Elo-MMR: A Rating System for Massive Multiplayer Competitions

Anonymous Author(s)

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, dance competitions, and some video games. The simplicity of our system allows us to prove theoretical bounds regarding properties such as robustness and runtime. In particular, we show that the system *aligns incentives*: that is, intentional losses by a competitor will never raise their rating. Experimentally, the rating system outperforms existing systems in prediction accuracy, and computes faster than existing systems by up to an order of magnitude.

1 INTRODUCTION

Competitions, in the form of sports, games, and examinations, have been with us since antiquity. In many competitions, skill can be quantified by a standardized objective, such as a score on a test or a completion time in a race. Where the tests or races are not standardized, it may make more sense to instead measure performance relative to other players. If a player's skill and performance varies from day to day, it will be more informative to look at the entire history of such rankings. In these cases, a good player is simply one who can frequently win against most other players. By and large, players and spectators alike are interested in estimating the relative skills of different players, thus necessitating the need for 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 rating system must be efficient: in modern applications, rating systems are predominately used in providing services and matchmaking in massive online multiplayer games (such as Halo, CounterStrike, League of Legends, etc.) [17, 21, 24]. 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 [9]. 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 [4]. Finally, the ratings provided by the system must be easily interpretable and computable: 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 [14].

Classically, rating systems were designed as a way to measure the performance of players in two-player games. One of the first developed Bayesian rating systems is the well-known Elo system [12]. Elo, as well as its successors Glicko and Glicko-2, have been widely applied within classic two-player games such as Chess and Go [5, 14–16]. The key idea behind the Elo and Glicko systems is to model each player i's performances as a distribution $\mathcal{D}_i$ (usually normal or logistic) with mean and variance μ_i and σ^2 (the variance parameter is the same for all players). In each game a player i draws their performance p_i from their distribution. Given a match between players i and j, one can use the prior distributions $\mathcal{D}_i$ and $\mathcal{D}_j$ to predict the probability that i beats j. After the game, the result is compared with the prediction, and the distributions are updated so that the observed result is more likely. In practice, the system only require storing one number per player (the μ_i), and updates to the ratings are easily computed in $O(1)$ time per game. Glicko improves upon Elo by introducing player specific deviations σ_i. Glicko-2 further refines the Glicko system by adding a rating volatility parameter to model skill decay over time. Unfortunately, Glicko-2 is is known to be flawed in practice, where users can be incentivized to lose (this was most notably exploited in the popular game of Pokemon Go [4]).

The family of Elo-like methods described above only utilizes the binary outcome of the match. In situations where score represent a measure of "closeness" between skill levels, Kovalchik [18] has shown variants of Elo that is able to take into account score information. For competitions with a few set "tasks" instead of score (such as an obstacle race), Forišek [13] develop a framework to model each task specifically. Each task then gives a different "response" to the player, using the total response as a predictor of the win probability. However, such systems are often highly application dependent, and is hard to calibrate in practice.

Though Elo-like systems are widely used, many modern competitions are now multiplayer in nature. This includes games such as CounterStrike and Halo, as well as coding platforms such as Codeforces and TopCoder [3, 8]. In these applications, the number of competitors can reach into thousands. In this regime, the most popular rating systems are variants of TrueSkill [11, 17, 21, 22], and multiplayer variants of Glicko and Elo. These variants are known to be efficient in practice, effectively rating user bases that are tens and hundreds of millions in size [2, 7]. In particular, Microsoft, TopCoder, and Codeforces have each attempted to develop their own open rating systems with varying approaches. TrueSkill, originally developed by Microsoft for the popular Halo video game, utilizes a factor graph model where nodes in the factor graph track players' skills and performance uncertainty [17]. Inference is performed by approximate message passing on the factor graph and iterated until convergence. Though more complicated than other systems, TrueSkill allows for the modelling of players in teams. In comparison, the methods of TopCoder and Codeforces are relatively simpler, as players compete individually on these websites. Topcoder uses a multiplayer variant of Glicko-2 (hence suffering from the same exploits as [4]), while Codeforces developed its own multiplayer version of Elo. Variants similar to Codeforces' and TopCoders' rating system have also been adopted by the popular websites LeetCode and CodeChef [1, 6]. There are two things that these systems have in common. First, all of the these systems propagate changes forward in time, never backward. This is a strict requirement of the system in most cases, as users prefer their historical ratings to state fixed (even if it's possible to infer more accurate historical ratings from future matches). Such a property will be a requirement of our system as well. Unfortunately, the other thing that these systems have in common is that their theoretical properties (e.g. convergence and aligned incentives) are not well understood, as many are created in an ad-hoc manner to fit the needs of the underlying platform.

Our contributions. In this paper, we describe the Elo-MMR rating system, a theoretically sound Bayesian system designed for use in large-scale multiplayer competitions. The system is best suited for competitions with tens to thousands of individual competitors, such as programming contests. The "MMR" in the name stands for "Massive", "Multiplayer", ‘and "Robust", as it operates on game outcomes given as a massive rank-ordered list of players, and its ratings are robust to outlier performances. We build Elo-MMR upon a rigorous probabilistic model, mirroring the Bayesian development of the Glicko system. This allows us to show theoretical bounds for properties such as outlier robustness and running time, thus resolving known issues with the Codeforces, LeetCode, CodeChef, and TopCoder systems. In particular, we show that the Elo-MMR system *aligns incentives*: that is, purposeful losses by a competitor will never raise their rating. In practice, we show that it achieves competitive performance and accuracy, even outperforming the linear time TrueSkill rating system in speed. In particular, we process the entire Codeforces contest database of over 850K users and 1000 contests in less than 30ms per contest on average (beating the existing Codeforces system by an order of magnitude).

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. Following Section 3, the reader will have an intuitive understanding of our algorithm. To further refine our algorithm, Section 4 models skill evolutions from players training between competitions. This modeling is quite tricky, as several conditions are required to ensure it cannot be exploited for abnormally large rating gains. Section 5 proves that the basic system with the skill evolution modeling from Section 4 satisfies several salient properties, including aligned incentives. Finally, we conclude with experiments 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 $P_{i,t}$, which approximates 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. samples from a log-concave distribution (see Definition 3.1) independent of $S_{i,t}$. Finally, an important assumption we make is that the number of players $|\mathcal{P}_t|$ is large (in practice, in the tens to thousands).

While performances are not observed directly, a ranking is observed which gives the relative ordering among all performances $\{P_{i,t}\}_{i\in\mathcal{P}_t}$. In particular, ties are modelled to occur with probability zero, when performances are exactly equal. This ranking will be the observational **evidence** E_t in our Bayesian updates. The goal of a rating system is to estimate the skill $S_{i,t}$ of all players at the present time, given the historical round rankings $\{E_1, \ldots, E_t\}$.

We overload the notation Pr for both probabilities and probability densities: the latter interpretation applies to events that obviously have probability zero, such as $\Pr(S_{i,t} = s)$.[1]

Let **S**, **P**, and **E** be the sets of all skills $S_{i,t}$, performanes $P_{i,t}$, and evidence $E_{i,t}$ respectively. In summary, the joint distribution described by are Bayesian model factorizes as follows:

$$\Pr(\mathbf{S}, \mathbf{P}, \mathbf{E}) = \prod_i \Pr(S_{i,0}) \prod_{i,t} \Pr(S_{i,t} \mid S_{i,t-1}) \prod_{i,t} \Pr(P_{i,t} \mid S_{i,t}) \prod_i \Pr(E_t \mid P_{\cdot,t}) \quad (1)$$

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
$\Pr(E_t \mid P_{\cdot,t})$ is the evidence model.

While we will specify log-concave distribution for the first three factors, the evidence model is a deterministic indicator. It equals one when E_t is consistent with the relative ordering among $\{P_{i,t}\}_{t\in\mathcal{P}_t}$, and zero otherwise. The main intuition behind our algorithm is that, in sufficiently massive competitions, the evidence E_t is sufficient to infer a very precise estimate for $\{P_{i,t}\}_{t\in\mathcal{P}_t}$, so we can treat the performances as if they were observed directly.

Denote the prior skill belief distribution before round t by

$$\pi_{i,t}(s) := \Pr(S_{i,t} = s \mid E_1 = e_1, \ldots, E_{t-1} = e_{t-1}) \quad (2)$$

We are interested in the posterior skill distribution. Applying Bayes' rule, together with the conditional dependences implied by the performance and evidence models of Equation 1, the posterior is

$$\Pr(S_{i,t} = s \mid E_1 = e_1, \ldots, E_t = e_t) \propto \pi_{i,t}(s) \Pr(E_t = e_t \mid S_{i,t} = s) \quad (3)$$

The **rating** $\mu_{i,t}$ can be any reasonable statistic summarizing this posterior distribution: we'll use the maximum a posteriori (MAP) estimate, obtained by setting s to maximize Equation (3).

Where it's clear from context, we'll omit the subscript t and the identities of the random variables. Thus, from Equation (3) we have

$$\mu_i = \arg\max_s \pi_i(s) \Pr(e \mid s)$$

Using the law of total probability to break $\Pr(E = e \mid S_i = s)$ down by P_i, and the conditional independences implied by the

[1]If e contains ties, even $\Pr(E_t = e)$ could have probability zero as ties have probability zero in our model. In the latter case, the relevant limiting procedure is that of treating performances within ϵ-width buckets as ties, and letting $\epsilon \to 0$. This is mainly a technicality, used in the proof of Theorem 3.2 in the appendix.

factorization in Equation 1, we obtain

$$\Pr(e \mid s) = \int \Pr(e \mid p) \Pr(p \mid s)\, \mathrm{d}p. \tag{4}$$

This integral is intractable in general, since $\Pr(e \mid p)$ depends not just on player i but also on our belief regarding the skills of all other players. However, in the limit of very many participants, Doob's consistency theorem [20] allows us to determine the value of $P_{i,t}$ and simplify the integral.

Indeed, we don't even need the full evidence E. Let $E_i^L = \{j \in \mathcal{P} : P_j > P_i\}$ be the set of players against whom i lost, and $E_i^W = \{j \in \mathcal{P} : P_j < P_i\}$ be the set of players against whom i won. That is, we only look at who wins, draws, or loses against i. Then, applying the classic result:

Theorem 2.1 (Doob's consistency theorem [20]). *Consider a round with a player i having fixed performance P_i, and n i.i.d. participants drawn from any fixed distribution. Suppose our prior belief on P_i has positive density at its true value. Then with probability 1, in the limit as $n \to \infty$, the posterior belief on P_i conditioned on (E_i^L, E_i^W) concentrates around its true value. Furthermore,*

$$\lim_{n\to\infty} \int \Pr(e \mid p) \Pr(p \mid s)\, \mathrm{d}p = \Pr(P_i \mid s)$$

Thus, we approximate Equation (3) by taking its infinite-player limit:

$$\mu_i = \arg\max_s \pi_i(s) \Pr(p_i \mid s) \tag{5}$$

This simplified objective allows us to develop a two-phase update algorithm for each player i. In phase one, we want to estimate p_i. By Doob's theorem, our estimate should be extremely precise when $|\mathcal{P}_t|$ is large, so we use it as the true value of P_i. In phase two, we updating our posterior for S_i and the rating μ_i according to Equation (5).

3 A TWO PHASE ALGORITHM FOR RATING ESTIMATION

3.1 Performance estimation

In this section, we describe the first phase of the Elo-MMR algorithm. To simplify the notation, we assume we are estimating the skills for round t, give all previous rounds before t. Our prior belief on each player's skill S_i implies a prior distribution on P_i. Let's denote its density by:

$$f_i(p) = \Pr(P_i = p \mid e_1, \ldots, e_{t-1}) = \int \pi_i(s) \Pr(P_i = p \mid s)\, ds \tag{6}$$

where $\pi_i(s)$ is defined in Equation (2). Let $F_i(p) = \int_{-\infty}^{p} f_i(x)\, dx$ be the corresponding cumulative distribution function. For the purpose of analysis, we'll also define the following "loss", "draw", and "victory" functions:

$$l_i(p) = \frac{d}{dp} \ln(1 - F_i(p)) = \frac{-f_i(p)}{1 - F_i(p)}$$
$$d_i(p) = \frac{d}{dp} \ln f_i(p) = \frac{f_i'(p)}{f_i(p)}$$
$$v_i(p) = \frac{d}{dp} \ln F_i(p) = \frac{f_i(p)}{F_i(p)}$$

As previously stated, we assume the performance deviations $P_i - S_i$ to come from a log-concave distribution:

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 includes the normal distribution and logistic distributions used in TrueSkill, Glicko, and Elo. The restriction to log-concave distributions allows us to prove some salient properties (shown in the 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*

$$l_j(p) < d_j(p) < v_j(p) \text{ for all } p.$$

For the remainder of this section, we fix the analysis with respect to a player i. As argued in Theorem 2.1, P_i can be very accurately estimated by its MAP, so we seek 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, together with Doob's, allows us to recover P_i from these orderings relative to i:

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 intepretation of the equation above is that the performance is the exact balance point between the (non-linearly weighted) wins, draws, and losses. Since the proof is computational, we relegate it to the appendix.

Gaussian skill prior and performance model. The performance prior $f_i(p)$ in Equation (6) is a convolution of two densities, or a sum of two independent variables $P_i = S_i + (P_i - S_i)$. If both the skill prior $\pi_i(s)$ and the performance distribution $\Pr(p \mid s)$ are assumed to be Gaussian with known mean and variance, then P_i will also be Gaussian distributed. It is analytic and log-concave, so Theorem 3.2 applies, plugging in the well-known Gaussian density and distribution functions. A simple binary search, or faster numerical techniques such as Newton's method, can be employed to solve for the maximizing p.

Logistic performance model. Now we assume the performance residual $P_i - S_i$ has a logistic distribution with mean 0 and variance β^2. Given the mean and variance of the skill prior, the independent sum $P_i = S_i + (P_i - S_i)$ would have the same mean and a variance that's increased by β^2. Unfortunately, we are presented with two problems. Firstly, 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. Secondly, adding a logistic variable to the prior does not yield a simple distribution.

As a heuristic approximation, we treat P_i as a logistic distribution with mean equal to the the maximum a priori estimate $\mu_i^- = \arg\max \pi_i$ and variance $\delta_i^2 = \sigma_i^2 + \beta^2$, where σ_i will be given by Equation (9). This distribution is analytic and log-concave, so the same methods based on Theorem 3.2 apply.

The resulting formulas turn out to have nice interpretations. In terms of $\bar{\delta}_i = \frac{\sqrt{3}}{\pi}\delta_i$, the c.d.f. and p.d.f. of our approximate performance prior are given by:

$$F_i(x) = \frac{1}{1+e^{(x-\mu_i^-)/\bar{\delta}_i}} = \frac{1}{2}\left(1+\tanh\frac{x-\mu_i^-}{2\bar{\delta}_i}\right)$$

$$f_i(x) = \frac{e^{(x-\mu_i^-)/\bar{\delta}_i}}{\bar{\delta}_i\left(1+e^{(x-\mu_i^-)/\bar{\delta}_i}\right)^2} = \frac{1}{4\bar{\delta}_i}\operatorname{sech}^2\frac{x-\mu_i^-}{2\bar{\delta}_i}$$

The logistic distribution satisfies two very convenient relations:

$$F_i'(x) = f_i(x) = F_i(x)(1-F_i(x))/\bar{\delta}_i$$
$$f_i'(x) = f_i(x)(1-2F_i(x))/\bar{\delta}_i$$

from which it follows that

$$d_i(p) = \frac{1-2F_i(p)}{\bar{\delta}} = \frac{-F_i(p)}{\bar{\delta}} + \frac{1-F_i(p)}{\bar{\delta}} = l_i(p) + v_i(p)$$

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

Finally, putting everything together:

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

The terms of the latter expression correspond to probabilities of winning and losing against a player j, weighted by $1/\bar{\delta}_j$. Accordingly, we can interpret $\sum_{j\in\mathcal{P}_t}(1-F_j(p))/\bar{\delta}_j$ as a weighted expected rank of a player with performance p in round t. Thus the zero of Q_i can be interpreted as the performance level at which the expected rank of i would equal the observed rank.

3.2 Belief update

Given an estimate that $P_i = p$, we now compute the posterior skill distribution for S_i, up to a normalizing factor. Recalling Equation (5), this is given by

$$\pi_{i,t}(s)\Pr(P_{i,t} = p \mid S_{i,t} = s)$$

When the skill prior and performance models both have differentiable log-concave densities, then so does the posterior. We can expand it out and compute the MAP using the same techniques seen in the previous section.

Gaussian skill prior and performance model. When the skill prior and performance models are both Gaussian, multiplying their probability densities yields another Gaussian. Hence, the posterior is compactly represented by its mean and variance.

[2]Our system can be modified to split ties into half win 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.

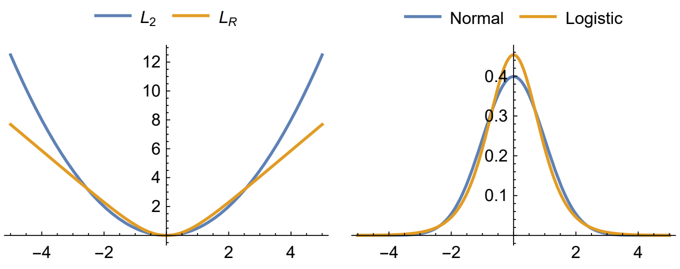

Figure 1: L_2 versus L_R for typical values (left). Logistic versus normal distribution (right).

Logistic performance model. When $\Pr(p \mid s)$ is logistic, the product must be written out explicitly. For now, let's assume the trivial skill evolution model, in which $S_{i,0} = S_{i,t}$ for all time t. Then each round's skill prior is simply the previous round's posterior. To get a new round's posterior, we multiply the prior by a new logistic factor corresponding to the performance model. Thus, the general posterior will be a product of many logistic densities, and an initial prior which we'll find convenient to treat as Gaussian.

Define

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

to be the set of rounds in which player i has participated thus far. Since this phase considers the history of player i in isolation, from here on we'll omit the subscript i but restore the round subscript t where disambiguation is needed.

Each $k \in \mathcal{H}$ contributes a logistic factor to $\pi_t(s)$, with mean p_k and variance β_k^2. Denoting the prior's mean and variance by p_0 and β_0^2, the posterior density is, up to normalization,

$$\pi_0(s)\prod_{k\in\mathcal{H}}\Pr(p_k \mid s) = \exp\left(-\frac{(s-p_0)^2}{2\beta_0^2}\right)\prod_{k\in\mathcal{H}}\operatorname{sech}^2\left(\frac{\pi}{\sqrt{12}}\frac{s-p_k}{\beta_k}\right) \tag{7}$$

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

$$L(s) := L_2\left(\frac{s-p_0}{\beta_0}\right) + \sum_{k\in\mathcal{H}} L_R\left(\frac{s-p_k}{\beta_k}\right)$$

$$\text{where } L_2(x) = \frac{1}{2}x^2 \text{ and } L_R(x) = 2\ln\left(\cosh\frac{\pi x}{\sqrt{12}}\right).$$

$$\text{Thus, } \frac{d}{ds}L(s) = \frac{s-p_0}{\beta_0^2} + \sum_{k\in\mathcal{H}}\frac{\pi}{\beta_k\sqrt{3}}\tanh\frac{(s-p_k)\pi}{\beta_k\sqrt{12}} \tag{8}$$

dL/ds is continuous and strictly increasing in s, so its zero is unique. This zero is the MAP μ_t in Equation (5), and we solve for it using the same methods associated with Theorem 3.2.

We pause to make an important observation. From Equation (8), 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's 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

$$\mu_t = \frac{\sum_k w_k p_k}{\sum_k w_k} \text{ where } w_0 = \frac{1}{\beta_0^2} \text{ and}$$

$$w_k = \frac{\pi}{(\mu_t - p_k)\beta_k\sqrt{3}} \tanh \frac{(\mu_t - p_k)\pi}{\beta_k\sqrt{12}} \text{ for } k \in \mathcal{H}.$$

w_k is close to $1/\beta_k^2$ for typical performances, but can be up to $\pi^2/6$ times more as $\mu_t \to p_k$, or vanish as $\mu_t \to \pm\infty$. This feature is due to the thicker tails of the logistic distribution, as compared to the normal, resulting in a rating algorithm that resists drastic rating changes in the presence of a few outliers. We'll prove this robustness formally in Theorem 5.7.

Estimating skill uncertainty. The variance of a player's posterior skill distribution provides a measure of uncertainty to accompany their rating. While we were able to solve for the mode μ_t of a posterior in the form of Equation (7), its variance is generally intractable to compute. Nonetheless, 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 in our setting, estimating the skill uncertainty σ_t^2 via:

$$\frac{1}{\sigma_t^2} = \sum_{k \in \{0\} \cup \mathcal{H}} \frac{1}{\beta_k^2} \tag{9}$$

4 SKILL EVOLUTION OVER TIME

In practice, players' skills often vary between rounds due to training or resting. In this section, we introduce a nontrivial skill evolution model, so that in general $S_{i,t} \neq S_{i,t'}$ for $t \neq t'$. Continuing to omit the fixed player subscript i and still taking advantage of the conditional dependences modeled by Equation 1, the skill prior at round t is given by

$$\begin{aligned}\pi_t(s) &= \Pr(S_t = s \mid e_1, \ldots, e_{t-1}) \\ &= \int \Pr(S_t = s \mid S_{t-1} = s^-) \Pr(S_{t-1} = s^- \mid e_1, \ldots, e_{t-1})\, ds^-\end{aligned} \tag{10}$$

The factors in the integrand are the skill evolution model and the previous round's posterior, respectively. As in other Bayesian rating systems [3, 8, 12, 15–17], we model skill evolution by treating the changes $S_t - S_{t-1}$ as independent zero-mean Gaussian increments; that is, $\Pr(S_t \mid S_{t-1})$ is a Gaussian density centered at S_{t-1} with some variance γ_t^2. These increments serve as a very simple means to diffuse the player's skill, encouraging rating changes without biasing in a particular direction. The system designer may set γ_t^2 according to the pace at which they expect players' skills to change: for example, the Glicko system makes them proportional to the time elapsed, whereas the TopCoder and Codeforces systems effectively treat each round as a discrete unit of time for the participants, keeping time frozen for all non-participants.

Gaussian skill prior and performance model. When both the performance and prior distributions are Gaussian, the round's posterior also becomes Gaussian. In this case, adding an independent Gaussian to S_{t-1} makes the prior over S_t another Gaussian, whose mean and variance are sums of the corresponding moments of the previous posterior and the diffusive Gaussian increment. By induction, the belief distribution forever remains Gaussian. We call this Gaussian specialization of our rating system Elo-MMRχ.

Logistic performance model. When the performance model is logistic, even the simplest Gaussian skill prior would turn the posterior into a product of the form in Equation (7), rendering the integral in Equation (10) intractable.

Hence, in the logistic setting, we no longer apply the diffusion model directly. Instead, we seek a heuristic derivation of the next round's prior, of a form similar to Equation (7), while satisfying many of the same properties as the diffusion model.

A very simple approach would be to replace the full posterior in Equation (7) by a Gaussian approximation with mean μ_t (equal to the posterior MAP) and variance σ_t^2 (given by Equation (9)). As we've seen, applying diffusion in the Gaussian setting is a simple matter of adding means and variances.

With this approximation, no memory is kept of past performances, and the general posterior will only retain one Gaussian and one logistic factor. The result is a rating system which we call Elo-R(∞). As the name implies, Elo-MMR(∞) turns out to be a special case of a more general system Elo-MMR(ρ), which we motivate next.

4.1 A heuristic diffusion algorithm

Instead of deriving the algorithm from first principles, we first identified some properties that we'd like it to have. As we'll see if the proof of the properties, the Elo-MMR(∞) method described earlier meets all but one of them. Therefore, we sought a method that remedies this flaw without introducing additional violations. The algorithm must be parametrized by a noise magnitude γ^2 and satisfy the following:

- *Aligned incentives.* The resulting rating system should be one for which we can prove Theorem 5.5.
- *Rating preservation.* The diffusion algorithm should not alter the arg max of the belief distribution: it should remain at μ_{t-1}.
- *Correct magnitude.* A diffusion with parameter γ^2 should increase the skill uncertainty, as measured by Equation (9), 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 $E_1, \ldots, E_{t-1}$ will be erased.

The first two properties are naturally of interest, while the remaining properties are true of Gaussian diffusions, which we sought to emulate. Having laid out our wishlist, we now present an algorithm that meets all six criteria.

We generalize the posterior from *Equation* (7) with *multiplicities* ω_k ($k \in \{0\} \cup \mathcal{H}$), initially set to 1. The k'th factor is raised to the

Algorithm 1 Elo-MMR(ρ, β, γ)

```
for all rounds $t$ do
    for all players $i \in \mathcal{P}_t$ in parallel do
        if $i$ is new then
            $\mu_i, \sigma_i \leftarrow 1500, 300$ // the scale is arbitrary
            $p_i, w_i \leftarrow [\mu_i], [1/\sigma_i^2]$
        diffuse($i, \gamma, \rho$)
        $\mu_i^-, \delta_i \leftarrow \mu_i, \sqrt{\sigma_i^2 + \beta^2}$
    for all $i \in \mathcal{P}_t$ in parallel do
        update($i, E_t, \beta$)
```

Algorithm 2 diffuse(i, γ, ρ)

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

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

power ω_k, which multiplies the corresponding term of Equation (8) by ω_k. We describe our changes to these multiplicities indirectly, via their effect on the *weights* $w_k = \omega_k/\beta_k^2$, generalized from Equation (9). Given a fixed $\rho \in [0, \infty]$, our algorithm consists of two stages, which may be performed in either order (the effect is the same):

In the *decay stage*, each weight w_k is multiplied by a *decay factor*

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

In the *transfer stage*, each weight is multiplied by κ^ρ, effectively subtracting out a fraction $1 - \kappa^\rho$. A new Gaussian term is created, centered at the old rating μ_{t-1}, with weight equal to the sum of the weights subtracted from the original terms. Thus, the transfer stage preserves the total weight. Note that the newly created Gaussian term can be fused with the prior term to save memory. ρ can be thought of as the relative rate of the two operations, decay and transfer. In the discussion following Theorem 5.7, we'll find another interpretation of ρ as momentum.

The algorithmic details are presented in Algorithm 1 and its helper functions. In the first main loop, new players are initialized to a Gaussian prior. The means and variances can be selected arbitrarily: the rating system is invariant to linear transformations applied uniformly to all of its location and scale parameters and outputs. The remaining hyperparameters β, γ, ρ are domain-dependent, and can be set by intuition or by standard hyperparameter search techniques.

At each round, changes in player skill are modeled by the two-stage diffusion algorithm just described, listed in Algorithm 2. Then finally, the two-phase update algorithm of Algorithm 3 solves an equation to estimate P_i in the first phase, which it then uses to solve for the new rating in the second phase. For notational convenience, we assume β is fixed for all time and use the shorthand $\bar{\beta}_k = \frac{\sqrt{3}}{\pi}\beta_k$. This completes the Elo-MMR(ρ) algorithm.

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

The *aligned incentives* property will be proved in Theorem 5.5. The relevant properties of the diffusion algorithm are that the weights of the terms have no dependence on the performances, and that the Gaussian term created by the transfer stage is centered at μ_{t-1}, making it trivially monotonic in the current rating. The remaining five properties are straightforward to verify, so we leave them to the appendix.

5 ALGORITHMIC PROPERTIES

Due to the simplicity of our system, a distinct advantage of Elo-MMR is that it is possible to prove several salient theoretical properties regarding its performance. In this section, we state and prove properties such as robustness, aligned incentives, and computational efficiency.

5.1 Aligned incentives

Aligned incentives is one of our system's most important properties, so we devote this section to motivating, stating, and proving it. The main result, Theorem 5.5, essentially guarantees that a player who seeks to improve their rating will never strategically lose rounds.

To demonstrate the need for aligned incentives, consider the following example taken from the TopCoder and Glicko-2 rating systems. In these two rating systems, parameters in the system exist to track "volatility", i.e. the amount by which a player will deviate their typical performance. This quantity is usually modelled to better differentiate between steady consistent players and inconsistent players with extraordinarily good or bad performances. In these systems, the "volatility" factor serves as a multiplier to the rating change, enhancing the rating change from particularly good or bad performances. Although this may seem like a good idea at the outset, the system has a simple exploit. By intentionally performing at a weaker level, a player can exert a fine degree of control over their performance over what the system predicts. Thus a player may alternate between extraordinarily good and bad performances, essentially "farming" volatility. Once the volatility is high enough, the player exerts their actual performance level for all future contests. Due to the farmed volatility, the final rating of the player will far exceed their original rating, despite being at the same performance level as before. This type of exploit was discovered in both the TopCoder rating system and the Pokemon Go rating system [4, 13].

In the analysis below, we show that with our particular modeling assumptions and derivations, no such strategic incentive exists. Recall from Section 3 that the performance is the unique zero of the function $Q_i(p) := \sum_{j \succ i} l_j(p) + \sum_{j \sim i} d_j(p) + \sum_{j \prec i} v_j(p)$, where l_i, d_i, v_i represent measures of the loss, draw, and victory contributions to the expected ranking.

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. □

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

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

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

By Lemma 5.1, it follows that $p_i > p_j$. By induction, this result extends to the case where i, j are not adjacent in the rankings. □

Lemma 5.3. *In any given round, holding fixed all past rounds and the relative round ranking of all players other than i, the performance p_i is a monotonic function of the pre-round rating and of player i's placement in this contest.*

Proof. Monotonicity in the rating follows directly from monotonicity of the prior term in the Theorem 3.1 expression. Since each upward shift in the rankings simply converts losses to ties to wins, monotonicity in contest placement follows from Lemma 5.1. □

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

Proof. The belief update expression is increasing in s_i and decreasing in each of the performances. Hence, changing a performance requires changing s_i in the same direction in order to restore the sum to zero. □

We're now ready for the main result of this section.

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

Proof. Let's consider changing one past round result. By Lemma 5.3, p_i in this round is increased, which by Lemma 5.4 increases player i's post-round rating. We then proceed inductively over each successive round, noting that the increased rating can only increase each p_i (Lemma 5.3) which again ensures a rating increase by Lemma 5.4. □

By Theorem 5.5, an exploit of the type exhibited by TopCoder and Pokemon Go cannot happen in our system. As previously discussed, these exploits require the player to wildly vary their performance in order to build up volatility. However, by Theorem 5.5, a local fix where we improve one of their performances can only improve their rating. Tepeat this fix many times to inductively shows that better results would have been achieved by performing consistently at the highest skill level in the first place.

When Elo-MMR(∞) or Elo-MMR$^\chi$ is used, the Algorithm 1 simplifies to be "memoryless", meaning that the history of contests do not need to be saved; only the rating r and the skill variance σ. In these cases, we present a natural "monotonicity" theorem. The monotonicity theorem shows that intentionally losing to an opponent in the current round can only decrease your rating gains, and that lower rated participants gain more rating upon beating higher rated participants. A similar theorem was stated for the Codeforces system in [3], 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 $\mu_{i,t-1} \geq \mu_{j,t-1}$ and skill variances $\sigma_{i,t-1} = \sigma_{j,t-1}$. Then $\sigma_{i,t} = \sigma_{j,t}$ and:*
If $i \succ j$ in round t, then $\mu_{i,t} > \mu_{j,t}$.
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 round update consists of a diffusion operation with parameter γ_t^2, followed by a new performance with deviation 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: monotonicity (Theorem 5.5); translation-invariance (skills, ratings and performances are relative); and memoryless-ness.

Since the Elo-MMR$^\chi$ or Elo-MMR(∞) systems are memoryless, we may replace the initial prior and performance histories of i and j with any alternate histories of our choosing, compatible with their ratings and uncertainties. For example, both 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 the first conclusion.

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 relative rating changes will be unaffected. Players i and j now have identical histories except at round t, in which the improvement to j preserves $r \succ i$. Thus, $\mu_{j,t-1} = \mu_{i,t-1}$ 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 after any one contest, no matter how extreme their performance. The Codeforces and TrueSkill systems lack this property: winning against players with extremely high ratings will induce a change proportional to our distance to their ratings. TopCoder achieves robustness by undoing any changes that exceed a cap, which is high for new players but much lower for experienced players. However, this method introducing an artificial discontinuity, which makes the method harder to analyze in theory.

When $\rho > 0$, Elo-MMR(ρ) achieves robustness in a natural and continuous manner. It comes out of the interaction between logistic and Gaussian factors ($\rho > 0$ is necessary to ensure the Gaussian component doesn't vanish) in the posterior. Recall the notation used to describe the general posterior in Equation (7), enhanced with the decaying multiplicities ω_k from Section 4.1.

Theorem 5.7. *Let*

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

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

Then,

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

Proof. Using the fact that $0 < \frac{d}{dx} \tanh(x) \le 1$, differentiating Equation (8) yields

$$\frac{1}{\beta_0^2} \le \frac{d^2}{ds^2} L(s) \le \frac{1}{\beta_0^2} + \frac{\pi^2}{6} \sum_{k \in \mathcal{R} \setminus \{t\}} \frac{\omega_k}{\beta_k^2}.$$

Now, 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 monotonicity.

By comparing against Equation (9), 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

Consider a round with participant set $\mathcal{P}$, where player i has round history $\mathcal{H}_i$. For each player i, estimating P_i entails finding the zero of a monotonic expression with $O(|\mathcal{P}|)$ terms, and then obtaining the MAP rating μ_i entails finding the zero of another monotonic expression with $O(|\mathcal{H}_i|)$ terms. Since it's difficult to bound the complexity of Newton's method, our implementation falls back to binary search in the worst-case. Hence, solving these equations to precision ϵ conservatively takes $O(\log \frac{1}{\epsilon})$ iterations. As a result, the total runtime needed to process one round of competition is

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

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, the quadratic terms may be cost-prohibitive in truly massive settings, where $|\mathcal{P}|$ or $|\mathcal{H}_i|$ number in the millions. In practice, we find that the expressions may be compressed down to finitely many terms, with negligible loss of precision.

Adaptive subsampling. In Section 2, we used Doob's theorem to argue that our estimate p_i is consistent in the limit as the number of terms becomes large. However, there is no reason for the number of terms to grow with $|\mathcal{P}|$. Thus, we can sample a smaller set of opponents to include in the expression, omitting the rest. Random sampling is one approach. We instead recommend choosing a fixed number of participants whose ratings are closest to that of player i, as players close in performance provide the most amount of information to determine player i's skill.

History compression. Similarly, it's possible to bound the history length $|\mathcal{H}_i|$. Our time-evolution operation causes the weights of old performance terms to decay exponentially. Thus, the contribution of all but the most recent terms is negligible. Rather than erase these terms completely, we recommend replacing all but a fixed number of the most recent logistic terms with moment-matched Gaussian terms. This allows us to summarize a large portion of the history with a single Gaussian, as Gaussians compose easily.

These two optimizations effectively replace the inner $|\mathcal{P}| + |\mathcal{H}_i|$ factor by a constant bound. 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. Altogether, the parallel span of our approximate Elo-MMR, treating the precision level as a fixed constant, is $O\left(\frac{|\mathcal{P}|}{\#\text{CPU}}\right)$.

6 COMPUTATIONAL EXPERIMENTS

In this section, we test our rating framework on several real-world datasets. The datasets were mined from a variety of sources, which we describe in Section 6.1.

We compare our rating system against the industry-tested rating systems of Codeforces and TopCoder, as well as the improved TrueSkill algorithm of [22]. We find that our rating performs slightly better than all competitors in terms of predictive power. In terms of computational time however, we show that Elo-MMR is up to an order of magnitude faster than CodeForces.

Practical optimizations. As discussed in Section 5.3, the Elo-MMR algorithm is trivially parallelizable. Furthermore, via the subsampling procedure in Section 5.3 we can attain large speed-ups with only a small loss in accuracy. In our tests below, we do our experiments on a 2.0 GHZ 24-core machine with 24 GB of memory (Skylake architecture). For the sub-sampling procedure, we set the number of subsamples to 500 across all datasets. In order to ensure a fairer comparison, we also parallelized the competing rating systems where we could. Unfortunately, we were not able to parallelize TrueSkill, due to the inherent sequentiality of the its internal message passing procedure.

We provide an open-source parallel implementation of our algorithm (as well as those of Codeforces, TopCoder, and TrueSkill), implemented entirely within the safe subset of Rust using the Rayon crate; as such, the Rust compiler verifies that it is memory-safe and contains no data races [23]. The open-source code will be available following the review process.

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. These datasets will publicly available following the review process. 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.

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

Table 1: Summary of test datasets.

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 and has hosted over 1000 contests to date. Each contest has a couple thousand competitors on average. The contest format and scoring system has varied, but a typical contest is 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. The sheer number of highly motivated participants in these competitions, as well as their very ergonomic data API, made it the top choice for our explorations.

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 https://www.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 (with the aim to get the highest number of votes). 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 then be interpreted as a mini-competition between the bots.

Synthetic data. This dataset contains 10000 players, with skills and performances generated according to the generative model in Section 2. Players are drawn from an initial skill distribution normally distributed around 1500 with variance 300. For each contest, players draw a performance value normally distributed around their skill with variance 50. Following each contest, a random gaussian distributed drift is applied to each player's skill according to the generative model in Section 4 (i.e. zero-centred gaussian noise with variance 10 is added to the skill value).

6.2 Evaluation metrics

To analyze the performance of the different algorithms, we define two simple metrics. Our metrics will be defined on individual contestants in each round, and then averaged:

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

Pair inversion metric [17]. For this metric, we predict the final rankings all competitors of each round (given information from all prior rounds), and then computed the fraction of pairs of competitors for which the relative ranking was correct:

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

A matchup between i and $j \neq i$ is considered to be correctly predicted if the higher-rated member wins or they tie. This metric was used in the evaluation of TrueSkill [17].

Average ranking deviation. For this metric, we compare the ranking implied by the system's ratings against the actual rankings, penalizing the system according to how much these ranks differ:

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

In the even of ties, we choose the most rank within the tied range that comes closest to the prediction.

6.3 Experimental results

We now evaluate the performance of several different rating systems. We compare our two variants of our algorithm (Elo-MMR and Elo-MMR$^{\mathcal{X}}$) against the well-known Codeforces, TopCoder, and TrueSkill rating systems, all of which have found massive success in

dataset	Codeforces		TopCoder		TrueSkill		Elo-MMR		Elo-MMR$^{\mathcal{X}}$	
	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%	**78.6%**	**14.7%**	78.5%	14.8%
TopCoder	72.6%	18.5%	72.3%	18.7%	68.7%	20.9%	**73.1%**	**18.2%**	73.0%	18.3%
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	Elo-MMR$^{\mathcal{X}}$
Codeforces	212.9	72.5	67.2	35.4	**31.4**
TopCoder	9.60	**4.25**	16.8	7.52	7.00
Reddit	1.19	1.14	**0.44**	1.42	1.14
Synthetic	3.26	1.00	2.93	0.85	**0.81**

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

industry applications. The Elo-MMR and Elo-MMR$^{\mathcal{X}}$ variants differ in the underlying distribution the data is assumed to have come from. For Elo-MMR, a logistic distribution is assumed, matching the assumptions used in the development of Codeforces, Elo, and Glicko. For Elo-MMR$^{\mathcal{X}}$, a gaussian distribution is assumed, matching the assumptions used in developing TopCoder and TrueSkill. In practice, we find that the ρ parameter described in Section 4 does not significantly affect results. Thus we omit ρ from the algorithm names in the tables.

In measuring our metrics, we additionally excluded players who have 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 however, we show below that both Elo-MMR and Elo-MMR$^{\mathcal{X}}$ outperform competitors significantly 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. The hyper-parameters for each dataset will be released in the publicly available repository following the review process.

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 (and its Gaussian variant, Elo-MMR$^{\mathcal{X}}$), 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, even though both are designed specifically for the needs of each platform. 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 and Elo-MMR$^{\mathcal{X}}$ 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 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 is more accurate, Elo-MMR$^{\mathcal{X}}$ is always faster. This has to do with the skill drift modelling described in Section 4, as the logistic version of Elo-MMR requires storing the entire competition history of a user. We note that in practical applications, this history is usually available, as it is used to summarize to the user their skill progress over time.

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. The core insight of the algorithm is that in the limit of a large number of players, player performances can be estimated almost exactly. Due to the simplicity of the algorithm, we are able to theoretically analyze desirable properties such as *aligned incentives*, robustness to extreme performances, and asymptotic running time. To our knowledge, our system is the first to rigorously all these properties in a setting with arbitrarily many players.

In terms of performance, we show that it outperforms existing industry systems in terms of both accuracy and speed. In particular, we compare against the popular CodeForces, TopCoder, and TrueSkill systems, which are deployed on platforms with millions of users. The algorithm itself is trivially parallelizable, and further speedup can be attained through a simple sub-sampling strategy. We believe there is potential to improve the performance even more, either through a more sophisticated sub-sampling strategy, interpolation, or by combining our two-phase approach with a factor graph framework similar to that of TrueSkill [17, 19].

Over the past decade, online competitive communities such as Codeforces have grown exponentially. As such, an incredible amount of work has gone into engineering scalable and robust rating systems. Unfortunately, many of these systems have not have not been rigorously analyzed in the academic community. We hope that our work will open up new explorations in this area. To encourage this, we plan to release a suite of datasets and an open-source version of our code following the review.

A 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*

$$l_j(p) < d_j(p) < v_j(p) \text{ for all } p.$$

Proof. Continuity of l_j, d_j, v_j follows from that of F_i, f_i, f_i'. By [10], 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 $\epsilon \to 0$:

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

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.

$$\begin{aligned}
\Pr(E_i^W, E_i^L \mid P_i = p) &= \lim_{\epsilon \to 0} \prod_{j \succ i} \Pr(L_{jp}^\epsilon) \prod_{j \prec i} \Pr(W_{jp}^\epsilon) \prod_{j \sim i, j \ne i} \frac{\Pr(D_{jp}^\epsilon)}{\epsilon} \\
&= \prod_{j \succ i} (1 - F_j(p)) \prod_{j \prec i} F_j(p) \prod_{j \sim i, j \ne i} f_j(p) \\
\Pr(P_i = p \mid E_i^L, E_i^W) &\propto f_i(p) \Pr(E_i^L, E_i^W \mid P_i = p) \\
&= \prod_{j \succ i} (1 - F_j(p)) \prod_{j \prec i} F_j(p) \prod_{j \sim i} f_j(p)
\end{aligned}$$

$$\frac{d}{dp} \ln \Pr(P_i = p \mid E_i^L, E_i^W) = \sum_{j \succ i} l_j(p) + \sum_{j \prec i} v_j(p) + \sum_{j \sim i} d_j(p) = Q_i(p)$$

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 $f_j'(p) \ge 0$ 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

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

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 (8). 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 (9), 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 $\gamma \longrightarrow 0$, $\kappa \longrightarrow 1$, so the decay stage has no effect. Provided that $\rho < \infty$, we also have that $\kappa^\rho \longrightarrow 1$, 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 $\kappa^\rho \longrightarrow 0$, 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. https://www.codechef.com/ratings.
[2] Codeforces. https://en.wikipedia.org/wiki/Codeforces
[3] Codeforces Rating System. http://codeforces.com/blog/entry/20762.
[4] Farming Volatility: How a major flaw in a well-known rating system takes over the GBL leaderboard. https://www.reddit.com/r/TheSilphRoad/comments/hwff2d/farming_volatility_how_a_major_flaw_in_a/
[5] Glicko Rating System. https://en.wikipedia.org/wiki/Glicko_rating_system
[6] LeetCode Rating System. https://leetcode.com/discuss/general-discussion/468851/New-Contest-Rating-Algorithm-(Coming-Soon).
[7] Topcoder. https://en.wikipedia.org/wiki/Topcoder
[8] TopCoder Algorithm Rating System. http://cshelp.wpengine.com/data-science/srm-and-mm-rating-systems/algorithm-rating-system.
[9] Sharad Agarwal and Jacob R. Lorch. 2009. Matchmaking for online games and other latency-sensitive P2P systems. In *Proceedings of the ACM SIGCOMM 2009 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Barcelona, Spain, August 16-21, 2009*, Pablo Rodriguez, Ernst W. Biersack, Konstantina Papagiannaki, and Luigi Rizzo (Eds.). ACM, 315–326. https://doi.org/10.1145/1592568.1592605
[10] Mark Bagnoli and Ted Bergstrom. 2005. Log-concave probability and its applications. *Economic theory* 26, 2 (2005), 445–469.
[11] Pierre Dangauthier, Ralf Herbrich, Tom Minka, and Thore Graepel. 2007. TrueSkill Through Time: Revisiting the History of Chess. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis (Eds.). Curran Associates, Inc., 337–344. http://papers.nips.cc/paper/3331-trueskill-through-time-revisiting-the-history-of-chess
[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] Mark E Glickman. 1995. A comprehensive guide to chess ratings. *American Chess Journal* 3, 1 (1995), 59–102.
[15] Mark E Glickman. 1999. Parameter estimation in large dynamic paired comparison experiments. *Applied Statistics* (1999), 377–394.
[16] Mark E Glickman. 2012. Example of the Glicko-2 system. *Boston University* (2012), 1–6.
[17] Ralf Herbrich, Tom Minka, and Thore Graepel. 2006. TrueSkill$^{\text{TM}}$: A Bayesian Skill Rating System. In *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*, Bernhard Schölkopf, John C. Platt, and Thomas Hofmann (Eds.). MIT Press, 569–576. http://papers.nips.cc/paper/3079-trueskilltm-a-bayesian-skill-rating-system
[18] Stephanie Kovalchik. 2020. Extension of the Elo rating system to margin of victory. *International Journal of Forecasting* (2020). https://doi.org/10.1016/j.ijforecast.2020.01.006
[19] Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger. 2001. Factor graphs and the sum-product algorithm. *IEEE Trans. Inf. Theory* 47, 2 (2001), 498–519. https://doi.org/10.1109/18.910572
[20] Jeffrey W Miller. 2018. A detailed treatment of Doob's theorem. *arXiv preprint arXiv:1801.03122* (2018).
[21] Tom Minka, Ryan Cleven, and Yordan Zaykov. [n.d.]. TrueSkill 2: An improved Bayesian skill rating system. ([n. d.]), 24.
[22] Sergey I. Nikolenko, Alexander, and V. Sirotkin. 2010. Extensions of the TrueSkill TM rating system. In *In Proceedings of the 9 th International Conference on Applications of Fuzzy Systems and Soft Computing*. 151–160.
[23] Josh Stone and Nicholas D Matsakis. 2017. The Rayon library. *Rust crate* (2017).
[24] 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. https://doi.org/10.1057/rpm.2014.26