Elo-MMR: A Rating System for Massive Multiplayer Competitions

Aram Ebtekar
Vancouver, BC, Canada
aramebtech@gmail.com

Paul Liu
Stanford University
Stanford, CA, USA
paul.liu@stanford.edu

ABSTRACT

Rating systems play an important role in competitive sports and games. They provide a measure of player skill, incentivize competitive performances, and are crucial to providing balanced match-ups. In this paper, we present a novel Bayesian rating system for contests with many participants. It is widely applicable to competition formats with discrete ranked matches, such as online programming competitions, obstacle courses races, and some video games. The simplicity of our system allows us to prove theoretical bounds on robustness and runtime. In addition, we show that the system *aligns incentives*: that is, a player who seeks to maximize their rating will never want to underperform. Experimentally, the rating system rivals or surpasses existing systems in prediction accuracy, and computes faster than existing systems by up to an order of magnitude.

ACM Reference Format:
Aram Ebtekar and Paul Liu. 2021. Elo-MMR: A Rating System for Massive Multiplayer Competitions. In *Proceedings of The Web Conference 2021 (WWW21), April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3357713.3384311

1 INTRODUCTION

Competitions, in the form of sports, games, and examinations, have been with us since antiquity. Many competitions grade performances along a numerical scale, such as a score on a test or a completion time in a race. In the case of a college admissions exam or a track race, scores are standardized so that a given score on two different occasions carries the same meaning. However, in events that feature either novelty or subjectivity, standardization is difficult. The Spartan Races, completed by millions of runners, feature a variety of obstacles placed on hiking trails around the world [7]. Rock climbing, a sport to be added to the 2020 Olympics, likewise has routes set specifically for each competition. Academic olympiads, too, feature creative challenges. DanceSport, gymnastics, and figure skating competitions have a panel of judges who rank contestants against one another; these subjective scores are known to be noisy [25]. In all these cases, scores can only be used to compare and rank participants at the same event. Players, spectators, and contest organizers who are interested in comparing players' skill levels across different competitions will need to aggregate the entire history of such rankings. A strong player, then, is one who consistently wins against weaker players. To quantify skill, we need a *rating system.*

Good rating systems are difficult to create, as they must balance several mutually constraining objectives. First and foremost, the rating system must be accurate, in that ratings provide useful predictors of contest outcomes. Second, the ratings must be efficient to compute: in video game applications, rating systems are predominantly used for matchmaking in massively multiplayer online games (such as Halo, CounterStrike, League of Legends, etc.) [19, 23, 27]. These games have hundreds of millions of players playing tens of millions of games per day, necessitating certain latency and memory requirements for the rating system [8]. Third, the rating system must align incentives. That is, players should not modify their performance to "game" the rating system. Rating systems that can be gamed often create disastrous consequences to player-base, more often than not leading to the loss of players from the game [3]. Finally, the ratings provided by the system must be human-interpretable: ratings are typically represented to players as a single number encapsulating their overall skill, and players often desire the ability to precisely predict the effect of their own performance on their rating [15].

Classically, rating systems were designed for two-player games. The famous Elo system [12], as well as its Bayesian successors Glicko and Glicko-2, have been widely applied to games such as Chess and Go [15–17]. Both Glicko versions model each player's skill as a real random variable that evolves with time according to Brownian motion. Inference is done by entering these variables into the Bradley-Terry model, which predicts probabilities of game outcomes. Glicko-2 refines the Glicko system by adding a rating volatility parameter. Unfortunately, Glicko-2 is known to be flawed in practice, potentially incentivising players to lose. This was most notably exploited in the popular game of Pokemon Go [3]; see Section 5.1 for further discussion.

The family of Elo-like methods just described only utilize the binary outcome of a match. In settings where a scoring system provides a more fine-grained measure of match performance, Kovalchik [21] has shown variants of Elo that are able to take advantage of score information. For competitions consisting of several set tasks (such as an academic olympiad), Forišek [13] developed a model in which each task gives a different "response" to the player: the total response then predicts match outcomes. However, such systems are often highly application-dependent and hard to calibrate.

Though Elo-like systems are widely used in two-player contests, one needn't look far to find competitions that involve much more

WWW '21, April 19–23, 2021, Ljubljana, Slovenia

ACM ISBN 978-1-4503-6979-4/20/06…$15.00
https://doi.org/10.1145/3357713.3384311

than two players. Examples include video games such as Counter-Strike and Halo, as well as academic olympiads: notably, programming contest platforms such as Codeforces and TopCoder [2, 6]. In these applications, the number of contestants can easily reach into the thousands. Much of recent work present interesting approaches to deal with competitions between two teams [10, 18, 20, 22], but they do not present efficient extensions for settings in which players are sorted into *more than two* (e.g., thousands of) distinct places.

For individual competitions, a player's final ranking is not the result of independent 1v1 matchups. Instead, variants of TrueSkill [11, 19, 23, 24] model a player's performance during each contest as a single random variable. The overall rankings are assumed to reveal the total order among these hidden performance variables, with various strategies used to model ties and teams. TrueSkill and its variants are efficient in practice, successfully rating userbases that number in the tens of millions (the Halo series, for example, has over 60 million sales since 2001 [4]).

The main disadvantage of TrueSkill is its complexity: originally developed by Microsoft for the popular Halo video game, TrueSkill performs approximate belief propagation on a factor graph, which is iterated until convergence [19]. Aside from being less human-interpretable and difficult to implement correctly, this complexity means that, to our knowledge, there are no proofs of key properties such as runtime and incentive alignment. Even when these properties are discussed [23], no rigorous justification is provided. In addition, we are not aware of any work that extends TrueSkill to non-Gaussian performance models, which might be desirable to limit the influence of outlier performances (see Section 5.2).

It might be for these reasons that platforms such as Codeforces and TopCoder opted for their own custom rating systems. These systems are not published in academia and do not come with Bayesian justifications. However, they retain the formulaic simplicity of Elo and Glicko, extending them to settings with much more than two players. The Codeforces system includes ad hoc heuristics to distinguish top players, while curbing rampant inflation. TopCoder's formulas are more principled from a statistical perspective; however, it has a volatility parameter similar to Glicko-2, and hence suffers from similar exploits [13]. Despite their flaws, these systems have been in place for over a decade, and have more recently been adopted by additional platforms such as LeetCode and CodeChef [1, 5].

Our contributions. In this paper, we describe the Elo-MMR rating system, obtained by a principled approximation of a Bayesian model very similar to TrueSkill. It is fast, embarrassingly parallel, and makes accurate predictions. Most interesting of all, its simplicity allows us to rigorously analyze its properties: the "MMR" in the name stands for "Massive", "Monotonic", and "Robust". "Massive" means that it supports any number of players with a runtime that scales linearly; "monotonic" means that it *aligns incentives* so that a rating-maximizing player always wants to perform well; "robust" means that rating changes are bounded, with the bound being smaller for more consistent players than for volatile players. Robustness turns out to be a natural byproduct of accurately modeling performances with heavy-tailed distributions, such as the logistic. TrueSkill is believed to satisfy the first two properties, albeit without proof, but fails robustness. Codeforces only satisfies aligned incentives, and TopCoder only satisfies robustness.

Experimentally, we show that Elo-MMR achieves state-of-the-art performance in terms of both runtime and prediction accuracy. In particular, we process the entire Codeforces contest database of over 300K users and 1000 contests in less than 30ms per contest on average, beating the existing Codeforces system by an order of magnitude while improving upon its accuracy. A difficulty we faced was the scarcity of efficient open-source rating system implementations. In an effort to aid researchers and practitioners alike, we provide open-source implementations of all rating systems, datasets, and additional processing used in our experiments at https://github.com/EbTech/EloR/.

Organization. In Section 2, we formalize the details of our Bayesian model. We then show how to estimate ratings under this model in Section 3, leaving the reader with an intuitive understanding of our algorithm. As a further refinement, Section 4 models skill evolutions from players training between competitions. This modeling is quite tricky as we choose to retain players' momentum while ensuring it cannot be exploited for incentive-misaligned rating gains. Section 5 proves that the system as a whole satisfies several salient properties, the most critical of which is aligned incentives. Finally, we present experimental evaluations in Section 6.

2 A BAYESIAN MODEL FOR MASSIVE COMPETITIONS

We now describe the setting formally, denoting random variables by capital letters. A series of competitive **rounds**, indexed by $t = 1, 2, 3, \ldots$, take place sequentially in time. Each round has a set of participating **players** $\mathcal{P}_t$, which may in general overlap between rounds. A player's **skill** is likely to change with time, so we represent the skill of player i at time t by a real random variable $S_{i,t}$.

In round t, each player $i \in \mathcal{P}_t$ competes at some **performance** level $P_{i,t}$, typically close to their current skill $S_{i,t}$. The deviations $\{P_{i,t} - S_{i,t}\}_{i \in \mathcal{P}_t}$ are assumed to be i.i.d. and independent of $\{S_{i,t}\}_{i \in \mathcal{P}_t}$.

Performances are not observed directly; instead, a ranking gives the relative order among all performances $\{P_{i,t}\}_{i \in \mathcal{P}_t}$. In particular, ties are modelled to occur when performances are exactly equal, a zero-probability event when their distributions are continuous.[1] This ranking constitutes the observational **evidence** E_t for our Bayesian updates. The rating system seeks to estimate the skill $S_{i,t}$ of every player at the present time t, given the historical round rankings $E_{\le t} := \{E_1, \ldots, E_t\}$.

We overload the notation Pr for both probabilities and probability densities: the latter interpretation applies to zero-probability events, such as in $\Pr(S_{i,t} = s)$. We also use colons as shorthand for collections of variables differing only in a subscript: for instance, $P_{:,t} := \{P_{i,t}\}_{i \in \mathcal{P}_t}$. The joint distribution described by our Bayesian model factorizes as follows:

$$\begin{aligned} &\Pr(S_{:,:}, P_{:,:}, E_:) &(1)\\ &= \prod_i \Pr(S_{i,0}) \prod_{i,t} \Pr(S_{i,t} \mid S_{i,t-1}) \prod_{i,t} \Pr(P_{i,t} \mid S_{i,t}) \prod_t \Pr(E_t \mid P_{:,t}), \end{aligned}$$

[1] The relevant limiting procedure is to treat performances within ϵ-width buckets as ties, and letting $\epsilon \to 0$. This technicality appears in the proof of Theorem 3.2.

where $\Pr(S_{i,0})$ is the initial skill prior,
$\Pr(S_{i,t} \mid S_{i,t-1})$ is the skill evolution model (Section 4),
$\Pr(P_{i,t} \mid S_{i,t})$ is the performance model, and
$\Pr(E_t \mid P_{:,t})$ is the evidence model.

For the first three factors, we will specify log-concave distributions (see Definition 3.1). The evidence model, on the other hand, is a deterministic indicator. It equals one when E_t is consistent with the relative ordering among $\{P_{i,t}\}_{i \in \mathcal{P}_t}$, and zero otherwise.

Finally, our model assumes that the number of participants $|\mathcal{P}_t|$ is large. The main idea behind our algorithm is that, in sufficiently massive competitions, from the evidence E_t we can infer very precise estimates for $\{P_{i,t}\}_{i \in \mathcal{P}_t}$. Hence, we can treat these performances as if they were observed directly.

That is, suppose we have the skill prior at round t:

$$\pi_{i,t}(s) := \Pr(S_{i,t} = s \mid P_{i,<t}). \tag{2}$$

Now we observe E_t. By Equation (1), it is conditionally independent of $S_{i,t}$, given $P_{i,\le t}$. Using the law of total probability,

$$\begin{aligned}&\Pr(S_{i,t} = s \mid P_{i,<t}, E_t)\\ &= \int \Pr(S_{i,t} = s \mid P_{i,<t}, P_{i,t} = p) \Pr(P_{i,t} = p \mid P_{i,<t}, E_t)\, \mathrm{d}p\\ &\to \Pr(S_{i,t} = s \mid P_{i,\le t}) \quad \text{almost surely as } |\mathcal{P}_t| \to \infty.\end{aligned}$$

The integral is intractable in general, since the performance posterior $\Pr(P_{i,t} = p \mid P_{i,<t}, E_t)$ depends not only on player i, but also on our belief regarding the skills of all $j \in \mathcal{P}_t$. However, in the limit of infinite participants, Doob's consistency theorem [14] implies that it concentrates at the true value $P_{i,t}$. Since our posteriors are continuous, the convergence holds for all s simultaneously.

Indeed, we don't even need the full evidence E_t. Let $E^L_{i,t} = \{j \in \mathcal{P} : P_{j,t} > P_{i,t}\}$ be the set of players against whom i lost, and $E^W_{i,t} = \{j \in \mathcal{P} : P_{j,t} < P_{i,t}\}$ be the set of players against whom i won. That is, we only see who wins, draws, and loses against i. $P_{i,t}$ remains identifiable using only $(E^L_{i,t}, E^W_{i,t})$, which will be more convenient for our purposes.

Passing to the limit $|\mathcal{P}_t| \to \infty$ serves to justify another common simplification made by algorithms such as TrueSkill: since conditioning on $P_{i,\le t}$ makes the skills of different players independent of one another, it becomes accurate to model them as such. In addition to simplifying derivations, this fact ensures that a player's posterior is unaffected by rounds in which they are not a participant, arguably a desirable property in its own right.

Finally, a word on the rate of convergence. Suppose we want our estimate to be within ϵ of $P_{i,t}$, with probability at least $1 - \delta$. By asymptotic normality of the posterior [14], it suffices to have $O(\frac{1}{\epsilon^2}\sqrt{\log \frac{1}{\delta}})$ participants.

When the initial prior, performance model, and evolution model are all Gaussian, treating $P_{i,t}$ as certain is the *only* simplifying approximation we will make; that is, in the limit $|\mathcal{P}_t| \to \infty$, our method performs *exact* inference on Equation (1). In the following sections, we focus some attention on generalizing the performance model to non-Gaussian log-concave families, parametrized by location and scale. We will use the logistic distribution as a running example, though our framework is agnostic to the specific distributions used.

The **rating** $\mu_{i,t}$ of player i after round t should be a statistic that summarizes their posterior distribution: we'll use the maximum a posteriori (MAP) estimate, obtained by setting s to maximize the posterior $\Pr(S_{i,t} = s \mid P_{i,\le t})$. By Bayes' rule,

$$\mu_{i,t} := \arg\max_s \pi_{i,t}(s) \Pr(P_{i,t} \mid S_{i,t} = s). \tag{3}$$

This objective suggests a two-phase algorithm to update each player $i \in \mathcal{P}_t$ at round t. In phase one, we estimate $P_{i,t}$ from $(E^L_{i,t}, E^W_{i,t})$. By Doob's consistency theorem, our estimate is extremely precise when $|\mathcal{P}_t|$ is large, so we assume it to be exact. In phase two, we update our posterior for $S_{i,t}$ and the rating $\mu_{i,t}$ according to Equation (3).

We will occasionally make use of the **prior rating**, defined as

$$\mu^\pi_{i,t} := \arg\max_s \pi_{i,t}(s).$$

3 A TWO-PHASE ALGORITHM FOR SKILL ESTIMATION

3.1 Performance estimation

In this section, we describe the first phase of Elo-MMR. For notational convenience, we assume all probability expressions to be conditioned on the **prior context** $P_{i,<t}$, and omit the subscript t.

Our prior belief on each player's skill S_i implies a prior distribution on P_i. Let's denote its probability density function (pdf) by

$$f_i(p) := \Pr(P_i = p) = \int \pi_i(s) \Pr(P_i = p \mid S_i = s)\, \mathrm{d}s, \tag{4}$$

where $\pi_i(s)$ was defined in Equation (2). Let

$$F_i(p) := \Pr(P_i \le p) = \int_{-\infty}^{p} f_i(x)\, \mathrm{d}x,$$

be the corresponding cumulative distribution function (cdf). For the purpose of analysis, we'll also define the following "loss", "draw", and "victory" functions:

$$\begin{aligned}l_i(p) &:= \frac{d}{dp}\ln(1 - F_i(p)) = \frac{-f_i(p)}{1 - F_i(p)},\\ d_i(p) &:= \frac{d}{dp}\ln f_i(p) = \frac{f_i'(p)}{f_i(p)},\\ v_i(p) &:= \frac{d}{dp}\ln F_i(p) = \frac{f_i(p)}{F_i(p)}.\end{aligned}$$

Evidently, $l_i(p) < 0 < v_i(p)$. Now we define what it means for the deviation $P_i - S_i$ to be log-concave.

Definition 3.1. *An absolutely continuous random variable on a convex domain is* ***log-concave*** *if its probability density function f is positive on its domain and satisfies*

$$f(\theta x + (1 - \theta)y) > f(x)^\theta f(y)^{1-\theta},\ \forall \theta \in (0, 1), x \neq y.$$

We note that log-concave distributions appear widely, and include the Gaussian and logistic distributions used in Glicko, TrueSkill, and many others. We'll see inductively that our prior π_i is log-concave at every round. Since log-concave densities are closed under convolution [9], the independent sum $P_i = S_i + (P_i - S_i)$ is also log-concave. The following lemma (proved in the appendix) makes log-concavity very convenient:

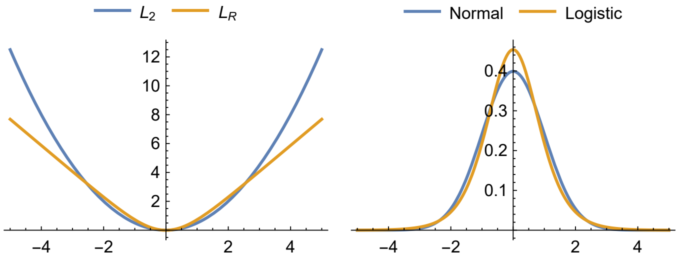

Figure 1: L_2 versus L_R for typical values (left). Logistic versus Gaussian distribution (right).

Lemma 3.1. *If f_i is continuously differentiable and log-concave, then the functions l_i, d_i, v_i are continuous, strictly decreasing, and*

$$l_i(p) < d_i(p) < v_i(p) \text{ for all } p.$$

For the remainder of this section, we fix the analysis with respect to some player i. As argued in Section 2, P_i concentrates very narrowly in the posterior. Hence, we can estimate P_i by its MAP, choosing p so as to maximize:

$$\Pr(P_i = p \mid E_i^L, E_i^W) \propto f_i(p) \Pr(E_i^L, E_i^W \mid P_i = p).$$

Define $j \succ i$, $j \prec i$, $j \sim i$ as shorthand for $j \in E_i^L$, $j \in E_i^W$, $j \in \mathcal{P} \setminus (E_i^L \cup E_i^W)$ (that is, $P_j > P_i$, $P_j < P_i$, $P_j = P_i$), respectively. The following theorem yields our MAP estimate:

Theorem 3.2. *Suppose that for all j, f_j is continuously differentiable and log-concave. Then the unique maximizer of* $\Pr(P_i = p \mid E_i^L, E_i^W)$ *is given by the unique zero of*

$$Q_i(p) := \sum_{j \succ i} l_j(p) + \sum_{j \sim i} d_j(p) + \sum_{j \prec i} v_j(p).$$

The proof is relegated to the appendix. Intuitively, we're saying that the performance is the balance point between appropriately weighted wins, draws, and losses. Let's look at two specializations of our general model, to serve as running examples in this paper.

Gaussian performance model. If both S_j and $P_j - S_j$ are assumed to be Gaussian with known means and variances, then their independent sum P_j will also be a known Gaussian. It is analytic and log-concave, so Theorem 3.2 applies.

We substitute the well-known Gaussian pdf and cdf for f_j and F_j, respectively. A simple binary search, or faster numerical techniques such as the Illinois algorithm or Newton's method, can be employed to solve for the maximizing p.

Logistic performance model. Now we assume the performance deviation $P_j - S_j$ has a logistic distribution with mean 0 and variance β^2. In general, the rating system administrator is free to set β differently for each contest. Since shorter contests tend to be more variable, one reasonable choice might be to make $1/\beta^2$ proportional to the contest duration.

Given the mean and variance of the skill prior, the independent sum $P_j = S_j + (P_j - S_j)$ would have the same mean, and a variance that's increased by β^2. Unfortunately, we'll see that the logistic performance model implies a form of skill prior from which it's tough to extract a mean and variance. Even if we could, the sum does not yield a simple distribution.

For experienced players, we expect S_j to contribute much less variance than $P_j - S_j$; thus, in our heuristic approximation, we take P_j to have the same form of distribution as the latter. That is, we take P_j to be logistic, centered at the prior rating $\mu_j^\pi = \arg\max \pi_j$, with variance $\delta_j^2 = \sigma_j^2 + \beta^2$, where σ_j will be given by Equation (8). This distribution is analytic and log-concave, so the same methods based on Theorem 3.2 apply. Define the scale parameter $\bar{\delta}_j = \frac{\sqrt{3}}{\pi}\delta_j$. A logistic distribution with variance δ_j^2 has cdf and pdf:

$$F_j(x) = \frac{1}{1 + e^{-(x-\mu_j^\pi)/\bar{\delta}_j}} = \frac{1}{2}\left(1 + \tanh\frac{x - \mu_j^\pi}{2\bar{\delta}_j}\right),$$

$$f_j(x) = \frac{e^{(x-\mu_j^\pi)/\bar{\delta}_j}}{\bar{\delta}_j\left(1 + e^{(x-\mu_j^\pi)/\bar{\delta}_j}\right)^2} = \frac{1}{4\bar{\delta}_j}\operatorname{sech}^2\frac{x - \mu_j^\pi}{2\bar{\delta}_j}.$$

The logistic distribution satisfies two very convenient relations:

$$F_j'(x) = f_j(x) = F_j(x)(1 - F_j(x))/\bar{\delta}_j,$$

$$f_j'(x) = f_j(x)(1 - 2F_j(x))/\bar{\delta}_j,$$

from which it follows that

$$d_j(p) = \frac{1 - 2F_j(p)}{\bar{\delta}} = \frac{-F_j(p)}{\bar{\delta}} + \frac{1 - F_j(p)}{\bar{\delta}} = l_j(p) + v_j(p).$$

In other words, a tie counts as the sum of a win and a loss. This can be compared to the approach (used in Elo, Glicko, TopCoder, and Codeforces) of treating each tie as half a win plus half a loss.[2]

Finally, putting everything together:

$$Q_i(p) = \sum_{j \succeq i} l_j(p) + \sum_{j \preceq i} v_j(p) = \sum_{j \succeq i} \frac{-F_j(p)}{\bar{\delta}_j} + \sum_{j \preceq i} \frac{1 - F_j(p)}{\bar{\delta}_j}.$$

Our estimate for P_i is the zero of this expression. The terms on the right correspond to probabilities of winning and losing against each player j, weighted by $1/\bar{\delta}_j$. Accordingly, we can interpret $\sum_{j \in \mathcal{P}} (1 - F_j(p))/\bar{\delta}_j$ as a weighted expected rank of a player whose performance is p. Similar to the performance computations in Codeforces and TopCoder, P_i can thus be viewed as the performance level at which one's expected rank would equal i's actual rank.

3.2 Belief update

Having estimated $P_{i,t}$ in the first phase, the second phase is rather simple. Ignoring normalizing constants, Equation (3) tells us that the pdf of the skill posterior can be obtained as the pointwise product of the pdfs of the skill prior and the performance model. When both factors are differentiable and log-concave, then so is their product. Its maximum is the new rating $\mu_{i,t}$; let's see how to compute it for the same two specializations of our model.

Gaussian skill prior and performance model. When the skill prior and performance model are Gaussian with known means and variances, multiplying their pdfs yields another known Gaussian. Hence, the posterior is compactly represented by its mean $\mu_{i,t}$, which coincides with the MAP and rating; and its variance $\sigma_{i,t}^2$, which is our **uncertainty** regarding the player's skill.

[2] Elo-MMR, too, can be modified to split ties into half win plus half loss. It's easy to check that Lemma 3.1 still holds if $d_j(p)$ is replaced by $w_l l_j(p) + w_v v_j(p)$ for some $w_l, w_v \in [0, 1]$ with $|w_l - w_v| < 1$. In particular, we can set $w_l = w_v = 0.5$. The results in Section 5 won't be altered by this change.

Logistic performance model. When the performance model is non-Gaussian, the multiplication does not simplify so easily. By Equation (3), each round contributes an additional factor to the belief distribution. In general, we allow it to consist of a collection of simple log-concave factors, one for each round in which player i has participated. Denote the participation history by

$$\mathcal{H}_{i,t} := \{k \in \{1, \dots, t\} : i \in \mathcal{P}_k\}.$$

Since each player can be considered in isolation, we'll omit the subscript i. Specializing to the logistic setting, each $k \in \mathcal{H}_t$ contributes a logistic factor to the posterior, with mean p_k and variance β_k^2. We still use a Gaussian initial prior, with mean and variance denoted by p_0 and β_0^2, respectively. Postponing the discussion of skill evolution to Section 4, for the moment we assume that $S_k = S_0$ for all k. The posterior pdf, up to normalization, is then

$$\pi_0(s) \prod_{k \in \mathcal{H}_t} \Pr(P_k = p_k \mid S_k = s)$$
$$\propto \exp\left(-\frac{(s-p_0)^2}{2\beta_0^2}\right) \prod_{k \in \mathcal{H}_t} \operatorname{sech}^2\left(\frac{\pi}{\sqrt{12}} \frac{s-p_k}{\beta_k}\right). \quad (5)$$

Maximizing the posterior density amounts to minimizing its negative logarithm. Up to a constant offset, this is given by

$$L(s) := L_2\left(\frac{s-p_0}{\beta_0}\right) + \sum_{k \in \mathcal{H}_t} L_R\left(\frac{s-p_k}{\beta_k}\right),$$
$$\text{where } L_2(x) := \frac{1}{2}x^2 \text{ and } L_R(x) := 2\ln\left(\cosh\frac{\pi x}{\sqrt{12}}\right).$$
$$\text{Thus, } \frac{d}{ds}L(s) = \frac{s-p_0}{\beta_0^2} + \sum_{k \in \mathcal{H}_t} \frac{\pi}{\beta_k\sqrt{3}} \tanh\frac{(s-p_k)\pi}{\beta_k\sqrt{12}}. \quad (6)$$

dL/ds is continuous and strictly increasing in s, so its zero is unique: it is the MAP μ_t. Similar to what we did in the first phase, we can solve for μ_t with either binary search or Newton's method.

We pause to make an important observation. From Equation (6), the rating carries a rather intuitive interpretation: Gaussian factors in L become L_2 penalty terms, whereas logistic factors take on a more interesting form as L_R terms. From Figure 1, we see that the L_R term behaves quadratically near the origin, but linearly at the extremities, effectively interpolating between L_2 and L_1 over a scale of magnitude β_k

It is well-known that minimizing a sum of L_2 terms pushes the argument towards a weighted mean, while minimizing a sum of L_1 terms pushes the argument towards a weighted median. With L_R terms, the net effect is that μ_t acts like a robust average of the historical performances p_k. Specifically, one can check that

$$\mu_t = \frac{\sum_k w_k p_k}{\sum_k w_k}, \text{ where } w_0 = \frac{1}{\beta_0^2} \text{ and}$$
$$w_k = \frac{\pi}{(\mu_t - p_k)\beta_k\sqrt{3}} \tanh\frac{(\mu_t - p_k)\pi}{\beta_k\sqrt{12}} \text{ for } k \in \mathcal{H}_t. \quad (7)$$

w_k is close to $1/\beta_k^2$ for typical performances, but can be up to $\pi^2/6$ times more as $|\mu_t - p_k| \to 0$, or vanish as $|\mu_t - p_k| \to \infty$. This feature is due to the thicker tails of the logistic distribution, as compared to the Gaussian, resulting in an algorithm that resists drastic rating changes in the presence of a few unusually good or bad performances. We'll formally state this *robustness* property in Theorem 5.7.

Estimating skill uncertainty. While there is no easy way to compute the variance of a posterior in the form of Equation (5), it will be useful to have some estimate σ_t^2 of uncertainty. There is a simple formula in the case where all factors are Gaussian. Since moment-matched logistic and normal distributions are relatively close (c.f. Figure 1), we apply the same formula:

$$\frac{1}{\sigma_t^2} := \sum_{k \in \{0\} \cup \mathcal{H}_t} \frac{1}{\beta_k^2}. \quad (8)$$

4 SKILL EVOLUTION OVER TIME

Factors such as training and resting will change a player's skill over time. If we model skill as a static variable, our system will eventually grow so confident in its estimate that it will refuse to admit substantial changes. To remedy this, we introduce a skill evolution model, so that in general $S_t \neq S_{t'}$ for $t \neq t'$. Now rather than simply being equal to the previous round's posterior, the skill prior at round t is given by

$$\pi_t(s) = \int \Pr(S_t = s \mid S_{t-1} = x) \Pr(S_{t-1} = x \mid P_{<t})\,\mathrm{d}x. \quad (9)$$

The factors in the integrand are the skill evolution model and the previous round's posterior, respectively. Following other Bayesian rating systems [2, 6, 12, 16, 17, 19], we model the skill diffusions $S_t - S_{t-1}$ as independent zero-mean Gaussians. That is, $\Pr(S_t \mid S_{t-1} = x)$ is a Gaussian with mean x and some variance γ_t^2. The Glicko system sets γ_t^2 proportionally to the time elapsed since the last update, corresponding to a continuous Brownian motion. Codeforces and TopCoder simply set γ_t to a constant when a player participates, and zero otherwise, corresponding to changes that are in proportion to how often the player competes. Now we are ready to complete the two specializations of our rating system.

Gaussian skill prior and performance model. If both the prior and performance distributions at round $t-1$ are Gaussian, then the posterior is also Gaussian. Adding an independent Gaussian diffusion to our posterior on S_{t-1} yields a Gaussian prior on S_t. By induction, the skill belief distribution forever remains Gaussian. We call this Gaussian specialization of our rating system Elo-MMR$^{\mathcal{X}}$.

Logistic performance model. After a player's first contest round, the posterior in Equation (5) becomes non-Gaussian, rendering the integral in Equation (9) intractable.

A very simple approach would be to replace the full posterior in Equation (5) by a Gaussian approximation with mean μ_t (equal to the posterior MAP) and variance σ_t^2 (given by Equation (8)). As in the previous case, applying diffusions in the Gaussian setting is a simple matter of adding means and variances.

With this approximation, no memory is kept of the individual performances P_t. Priors are simply Gaussian, while posterior densities are the product of two factors: the Gaussian prior, and a logistic factor corresponding to the latest performance. To ensure robustness (see Section 5.2), μ_t is computed as the argmax of this posterior *before* replacement by its Gaussian approximation. We call the rating system that takes this approach Elo-MMR(∞).

As the name implies, it turns out to be a special case of Elo-MMR(ρ). In the general setting with $\rho \in [0, \infty)$, we keep the full posterior from Equation (5). Since we cannot tractably compute the effect of a Gaussian diffusion, we seek a heuristic derivation of the next round's prior, retaining a form similar to Equation (5) while satisfying many of the same properties as the intended diffusion.

4.1 Desirable properties of an evolution model

Inspired from the Gaussian diffusions used by other rating systems (e.g. Glicko, Glicko-2, and TrueSkill [15, 17, 19]), we formalize a list of properties that would be natural to a skill evolution model.

- *Aligned incentives.* First and foremost, the skill evolution model must not break the aligned incentive property of our rating system. That is, a player should never be motivated to lose on purpose (Theorem 5.5).
- *Rating preservation.* The diffusion algorithm should not alter the arg max of the belief distribution: that is, the rating of a player should not change after applying the skill evolution (it should remain at μ_{t-1}).
- *Correct magnitude.* A diffusion with parameter γ^2 should increase the skill uncertainty, as measured by Equation (8), by γ^2.
- *Composability.* Two diffusions applied in sequence, first with parameter γ_1^2 and then with γ_2^2, should have the same effect as a single diffusion with parameter $\gamma_1^2 + \gamma_2^2$.
- *Zero diffusion.* In the limit as $\gamma \to 0$, diffusion should not alter the posterior.
- *Zero uncertainty.* In the limit as $\sigma_{t-1} \to 0$, where the previous rating μ_{t-1} becomes a perfect estimate of S_{t-1}, our belief on S_t should become a Gaussian with moments (μ_{t-1}, γ^2) Any finer-grained information regarding the prior history $P_{\leq t}$ will be erased.

The first two properties are naturally of interest, while the rest are true of Gaussian diffusions, which we sought to emulate.

4.2 A heuristic diffusion algorithm

Each factor in the posterior (see Equation (5)) has a parameter β_k. Define a factor's **weight** to be $1/\beta_k^2$, which by Equation (8) contributes to the **total weight** $1/\sigma_t^2$. Unlike in Equation (7), these weights don't depend on $|\mu_t - p_k|$.

The approximation step of Elo-MMR(∞) replaces all the logistic factors by a single Gaussian whose variance is chosen to ensure that the total weight is preserved. In addition, its mean is chosen to preserve the player's rating, given by the unique zero of Equation (6). Finally, the diffusion step of Elo-MMR(∞) increases the Gaussian's variance, and hence the player's skill uncertainty, by γ_t^2; this corresponds to a decay in the weight.

To generalize the idea, we interleave the two steps in a continuous manner. The approximation step becomes a **transfer step**: rather than replace the logistic factors outright, we take away the same fraction from each of their weights, and place the sum of removed weights onto a new Gaussian factor. The diffusion step becomes a **decay step**, reducing each factor's weight by the same fraction, chosen such that the overall uncertainty is increased by γ_t^2.

To make the idea precise, we generalize the posterior from Equation (5) with fractional **multiplicities** ω_k, initially set to 1 for each $k \in \{0\} \cup \mathcal{H}_t$. The k'th factor is raised to the power ω_k; in Equations (6) and (8), the corresponding term is multiplied by ω_k. In other words, the weights are now given by

$$w_k := \frac{\omega_k}{\beta_k^2}.$$

For $\rho \in [0, \infty]$, the Elo-MMR(ρ) algorithm continuously and simultaneously performs transfer and decay, with transfer proceeding at ρ times the rate of decay. Holding β_k fixed, changes to ω_k can be described in terms of changes to w_k:

$$\dot{w}_0 = -r(t) w_0 + \rho r(t) \sum_{k \in \mathcal{H}_t} w_k,$$
$$\dot{w}_k = -(1+\rho) r(t) w_k \quad \text{for } k \in \mathcal{H}_t,$$

where the arbitrary decay rate $r(t)$ can be eliminated by a change of variable $\mathrm{d}\tau = r(t)\mathrm{d}t$. After some time $\Delta\tau$, the total weight will have decayed by a factor $\kappa = e^{-\Delta\tau}$, resulting in the new weights:

$$w_0^{final} = \kappa w_0 + \left(\kappa - \kappa^{1+\rho}\right) \sum_{k \in \mathcal{H}_t} w_k,$$
$$w_k^{final} = \kappa^{1+\rho} w_k \quad \text{for } k \in \mathcal{H}_t.$$

In order for the uncertainty to increase from σ_{t-1}^2 to $\sigma_{t-1}^2 + \gamma_t^2$, we must solve $\kappa/\sigma_{t-1}^2 = 1/(\sigma_{t-1}^2 + \gamma_t^2)$ for the decay factor:

$$\kappa = \left(1 + \frac{\gamma_t^2}{\sigma_{t-1}^2}\right)^{-1}.$$

In order for this operation to preserve ratings, the transferred weight must be centered at μ_{t-1}; see Algorithm 2 for details.

Algorithm 1 details the full Elo-MMR(ρ) rating system. The main loop runs whenever a round of competition takes place. First, new players are initialized with a Gaussian prior. Then, changes in player skill are modeled by Algorithm 2. Given the round rankings E_t, the first phase of Algorithm 3 solves an equation to estimate P_t. Finally, the second phase solves another equation for the rating μ_t.

The hyperparameters ρ, β, γ are domain-dependent, and can be set by standard hyperparameter search techniques. For convenience, we assume β and γ are fixed and use the shorthand $\bar{\beta}_k := \frac{\sqrt{3}}{\pi}\beta_k$.

Algorithm 1 Elo-MMR(ρ, β, γ)

```
for all rounds $t$ do
    for all players $i \in \mathcal{P}_t$ in parallel do
        if $i$ has never competed before then
            $\mu_i, \sigma_i \leftarrow \mu_{newcomer}, \sigma_{newcomer}$
            $p_i, w_i \leftarrow [\mu_i], [1/\sigma_i^2]$
        diffuse($i, \gamma, \rho$)
        $\mu_i^\pi, \delta_i \leftarrow \mu_i, \sqrt{\sigma_i^2 + \beta^2}$
    for all $i \in \mathcal{P}_t$ in parallel do
        update($i, E_t, \beta$)
```

Theorem 4.1. *Algorithm 2 with $\rho \in (0, \infty)$ meets all of the properties listed in Section 4.1.*

Algorithm 2 diffuse(i, γ, ρ)

```
$\kappa \leftarrow (1 + \gamma^2/\sigma_i^2)^{-1}$
$w_G, w_L \leftarrow \kappa^\rho w_{i,0}, (1 - \kappa^\rho) \sum_k w_{i,k}$
$p_{i,0} \leftarrow (w_G p_{i,0} + w_L \mu_i)/(w_G + w_L)$
$w_{i,0} \leftarrow \kappa(w_G + w_L)$
for all $k \neq 0$ do
    $w_{i,k} \leftarrow \kappa^{1+\rho} w_{i,k}$
$\sigma_i \leftarrow \sigma_i/\sqrt{\kappa}$
```

Algorithm 3 update(i, E, β)

```
$p \leftarrow \text{zero}\left(\sum_{j \preceq i} \frac{1}{\delta_j}\left(\tanh \frac{x-\mu_j^\pi}{2\bar{\delta}_j} - 1\right) + \sum_{j \succeq i} \frac{1}{\delta_j}\left(\tanh \frac{x-\mu_j^\pi}{2\bar{\delta}_j} + 1\right)\right)$
$p_i$.push($p$)
$w_i$.push($1/\beta^2$)
$\mu_i \leftarrow \text{zero}\left(w_{i,0}(x - p_{i,0}) + \sum_{k \neq 0} \frac{w_{i,k}\beta^2}{\bar{\beta}} \tanh \frac{x-p_{i,k}}{2\bar{\beta}}\right)$
```

The *aligned incentives* property will be proved in Theorem 5.5. The proof relies on the diffusion algorithm ignoring the performances p_k, and centering the transferred Gaussian weight at the rating μ_{t-1}, which is trivially monotonic in μ_{t-1}.

The remaining five properties are straightforward to verify, so we leave them to the appendix. Notably, one property fails in each of the cases $\rho = 0$ and $\rho = \infty$. In Section 5.2, we'll come to interpret ρ as a kind of inverse momentum.

5 THEORETICAL PROPERTIES

In this section, we show that the simplicity of the Elo-MMR formulate enables us to rigorously prove Elo-MMR's incentive alignment, robustness, and computational efficiency.

5.1 Aligned incentives

To demonstrate the need for *aligned incentives*, let's look at the consequences of violating this property in the TopCoder and Glicko-2 rating systems. These systems track a "volatility" for each player, which estimates the variance of their performances. A player whose recent performance history is more consistent would be assigned a lower volatility score, than one with wild swings in performance. The volatility acts as a multiplier on rating changes; thus, players with an extremely low or high performance will have their subsequent rating changes amplified.

While it may seem like a good idea to boost changes for players whose ratings are poor predictors of their performance, this feature has an exploit. By intentionally performing at a weaker level, a player can amplify future increases to an extent that more than compensates for the immediate hit to their rating. A player may even "farm" volatility by alternating between very strong and very weak performances. After acquiring a sufficiently high volatility score, the strategic player exerts their honest maximum performance over a series of contests. The amplification eventually results in a rating that exceeds what would have been obtained via honest play. This type of exploit was discovered in both TopCoder competitions and the Pokemon Go video game [3, 13]. For a detailed example, see Table 5.3 of [13].

Remarkably, Elo-MMR combines the best of both worlds: we'll see in Section 5.2 that, for $\rho \in (0, \infty)$, Elo-MMR(ρ) also boosts changes to inconsistent players. And yet, as we'll prove in this section, no such strategic incentive exists in *any* version of Elo-MMR.

Recall that, for the purposes of the algorithm, the performance p_i is defined to be the unique zero of the function $Q_i(p) := \sum_{j \succ i} l_j(p) + \sum_{j \sim i} d_j(p) + \sum_{j \prec i} v_j(p)$, whose terms l_i, d_i, v_i are contributed by opponents against whom i lost, drew, or won, respectively. Wins (losses) are always positive (negative) contributions to a player's performance score:

Lemma 5.1. *Adding a win term to $Q_i(\cdot)$, or replacing a tie term by a win term, always increases its zero. Conversely, adding a loss term, or replacing a tie term by a loss term, always decreases it.*

Proof. By Lemma 3.1, $Q_i(p)$ is decreasing in p. Thus, adding a positive term will increase its zero whereas adding a negative term will decrease it. The desired conclusion follows by noting that, for all j and p, $v_j(p)$ and $v_j(p) - d_j(p)$ are positive, whereas $l_j(p)$ and $l_j(p) - d_j(p)$ are negative. □

While not needed for our main result, a similar argument shows that performance scores are monotonic across the round standings:

Theorem 5.2. *If $i \succ j$ (that is, player i beats j) in a given round, then player i and j's performances satisfy $p_i > p_j$.*

Proof. If $i \succ j$ with i, j adjacent in the rankings, then

$$Q_i(p) - Q_j(p) = \sum_{k \sim i}(d_k(p) - l_k(p)) + \sum_{k \sim j}(v_k(p) - d_k(p)) > 0.$$

for all p. Since Q_i and Q_j are decreasing functions, it follows that $p_i > p_j$. By induction, this result extends to the case where i, j are not adjacent in the rankings. □

What matters for incentives is that performance scores be *counterfactually* monotonic: meaning, if we were to alter the round standings, a strategic player will always prefer to place higher:

Lemma 5.3. *In any given round, holding fixed the relative ranking of all players other than i (and holding fixed all preceding rounds), the performance p_i is a monotonic function of player i's prior rating and of player i's rank in this round.*

Proof. Monotonicity in the rating follows directly from monotonicity of the self-tie term d_i in Q_i. Since an upward shift in the rankings can only convert losses to ties to wins, monotonicity in contest rank follows from Lemma 5.1. □

Having established the relationship between round rankings and performance scores, the next step is to prove that, even with hindsight, players will always prefer their performance scores to be as high as possible:

Lemma 5.4. *Holding fixed the set of contest rounds in which a player has participated, their current rating is monotonic in each of their past performance scores.*

Proof. The player's rating is given by the zero of Equation (6), which is monotonically increasing in s and decreasing in each of the p_k. Thus, the zero will be monotonically increasing in each of the p_k, provided that all of the β_k are held fixed.

To see that the β_k are indeed fixed, note that the skill evolution algorithms in Section 4 set each of the β_k in a manner that does not depend on any of the p_k. The prior term p_0 may change, but only towards past ratings. By induction, we can assume these past ratings only change monotonically, so the proof is complete. □

Finally, we conclude that the player's incentives are aligned with optimizing round rankings, or raw scores:

Theorem 5.5 (Aligned Incentives). *Holding fixed the set of contest rounds in which each player has participated, and the historical ratings and relative rankings of all players other than i, player i's current rating is monotonic in each of their past rankings.*

Proof. Choose any contest round in player i's history, and consider improving player i's rank in that round while holding everything else fixed. It suffices to show that player i's current rating would necessarily increase as a result.

In the altered round, by Lemma 5.3, p_i is increased; and by Lemma 5.4, player i's post-round rating is increased. By Lemma 5.3 again, this increases player i's performance score in the following round. Proceeding inductively, we find that performance scores and ratings from this point on are all increased. □

In the special cases of Elo-MMR$^\chi$ or Elo-MMR(∞), the rating system is "memoryless": the only data retained for each player are the current rating $\mu_{i,t}$ and uncertainty $\sigma_{i,t}$; detailed performance history is not saved. In this setting, we present a natural monotonicity theorem. A similar theorem was stated for the Codeforces system in [2], but no proofs were given.

Theorem 5.6 (Memoryless Monotonicity Theorem). *In either the Elo-MMR$^\chi$ or Elo-MMR(∞) system, suppose i and j are two participants of round t. Suppose that the ratings and corresponding uncertainties satisfy $\mu_{i,t-1} \geq \mu_{j,t-1},\ \sigma_{i,t-1} = \sigma_{j,t-1}$. Then, $\sigma_{i,t} = \sigma_{j,t}$. Furthermore, if $i \succ j$ in round t, then $\mu_{i,t} > \mu_{j,t}$. On the other hand, if $j \succ i$ in round t, then $\mu_{j,t} - \mu_{j,t-1} > \mu_{i,t} - \mu_{i,t-1}$.*

Proof. The new contest round will add a rating perturbation with variance γ_t^2, followed by a new performance with variance β_t^2. As a result,

$$\sigma_{i,t} = \left(\frac{1}{\sigma_{i,t-1}^2+\gamma_t^2}+\frac{1}{\beta_t^2}\right)^{-\frac{1}{2}} = \left(\frac{1}{\sigma_{j,t-1}^2+\gamma_t^2}+\frac{1}{\beta_t^2}\right)^{-\frac{1}{2}} = \sigma_{j,t}.$$

The remaining conclusions are consequences of three properties: memorylessness, aligned incentives (Theorem 5.5), and translation-invariance (ratings, skills, and performances are quantified on a common interval scale relative to one another).

Since the Elo-MMR$^\chi$ or Elo-MMR(∞) systems are memoryless, we may replace the initial prior and performance histories of players with any alternate histories of our choosing, as long as our choice is compatible with their current rating and uncertainty. For example, both i and j can be considered to have participated in the same set of rounds, with i always performing at $\mu_{i,t-1}$. and j always performing at $\mu_{j,t-1}$. Round t is unchanged.

Suppose $i \succ j$. Since i's historical performances are all equal or stronger than j's, Theorem 5.5 implies $\mu_{i,t} > \mu_{j,t}$.

Suppose $j \succ i$. By translation-invariance, if we shift each of j's performances, up to round t and including the initial prior, upward by $\mu_{i,t-1} - \mu_{j,t-1}$, the rating changes between rounds will be unaffected. Players i and j now have identical histories, except that we still have $j \succ i$ at round t. Therefore, $\mu_{j,t-1} = \mu_{i,t-1}$ and, by Theorem 5.5, $\mu_{j,t} > \mu_{i,t}$. Subtracting the equation from the inequality proves the second conclusion. □

5.2 Robustness to outliers

Another desirable property in many settings in robustness: a player's rating should not change too much in response to any one contest, no matter how extreme their performance. The Codeforces and TrueSkill systems lack this property, allowing for unbounded rating changes. TopCoder has a hack to achieve robustness by clamping any changes that exceed a cap, which is initially high for new players but decreases with experience.

When $\rho > 0$, Elo-MMR(ρ) achieves robustness in a natural and continuous manner. It comes out of the interplay between logistic and Gaussian factors in the posterior; $\rho > 0$ ensures that the Gaussian component doesn't vanish. Recall the notation used to describe the general posterior in Equations (5) and (6), enhanced with the fractional multiplicities ω_k from Section 4.1.

Theorem 5.7. *Let*

$$\Delta_+ := \lim_{p_t \to +\infty} \mu_t - \mu_{t-1}, \quad \Delta_- := \lim_{p_t \to -\infty} \mu_{t-1} - \mu_t.$$

Then,

$$\frac{\pi}{\beta_t\sqrt{3}}\left(\frac{1}{\beta_0^2}+\frac{\pi^2}{6}\sum_{k\in\mathcal{H}_{t-1}}\frac{\omega_k}{\beta_k^2}\right)^{-1} \leq \Delta_\pm \leq \frac{\pi\beta_0^2}{\beta_t\sqrt{3}}.$$

Proof. Using the fact that $0 < \frac{d}{dx}\tanh(x) \leq 1$, differentiating Equation (6) yields

$$\frac{1}{\beta_0^2} \leq \frac{d^2}{ds^2}L(s) \leq \frac{1}{\beta_0^2}+\frac{\pi^2}{6}\sum_{k\in\mathcal{H}_{t-1}}\frac{\omega_k}{\beta_k^2}.$$

For every $s \in \mathbb{R}$, in the limit as $p_t \to \pm\infty$, the new term corresponding to the performance at round t will increase $\frac{d}{ds}L(s)$ by $\mp\frac{\pi}{\beta_t\sqrt{3}}$. Since μ_{t-1} was a zero of $\frac{d}{ds}L(s)$ without this new term, we now have $\frac{d}{ds}L(s)\mid_{s=\mu_{t-1}} \to \mp\frac{\pi}{\beta_t\sqrt{3}}$. Dividing by the former inequalities yields the desired result. □

The proof reveals that the magnitude of $\Delta_\pm$ depends inversely on that of $\frac{d^2}{ds^2}L(s)$ in the vicinity of the current rating, which in turn is related to the derivative of the tanh terms. If a player's performances vary wildly, then most of the tanh terms will be in their tails, which contribute small derivatives, enabling larger rating changes. Conversely, the tanh terms of a player with a very consistent rating history will contribute large derivatives, so the bound on their rating change will be small.

Thus, Elo-MMR(ρ) naturally caps the rating change of all players, and puts a smaller cap on the rating change of consistent players. The cap will increase after an extreme performance, providing a similar "momentum" to the TopCoder and Glicko-2 systems, but without sacrificing aligned incentives (Theorem 5.5).

By comparing against Equation (8), we see that the lower bound in Theorem 5.7 is on the order of σ_t^2/β_t, while the upper bound is on the order of β_0^2/β_t. As a result, the momentum effect is more pronounced when β_0 is much larger than σ_t. This occurs when ρ is set to a small value; thus, a system designer may adjust ρ according to the desired strength of momentum.

5.3 Runtime analysis and optimizations

Let's look at the computation time needed to process a round with participant set $\mathcal{P}$, where we again omit the round subscript. Each player i has a participation history $\mathcal{H}_i$.

Estimating P_i entails finding the zero of a monotonic function with $O(|\mathcal{P}|)$ terms, and then obtaining the rating μ_i entails finding the zero of another monotonic function with $O(|\mathcal{H}_i|)$ terms. Using the Illinois or Newton methods, solving these equations to precision ϵ takes $O(\log\log\frac{1}{\epsilon})$ iterations. As a result, the total runtime needed to process one round of competition is

$$O\left(\sum_{i\in\mathcal{P}}(|\mathcal{P}|+|\mathcal{H}_i|)\log\log\frac{1}{\epsilon}\right).$$

This complexity is more than adequate for Codeforces-style competitions with thousands of contestants and history lengths up to a few hundred. Indeed, we were able to process the entire history of Codeforces on a small laptop in less than half an hour. Nonetheless, it may be cost-prohibitive in truly massive settings, where $|\mathcal{P}|$ or $|\mathcal{H}_i|$ number in the millions. Fortunately, it turns out that both functions may be compressed down to a bounded number of terms, with negligible loss of precision.

Adaptive subsampling. In Section 2, we used Doob's consistency theorem to argue that our estimate for P_i is consistent. Specifically, we saw that $O(1/\epsilon^2)$ opponents are needed to get the typical error below ϵ. Thus, we can subsample the set of opponents to include in the estimation, omitting the rest. Random sampling is one approach. A more efficient approach chooses a fixed number of opponents whose ratings are closest to that of player i, as these are more likely to provide informative match-ups. On the other hand, if the setting requires aligned incentives to hold exactly, then one must avoid choosing different opponents for each player.

History compression. Similarly, it's possible to bound the number of stored factors in the posterior. Our skill-evolution algorithm decays the weights of old performances at an exponential rate. Thus, the contributions of all but the most recent $\log\frac{1}{\epsilon}$ terms are negligible. Rather than erase the older logistic terms outright, we recommend replacing them with moment-matched Gaussian terms, similar to the transfers in Section 4 with $\kappa = 0$. Since Gaussians compose easily, a single term can then summarize an arbitrarily long prefix of the history.

Substituting $1/\epsilon^2$ and $\log\frac{1}{\epsilon}$ for $|\mathcal{P}|$ and $|\mathcal{H}_i|$, respectively, the runtime of Elo-MMR with both optimizations becomes

$$O\left(\frac{|\mathcal{P}|}{\epsilon^2}\log\log\frac{1}{\epsilon}\right).$$

Finally, we note that the algorithm is embarrassingly parallel, with each player able to solve its equations independently. The threads can read the same global data structures, so each additional thread only contributes $O(1)$ memory overhead.

dataset	# contests	avg. # participants / contest
Codeforces	1087	2999
TopCoder	2023	403
Reddit	1000	20
Synthetic	50	2500

Table 1: Summary of test datasets.

6 EXPERIMENTS

In this section, we compare various rating systems on real-world datasets, mined from several sources that will be described in Section 6.1. The metrics are runtime and predictive accuracy, as described in Section 6.2.

We compare Elo-MMR$^\chi$ and Elo-MMR(ρ) against the industry-tested rating systems of Codeforces and TopCoder, as well as the improved TrueSkill algorithm of [24]. To encourage a fairer comparison, we hand-coded efficient implementations of all the algorithms except TrueSkill, entirely within the safe subset of Rust, parallelized using the Rayon crate; as such, the Rust compiler verifies that they are memory-safe and contain no data races [26].

Our implementation of Elo-MMR(ρ) makes use of the optimizations in Section 5.3, bounding both the number of sampled opponents and the history length to 500. TrueSkill and its variants are very difficult to implement and tune, so we used an open-source implementation of the algorithm in [24]. The inherent seqentiality of its message passing procedure also prevented us from parallelizing it. We were unable to get this algorithm to perform well, but we report the best results that we obtained for completeness.

The experiments were run on a 2.0 GHz 24-core Skylake machine with 24 GB of memory. The hyperparameters of all five algorithms were tuned separately for each dataset, using a simple grid search. Implementations of all rating systems, hyperparameters, datasets, and additional processing used in our experiments can be found at https://github.com/EbTech/EloR/.

6.1 Datasets

Due to the scarcity of public domain datasets for rating systems, we mined three datasets to analyze the effectiveness of our system. A synthetic dataset following the parameters of our generative model was also created for scaling tests. Summary statistics regarding the datasets are shown in Table 1.

Codeforces contest history. This dataset contains the entire history of rated contests ever run on CodeForces.com, the dominant platform for online programming competitions. The CodeForces platform has over 850K users, over 300K of whom are rated, and has hosted over 1000 contests to date. Each contest has a couple thousand competitors on average. A typical contest takes 2 to 3 hours and contains 5 to 8 problems. Players are ranked by total points, with more points typically awarded for tougher problems, and for early solves. Users may also attempt to "hack" one another's submissions for bonus points, identifying test cases that break their solutions.

TopCoder contest history. This dataset contains the entire history of algorithm contests ever run on the TopCoder.com. TopCoder is a predecessor to Codeforces, with over 1.4 million total users and a long history as a pioneering platform for programming contests.

It hosts a variety of contest types, including over 2000 algorithm contests to date. The scoring system is similar to Codeforces but its rounds are shorter, typically 75 minutes with 3 problems.

SubRedditSimulator threads. This dataset contains data scraped from the top-1000 most upvoted threads on the website reddit.com/r/SubredditSimulator/. Reddit is a social news aggregation website with over 300 million users. The site itself is broken down into sub-sites called subreddits. Users then post and comment to the subreddits, where the posts and comments receive votes from other users. In the subreddit SubredditSimulator, users are language generation bots trained on text from other subreddits. Automated posts are made by these bots to SubredditSimulator every 3 minutes, and real users of Reddit vote on the best bot. Each post (and its associated comments) can thus be interpreted as a round of competition between the bots who commented.

Synthetic data. This dataset contains 10K players, with skills and performances generated according to the generative model in Section 2. Players are drawn from a Gaussian initial skill distribution with mean 1500 and variance 300. Each contest purturbs the skills of its participants with variance 35, and the players are ranked according to independent performances with variance 200.

6.2 Evaluation metrics

To compare the different algorithms, we define two measures of predictive accuracy. Our metrics will be defined on individual contestants in each round, and then averaged:

$$\mathbf{aggregate}(\mathbf{metric}) = \frac{\sum_t \sum_{i \in \mathcal{P}_t} \mathbf{metric}(i,t)}{\sum_t |\mathcal{P}_t|}.$$

Pair inversion metric [19]. Our first metric computes the fraction of opponents against whom our ratings predict the correct pairwise result, defined as the higher-rated player either winning or tying:

$$\mathbf{pair\_inversion}(i,t) = \frac{\text{\# correctly predicted matchups}}{|\mathcal{P}_t| - 1} \times 100\%.$$

This metric was used in the evaluation of TrueSkill [19].

Average ranking deviation. Our second metric compares the rankings with the total ordering that would be obtained by sorting players according to their prior rating. The penalty is proportional to how much these ranks differ:

$$\mathbf{rank\_deviation}(i,t) = \frac{|\text{actual\_rank}_i - \text{predicted\_rank}_i|}{|\mathcal{P}_t| - 1} \times 100\%.$$

In the event of ties, among the ranks within the tied range, we use the one that comes closest to the rating-based prediction.

6.3 Experimental results

Recall that Elo-MMR$^{\mathcal{X}}$ has a Gaussian performance model, matching the modeling assumptions of TopCoder and TrueSkill. Elo-MMR(ρ), on the other hand, has a logistic performance model, matching the modelling assumptions of Codeforces and Glicko. While ρ is included in the hyperparameter search, in practice we found that its value does not significantly affect results.

In computing our metrics, we excluded players who had competed in less than 5 total contests. This is to ensure that the initial prior distributions chosen by the algorithms do not overly affect the results. In the majority of datasets, this actually hurt the relative improvement of our results with respect to the other methods, as our method seems to have better convergence properties. Despite this, we show below that both and Elo-MMR$^{\mathcal{X}}$ and Elo-MMR(ρ) outperform competitors in accuracy and efficiency.

Hyperparameter search. In order to ensure fair comparisons between the different methods, we ran a grid-search over all hyperparameters for each method, and chose the best parameter set from this grid search. The hyperparameters were optimized by running the grid search over the first 10% of the dataset, and then using the optimal hyperparameters on the last 90% of the dataset.

In Table 2 and Table 3, we show the predictive performance and computation time of each rating system. We highlight a few important observations. First, as shown in Table 2, Elo-MMR outperforms competing rating systems across all datasets in both the pairwise inversion metric and the ranking deviation metric. In particular, significant performance gains are observed on the Codeforces and TopCoder datasets, despite these platforms' rating systems having been designed specifically for their needs. We note that the gains are the smallest for the Synthetic dataset, for which all algorithms perform similarly. This can be partly explained by the fact that the dataset is drawn from a simple distribution corresponding almost exactly to the assumptions of these rating systems. Furthermore, every round contains every player in the dataset. As such, each system is able to quickly converge to the correct skill distributions for every player.

Next, we observe that Elo-MMR$^{\mathcal{X}}$ and Elo-MMR(ρ) are both extremely computationally efficient. In particular, our Elo-MMR variants outperform Codeforces by an order of magnitude on the Codeforces dataset, and is comparable in speed on the smaller Reddit and TopCoder datasets. The relative slowdown on the smaller datasets can be explained: the subsampling optimization of Elo-MMR(ρ) is only effective for contests with an extremely large number of participants. For smaller contests, the optimization is ineffective, as the results of every participant is needed to get an accurate skill estimation.

Finally, in comparing between the two Elo-MMR variants, we note that whilst Elo-MMR(ρ) is more accurate, Elo-MMR$^{\mathcal{X}}$ is always faster. This has to do with the skill drift modelling described in Section 4, as every update in Elo-MMR(ρ) must access the user's recent competition history.

7 CONCLUSION

This paper introduces the Elo-MMR rating system, which is in part a generalization of the two-player Glicko system, allowing an unbounded number of players. By developing a Bayesian model and taking the limit as the number of participants go to infinity, we obtained simple, human-interpretable rating update formulas. Furthermore, we saw that the algorithm is asymptotically fast, embarrassingly parallel, robust to extreme performances, and satisfies the important *aligned incentives* property. To our knowledge, our system is the first to rigorously prove all these properties in a setting with more than two individually ranked players.

dataset	**Codeforces**		**TopCoder**		**TrueSkill**		**Elo-MMR$^\chi$**		**Elo-MMR(ρ)**	
	pair inv.	rank dev.	pair inv.	rank dev.	pair inv.	rank dev.	pair inv.	rank dev.	pair inv.	rank dev.
Codeforces	78.3%	14.9%	78.5%	15.1%	61.7%	25.4%	78.5%	14.8%	**78.6%**	**14.7%**
TopCoder	72.6%	18.5%	72.3%	18.7%	68.7%	20.9%	73.0%	18.3%	**73.1%**	**18.2%**
Reddit	61.5%	27.3%	61.4%	27.4%	61.5%	**27.2%**	61.6%	27.3%	**61.6%**	27.3%
Synthetic	**81.7%**	12.9%	**81.7%**	**12.8%**	81.3%	13.1%	**81.7%**	**12.8%**	**81.7%**	**12.8%**

Table 2: Performance of each rating system on the pairwise inversion and average ranking deviation metrics. The bolded text in each row denote the best performances for each metric across the datasets. Higher pair inv. and lower rank dev. correspond to better performance.

dataset	**CF**	**TC**	**TS**	**Elo-MMR$^\chi$**	**Elo-MMR(ρ)**
Codeforces	212.9	72.5	67.2	**31.4**	35.4
TopCoder	9.60	**4.25**	16.8	7.00	7.52
Reddit	1.19	1.14	**0.44**	1.14	1.42
Synthetic	3.26	1.00	2.93	**0.81**	0.85

Table 3: Average compute time per dataset (time in seconds).

In terms of practical performance, we show that it outperforms existing industry systems in both prediction accuracy and computation speed. In particular, we compare against the popular CodeForces, TopCoder, and TrueSkill systems, which are deployed on platforms with hundreds of thousands to millions of users.

Over the past decade, online competitive communities such as Codeforces have grown exponentially. As such, considerable work has gone into engineering scalable and reliable rating systems. Unfortunately, many of these systems have not been rigorously analyzed in the academic community. We hope that our paper and open-source release will open new explorations in this area.

ACKNOWLEDGEMENTS

The authors are indebted to Daniel Sleator and D. J. Sutherland for initial discussions that contributed to this work. Experiments in this paper were funded by a Google Cloud Research Grant. The second author is supported by a VMWare Fellowship and the Natural Sciences and Engineering Research Council of Canada.

APPENDIX

Lemma 3.1. *If f_i is continuously differentiable and log-concave, then the functions l_i, d_i, v_i are continuous, strictly decreasing, and*

$$l_i(p) < d_i(p) < v_i(p) \text{ for all } p.$$

Proof. Continuity of l_j, d_j, v_j follows from that of F_i, f_i, f_i'. By [9], log-concavity of f_i implies log-concavity of both F_i and $1 - F_i$. As a result, each of l_i, d_i, v_i is the derivative of a strictly concave function, which is therefore strictly decreasing. In particular, since v_i is decreasing,

$$0 > \frac{d}{dp} v_i(p) = \frac{f_i'(p)}{F_i(p)} - \frac{f_i(p)^2}{F_i(p)^2}.$$

Multiplying this inequality by $F_i(p)/f_i(p)$ yields

$$d_i(p) - v_i(p) = \frac{f_i'(p)}{f_i(p)} - \frac{f_i(p)}{F_i(p)} < 0.$$

Similarly, multiplying $\frac{d}{dp} l_i(p) < 0$ by $(1 - F_i(p))/f_i(p)$ yields $l_i(p) - d_i(p) < 0$. □

Theorem 3.2. *Suppose that for all j, f_j is continuously differentiable and log-concave. Then the unique maximizer of* $\Pr(P_i = p \mid E_i^L, E_i^W)$ *is given by the unique zero of*

$$Q_i(p) = \sum_{j \succ i} l_j(p) + \sum_{j \sim i} d_j(p) + \sum_{j \prec i} v_j(p).$$

Proof. First, we rank the players by their buckets according to $\lfloor P_j/\epsilon \rfloor$, and take the limiting probabilities as $\epsilon \to 0$:

$$\begin{aligned}
\Pr(\lfloor \tfrac{P_j}{\epsilon} \rfloor > \lfloor \tfrac{p}{\epsilon} \rfloor) &= \Pr(p_j \geq \epsilon \lfloor \tfrac{p}{\epsilon} \rfloor + \epsilon) \\
&= 1 - F_j(\epsilon \lfloor \tfrac{p}{\epsilon} \rfloor + \epsilon) \to 1 - F_j(p), \\
\Pr(\lfloor \tfrac{P_j}{\epsilon} \rfloor < \lfloor \tfrac{p}{\epsilon} \rfloor) &= \Pr(p_j < \epsilon \lfloor \tfrac{p}{\epsilon} \rfloor) \\
&= F_j(\epsilon \lfloor \tfrac{p}{\epsilon} \rfloor) \to F_j(p), \\
\frac{1}{\epsilon} \Pr(\lfloor \tfrac{P_j}{\epsilon} \rfloor = \lfloor \tfrac{p}{\epsilon} \rfloor) &= \frac{1}{\epsilon} \Pr(\epsilon \lfloor \tfrac{p}{\epsilon} \rfloor \leq P_j < \epsilon \lfloor \tfrac{p}{\epsilon} \rfloor + \epsilon) \\
&= \frac{1}{\epsilon} \left(F_j(\epsilon \lfloor \tfrac{p}{\epsilon} \rfloor + \epsilon) - F_j(\epsilon \lfloor \tfrac{p}{\epsilon} \rfloor) \right) \to f_j(p).
\end{aligned}$$

Let L_{jp}^ϵ, W_{jp}^ϵ, and D_{jp}^ϵ be shorthand for the events $\lfloor \frac{P_j}{\epsilon} \rfloor > \lfloor \frac{p}{\epsilon} \rfloor$, $\lfloor \frac{P_j}{\epsilon} \rfloor < \lfloor \frac{p}{\epsilon} \rfloor$, and $\lfloor \frac{P_j}{\epsilon} \rfloor = \lfloor \frac{p}{\epsilon} \rfloor$. respectively. These are the events of a player who performs at p losing, winning, and drawing against j, when performances are discretized into ϵ-buckets.

$$\begin{aligned}
\Pr(E_i^W, E_i^L \mid P_i = p) &= \lim_{\epsilon \to 0} \prod_{j \succ i} \Pr(L_{jp}^\epsilon) \prod_{j \prec i} \Pr(W_{jp}^\epsilon) \prod_{j \sim i, j \neq i} \frac{\Pr(D_{jp}^\epsilon)}{\epsilon} \\
&= \prod_{j \succ i} (1 - F_j(p)) \prod_{j \prec i} F_j(p) \prod_{j \sim i, j \neq i} f_j(p), \\
\Pr(P_i = p \mid E_i^L, E_i^W) &\propto f_i(p) \Pr(E_i^L, E_i^W \mid P_i = p) \\
&= \prod_{j \succ i} (1 - F_j(p)) \prod_{j \prec i} F_j(p) \prod_{j \sim i} f_j(p), \\
\frac{d}{dp} \ln \Pr(P_i = p \mid E_i^L, E_i^W) &= \sum_{j \succ i} l_j(p) + \sum_{j \prec i} v_j(p) + \sum_{j \sim i} d_j(p) = Q_i(p).
\end{aligned}$$

Since Lemma 3.1 tells us that Q_i is strictly decreasing, it only remains to show that it has a zero. If so, then this zero must be unique and it will be the unique maximum of $\Pr(P_i = p \mid E_i^L, E_i^W)$.

To start, we want to prove the existence of p^* such that $Q_i(p^*) < 0$. Note that it's not possible to have $f_j'(p) \geq 0$ for all p, as in that case the density would integrate to either zero or infinity. Thus, for each j such that $j \sim i$, we can choose p_j such that $f_j'(p_j) < 0$, and so $d_j(p_j) < 0$. Let $\alpha = -\sum_{j \sim i} d_j(p_j) > 0$.

Let $n = |\{j : j \prec i\}|$, and note that $\lim_{p\to\infty} F_j(p) = 1$ and $\lim_{p\to\infty} f_j(p) = 0$. Hence, for each $j \prec i$, we can choose p_j such that $F_j(p_j) > 1/2$ and $f_j(p_j) < \alpha/(2n)$, so that $v_j(p_j) < \alpha/n$. Let $p^* = \max_{j \preceq i} p_j$. Then

$$\sum_{j \succ i} l_j(p^*) \le 0, \quad \sum_{j \sim i} d_j(p^*) \le -\alpha, \quad \sum_{j \prec i} v_j(p^*) < \alpha.$$

Therefore,

$$\begin{aligned} Q_i(p^*) &= \sum_{j \succ i} l_j(p^*) + \sum_{j \sim i} d_j(p^*) + \sum_{j \prec i} v_j(p^*) \\ &< 0 - \alpha + \alpha = 0. \end{aligned}$$

By a symmetric argument, there also exists a point q^* such that $Q_i(q^*) > 0$. Since Q_i is continuous, by the intermediate value theorem, there exists $p \in (q^*, p^*)$ such that $Q_i(p) = 0$, as desired. □

Theorem 4.1. *Algorithm 2 with $\rho \in (0, \infty)$ meets all of the properties listed in Section 4.1.*

Proof. Having already proved aligned incentives, we now verify the five remaining properties.

- *Rating preservation.* Recall that the rating is the unique zero of dL/ds as defined in Equation (6). Multiplying every weight by a common constant, whether it be κ or κ^ρ, has the effect of multiplying dL/ds uniformly by that same constant, so its zero at μ_{t-1} is preserved. Adding a new Gaussian term centered at μ_{t-1} adds zero to dL/ds evaluated at μ_{t-1}, so once again the zero is preserved.
- *Correct magnitude.* By Equation (8), multiplying every weight by κ has the effect of multiplying σ_{t-1}^2 by $1/\kappa = 1 + \gamma_t^2/\sigma_{t-1}^2$, so the decay stage raises the skill uncertainty to $\sigma_{t-1}^2 + \gamma_t^2$. The transfer stage does not change the sum of weights, so it has no bearing on the uncertainty.
- *Composability.* First, we prove an analoguous composability property in terms of the decay factor κ. Wether we apply one diffusion with factor $\kappa_1\kappa_2$, or two diffusions with factors κ_1 and κ_2, the result is that all existing terms have their weights reduced by a factor $(\kappa_1\kappa_2)^{1+\rho}$, with a fraction $1 - \kappa_1 + \kappa_1(1 - \kappa_2) = 1 - \kappa_2$ of that weight being gone for good, and the remainder going into a new Gaussian term. Thus, κ composes multiplicatively.
  It remains to show that γ^2 composes additively. Starting with uncertainty σ^2, we first apply diffusion with $\kappa_1 = \sigma^2/(\sigma^2+\gamma_1^2)$. By the *correct magnitude* property, the uncertainty becomes $\sigma^2 + \gamma_1^2$, so the second diffusion applies with $\kappa_2 = (\sigma^2+\gamma_1^2)/(\sigma^2+\gamma_1^2+\gamma_2^2)$. Their product $\kappa_1\kappa_2 = \sigma^2/(\sigma^2 + \gamma_1^2 + \gamma_2^2)$ corresponds to s single diffusion with parameter $\sigma_1^2 + \sigma_2^2$.
- *Zero-diffusion.* As $\gamma \to 0$, $\kappa \to 1$, so the decay stage has no effect. Provided that $\rho < \infty$, we also have that $\kappa^\rho \to 1$, so the transfer stage also has no effect. Note that this property fails for $\rho = \infty$.
- *Zero uncertainty.* If the skill uncertainty was very close to 0, the decay simply grows it to γ^2. Provided that $\rho > 0$, we have $\kappa^\rho \to 0$, so all of the weight is transferred away, resulting in a single Gaussian term with mean μ_{t-1}, variance γ^2, and no additional history. Note that this property fails for $\rho = 0$. □

REFERENCES

[1] CodeChef Rating System. codechef.com/ratings

[2] Codeforces Rating System. codeforces.com/blog/entry/20762

[3] Farming Volatility: How a major flaw in a well-known rating system takes over the GBL leaderboard. reddit.com/r/TheSilphRoad/comments/hwff2d/farming_volatility_how_a_major_flaw_in_a/

[4] Halo Xbox video game franchise: in numbers. telegraph.co.uk/technology/video-games/11223730/Halo-in-numbers.html

[5] LeetCode Rating System. leetcode.com/discuss/general-discussion/468851/New-Contest-Rating-Algorithm-(Coming-Soon)

[6] TopCoder Algorithm Rating System. topcoder.com/community/competitive-programming/how-to-compete/ratings

[7] Why Are Obstacle-Course Races So Popular? theatlantic.com/health/archive/2018/07/why-are-obstacle-course-races-so-popular/565130/

[8] Sharad Agarwal and Jacob R. Lorch. 2009. Matchmaking for online games and other latency-sensitive P2P systems. In *SIGCOMM 2009*. 315–326.

[9] Mark Yuying An. 1997. Log-concave probability distributions: Theory and statistical testing. *Duke University Dept of Economics Working Paper* 95-03 (1997).

[10] Shuo Chen and Thorsten Joachims. 2016. Modeling Intransitivity in Matchup and Comparison Data. In *WSDM 2016*. 227–236.

[11] Pierre Dangauthier, Ralf Herbrich, Tom Minka, and Thore Graepel. 2007. TrueSkill Through Time: Revisiting the History of Chess. In *NeurIPS 2007*. 337–344.

[12] Arpad E. Elo. 1961. New USCF rating system. *Chess Life* 16 (1961), 160–161.

[13] RNDr Michal Forišek. 2009. Theoretical and Practical Aspects of Programming Contest Ratings. (2009).

[14] David A Freedman. 1963. On the asymptotic behavior of Bayes' estimates in the discrete case. *The Annals of Mathematical Statistics* (1963), 1386–1403.

[15] Mark E Glickman. 1995. A comprehensive guide to chess ratings. *American Chess Journal* 3, 1 (1995), 59–102.

[16] Mark E Glickman. 1999. Parameter estimation in large dynamic paired comparison experiments. *Applied Statistics* (1999), 377–394.

[17] Mark E Glickman. 2012. Example of the Glicko-2 system. *Boston University* (2012), 1–6.

[18] Linxia Gong, Xiaochuan Feng, Dezhi Ye, Hao Li, Runze Wu, Jianrong Tao, Changjie Fan, and Peng Cui. 2020. OptMatch: Optimized Matchmaking via Modeling the High-Order Interactions on the Arena. In *KDD 2020*. 2300–2310.

[19] Ralf Herbrich, Tom Minka, and Thore Graepel. 2006. TrueSkill$^{\mathrm{TM}}$: A Bayesian Skill Rating System. In *NeurIPS 2006*. 569–576.

[20] Tzu-Kuo Huang, Chih-Jen Lin, and Ruby C. Weng. 2006. Ranking individuals by group comparisons. In *ICML 2006*. ACM, 425–432.

[21] Stephanie Kovalchik. 2020. Extension of the Elo rating system to margin of victory. *International Journal of Forecasting* (2020).

[22] Yao Li, Minhao Cheng, Kevin Fujii, Fushing Hsieh, and Cho-Jui Hsieh. 2018. Learning from Group Comparisons: Exploiting Higher Order Interactions. In *NeurIPS 2018*. 4986–4995.

[23] Tom Minka, Ryan Cleven, and Yordan Zaykov. 2018. *TrueSkill 2: An improved Bayesian skill rating system.* Technical Report MSR-TR-2018-8. Microsoft.

[24] Sergey I. Nikolenko, Alexander, and V. Sirotkin. 2010. Extensions of the TrueSkill TM rating system. In *In Proceedings of the 9th International Conference on Applications of Fuzzy Systems and Soft Computing*. 151–160.

[25] Jerneja Premelč, Goran Vučković, Nic James, and Bojan Leskošek. 2019. Reliability of judging in DanceSport. *Frontiers in psychology* 10 (2019), 1001.

[26] Josh Stone and Nicholas D Matsakis. 2017. The Rayon library. *Rust crate* (2017).

[27] Lin Yang, Stanko Dimitrov, and Benny Mantin. 2014. Forecasting sales of new virtual goods with the Elo rating system. *Journal of Revenue and Pricing Management* 13, 6 (Dec. 2014), 457–469.