, \dots, A_K$ such that $W_{\hat{r}}(A_1, \dots, A_K) = \mathcal{O}(\max_{i \in [N]} |\mathcal{N}_{2\varepsilon}(i)|)$.

The proof of this lemma is relegated to Appendix C.2.

2.3.3. FROM (ε_t, δ)-PAC TO $(\varepsilon_{t+1}, \delta)$-PAC RANKING

In this section, we describe how the Algorithm CASCADINGAKS builds an ε_s-correct ranking from a ε_s-correct one. The main idea is relatively natural: to improve the estimated ranking, it applies AKS$(\varepsilon_{s+1}, \delta_{s+1})$ to the different elements of a partition induced by the clusters (but twice, and not only once). At a high level, the first iteration of MULTIAKS produces a ranking that is locally ε_{s+1}-correct (on the red cells $U_1, U_2, \dots$ in Figure 4), but not globally, as mis-rankings between elements of U_j and U_{j-1} or U_{j+1} could still exist but not with cells further away: through a second application of MULTIAKS on the blue cells $U'_1, U'_2, \dots$ Figure 4, the mis-rankings left are eliminated. It thus produces an ε_{s+1}-correct ranking $\hat{r}_s$.

The pseudocode is given in Algorithm 1. For the sake of readability, CASCADINGAKS has been separated into subprotocoles, detailed in Appendix F.3. Figure 4 illustrates the behavior of CASCADINGAKS.

Lemma 2.6. *The time complexity of transitioning from stage s to stage $s+1$ of Algorithm 1 is*

$$\mathcal{O}\left(\frac{1}{\varepsilon_{s+1}^2} \log\left(\frac{N}{\delta_{s+1}}\right) \log^3\left(\max_{i \in [N]} \mathcal{N}_{2\varepsilon_{s+1}}(i)\right)\right). \quad (2)$$

Furthermore, the ranking obtained at the end of stage s is ε_{s+1}-correct with probability at least $1 - \sum_{l=1}^{s+1} \delta_l$.

The instance-dependent bound on the sample complexity depends on the size of the skill gaps between a player and those ranked immediately above/below him, as shown by the following result.

Theorem 2.7. *Each player i is involved in at most*

$$\mathcal{O}\left(\frac{\log^3 N}{\min\{\Delta_i^2, \Delta_{i-1}^2\}}\left(\log\frac{N}{\delta} + \log\log\frac{1}{\min\{\Delta_i, \Delta_{i-1}\}}\right)\right) \quad (3)$$

comparisons before its rank can be confidently identified.

For the proof of Theorem 2.7, see Appendix C.4.

Algorithm 1 CASCADINGAKS(δ)

Input: items $\{1,\dots,N\}$, confidence level $\delta > 0$;
Initialize: $\hat{r}_0$ (a random permutation $[N]$), $s = 1$

1: **while** $L \neq [1,\dots,1]$ **do**
2: $\quad\varepsilon_s \leftarrow (1/2)^s$;
3: $\quad(A_1,\dots,A_K) \leftarrow$ ANCHORING$(N, \hat{r}_{s-1}, \varepsilon_s, \delta_s/3)$;
4: $\quad\ell = \lfloor \frac{K+1}{2} \rfloor$;
5: $\quad\mathcal{C}_1,\dots,\mathcal{C}_{k+1} \leftarrow$ CLUSTERING$((A_1,..,A_{K+1}), \hat{r}_s)$;
6: $\quad(U_1,..,U_\ell) \leftarrow$ GRP1$((A_1,..,A_K),(\mathcal{C}_1,..,\mathcal{C}_{K+1}))$;
7: $\quad\hat{r}' =$ MULTIAKS$((U_1,\dots,U_\ell),\varepsilon_s,\delta_s/6)$;
8: $\quad\mathcal{C}_1,\dots,\mathcal{C}_{K+1} \leftarrow$ CLUSTERING$((A_1,\dots,A_K),\hat{r}')$;
9: $\quad(U'_1,\dots,U'_\ell) \leftarrow$ GRP2$((A_1,\dots,A_K),(\mathcal{C}'_1,\dots,\mathcal{C}'_{K+1}))$;
10: $\quad\hat{r}_s =$ MULTIAKS$((U'_1,\dots,U'_\ell),\varepsilon_s,\delta_s/6)$
11: $\quad L =$ CHECK$(\varepsilon_s,\delta_s/3)(\hat{r}_s)$;
12: $\quad$**for** $i \in [N]$ **do**
13: $\quad\quad$**if** i is correctly ranked ($L[i] = 1$) **then**
14: $\quad\quad\quad i$ can be moved to the gap estimation phase;
15: $\quad\quad$**end if**
16: $\quad$**end for**
17: $\quad s \leftarrow s + 1$;
18: **end while**
19: **return** $\hat{r}_s$

Depending on $\min\{\Delta_i, \Delta_{i-1}\}$, the rank of player i can be confidently determined sooner than the exact ranking (Algorithm 1, line 14). Thus CASCADINGAKS can move players easily ranked (large $\min\{\Delta_i, \Delta_{i-1}\}$) to the gap estimation phase before exact ranking is achieved. Precisely, as soon as $\hat{r}(i-1), \hat{r}(i), \hat{r}(i+1)$ are confidently ranked, GAPESTIMATION proceeds to estimate the gaps between $\hat{r}(i-1), \hat{r}(i)$ and between $\hat{r}(i), \hat{r}(i+1)$. This is relevant in practice, but there are instances where this does not happen (see the discussion in Section 2.1). This motivates the following corollary whose proof is postponed to Appendix C.4.

Corollary 2.8. *Algorithm 1 has a time complexity of*

$$\mathcal{O}\left(\frac{\log^3 N}{\min_{i\in[N]}\Delta_i^2}\left(\log(\frac{N}{\delta}) + \log\log\frac{1}{\min_{i\in[N]}\Delta_i^2}\right)\right). \tag{4}$$

It returns the true ranking with probability at least $1-\delta$.

3. Gap estimation and Regret

In this section, we assume that Algorithm 1 has output an exact ranking, hence the remaining objective is to identify an optimal matching. Without loss of generality and for the sake of notations, we can assume that the true ranking is the identity, $r(i) = i$.

Lemma 3.1 show that, in the optimal matching, all pairs with a cost of zero are between two players of adjacent rank i and $j = i+1$. This implies that it only remains to detect which of the skill gaps $\varepsilon(i, i+1)$ are above, or below ε^*.

Lemma 3.1. *Let $V = \{1,\dots,N\}$ and E the set of edges such that $G = (V, E)$ is a graph verifying: if $\{i,j\} \in E$, then, for every k and l such that $i \leq k, l \leq j$, $\{k,l\} \in E$. Then the matching obtained by traversing the graph from 1 to N , matching i to $i+1$ whenever possible, is a matching with maximal size.*

This lemma will be applied on the ε^*-adjacency graph $G_{\varepsilon^*} = ([N], E_{\varepsilon^*})$ defined by: the edge (i,j) belongs to E_{ε^*} if and only if $\varepsilon(i,j) \leq \varepsilon^*$. The maximal matching of this graph is an optimal matching with respect to the cost function. The fact that G satisfies the condition of Lemma 3.1 is a direct consequence of Assumption 1.1

The pseudo-code of Algorithm GAPESTIMATION learning the ε^*-adjacency is given in Appendix F.5, as it is quite similar to standard multi-armed bandit techniques (and more precisely on best arm identification). The main idea is to start from the graph $E_0 = ([N], E_0)$ with $E_0 = \{(i, i+1), i \in [N-1]\}$ and to sequentially remove edges once the algorithm detects that they do not belong to G_{ε^*} with high probability. This creates a nested sequence of graphs $G_t = ([N], E_t)$, i.e., such that $E_{\varepsilon^*} \subset E_{t+1} \subset E_t$, with high probability. The algorithm, using the subprotocol SAMPLEMATCHING, whose pseudo-codes is postponed to Appendix F.4, chooses at stage t a maximal matching of G_t (completed arbitrarily into a perfect matching of $[N]$).

The main intuition is the following. Each time the matching selected contains an edge $\{i, i+1\} \in E_t$ that does not belong to E_{ε^*}, the regret might increase by one, but this can only happen finitely many times. Indeed, after some time, the algorithm will eventually learn that $\{i, i+1\} \notin E_{\varepsilon^*}$. It only remains to control when this happens, using a variant of LIL-UCB (Jamieson et al., 2014).

An edge $\{i, i+1\}$ is removed from E_t as soon as

$$\hat{p}_{i,i+1}(t) - \frac{1}{2} \geq \varepsilon + u(T_{i,i+1}(t), \delta), \qquad \text{where}$$

$$T_{i,i+1}(t) = \sum_{s=1}^{t} \mathbb{1}\{\{i,i+1\} \text{ was sampled at step } s\} \times \mathbb{1}\{\{i,i+1\} \in E_s\}$$

is the number of times the selected matching contains the pair $\{i, i+1\}$ that belonged (erroneously) to E_s, $\hat{p}_{i,i+1}(t)$ is the empirical frequency of wins of player i against $i+1$ and $u(n,\delta) = 2\sqrt{\frac{\log(\frac{2n\log(t)}{\delta})}{t}}$. This algorithm yields the guarantees given in the following Lemma 3.2.

Lemma 3.2. *With probability at least $1-\delta$, for any $i \in [N]$ such that $\Delta_{\varepsilon^*,i} := \Delta_i - \varepsilon^* > 0$, it holds*

$$\sup_t T_{i,i+1}(t) \leq \frac{32}{\Delta^2_{\varepsilon^*,i}} \log\left(\frac{2N}{\delta}\log\frac{32}{\Delta^2_{\varepsilon^*,i}}\right). \tag{5}$$

The lemma is a consequence of the law of iterated logarithm, see (Jamieson et al., 2014), and its proof is somewhat standard in multi-armed bandit literature, hence omitted. It yields the following regret bound.

Corollary 3.3. *The regret of learning an optimal matching is upper-bounded by*

$$\mathcal{O}\left(\sum_{i\in[N-1],\varepsilon(i,i+1)>\varepsilon^*} \frac{1}{\Delta^2_{\varepsilon^*,i}} \log\left(\frac{N}{\delta}\log\frac{1}{\Delta^2_{\varepsilon^*,i}}\right)\right). \quad (6)$$

Proof. Lemma 3.2 provides an upper bound on the number of comparisons between i and $i+1$ required to learn that $\Delta_i > \varepsilon^*$ and that this comparison is costly.

Let us assume that those comparisons are indeed costly (otherwise, there is no issue) and that the algorithm keeps sampling it after learning it. Since the graph E_t contains E_{ε^*}, the estimated cost of SampleMatching is smaller than the cost of the optimal matching. Consequently, any costly edge willingly put in the matching (line 6 of SampleMatching) corresponds to one costly edge of the optimal matching.

Hence the total regret is the number of times a costly edge is sampled while belonging to the current edge set E_t. □

3.1. Wrapping up and Minimax Optimality

By combining Theorem 2.7 and Corollary 3.3, we get the following Theorem 3.4. It is minimax optimal (up to polylog term) in the sense that, for any given values of Δ_i and $\Delta_{\varepsilon^*,i}$, there exists an instance where this bound cannot be improved (up to the $\log^3$ term). We however acknowledge that the regret can be, in some instances, lower than the above bound and refer to Section D).

Theorem 3.4. *The regret of learning an optimal matching without knowing the ranking is upper bounded by*

$$\mathcal{O}\Bigg(\sum_{i\in[N-1]:\Delta_i>\varepsilon^*} \frac{1}{\Delta^2_{\varepsilon^*,i}} \log\left(\frac{N}{\delta}\log(\frac{1}{\Delta^2_{\varepsilon^*,i}})\right) + \sum_{i\in[N-1]} \frac{1}{\Delta^2_i}\log(\frac{N}{\delta}\log\frac{1}{\Delta^2_i})\Bigg). \quad (7)$$

The optimality proof relies on the following three 4-vertices instances, where $\eta_1 < \eta_2 \ll \varepsilon^*$, illustrated in Figure 5:

Instance A: $\varepsilon(1,2) = \varepsilon(3,4) = \varepsilon^* + \eta_1$ and $\varepsilon(2,3) = \eta_2$
Instance B: $\varepsilon(1,2) = \varepsilon(3,4) = \varepsilon^* - \eta_1$ and $\varepsilon(2,3) = \eta_2$
Instance C: $\varepsilon(1,3) = \varepsilon(2,4) = \varepsilon^* - \eta_1$ and $\varepsilon(3,2) = \eta_2$.

and the other $\varepsilon(i,j)$ are all strictly bigger than ε^*. Straightforward computations show that the optimal matching in Instance A is$\{2 \sim 3, 1 \sim 4\}$ with a cost 1. In Instance B (resp. C), on the other hand, the optimal matching is $\{1 \sim 2, 4 \sim 3\}$ (resp. $\{1 \sim 3, 4 \sim 2\}$), for a cost of 0. In all instances, any other matching has an additional cost of at least 1. Notice that in instances B and C, computing the optimal matching requires the true ranking (no matter η).

Standard online learning arguments (Kaufmann et al., 2016; Bubeck et al., 2013) yield that in order to distinguish between Instances A and B/C, one must sample an edge $\{1,2\}$ or $\{3,4\}$ $\Omega(\frac{\log(1/\delta)}{\eta_1^2})$ times. And the ranking of 2 or 3, to distinguish between instances B and C, requires $\Omega(\frac{\log(1/\delta)}{\eta_2^2})$ samples of the edge $\{2,3\}$. An interesting feature of this example is that it illustrates that even though matching players 2 to 3 has no direct cost (as the skill gap is very small), this pairing induces some "indirect" but unavoidable loss to other players, that are matched in unbalanced games.

As a consequence, distinguishing between A and B/C incurs a cost of $\Omega(\frac{\log(1/\delta)}{\eta_1^2})$ in Instance A, while distinguishing between B and C incurs a cost of $\Omega(\frac{\log(1/\delta)}{\eta_2^2})$ in both Instances B and C. In particular, if the choice of the instance is made uniform at random, the expected cost is of order $\Omega\Big(\frac{\log 1/\delta}{\eta_1^2} + \frac{\log 1/\delta}{\eta_2^2}\Big)$. Duplicating this 4-vertices example, independently $N/4$ times, gives a regret that has to scale as

$$\begin{aligned} R(\delta) &\geq \Omega\Big(N\frac{\log N/\delta}{\eta_1^2} + N\frac{\log N/\delta}{\eta_2^2}\Big) \\ &= \Omega\Big(\sum_{i\in[N-1],\varepsilon(i,i+1)>\varepsilon^*} \frac{1}{\Delta^2_{\varepsilon^*,i}}\log(\frac{N}{\delta}) \\ &\qquad + \sum_{i\in[N]}\frac{1}{\Delta_i^2}\log(\frac{N}{\delta})\Big). \end{aligned}$$

This construction can be generalized to any values of $\Delta_{\varepsilon^*,i}$ and Δ_i, by adapting values of η_1 and η_2 per 4- instances.

4. Conclusion and future work

We designed an algorithm with time complexity optimal for ranking and optimal matching, but up to $\log^3 N$. A natural research direction would be to close this gap. Moreover, these algorithms are based on the AKS sorting network and suffer from its drawback, a large constant hidden in the $\mathcal{O}$. Avoiding AKS network, with the same result, is another exciting challenge.

Impact Statement This paper presents work whose goal is to advance the field of Machine Learning and Sorting. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

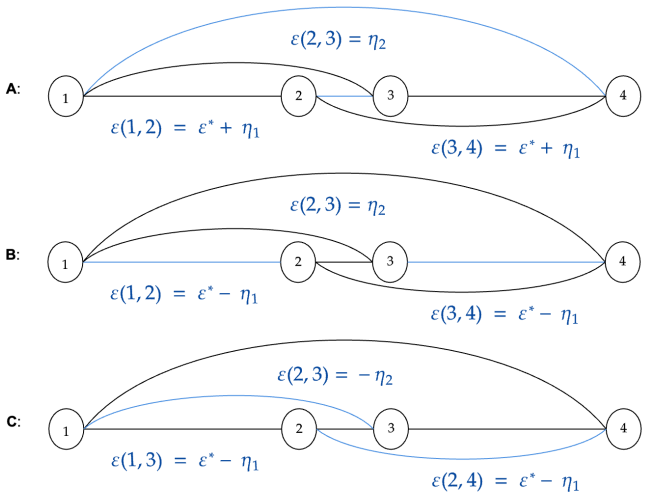

Figure 5. The 3 different instances and their optimal matching

References

Ajtai, M., Komlós, J., and Szemerédi, E. An 0 (n log n) sorting network. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pp. 1–9, 1983.

Batcher, K. E. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, pp. 307–314, 1968.

Bengs, V., Busa-Fekete, R., El Mesaoudi-Paul, A., and Hüllermeier, E. Preference-based online learning with dueling bandits: A survey. *The Journal of Machine Learning Research*, 22(1):278–385, 2021.

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

Bubeck, S., Perchet, V., and Rigollet, P. Bounded regret in stochastic multi-armed bandits. In *Conference on Learning Theory*, pp. 122–134. PMLR, 2013.

Cesa-Bianchi, N. and Lugosi, G. Combinatorial bandits. *Journal of Computer and System Sciences*, 78(5):1404–1422, 2012.

Chvatal, V. Lecture notes on the aks sorting network. URL https://users.encs.concordia.ca/~chvatal/notes/aks.pdf.

Devroye, L. Laws of the iterated logarithm for order statistics of uniform spacings. *The Annals of Probability*, 9(5): 860–867, 1981.

Elo, A. E. *The Rating of Chessplayers, Past and Present*. Arco Pub., New York, 1978. ISBN 0668047216.

Falahatgar, M., Hao, Y., Orlitsky, A., Pichapati, V., and Ravindrakumar, V. Maxing and ranking with few assumptions. *Advances in Neural Information Processing Systems*, 30, 2017a.

Falahatgar, M., Orlitsky, A., Pichapati, V., and Suresh, A. T. Maximum selection and ranking under noisy comparisons. In *International Conference on Machine Learning*, pp. 1088–1096. PMLR, 2017b.

Falahatgar, M., Jain, A., Orlitsky, A., Pichapati, V., and Ravindrakumar, V. The limits of maxing, ranking, and preference learning. In *International conference on machine learning*, pp. 1427–1436. PMLR, 2018.

Feige, U., Raghavan, P., Peleg, D., and Upfal, E. Computing with noisy information. *SIAM Journal on Computing*, 23 (5):1001–1018, 1994.

Heckel, R., Shah, N. B., Ramchandran, K., and Wainwright, M. J. Active ranking from pairwise comparisons and when parametric assumptions do not help. 2019.

Herbrich, R., Minka, T., and Graepel, T. Trueskill™: a bayesian skill rating system. *Advances in neural information processing systems*, 19, 2006.

Hoare, C. A. Quicksort. *The computer journal*, 5(1):10–16, 1962.

Jamieson, K., Malloy, M., Nowak, R., and Bubeck, S. lil'ucb: An optimal exploration algorithm for multi-armed bandits. In *Conference on Learning Theory*, pp. 423–439. PMLR, 2014.

Kaufmann, E., Cappé, O., and Garivier, A. On the complexity of best arm identification in multi-armed bandit models. *Journal of Machine Learning Research*, 17:1–42, 2016.

Knuth, D. E. et al. *The art of computer programming*, volume 3. Addison-Wesley Reading, MA, 1973.

Kveton, B., Wen, Z., Ashkan, A., and Szepesvari, C. Tight regret bounds for stochastic combinatorial semi-bandits. In *Artificial Intelligence and Statistics*, pp. 535–543. PMLR, 2015.

Merlis, N. and Mannor, S. Tight lower bounds for combinatorial multi-armed bandits. In *Conference on Learning Theory*, pp. 2830–2857. PMLR, 2020.

Merlis, N. and Mannor, S. Lenient regret for multi-armed bandits. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 8950–8957, 2021.

Minka, T., Cleven, R., and Zaykov, Y. Trueskill 2: An improved bayesian skill rating system. *Technical Report*, 2018.

Natvig, L. Logarithmic time cost optimal parallel sorting is not yet fast in practice! In *Conference on High Performance Networking and Computing: Proceedings of the 1990 ACM/IEEE conference on Supercomputing*, volume 12, pp. 486–494. Citeseer, 1990.

Paterson, M. S. Improved sorting networks with o (log n) depth. *Algorithmica*, 5(1-4):75–92, 1990.

Perrault, P., Boursier, E., Valko, M., and Perchet, V. Statistical efficiency of thompson sampling for combinatorial semi-bandits. *Advances in Neural Information Processing Systems*, 33:5429–5440, 2020.

Ren, W., Liu, J., and Shroff, N. B. Pac ranking from pairwise and listwise queries: Lower bounds and upper bounds. *arXiv preprint arXiv:1806.02970*, 2018.

Ren, W., Liu, J. K., and Shroff, N. On sample complexity upper and lower bounds for exact ranking from noisy comparisons. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Saha, A. and Gopalan, A. Active ranking with subset-wise preferences. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 3312–3321. PMLR, 2019.

Saha, A. and Gopalan, A. From pac to instance-optimal sample complexity in the plackett-luce model. In *International Conference on Machine Learning*, pp. 8367–8376. PMLR, 2020.

Seiferas, J. Sorting networks of logarithmic depth, further simplified. *Algorithmica*, 53(3):374–384, 2009.

Szörényi, B., Busa-Fekete, R., Paul, A., and Hüllermeier, E. Online rank elicitation for plackett-luce: A dueling bandits approach. *Advances in Neural Information Processing Systems*, 28, 2015.

Tversky, A. and Russo, J. E. Substitutability and similarity in binary choices. *Journal of Mathematical psychology*, 6(1):1–12, 1969.

Valiant, L. G. Parallelism in comparison problems. *SIAM Journal on Computing*, 4(3):348–355, 1975.

Wang, S. and Chen, W. Thompson sampling for combinatorial semi-bandits. In *International Conference on Machine Learning*, pp. 5114–5122. PMLR, 2018.

Yue, Y., Broder, J., Kleinberg, R., and Joachims, T. The k-armed dueling bandits problem. *Journal of Computer and System Sciences*, 78(5):1538–1556, 2012.

Zoghi, M., Tunys, T., Ghavamzadeh, M., Kveton, B., Szepesvari, C., and Wen, Z. Online learning to rank in stochastic click models. In *International conference on machine learning*, pp. 4199–4208. PMLR, 2017.

A. Discussion of Assumptions 1.1 and 1.2

The main results of the different algorithms rely on Assumptions 1.1 and 1.2 that we restate here for completeness.

Assumption A.1. Strong Stochastic Transitivity (SST). For any $i, j, k \in [N]$, if $r(i) < r(j) < r(k)$ then:

$$\varepsilon(i,k) \geq \max\{\varepsilon(i,j), \varepsilon(j,k)\}. \tag{8}$$

Assumption A.2. Stochastic Triangular inequality (STI). For any $i, j, k \in [N]$, if $r(i) < r(j) < r(k)$ then:

$$\varepsilon(i,k) \leq \varepsilon(i,j) + \varepsilon(j,k). \tag{9}$$

Consequences of (SST) (SST) have two main consequences:

1. There is a ranking r that is consistent with $(\varepsilon(i,j))_{i,j\in[N]}$.
2. If $r(i) < r(j) < r(k)$ and $\varepsilon(i,j) > \varepsilon^*$ then $\varepsilon(i,k) > \varepsilon^*$.

Consequence 1 can be obtained with less restrictive assumptions like Weak Stochastic transitivity (WST), the minimum assumption required for the existence of a ranking consistent with the skill gaps $(\varepsilon(i,j))_{i,j\in[N]}$.

Assumption A.3. Weak Stochastic transitivity (WST). For any $i, j, k \in [N]$, if $\varepsilon(i,j) \geq 0$ and $\varepsilon(j,k) \geq 0$ then $\varepsilon(i,k) \geq 0$.

If only A.3 holds, even though a ranking exists and is consistent with the skill gaps, its existence is not useful.

Indeed, to decide whether a given player i has an opponent j with $|\varepsilon(i,j)| \leq \varepsilon^*$, the gaps between i and any other player must be estimated, which yields a sample complexity of $\Omega\left(\sum_{i<j} \frac{1}{\varepsilon^2(i,j)}\left(\log\log\frac{1}{\varepsilon^2(i,j)} + \log(1/\delta)\right)\right)$ (Ren et al., 2019).

It should be noted that models where every element i has a "strength parameter" θ_i and there is an increasing function F such that $\varepsilon(i,j) = F(\theta_i - \theta_j)$ and $F(0) = 0$, which are called parametric models, all satisfy the assumption (SST). Among notable parametric models is the Bradley-Terry-Luce (BTL) model (Bradley & Terry, 1952), where F is the logistic function. Elo ranking system (Elo, 1978) is based on (BTL) model.

Consequences of (STI) The consequences of forgoing STI have been elucidated (Falahatgar et al., 2017a): there is an instance of verifying (SST) where any $(1/4, 1/8)$-PAC ranking algorithm requires $\Omega(N^2)$ comparisons, see (Falahatgar et al., 2017a) Theorem 7. It is perhaps convenient to recall that (STI) implies that [Lemma 19(Falahatgar et al., 2017b)] the following holds without conditions on the ordering of i, j, k:

$$(\varepsilon(i,j) \leq \varepsilon_1) \;\&\; (\varepsilon(j,k) \leq \varepsilon_2) \implies \varepsilon(i,k) \leq \varepsilon_1 + \varepsilon_2 \tag{10}$$

BTL and STT/STI Notice that the Bradley-Terry-Luce model (BTL) also verifies the (STI) condition. Indeed: consider i, j, k three elements such that $\theta_i \leq \theta_j \leq \theta_k$. The (BTL) model indicates that $\varepsilon(i,j) = F(\theta_i - \theta_k)$, where $F(x) = 1/(1+\exp(-x))$. To show that (BTL) verifies (STI), it remains to show that $\varepsilon(k,i) \leq \varepsilon(k,j) + \varepsilon(j,i)$. This follows from the fact that F is concave on $\mathbb{T}_+$. Consider $\tau(x) = \frac{F(x)}{x}$ the rate of change of F between 0 and x. Since F is concave, this rate is decreasing. It follows that $\tau(\theta_k - \theta_i) \leq \min(\tau(\theta_k - \theta_j), \tau(\theta_j - \theta_i))$. Hence

$$\begin{aligned}\varepsilon(k,i) &= \tau(\theta_k - \theta_i)(\theta_k - \theta_i) \\ &\leq \tau(\theta_k - \theta_j)(\theta_k - \theta_i) + \tau(\theta_j - \theta_i)(\theta_j - \theta_i) \\ &= \varepsilon(k,j) + \varepsilon(j,i).\end{aligned}$$

B. Short reminder of AKS sorting network

In this section, we present a comprehensive overview of the AKS sorting network (Ajtai et al., 1983; Paterson, 1990). For a more accessible yet detailed exposition of the algorithm, we direct readers to (Chvatal). The organization of this section unfolds as follows: we commence by providing a brief introduction to sorting networks, a category of sorting algorithms well-suited for parallel computing. Subsequently, recognizing the intricacy of the AKS algorithm, we initially introduce an algorithm known as Halver-Sort. Although slower than the AKS sorting network, Halver-Sort serves as a more easily graspable precursor. We then delve into the step-by-step transformations required to transition from Halver-Sort to the AKS sorting network, elucidating the performance gains achieved through this process.

B.1. Sorting networks

A sorting network, as introduced by Batcher (Batcher, 1968), possesses a noteworthy characteristic: the predetermined set of comparisons it is designed to perform is independent of the input data, rendering the algorithm "oblivious". This property is particularly advantageous when leveraging parallel computing for sorting, as it minimizes communication between processors responsible for the comparisons.

In the context of sorting a list l of size N, envision containers $C_1, C_2, \ldots, C_N$, representing memory slots to store the list l. Initially unsorted, the goal is for C_i to contain the i-th smallest element at the end, resulting in C_1 containing the smallest and C_N the largest.

A sorting network entails a specified scheme of comparisons $((C_{i_1}, C_{j_1}), \ldots, (C_{i_t}, C_{j_t}), \ldots)$, where each pair

(C_{i_t}, C_{j_t}) indicates a comparison instruction for the values inside the respective containers. If C_{j_t} is observed to be smaller than C_{i_t}, the containers exchange their contents. Importantly, the choice of the comparison scheme remains independent of the input list.

Note that if $\{i_t, j_t\} \cap \{i_{t+1}, j_{t+1}\} = \emptyset$, the corresponding comparisons are deemed independent, allowing parallel execution. Likewise, comparisons can be grouped into batches of independent comparisons, facilitating simultaneous execution. Naturally, the batch size cannot exceed $N/2$.

For any sorting network, the comparisons can be organized into batches, with the number of batches D representing the time required for the algorithm to conclude if each batch is processed in unit time. Thus, D serves as the depth of the sorting network.

It is well-established that any comparison-based sorting algorithm demands at least $\mathcal{O}(N \log N)$ comparisons to sort a list of size N. As sorting networks fall under the category of comparison-based algorithms, they too necessitate $\mathcal{O}(N \log N)$ comparisons. Furthermore, given that each batch has a size of at most $N/2$, it follows that any sorting network has a depth of $\Omega(\log N)$. This optimal depth is matched by the AKS sorting network, which will be elucidated below.

For an in-depth exploration of sorting networks, we refer readers to (Knuth et al., 1973).

B.2. Halver-Sort

In this section, we introduce a sorting algorithm termed Halver-Sort. The algorithm utilizes a comparison network, specifically a perfect halver denoted as H. When provided with an input list of size $2m$, this perfect halver H partitions the list into two sublists, each of size m. The first sublist comprises the larger m values, while the second contains the smaller m values. Employing this network, we devise an algorithm that constructs a binary tree. At each node, the algorithm utilizes H to split the list into superior and inferior lists of equal sizes, forwarding these lists to the respective child nodes. Notably, this algorithm bears a resemblance to QuickSort (Hoare, 1962); however, in Halver-Sort, it is as if the pivot is consistently the median, ensuring the division of the list into two equal-sized sublists. Consequently, each element steadily traverses the binary tree until reaching its corresponding leaf.

Regrettably, a perfect halver H is proven to have a minimum depth of at least $\frac{1}{2} \log m$. Consequently, when considering the depth of the binary tree (bounded by $\mathcal{O}(\log N)$), the overall depth amounts to $\mathcal{O}(\log^2 N)$. Subsequently, we will explore how deviating from the perfect halving property in favor of approximate halving, as defined below, can lead to a substantial acceleration of $\mathcal{O}(\log N)$.

B.3. AKS

There are two primary modifications required to transform Halver-Sort into the AKS sorting network. The first entails constructing an $(2N, \varepsilon)$-halver, which is permitted to make errors. Formally, an $(2N, \varepsilon)$-halver is defined as follows:

Definition B.1. An $(2N, \varepsilon)$-halver is a network of comparators that, given a list of $2N$ values, outputs two lists B_L and B_R such that:

- For $k \in [N]$, at most εk of the k largest values among the input are assigned to B_L
- For $k \in [N]$, at most εk of the k smallest values among the input are assigned to B_R

A proof and construction of an ε-halver, along with its constant depth independent of N (though dependent on ε), is provided by (Ajtai et al., 1983; Paterson, 1990).

The second modification addresses the adaptation necessary to accommodate the imperfections introduced by the non-perfect halver, which may lead to errors in the halving procedures. At a high level, the concept involves using an ε-halver to identify a fraction of B_R, which ideally consists of the weakest values in B_R, referred to as F_L in the subsequent definition.

Consider the binary tree employed in Halver-Sort. While each node in Halver-Sort contains a perfect halver, in AKS, some elements may take incorrect turns but are later singled out and redirected upward in the tree to rectify the error.

The error correction is achieved through a sophisticated module known as a $(2N, 2f, \varepsilon_B, \varepsilon_f)$-Separator, defined as follows:

Definition B.2. A $(2N, 2f, \varepsilon_B, \varepsilon_F)$-separator is a $(2a, \varepsilon_B)$-halver such that there are designated block F_L and F_R of output wires such that

$$F_L \subset B_L \tag{11}$$
$$F_R \subset B_R \tag{12}$$
$$|F_L| = |F_R| = f \tag{13}$$

and satisfying the following properties: $\forall k \in \{1, \ldots, a\}$

1. the network places at most $\epsilon_F k$ of its smallest inputs outside of F_L (that is in $B_L - F_L$)
2. the network places at most $\epsilon_F k$ of its largest inputs outside of F_R (that is in $B_L - F_L$)

The blocks F_L and F_R play a crucial role in error correction. With appropriately calibrated modules within a binary tree, involving choices for ε_B, ε_f, and f, it can be demonstrated that AKS indeed performs sorting.

For a comprehensive yet accessible presentation of the AKS sorting network, we recommend referring to (Chvatal).

C. Omitted Proofs

C.1. Proof of Lemmas 2.1 and 2.2

For convenience, Lemma 2.1 is restated here:

Lemma C.1 (Theoretical Performance of COMPARE)**.** COMPARE *terminates after* $b = O(\varepsilon^{-2}\log(1/\delta))$ *comparisons and returns the more preferred item with probability at least* $1/2$*. Further, if* $\varepsilon \leq |\varepsilon(i,j)|$*, then* COMPARE *returns the stronger player and "Confident" with probability at least* $1-\delta$*, and, if "Confident" is returned, that implies that* $\varepsilon(i,j) > \varepsilon$ *with probability at least* $1-\delta$

Proof. W.l.o.g, assume that $\varepsilon(i,j) > 0$, then COMPARE terminates after $b = \lceil 2\varepsilon^{-2}\log(1/\delta)\rceil = \mathcal{O}(\varepsilon^{-2}\log 1/\delta)$ comparisons. Since the algorithm is symmetric and $\varepsilon(i,j) > 0$, then COMPARE returns j with probability smaller than $1/2$. Now, suppose that $\varepsilon < \varepsilon(i,j)$, It remains to prove that COMPARE returns i and that it is "Confident" with probability at least $1-\delta$. Let $\mathcal{E}$ be the event that $\hat{p}_{i,j} < 1/2 + 2\varepsilon$. Then, by Chernoff bound

$$\mathbb{P}(\mathcal{E}) \leq \exp\left(-2\left(\varepsilon\right)^2 b\right) \leq \delta \tag{14}$$

If $\mathcal{E}$ does not happen, then COMPARE returns i (the stronger player) and "Confident". This holds with probability at least $1-\delta/2$ □

For convenience, we restate the Lemma 2.2

Lemma C.2. *Using* COMPARE *in the AKS-Paterson algorithm as a comparison procedure with the parameters* $\varepsilon/(2D\log(N))$ *and* $\delta' = \delta/(N\log^3 N)$*, the ranking retrieved is* (ε,δ)*-PAC in at most* $O\left(\varepsilon^{-2}N\log(N)\log(N/\delta)\right)$ *comparisons.*

To prove this lemma, the only used property of AKS-Paterson algorithm is that it is a network of comparators, meaning that it can be expressed as a finite sequence $(M_t, L_t, R_t)_t$, where M_t is a perfect bipartite matching between elements $L_t \subset [N]$ and those of $R_t \subset [N]$.

Let σ_0 be an arbitrary ordering ($\sigma(i)$ is the element in position i). $\hat{\sigma}_1$(and subsequently r_t) is obtain as follows: if $i \in L_1$ and $j \in R_1$ are paired in M_1, then $\hat{\sigma}_0(i)$ and $\sigma_0(j)$ are compared(For now, comparisons are noiseless). After the comparison: $\sigma_1(j)$ gets the larger value and $\sigma_1(i)$ gets the smaller one. In summary, positions in $[N]$ with indices in R_1 receive the larger values and elements with indices in L_1 receive the smaller values. σ_t is built similarly, by induction.

Hence, σ_t is the ranking resulting from applying the t first element of the sequence (M_t, L_t, R_t). Let $\hat{\sigma}_t$ the ranking resulting from applying the same sequence, but with a faulty comparator: COMPARE (with parameters (ε,δ)) instead of a perfect comparison. Then we have the following lemma:

Lemma C.3. *The following bounds hold that:*

$$|\varepsilon(\hat{\sigma}_t(i), \sigma_t(i))| \leq t\varepsilon. \tag{15}$$

and this, simultaneously on all $i \in [N]$*, with probability at least* $1 - \delta tN/2$

Proof. At time t, $tN/2$ comparisons have been queried to construct $\hat{\sigma}_t$. Hence, all queries of COMPARE are simultaneous correct with a probability at least $1 - \delta tN/2$. For the rest of the proof, suppose that all the queries of COMPARE are correct. The proof is by induction: for $t = 0$: $\hat{\sigma}_0 = \sigma_0$ (no comparison queried yet).

Let $t \in \mathbb{N}$. Suppose that equation 15 hold for t. Let $i \in L_{t+1}$ and $j \in R_{t+1}$ be two indices that are paired in M_{t+1}.

W.l.o.g, suppose that $\varepsilon(\sigma_t(i), \sigma_t(j)) > 0$, so much that $\sigma_{t+1}(i) = \sigma_t(j)$ and $\sigma_{t+1}(j) = \sigma_t(i)$. $\hat{\sigma}_t(i)$ and $\hat{\sigma}_t(j)$ are compared using COMPARE.

If $\hat{\sigma}_{t+1}(i) = \hat{\sigma}_t(j)$ and $\hat{\sigma}_{t+1}(j) = \hat{\sigma}_t(i)$, then Equation 15 trivially holds for i and j because $\varepsilon(\hat{\sigma}_t(i), \sigma_t(i)) = \varepsilon(\hat{\sigma}_{t+1}(i), \sigma_{t+1}(i))$ and $\varepsilon(\hat{\sigma}_t(j), \sigma_t(j)) = \varepsilon(\hat{\sigma}_{t+1}(j), \sigma_{t+1}(j))$.

If instead $\hat{\sigma}_{t+1}(i) = \hat{\sigma}_t(i)$ and $\hat{\sigma}_{t+1}(j) = \hat{\sigma}_t(j)$, this means that $\varepsilon(\hat{\sigma}_t(i), \hat{\sigma}_t(j)) \leq \varepsilon$. It follows from equation 10 (which is a consequence of assumption 1.2) that

$$\begin{aligned}
\varepsilon(\hat{\sigma}_{t+1}(i), \sigma_{t+1}(i)) &= \varepsilon(\hat{\sigma}_t(i), \sigma_t(j) && (16)\\
&\leq \varepsilon(\hat{\sigma}_t(i), \hat{\sigma}_t(j)) + \varepsilon(\hat{\sigma}_t(j), \sigma_t(j)) && (17)\\
&\leq \varepsilon + t\varepsilon = (t+1)\varepsilon && (18)
\end{aligned}$$

Moreover, $\varepsilon(\sigma_t(j), \sigma_t(i)) < 0$, hence the same reasoning yields

$$\begin{aligned}
\varepsilon(\sigma_{t+1}(i), \hat{\sigma}_{t+1}(i)) &= \varepsilon(\sigma_t(j), \hat{\sigma}_t(i)) && (19)\\
&\leq \varepsilon(\sigma_t(j), \sigma_t(i)) + \varepsilon(\sigma_t(i), \hat{\sigma}_t(i)) && (20)\\
&\leq t\varepsilon && (21)
\end{aligned}$$

In summary $|\varepsilon(\sigma_{t+1}(i), \hat{\sigma}_{t+1}(i))| \leq (t+1)\varepsilon$. The same reasoning applied to j yields $|\varepsilon(\sigma_{t+1}(j), \hat{\sigma}_{t+1}(j))| \leq (t+1)\varepsilon$. Hence the lemma. □

Proof. Lemma 2.2 is a direct consequence of Lemma C.3: for $\varepsilon' = \varepsilon/(2D\log N)$ and $\delta' = \delta/(DN\log N)$, the retrieved ranking $\hat{r}$ verifies the following relation with respect to the true ranking r:

$$\forall i \in [N], |\varepsilon(\sigma(i), \hat{\sigma}(i))| \leq \varepsilon/2 \tag{22}$$

Let $i < j \in [N]$ two positions. Let $\hat{\sigma}(i)$ and $\hat{\sigma}(j)$ be the players at those ranks according to $\hat{\sigma}$ (the rank of player $\hat{\sigma}(i)$ is i). It is sufficient to show that $\varepsilon(\hat{\sigma}(j), \hat{\sigma}(i)) \leq \varepsilon$ to conclude that $\hat{r}$ is indeed ε-correct. The Equation 10 and Lemma C.3 yield:

$$\begin{aligned}\varepsilon(\hat{\sigma}(j), \hat{\sigma}(i)) &\leq \varepsilon(\hat{\sigma}(j), \sigma(j)) && (23)\\ &+ \varepsilon(\sigma(j), \sigma(i)) && (24)\\ &+ \varepsilon(\sigma(i), \hat{\sigma}(i)) && (25)\\ &\leq \frac{\varepsilon}{2} + 0 + \frac{\varepsilon}{2} = \varepsilon && (26)\end{aligned}$$

Hence the lemma. □

C.2. Proof of Lemma 2.5

For convenience, the Lemma 2.5 is restated:

Lemma C.4. *Let $\hat{r}$ be an ε-correct ranking. Then Anchoring$(N, \hat{r}, \varepsilon, \delta)$ has a time complexity of order $\mathcal{O}\left(\frac{\log(N/\delta)}{\varepsilon^2} \max_{i\in[N]} \log\left(|\mathcal{N}_{2\varepsilon}(i)|\right)\right)$.*

It returns a set of anchor points $A_1, \ldots, A_K$ such that $W_{\hat{r}}(A_1, \ldots, A_K) = \mathcal{O}(\max_{i\in[N]} |\mathcal{N}_{2\varepsilon}(i)|)$. Both these claims hold simultaneously with probability at least $1 - \delta$

The proof of Lemma 2.5 relies on the following observation:

Lemma C.5. *Let $\hat{r}$ be a an ε-correct ranking. Let $i \in [N]$ and $j \in \mathcal{N}_\varepsilon(i)$. Then:*

$$|\hat{r}(i) - \hat{r}(j)| \leq |\mathcal{N}_{2\varepsilon}(i)| \quad (27)$$

Proof. First, notice that since $\hat{r}$ is ε-correct, $|\hat{r}(i) - r(i)| \leq |\mathcal{N}_\varepsilon(i)|$. Indeed, to shift the rank of i with s, at least s misrankings are needed. Since i cannot be misranked with other elements than those of $\mathcal{N}_\varepsilon(i)$, the shift $s \leq |\mathcal{N}_\varepsilon(i)|$. Similarly for every element $j \in \mathcal{N}_\varepsilon(i)$, we have that $|\hat{r}(j) - r(j)| \leq |\mathcal{N}_\varepsilon(j)|$.

Moreover, for every $j \in \mathcal{N}_\varepsilon(i)$, it holds that $\mathcal{N}_\varepsilon(j) \subset \mathcal{N}_{2\varepsilon}(i)$. This is a direct consequence of Assumption 1.2. Now, notice that $|\hat{r}(i) - \hat{r}(j)|$ counts the number of elements ranked between i and j in $\hat{r}$. Since $\hat{r}$ is ε-correct, there is two types of elements that are between i and j in $\hat{r}$:

- Elements that are between i and j in the true ranking r: these are included in $\mathcal{N}_\varepsilon(i)$ because $j \in \mathcal{N}_\varepsilon(i)$.
- Elements that are mismatched with respect to either i or j: these are elements of $\mathcal{N}_\varepsilon(i)$ and $\mathcal{N}_\varepsilon(j)$ which are both included in $\mathcal{N}_{2\varepsilon}(i)$.

Hence, every element ranked between i and j in $\hat{r}$ is an element of $\mathcal{N}_{2\varepsilon}(i)$ Hence, for every $j \in \mathcal{N}_\varepsilon(i)$, it holds that $|\hat{r}(i) - \hat{r}(j)| \leq |\mathcal{N}_{2\varepsilon}(i)|$. This concludes the proof. □

Now, the proof of Lemma 2.5 is presented

Proof. First, Anchoring$(N, \hat{r}, \varepsilon, \delta)$ queries at most $2N \log N$ times COMPARE, with each query having a confidence of $\frac{\delta}{2N \log N}$. It follows from a union bound that all queries are simultaneously correct with probability at least $1 - \delta$. For the rest of the proof, assume it is the case.

There are two claims to the lemma:

1. Prove that Anchoring$(N, \hat{r}, \varepsilon, \delta/(N \log N))$ finds an anchoring $(A_1, A_2, \ldots, A_K)$ satisfying $W_{\hat{r}}(A_1, \ldots, A_K) = \mathcal{O}(\max_{i\in[N]} |\mathcal{N}_{2\varepsilon}(i)|)$.
2. Prove that the comparisons queried can be organized into $\mathcal{O}\left(\frac{\log(N/\delta)}{\varepsilon^2} \max_{i\in[N]} \log |\mathcal{N}_{2\varepsilon}(i)|\right)$ matchings, proving the upper bound on the time complexity.

To prove the first claim, make use of Lemma C.5. Indeed, if $z = \lfloor 1 + \log(|\mathcal{N}_{2\varepsilon}(i)|)\rfloor$ and $m(i)$ and $M(i)$ are two elements such that $\hat{r}(i) - \hat{r}(M(i)) = 2^z$, and $\hat{r}(i) - \hat{r}(m(i)) = -2^z$. Since $2^z > |\mathcal{N}_{2\varepsilon}(i)|$, $m(i)$ and $M(i)$ are both outside of $\mathcal{N}_\varepsilon(i)$ making comparing them to i a confident comparison.

Now, consider the following anchoring:

- $A_1 = \hat{r}^{-1}(1)$
- $A_{i+1} = m(A_i)$

this anchoring has a weight function of

$$W_{\hat{r}}(A_1, \ldots, A_K) = \max_{i\in[K]} |\mathcal{C}_i| \leq \max_{i\in[K]} 2|\mathcal{N}_{2\varepsilon}(A_i)| \quad (28)$$

This is because $|\mathcal{C}_i| \leq 2 \max\{|\mathcal{N}_{2\varepsilon}(A_{i-1})|, |\mathcal{N}_{2\varepsilon}(A_i)|\}$. In conclusion, the anchoring $(A_1, \ldots, A_K)$ constructed satisfies the claim $W_{\hat{r}}(A_1, \ldots, A_K) = \mathcal{O}\left(\max_{i\in[N]} |\mathcal{N}_{2\varepsilon(i)}|\right)$. This prove the first claim.

The second claim states that Algorithm 3 has a time complexity of $\mathcal{O}\left(\frac{\log(N/\delta)}{\varepsilon^2} \max_{i\in[N]} \log |\mathcal{N}_{2\varepsilon}(i)|\right)$. First, it follows from the first claim that every element i will encounter an opponent that is confidently stronger than him $(M(i))$ and one that is confidently weaker than him $(m(i))$ at stage $z = \lfloor 1 + \log |\mathcal{N}_{2\varepsilon}(i)|\rfloor$ of Algorithm 3 at the latest. Hence, at stage $z_m = \lfloor 1 + \max_{i\in[N]} \log |\mathcal{N}_{2\varepsilon}(i)|\rfloor$, every element i has either encountered corresponding opponents $m(i)$ and $M(i)$, or either $\hat{r}(i) - 2^z < 1$ or $\hat{r}(i) + 2^Z > N$. In both cases, $L[i] = U[i] = 1$ for all $i \in [N]$ (step 3 in Algorithm 3), hence Algorithm 3 halts at stage z_m at the latest.

Now, it remains to show that each stage costs $\mathcal{O}(\frac{\log(N/\delta)}{\varepsilon^2})$ matchings. we proceed as follows: W.l.o.g., suppose that $\hat{r} = I_d$. Recall that at stage z, Algorithm 3 queries comparisons between elements i, j such that $|i - j| = 2^z$. It is sufficient to show that all the queries of stage can be covered with only 2 matchings. The proof of that is as follows:

$[N]$ is divided into consecutive segments $S_k = \{2^{z+1}(k-1)+1, \dots, 2^{z+1}k\}$ for $k \in \{1, \dots, P = \lfloor N/(2^{z+1}) \rfloor\}$ of size 2^{z+1} each, with the last segment being $S_{P+1} = 2^{z+1}P+1, \dots, N$ (possibly empty). For $k \leq P$, segment S_k is divided into two halves: a left half

$$S_k^L = \{2^{z+1}(k-1)+1, \dots, 2^{z+1}k + 2^z\} \tag{29}$$

and right half

$$S_k^R = \{2^{z+1}k + 2^z + 1, 2^{z+1}k\}. \tag{30}$$

S_{P+1} is also divided into

$$S_{P+1}^L = \{2^{z+1}(P)+1, \dots, \min\{N, 2^{z+1}(P+1) + 2^z\}\} \tag{31}$$

and

$$S_{P+1}^R = \{2^{z+1}(P+1) + 2^z + 1, \dots, N\} \tag{32}$$

whenever it makes sense. The following properties are straightforward:

- There is no queries involving elements that are both in the same set S_k^R or S_k^L for any $k = 1, \dots, P+1$.
- If a comparison between i and j is queried with $i \leq j$, then there is a $k \in [P+1]]$ such that $i \in S_k^R$ and $j \in S_{k+1}^L$ or $i \in S_k^L$ and $j \in S_k^R$.
- Elements of S_1^L and S_{P+1}^R are involved in one query each.

If we match the elements of S_k^L to those of S_k^R in increasing order, we get a perfect matching on $[N] \setminus S_{P+1}$ totaling to $\lfloor N/2^{z+1} \rfloor$ queries of COMPARE. Notice that in this way, all queries of COMPARE that involve elements of S_1^L are already addressed. If S_{P+1} has more than 2^z elements, match elements of S_{P+1}^L with those of S_{P+1} in an increasing order corresponding to the queries. Match what is left arbitrarily. Notice that this way, all queries between elements of S_{P+1}^L and those of S_{P+1}^R are addressed. In particular, all queries involving S_{P+1}^R are addressed too. If S_{P+1} has 2^z or less elements, match arbitrarily. In summary, all queries involving elements between S_k^L and elements of S_k^R for all $k \in [P+1]$.

The leftover queries (involving elements from S_k^R with elements from S_{k+1}^L, for $k \in [P]$) are addressed now in a similar way. Elements of S_k^R are matched with those of S_{k+1}^L in increasing order. Since no queries involving elements of S_1^L are left for this stage z, they can be matched arbitrarily. It remains to indicate the way in which elements of S_P^R are matched. They are matched to those of S_{P+1}^L whenever possible. Match potential leftover arbitrarily. This way, all queried comparisons of stage z have been addressed.

In conclusion, we separated the queries of COMPARE, queried by Algorithm 3 in a single stage into two matchings. Since each comparisons is performed $\mathcal{O}\left(\frac{\log N/\delta}{\varepsilon^2}\right)$ times by COMPARE, each matching is repeated $\mathcal{O}\left(\frac{\log N/\delta}{\varepsilon^2}\right)$ too, implying a time complexity of $\frac{\log N/\delta}{\varepsilon^2}$ per stage of Algorithm 3, hence the second claim. □

C.3. Proof of Lemma 2.6

For convenience, Lemma 2.6 is restated:

Lemma C.6. *The time complexity of transitioning from stage s to stage $s+1$ of Algorithm 1 is*

$$\mathcal{O}\left(\frac{1}{\varepsilon_{s+1}^2} \log\left(\frac{N}{\delta_{s+1}}\right) \log^3\left(\max_{i\in[N]} \mathcal{N}_{2\varepsilon_{s+1}}(i)\right)\right). \tag{33}$$

Furthermore, the ranking obtained at the end of stage s is ε_{s+1}-correct with probability at least $1 - \sum_{l=1}^{s+1} \delta_l$.

Proof. This lemma is composed of two claims:

1. At the end of stage s, the ranking $\hat{r}_s$ is ε_s-correct with probability at least $1 - \sum_{l=1}^{s} \delta_l$.
2. The time complexity of stage s is

$$\mathcal{O}\left(\frac{1}{\varepsilon_s^2} \log\left(\frac{N}{\delta_s}\right) \log^3\left(\max_{i\in[N]} \mathcal{N}_{2\varepsilon_{s-1}}(i)\right)\right). \tag{34}$$

The proof is conducted through induction Suppose that $\hat{r}_s$ is indeed ε_s-correct.

First(line 3), Algorithm 1 generates anchors $(A_1, \dots, A_K)$ that provides a clustering of maximum cluster size of $\mathcal{O}\left(\max_{i\in[N]} |\mathcal{N}_{2\varepsilon_{s-1}}|\right)$. This step have a time complexity of $\mathcal{O}\left(\frac{\log(N/\delta)}{\varepsilon_s} \max_{i\in[N]|\mathcal{N}_{2\varepsilon_{s-1}}|(i)}\right)$, and is sound with probability at least $1 - \delta_s/3$.

Now that a clustering is available, Algorithm 1 constructs $(U_j)_j$ as indicated in line 6. We have that $|U_j| \leq |\mathcal{C}_j| + |\mathcal{C}_j| + 2 = \mathcal{O}\left(\max_{i\in[N]} |\mathcal{N}_{2\varepsilon_{s-1}}(i)|\right)$. Hence, line 7 would have a time complexity of $\mathcal{O}\left(\frac{\log(N/\delta_s)}{\varepsilon_s^2} \log^3(\max_{i\in[N]} |\mathcal{N}_{2\varepsilon_{s-1}}(i)|)\right)$. This step is correct with a probability at least $1 - \delta_s/6$: this is because Algorithm F.3, is correct with a probability at least $1 - \delta/6$ (using a union bound).

A similar reasoning is applied to line 10. As will be shown later, $|\mathcal{C}'_j| \leq |\mathcal{C}_{j+1}| + |\mathcal{C}_j| + |\mathcal{C}_{j-1}|$. It follows that $\max_j |U'_j| = \mathcal{O}\left(\max_{i\in[N]} |\mathcal{N}_{2\varepsilon_{s-1}}(i)|\right)$. Hence, with the same reasoning as for step 1, the time complexity of step 1 is shown to also be $\mathcal{O}\left(\frac{\log(N/\delta_s)}{\varepsilon_s^2} \max_{i\in[N]} \log^3 |\mathcal{N}_{2\varepsilon_{s-1}}(i)|\right)$. This step is also correct with probability at least $1 - \delta_s/6$.

The last step of the stage is the checking step (halting condition). It also is correct with probability at least $1-\delta_s/3$.

Hence, the stage s has a time complexity of $\mathcal{O}\left(\frac{\log(N/\delta_s)}{\varepsilon_s^2}\log^3(\max_{i\in[N]}|\mathcal{N}_{2\varepsilon_{s-1}}(i)|)\right)$. All the steps of stage s are simultaneously correct with probability at least $1-\delta_s$.

To conclude the proof, it remains to show that if all the steps of stage s are correct, then the ranking $\hat{r}_s$ obtained is indeed ε_s-correct. We will show the following claims that hold for any set of anchors $A_1, ..., A_K$:

1. $\hat{r}'$ is ε_s-correct on U_j.
2. $\hat{r}'$ is ε_{s-1}-correct on $[N]$.
3. $\forall j \in \{1,\dots,\lfloor\frac{K+1}{2}\rfloor\}$, $\forall (i_1,i_2) \in \mathcal{C}'_{2j}\times[N]$; $(\hat{r}'(i_1)<\hat{r}'(i_2))$ & $(\varepsilon(i_1,i_2)<-\varepsilon_s) \implies i_2 \in \mathcal{C}_{2j+1}$.
4. $\hat{r}_s$ is indeed ε_s-correct.

The third claim simply means that the only remaining misrankings preventing ranking $\hat{r}'$ from being ε_s-correct are between elements of $\mathcal{C}'_{2j}$ and $\mathcal{C}'_{2j+1}$ for $j\in\{1,\dots,\lfloor\frac{K+1}{2}\rfloor\}$.

1) $\hat{r}'$ is ε_s-correct on U_j by direct application of Lemma 2.2.

2) Since each $\hat{r}'$ is ε_s-correct on each U_j, it is ε_{s-1}-correct too, for comparisons between elements $i_1\in U_{j_1}$ and $i_2\in U_{j_2}$ ($j_1\neq j_2$), ε_{s-1}-correctness is inherited from $\hat{r}_{s-1}$. (elements in U_j according to $\hat{r}_s$ are the same as according to $\hat{r}_{s-1}$.)

3) The remaining misrankings are narrowed down as follows:

- There is no misrankings between elements of $\mathcal{C}'_{2j}$ and those of $\mathcal{C}'_l$ for $l\in[\lfloor\frac{K+1}{2}\rfloor]\setminus\{2j-2,\dots,2j+2\}$. This is a direct application of Lemma 2.4.
- Elements $i\in\mathcal{C}'_{2j-2}$ satisfy $r(i_1)<r(A_{2j-1})$: this is a direct application of Lemma 2.3 on (A_{2j-2},A_{2j-2}) and $\hat{r}'$ (since it is ε_{s-1}-correct).

  Let $(i_1,i_2)\in\mathcal{C}'_{2j-2}\times\mathcal{C}'_{2j}$. Either $(r(i_1)<r(i_2))$, in which case there is nothing to prove because $\hat{r}'$ and r are in agreement, or $(r(i_2)<r(i_1))$: Recall that elements in $i_2\in\mathcal{C}'_{2j}$ verify $\varepsilon(i_2,A_{2j-1})<\varepsilon_s$ (because $\hat{r}'$ is ε_s-correct on U_j and $A_{2j-1}\in U_j$). If $(r(i_2)<r(i_1)<r(A_{2j-1}))$, then it follows from Assumption 1.1 that $\varepsilon(i_2,i_1)\leq\varepsilon(i_2,A_{2j-1})\leq\varepsilon_s$. This means that i_1 and i_2 are not misranked for a precision ε_s, thus eliminating misrankings between elements of $\mathcal{C}'_{2j}$ and those of $\mathcal{C}'_{2j-2}$.

  Misrankings between elements of $\mathcal{C}'_{2j}$ and those of $\mathcal{C}'_{2j+2}$ are eliminated with the same reasoning.
- Misrankings between elements of $\mathcal{C}'_{2j}$ and those of $\mathcal{C}'_{2j-1}$ are automatically eleminited since $\hat{r}'$ is ε_s - correct on U_j, which includes $\mathcal{C}'_{2j}$ and $\mathcal{C}'_{2j-1}$.

Hence, the only remaining misrankings are potentially between elements of $\mathcal{C}'_{2j}$ and those of $\mathcal{C}'_{2j+1}$.

4) Step 1 corrects these potential misrankings, as it applies AKS with a precision ε_s on U'_j which contains $\mathcal{C}'_{2j}$ and $\mathcal{C}'_{2j+1}$.

Hence, after step 1, the algorithm will have eliminated all potential misranking preventing $\hat{r}_s$ from being ε_s-correct. Hence, $\hat{r}_s$ is indeed ε_s-correct. This is true provided that all the steps are simultaneously correct. This happens with probability at least $1-2\delta_s/3$.

The proof is concluded by induction: For $s=0$, $\hat{r}_s$ is $\varepsilon_0=1/2$-correct with probability 1 (every permutation is such). Let $s\in\mathbb{N}$, Suppose that at the end of the stage s, we have $\hat{r}_s$ is ε_s-correct with probability at least $1-\sum_{l=1}^{s}2\delta_s/3$. Then, as shown, $\hat{r}_{s+1}$ is (ε_{s+1})-correct with probability at least $1-2\delta_{s+1}/3$ given that $\hat{r}_s$ is ε_s-correct. Since $\hat{r}_s$ is ε_s-correct with probability at least $1-\sum_{l=1}^{s}2\delta_l/3$, through a union bound, $\hat{r}_{s+1}$ is proven to be ε_{s+1}-correct with probability at least $1-\sum_{l=1}^{s+1}2\delta_s/3$.

□

C.4. Proofs of Theorem 2.7 and Corollary 2.8

Theorem C.7. *Each player i is involved in at most*

$$\mathcal{O}\left(\frac{\log^3 N}{\min\{\Delta_i^2,\Delta_{i-1}^2\}}\left(\log\frac{N}{\delta}+\log\log\frac{1}{\min\{\Delta_i,\Delta_{i-1}\}}\right)\right) \tag{35}$$

comparisons before its rank can be confidently identified.

Proof. Let $i\in[N]$. if $\varepsilon_s^2<\min\left\{\Delta_i^2,\Delta_{i-1}^2\right\}$, then i will be flagged at step 1 as correctly ranked. Hence, i is involved in at most:

$$\sum_{l=1}^{s}\frac{\log N/\delta_l}{\varepsilon_l^2}\log^3(N)\leq\log^3(N)\sum_{l=1}^{s}4^l\log\frac{\pi^2Nl^2}{6\delta} \tag{36}$$

$$\leq 2\log^3(N)4^s\log\frac{\pi^2Ns^2}{6\delta} \tag{37}$$

$$=\mathcal{O}(\log^3(N)4^s\log\frac{Ns^2}{\delta}) \tag{38}$$

For $s=\lceil\log(\frac{1}{\min\{\Delta_i,\Delta_{i-1}\}})\rceil$, i is correctly ranked and would have been involved in at most:

$$\mathcal{O}\left(\frac{\log^3 N}{\min\{\Delta_i^2,\Delta_{i-1}^2\}}\left(\log\frac{N}{\delta}+\log\log\frac{1}{\min\{\Delta_i,\Delta_{i-1}\}}\right)\right) \tag{39}$$

□

Corollary C.8. *Algorithm 1 has a time complexity of*

$$\mathcal{O}\left(\frac{\log^3 N}{\min_{i\in[N]}\Delta_i^2}\left(\log(\frac{N}{\delta})+\log\log\frac{1}{\min_{i\in[N]}\Delta_i^2}\right)\right). \tag{40}$$

It returns the true ranking with probability at least $1-\delta$

Proof. If $\varepsilon_s < \log\frac{1}{\min_{i\in[N]}\Delta_i}$, then $\hat{r}_s$ is ε_s-correct, hence it corresponds to the true ranking (no gap is small enough to induce an error). The probability of error of Algorithm 1 is upper bounded by (union bound):

$$\sum_{l=1}^{\infty}\mathbb{P}(\{\textbf{an error occured at stage } l\}) \leq \sum_{l=1}^{\infty}\frac{6\delta}{\pi^2 l^2} = \delta \tag{41}$$

Hence, Algorithm 1 retrieves the exact ranking in the claimed time complexity with probability at least $1-\delta$. □

D. Going beyond worst cases

The underlying concept behind thresholding the "cost" of a matching stems from the idea that obtaining an exact ranking is prohibitively expensive and that games do not have to be exactly balanced (i.e., $\varepsilon(i,j)$ exactly equal to 0) to be interesting. Unfortunately, and as discussed above, thresholding does not allow bypassing the exact ranking retrieval process, and in general, obtaining the exact ranking is unavoidable (as instances exist where the exact matching is unique, and retrieving it implies retrieving the ranking, see also Section 2.1).

However, there are also many situations, obviously not the worst-case ones, where retrieving an optimal matching is nearly as costly as obtaining a ε^*-correct ranking. One typical such situation is given in Theorem D.2. We proceed to prove and illustrate this result in the remainder of this section.

W.l.o.g., we suppose in this section that the items are correctly ranked by there index, that is, $i < j$ implies $\epsilon(i,j) > 0$.

For any estimated ranking r_ϵ, possibly indexed by ϵ, we let $\sigma_\epsilon = r_\epsilon^{-1}$ so any element i is immediately preceded in the ranking r_ϵ by $j = \sigma_\varepsilon(r_\varepsilon(i)-1)$.

The following lemma allows to bound the gap between j and i when r_ε is ε-correct, when $i-1$ and i are sufficiently close.

Lemma D.1. *Let r_ε denote a ε-correct ranking and assume that $\varepsilon + \varepsilon(i-1,i)) \leq \varepsilon^*$. Then $j = \sigma_\varepsilon(r_\varepsilon(i)-1)$ satisfies $|\varepsilon(j,i)| \leq \varepsilon^*$.*

Proof. Suppose first that $j > i$ is actually weaker than i so i and j are improperly compared by r_ϵ. As r_ε is ε-correct, this implies that $0 < \varepsilon(i,j) \leq \varepsilon \leq \varepsilon^*$.

Suppose now that $j < i$ is in reality stronger than i. Then either $j = i-1$, so by hypothesis

$$0 < \varepsilon(j,i) = \varepsilon(i-1,i) \leqslant \varepsilon^* \ ,$$

or $j < i-1$, so j and $i-1$ are improperly compared by r_ε. In this last case, by Assumption 1.2,

$$0 < \varepsilon(k,i) \leq \varepsilon(k,i-1)+\varepsilon(i-1,i) \leq \varepsilon+\varepsilon(i-1,i) \leq \varepsilon^*.$$

□

Lemma D.1 allows to prove the following theorem that shows a typical situation where a staisfying matching can be obtained using only an η-correct ranking.

Theorem D.2. *Assume that there exists $\varepsilon > 0$ such that, for any $i \in [N-1]$, $\varepsilon(i,i+1) \leq \varepsilon^* - \varepsilon$. Let r_ε denote any ε-correct ranking. Let M_ε denote the matching $(\sigma_\varepsilon(2i-1) \sim \sigma_\varepsilon(2i), i \in [N/2])$ that pairs the items ranked $2i-1$ and $2i$ by r_ϵ. Then M_ε is a costless matching: $C_{M_\varepsilon} = 0$.*

Proof. It is sufficient to prove that, for any $i \in [N/2]$,

$$|\varepsilon(\sigma_\varepsilon(2i-1), \sigma_\varepsilon(2i))| \leq \varepsilon^*.$$

Fix $j = \sigma_\varepsilon(2i)$, we have $\varepsilon(j-1,j) \leqslant \varepsilon^* - \varepsilon$ by assumption so by Lemma D.1, $\sigma_\varepsilon(2i-1) = \sigma_\varepsilon(r_\varepsilon(j)-1)$ satisfies $|\varepsilon(\sigma_\varepsilon(2i-1), \sigma_\varepsilon(2i))| \leq \varepsilon^*$. □

A really nice consequence of Theorem D.2 is that, if all $\varepsilon(i,i+1) \leqslant \varepsilon^*/2$, then costless matchings can be built from any $\varepsilon^*/2$-correct ranking. The condition $\varepsilon(i,i+1) \leqslant \varepsilon^*/2$ can easily be tested online and it is straightforward to adapt our sampling policy in this case to prevent it from looking at a perfect ranking. The details are left to the interested reader.

A natural question that may arise from the preceeding result is wether the condition that all $\varepsilon(i,i+1) \leqslant \varepsilon^*/2$ is met in some examples. It turns out to be typically the case as shown by the following example. Assume that any i has a "value" X_i that is a non negative random variable, say uniform on $[0,1]$ to fix ideas, sampled independently from the other X_j. Assume moreover that $\epsilon(i,j) = \frac{X_i - X_j}{2}$. The rank r is then the function whose inverse $\sigma = r^{-1}$ satisfies

$$X_{\sigma(1)} > \ldots > X_{\sigma(n)} \ .$$

In words, the strongest item is the one with the highest value. In this setting, the condition

$$\max_{i\in[N-1]} \varepsilon(\sigma(i), \sigma(i+1)) \leqslant \frac{\varepsilon^*}{2}$$

is thus met as soon as the maximal spacing between uniforms satisfies

$$\max_{i\in[N-1]} X_{\sigma(i)} - X_{\sigma(i+1)} \leqslant \varepsilon^*.$$

The maximal spacing of uniforms has been extensively studied in probability and precise asymptotics are available, see (Devroye, 1981) for example, showing that it scales asymptotically as $\log N/N$. Therefore, as long as $\frac{\log N}{N} < c\varepsilon^*$ for some small constant c, the condition $\max_{i\in[N-1]} \varepsilon(\sigma(i), \sigma(i+1)) \leqslant \frac{\varepsilon^*}{2}$ is met with high probability in this example.

E. Connections with CombUCB (Kveton et al., 2015)

In this section, we shall draw some connections between the online ranking problem and combinatorial bandits, even though the two problems are quite different. We shall try as much as possible to use the same notation as (Kveton et al., 2015), and we will only provide high-level arguments (as, again, CombUCB analysis does not hold in the ranking problem, where the loss of sampling an edge is not the expectation of some random variable, but a non-linear transformation of it).

Formally, a stochastic combinatorial semi-bandit is a tuple $B = (E, \Theta, P)$, where $E = \{1, \dots, L\}$ is a finite set of L items, $\Theta \subseteq 2^E$ is a non-empty set of feasible subsets of E, and P is a probability distribution over a unit cube $[0,1]^E$. The items in the set E are associated with a vector of stochastic weights/losses $w \sim P$, whose e-th component, $w(e)$, is the weight of item e. The expected weights of the items are defined as $\bar{w} = \mathbb{E}_{w\sim P}[w]$. The loss of choosing solution A under the realization of the weights w is simply (this is the where lies the main different between bandit and ranking)

$$f(A, w) = \sum_{c\in A} w(e). \tag{42}$$

The maximum number of chosen items is defined as $K = \max_{A\in\Theta} |A|$.

Let $(w_s)_{s=1}^t$ be an i.i.d. sequence of t weights drawn from P. At time s, the learning agents chooses solution $A_s \in \Theta$ based on its observations of the weights up to time s, loses $f(A_s, w_s)$, and observes the weights of all chosen items at time $s, \{(e, w_s(e)) : e \in A_s\}$. The learning agent interacts with the environment t times and its goal is to minimize its expected cumulative reward over this time. If the agent knew P a priori, the optimal action would be to choose the optimal solution:

$$A^* = \arg\min_{A\in\Theta} f(A, \bar{w}). \tag{43}$$

at all steps s. The quality of the agent's policy is measured by its expected cumulative regret:

$$R(t) = \mathbb{E}\left[\sum_{s=1}^{t} R(A_s, w_s)\right], \tag{44}$$

where $R(A_s, w_s) = f(A, w_s) - f(A^*, w_s)$ is the instantenous regret of the agent at time s.

CombUCB algorithm (Kveton et al., 2015) works as follows: First, it computes qn upper confidence bound (UCB) on the expected weight of each item e, which would rewrite in the ranking problem as follows (since the weight is 0, if the skill gap is smaller than ε^*

$$U_t(e) = \min\Big\{\mathbb{1}\{\hat{w}_{T_{t-1}(e)}(e) + c_{t-1,T_{t-1}(e)} > \frac{1}{2} - \varepsilon^*\}, \\ \mathbb{1}\{\hat{w}_{T_{t-1}(e)}(e) - c_{t-1,T_{t-1}(e)} < \frac{1}{2} + \varepsilon^*\}\Big\},$$

where $\hat{w}_s(e)$ is the average of s observed weights of item $e, T_t(e)$ is the number of times that item e is observed after t rounds, and:

$$c_{t,s} = \sqrt{\frac{1.5\log t}{s}} \tag{45}$$

is the radius of a confidence interval around $\hat{w}_s(e)$ at time t such that $\bar{w}(e) \in C(e, s, t) = [\hat{w}_s(e) - c_{t,s}, \hat{w}_s(e) + c_{t,s}]$ holds with high probability. Second, CombUCB calls the optimization oracle to solve the combinatorial problem on the UCBs:

$$A_t = \arg\min_{A\in\Theta} f(A, U_t). \tag{46}$$

Finally, CombUCB chooses A_t, observes the weights of all chosen items, and updates the estimates of $\bar{w}(e)$ for these items. In the case at hand, $E = \{\{i, j\}, i \neq j\}$, so much that $L = \frac{N(N-1)}{2}$, Θ the feasible set is the set of maximal size matchings, which implies that $K = N/2$ and P a product of independent Bernoulli variables such that $\omega_{i,j} \sim Ber(p(i, j))$.

Consider the following situation: $\varepsilon(i, i+1) = \varepsilon^* - \Delta$ and $\varepsilon(i, i+k) = \varepsilon^* + \Delta$ if $k \geq 2$.

To show that this situation yields a regret of $\frac{N^2\log(N/\delta)}{\Delta^2_{\min}}$, we can assume that the true edges of the graph G are kept in the estimated graph. To eliminate another edge, CombUCB - that does not take into account the ranking structure - must have sampled it $\frac{\log(N/\delta)}{\Delta^2}$ times. This adds up to a total cost of $O(\frac{N^2\log(N/\delta)}{\Delta^2})$. Checking that in this case $2\Delta = \Delta_{\min}$ is straightforward.

F. Omitted Pseudo-codes

F.1. Pseudo-code of COMPARE

Algorithm 2 COMPARE$(i, j, \varepsilon, \delta)$

Initialize: $b \leftarrow \lceil \frac{2}{\varepsilon^2} \log \frac{2}{\delta} \rceil$, $w_{i,j} \leftarrow 0$

for $t \leftarrow 1$ to b **do**
  Compare i and j; Update $w_{i,j} \leftarrow w_{i,j} + 1$ if i wins;
end for
$\hat{p}_{i,j} \leftarrow w_{i,j}/b$;
if $\hat{p}_{i,j} > 1/2$ **then**
  Output i;
else
  Output j;
end if
if $|\hat{p}_i - \frac{1}{2}| > 2\varepsilon$ **then**
  Output "Confident";
else
  Output "Not Confident";
end if

F.2. Pseudo-codes of ANCHORING

Algorithm 3 ANCHORING$(N, \hat{r}, \varepsilon, \delta)$

Input: $\hat{r}$: ranking, $\varepsilon > 0$, $\delta > 0$;
Initialize: $L = U = [0, \ldots, 0] \in \mathbb{R}^N$, $z = 0$, G: Empty graph (without edge);

while $L \neq [1, \ldots, 1]$ **or** $U \neq [1, \ldots, 1]$ **do**
  for $i, j \in [N]$ such that $|\hat{r}(i) - \hat{r}(j)| = 2^z$ **do**
    $(k, \text{string}) = \text{COMPARE}\left(\{i, j\}, \varepsilon, \frac{\delta}{2N \log N}\right)$;
    ▷ requires 2 perfect matchings;
    $\{\ell\} = \{i, j\} \setminus \{k\}$;
    if string ="Confident" **then**
      $L(k) = 1$; $U(\ell) = 1$;
      add the edge $\{i, j\}$ to the graph G;
    end if
  end for
  for $i \in [N]$ such that $\hat{r}(i) < 2^z$ **do**
    $L(i) = 1$;
  end for
  for $i \in [N]$ such that $\hat{r}(i) + 2^z > N$ **do**
    $U(i) = 1$;
  end for
  z=z+1;
end while
return; $\underset{s.t. \begin{cases} (A_1, \ldots, A_\ell) \text{ path of } G \\ \hat{r}(A_i) \leq \hat{r}(A_{i+1}), \forall i \end{cases}}{\arg\min} W_{\hat{r}}(A_1, \ldots, A_\ell)$.

F.3. Sub-protocols used in Algorithm 1

Algorithm 4 CHECK(ε, δ)

Input: a ranking $\hat{r}$ and its inverse $\hat{\sigma} = \hat{r}^{-1}$
Initialize: $L = [0, \ldots, 0] \in \mathbb{R}^N$,

for $i \in [N]$ **do**
  if COMPARE$(\{\hat{\sigma}(i), \hat{\sigma}(i+1)\}, 2\varepsilon, \frac{\delta}{N+1}) = (\hat{\sigma}(i)$,"Confident") **and** COMPARE$(\{\hat{\sigma}(i), \sigma(i-1)\}, 2\varepsilon, \frac{\delta}{N+1}) = (\hat{\sigma}(i-1)$,"Confident") **then**
    $L[\hat{\sigma}(i)] = 1 : \hat{\sigma}(i)$ is correctly ranked;
  end if
end for ▷ requires 2 perfect matchings;
return L;

Algorithm 5 GRP1$((A_1, \dots, A_K), (\mathcal{C}_1, \dots, \mathcal{C}_{K+1})$

Input: $(A_1, \dots, A_K)$: Anchors; $(\mathcal{C}_1, \dots, \mathcal{C}_{K+1})$: Clusters;
Initialize: $A_0 = \hat{\sigma}(1)$, $U_0 = \emptyset$, $\mathcal{C}_0 = \emptyset$, $A_{K+1} = \hat{\sigma}(N)$.

for $j \in \{1, \dots, \lfloor |\frac{K+1}{2}| \rfloor\}$ **do**
  $U_j \leftarrow \mathcal{C}_{2j-1} \cup \{A_{2j-1}\} \cup \mathcal{C}_{2j}$.
  if (size of U_j is odd) **and** $(A_{2j-2} \notin U_{j-1})$ **then**
    $U_j \leftarrow U_j \cup \{A_{2j-2}\}$.
  else if (size of U_j is odd) **then**
    $U_j \leftarrow U_j \cup \{A_{2j}\}$.
  end if
end for
return $(U_1, \dots, U_{\lfloor |\frac{K+1}{2}| \rfloor})$.

Algorithm 6 GRP2$((A_1, \dots, A_K), (\mathcal{C}_1, \dots, \mathcal{C}_{K+1})$

Input: $(A_1, \dots, A_K)$: Anchors; $(\mathcal{C}_1, \dots, \mathcal{C}_{K+1})$: Clusters;
Initialize: $A_0 = \hat{\sigma}(1)$, $U_0 = \emptyset$, $\mathcal{C}_0 = \emptyset$, $A_{K+1} = \hat{\sigma}(N)$.

$U_0 = \mathcal{C}_1$
if size of U_0 is odd **then**
  $U_0 \leftarrow U_0 \cup \{A_1\}$.
end if
for $j \in \{1, \dots, \lfloor |\frac{K+1}{2}| \rfloor\}$ **do**
  $U_j \leftarrow \mathcal{C}_{2j} \cup \{A_{2j}\} \cup \mathcal{C}_{2j+1}$.
  if (size of U_j is odd) **and** $(A_{2j-1} \notin U_{j-1})$ **then**
    $U_j \leftarrow U_j \cup \{A_{2j-1}\}$.
  else if (size of U_j is odd) **then**
    $U_j \leftarrow U_j \cup \{A_{2j+1}\}$.
  end if
end for
return $(U_1, \dots, U_{\lfloor |\frac{K+1}{2}| \rfloor})$.

Algorithm 7 CLUSTERING$((A_1, \dots, A_K), \hat{r})$

Input: $(A_1, \dots, A_K)$: Anchors; $\hat{r}$: Ranking;
Initialize: $A_0 = \hat{\sigma}(1)$, $A_{K+1} = \hat{\sigma}(N)$.

for $j \in \{1, \dots, K+1\}$ **do**
  $\mathcal{C}_j \leftarrow \left\{ i \in [N], \hat{r}(A_{j-1}) < \hat{r}(u) < \hat{r}_s(A_j) \right\}$,.
end for
return $(\mathcal{C}_1, \dots, \mathcal{C}_{K+1})$.

Algorithm 8 MULTIAKS$((U_1, \dots, U_L), \varepsilon, \delta)$

Input: $(U_1, \dots, U_L)$: list of sets;
Output: $(\hat{r}_1, \dots, \hat{r}_L)$ rankings on $U_1 \cup \dots \cup U_L$

for $j \in \{1, \dots, L\}$ **do**
  $\hat{r}_j \leftarrow$ AKS$(U_j, \varepsilon, \delta/L)$.
    ▷ This loop is run in parallel
end for
return $(\hat{r}_1, \dots, \hat{r}_L)$.

F.4. Pseudocode of SAMPLEMATCHING

Algorithm 9 SAMPLEMATCHING

Input Graph G, $n_{i,i+1} \in \mathbb{N}$, $r_{i,i+1} \in \mathbb{N}$
Output M: maximal matching on G

Construct $\mathcal{C}_1, \dots, \mathcal{C}_k$ connected components of G;
for $j \in [k]$ **do**
  $M_j = \underset{m \text{ max. match. on } \mathcal{C}_j}{\arg\min} \max_{\{i,i+1\} \in m} \frac{r_{i,i+1}}{n_{i,i+1}} - u(n_{i,i+1})$;
end for
$M = \cup_j M_j$;
Complete M arbitrarily into a perfect matching of $[N]$;
return M;

F.5. Pseudo-code of GAPESTIMATION

Algorithm 10 GAPESTIMATION

for $i \in [N-1]$ **do**
  Set $e_{i,i+1} = 1$, $n_{i,i+1} = r_{i,i+1} = l_{i,i+1} = 0$;
end for
while True **do**
  $G = ([N], \{e_{i,i+1}\})$;
  $M = \text{SampleMatching}(G, (n_{i,i+1})_i, (r_{i,i+1})_i)$;
    ▷ requires 1 perfect matching;
  Query comparisons according to M;
  for $i \in [N-1]$ **do**
    $n_{i,i+1} \leftarrow n_{i,i+1} + \mathbb{1}\left(\{i, i+1\} \in M\right)$;
    $r_{i,i+1} \leftarrow r_{i,i+1} + \mathbb{1}\left(i \textbf{ won against } i+1\right)$;
  end for
  for $i \in [N-1]$ **do**
    $\hat{\varepsilon}(i, i+1) = \frac{r_{i,i+1}}{n_{i,i+1}} - \frac{1}{2}$;
    $e_{i,i+1} = \mathbb{1}\{\hat{\varepsilon}(i, i+1) - u(n_{i,i+1}, \delta) \leq \varepsilon^*\}$;
    $l_{i,i+1} = \mathbb{1}\{|\hat{\varepsilon}(i, i+1) - \varepsilon^*| > u(n_{i,i+1}, \delta)\}$;
  end for
  if There is maximal matching M^* of G whose edges $e \in M^*$ all satisfy $l_e = 1$ **then**
    return M^*;
    break;
  end if
end while