# Skill-Based Matchmaking for Competitive Two-Player Games

CEM YUKSEL, University of Utah and Roblox, USA

Skill-based matchmaking is a crucial component of competitive multiplayer games and it is directly tied to how the players would enjoy the game. We present a simple matchmaking algorithm that aims to achieve a target win rate for all players, making long win/loss streaks less probable. It is based on the estimated skill levels of players. Therefore, we also present a rating estimation for players that does not require any game-specific information and purely relies on game outcomes. Our evaluation shows that our methods are effective in estimating a player's rating, responding to changes in rating, and achieving a desirable win rate that avoids long win/loss streaks in competitive two-player games.

CCS Concepts: • **Computing methodologies**;

Additional Key Words and Phrases: Matchmaking, competitive games, rating estimation

**ACM Reference Format:**

Cem Yuksel. 2024. Skill-Based Matchmaking for Competitive Two-Player Games. *Proc. ACM Comput. Graph. Interact. Tech.* 7, 1 (May 2024), 20 pages. https://doi.org/10.1145/3651303

## 1 INTRODUCTION

One of the primary metrics for evaluating a video game is its difficulty level. Games that are too easy for a player do not provide a sufficient level of challenge to make them engaging. On the other hand, games that are too difficult for a player can quickly discourage them. For a game to be fun, it must challenge the player just the right amount.

In competitive multiplayer games, the difficulty of a game depends on the relative skill levels of the players. Playing against an opponent with a much higher skill level, resulting in an extremely low probability of winning, often makes the effort of playing pointless. For such an opponent, easily winning the game may bring some joy, but it certainly would not provide a challenging and engaging experience. Therefore, *skill-based matchmaking* that matches players with similar skill levels is a critical component of making competitive multiplayer games fun.

Another important factor is the *win rate* of a player. While winning many games in a row can be fun, it comes at the cost of many opponents losing games. Though it is possible to enjoy a game that one loses, losing many games in a row is seldom associated with fun. Thus, a skill-based matchmaking system should also try to avoid long losing streaks at the cost of tempering long winning streaks.

Author's address: Cem Yuksel, cem@cemyuksel.com, University of Utah, Roblox, USA.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
© 2024 Copyright held by the owner/author(s).
2577-6193/2024/5-ART
https://doi.org/10.1145/3651303

The typical process of skill-based matchmaking involves the assumption that each player has a numerical *rating* value that can be used for computing the win probability against an opponent with a known rating. Therefore, an essential component is accurately estimating these ratings.

In this paper, we present methods for skill-based matchmaking for competitive two-player games that involve two distinct components:

(1) A novel rating estimation method that can accurately represent a player's skill level and quickly react to (sudden or gradual) changes in a player's skill level (Section 3).

(2) A model for skill-based matchmaking using the estimated ratings that promotes a desirable win rate, making long win/loss streaks less probable (Section 4).

Our rating estimation method (Section 3) stems from prior work (Section 2) but automatically avoids their shortcomings in noisy/divergent estimations, slow convergence, and inadequate reactions to changing skill levels. It adjusts the rating estimation after each game without the need to store players' match histories, keeping track of only a few variables per player. Also, it requires no parameter tuning, which is critical for practical applications where the accuracy of the rating estimation often cannot be measured for adjusting parameters before it is too late.

Our skill-based matchmaking (Section 4) evaluates an ideal opponent rating to promote a target win rate (i.e. 50% for two-player games) within a small window of games by only storing the outcomes of the past few games per player. To handle matchmaking for a group of players, we use a fast approximation that treats each player as if they have the same rating as their ideal target. Our matchmaking method is not tied to our rating estimation and can be used with any other rating estimation method, though its results are improved with more accurate ratings provided by our rating estimation method.

We evaluate our rating estimation and matchmaking methods (Section 5) jointly and independently using synthetic simulated data, showing that our rating estimation outperforms prominent prior alternatives and our skill-based matchmaking successfully promotes a desirable win rate, provided that the rating estimation is sufficiently accurate and there exist opponents in the player pool with the desired opponent ratings. We also show that our rating estimation produces reliable ratings that quickly converge to the correct values, does not suffer from divergence issues of prior methods, and does not require any parameter tuning.

## 2 BACKGROUND

The problem of skill-based matchmaking can be considered across multiple dimensions, involving a player's skill level on various tasks that comprise the gameplay experience, different playstyles and strategies, and how all of these elements interplay with each other. Such an approach may provide a comprehensive analysis of a player's skills and has the potential of accurately predicting game outcomes when considering players that match up against each other. However, this also requires a relatively deep exploration of a player's in-game activities, which would involve collecting a large amount of game-specific data and an extensive analysis for extracting the expected game outcomes.

Therefore, a common approach is to simplify this problem down to a single dimension by assuming that each player has a single numeric *rating* value and that comparing the ratings of players would be sufficient for predicting the outcome of a game. Obviously, this simpler approach ignores many aspects of a player's skills and it does not provide sufficient dimensions for representing circular relations (i.e. *intransitivity*), such as when a player $A$ is favored against player $B$, who is favored

against player $C$, who is in turn favored against player $A$. Nonetheless, it substantially simplifies the problem, makes it easier to evaluate with limited data, and allows completely ignoring game-specific details by simply considering observed game outcomes.

Methods using this simpler approach assume that each player has an *actual* rating that is unknown and aim to predict it. Regardless of the method used, these ratings are always population-dependent, as they are calculated by comparing the outcomes of games within a population. They may not be used for comparing different populations that have not played against each other, unless one can justifiably assert a similar skill-level distribution for different populations.

### 2.1 The Elo Rating System

Most methods for rating estimation are based on the Elo system [Elo 1978], developed by Arpad Elo in the early 1960's, and subsequently adopted by many chess federations and organizations for other games. Elo calculates the probability of winning for player $a$ with rating $r_a$ against an opponent $b$ with rating $r_b$ using

$$f_E(a,b) = \frac{1}{1+10^{-(r_a-r_b)/400}} \,. \tag{1}$$

This win probability function is consistent, such that $f_E(a,b) + f_E(b,a) = 1$. The constant *scale factor* of 400 in the denominator determines the rating difference between players beyond which the player with a higher rating has more than about 91% probability of winning.

Note that win probability depends on the difference between two players' ratings, rather than their individual values. Thus, we can freely shift all ratings by adding the same constant without impacting any win probability. Also, it does not make any sense to limit the ratings to a particular range: they can be as large or small as they need to be, including negative values.

However, common practices do not always follow this logic. For example, the US Chess Federation uses 100 as the lowest possible rating, which is problematic. In a game with such a limit, if a new player that should have a lower rating than the limit (based on observed game outcomes) enters the pool of players, either this player's rating will be represented incorrectly (because of the limit) or all other players' ratings should be increased to compensate for it.

It is also important to note that the choice of 400 for the scale factor is entirely inconsequential. Using a different value would simply scale the relative ratings accordingly. On the other hand, there is some evidence that an alternative scale factor of 561 might provide a better predictor of observed game outcomes for chess [Glickman and Jones 1999]. Yet, this is more likely an indication of chess ratings being inaccurate, rather than the scale factor being incorrect. Indeed, rescaling all chess ratings by 1.4025 would accomplish the same accuracy without modifying Elo's scale factor.

The main problem with Elo's system, however, is how ratings are updated. After a series of $m$ games (in a tournament), the previous rating of a player $r_a$ is replaced by $r'_a$, using

$$r'_a = r_a + K \sum_i^m \big(s_i - f_E(a,i)\big) \,, \tag{2}$$

where $K$ is the update scale, often called the *K-factor* and $s_i$ is the score of game $i$, which is 1 for a win, 0 for a loss, and 0.5 for a tie/draw.

The problem is that the choice of $K$ here is somewhat arbitrary. Larger values would make the system adapt quicker, but they negatively impact the convergence. Smaller values are more likely to

have the ratings converge after a large number of games, but they limit the possible rating change after a small number of games, making the ratings less responsive. US Chess Federation uses a range of values for $K$, depending on the number of games a player has played. Yet, if the goal is accurately estimating the player's actual rating, it is not possible to argue that any formulation that purely considers the number of games played would lead to an optimal $K$ value that would minimize the expected number of steps for convergence. The problem of picking a desirable $K$ value becomes even more challenging when we consider the possibility of a player's actual rating changing over time or suddenly (e.g. when the player learns a new strategy).

### 2.2 The Glicko Rating Systems

Another popular rating system is Glicko [Glickman 1999], which adds a *rating deviation* property $\phi$ for the player to Elo's formulation. Glicko also has a modified rating update rule, using a function $g(\phi)$ that scales the contribution of each match by the opponent's rating deviation, such that

$$r'_a = r_a + K_G \sum_i^m g(\phi_i)\big(s_i - f_G(a,i)\big)\,, \tag{3}$$

where the win probability function is also altered using the same scale factor, such that

$$f_G(a,b) = \frac{1}{1+10^{-g(\phi_b)(r_a-r_b)/400}}\,, \quad \text{using} \quad g(\phi) = \frac{1}{\sqrt{1+\frac{3q^2\phi^2}{\pi^2}}} \quad \text{and} \quad q = \frac{\ln(10)}{400}\,. \tag{4}$$

Note that this win probability function is not consistent, since $f_G(a,b) + f_G(b,a) < 1$ unless $\phi_a = \phi_b = 0$, though it is possible to normalize it using this sum. The update rule for $r'_a$ uses a $K$-factor that varies depending on the player's updated rating deviation $\phi'_a$, such that

$$K_G = {\phi'_a}^2\, q \qquad \text{and} \qquad \phi'_a = \frac{\phi_a}{\sqrt{1+\phi_a^2\, q^2 \sum_i^m \big(g(\phi_i)\big)^2 f_G(a,i)\big(1-f_G(a,i)\big)}}\,. \tag{5}$$

Note that with each new game, the player's rating variation approaches zero, thus $K_G$ goes to zero as well. As a result, the Glicko rating always converges to a value, though it may be different than the player's actual rating. The one factor that increases a player's rating deviation is the amount of time between games (i.e. tournaments, in the context of chess, for which this system was developed). Without sufficient time between games to increase the player's rating deviation, Glicko quickly becomes too slow for reacting to any changes in the player's actual rating.

The Glicko-2 system [Glickman 2001] introduces a *rating volatility* property per player to Glicko. This results in a more complicated system that reportedly improves the accuracy of the estimated game outcomes. However, the task of accurately estimating a player's *actual rating volatility* is a major challenge, particularly with proper matchmaking. This is because volatility is a measure of unexpected variance in game outcomes and, when players are matched against opponents with similar skill levels, the expected variance would be much higher. Therefore, accurately estimating volatility requires a large number of game outcomes and a player's actual volatility may change quicker than it can be estimated. In addition, Glicko-2 is prone to "volatility hacking," a process that incentivizes losing some games to artificially inflate one's rating [Ebtekar and Liu 2021]. Therefore, we do not include volatility in our system.

### 2.3 TrueSkill Rating Systems

TrueSkill [Herbrich et al. 2007] is a prominent alternative to Elo and Glicko. It is theoretically similar to Glicko, but uses the *sum-product algorithm* on a *factor graph* [Kschischang et al. 2001]. This allows representing games involving more than two players and handling teams using individual player ratings. It can also explicitly model draw conditions. In addition, it uses Gaussian distribution for evaluating all probabilities, instead of the logistic distribution used by Elo and Glicko. Gaussian distribution highly simplifies the calculations, but otherwise makes a negligible difference, since the two distributions are highly similar.

Like Glicko, TrueSkill also tracks a rating deviation property, but increases it before each game by a constant amount. This seemingly minor change can have a profound impact on how responsive the rating estimation is to any changes in a player's actual rating and the overall convergence behavior.

TrueSkill 2 [Minka et al. 2018] extends this system by introducing additional in-game data.

### 2.4 Other Related Work

Some prior work is based on variations of Elo. Sonas [2002] uses a linearized version of Elo's win probability function. Kovalchik [2020] extends Elo by considering the margin of victory, instead of only the game outcome. Powell [2023] introduces a generalization of Elo for games involving more than two players.

Games with more than two players have been targeted by other methods as well. Weng and Lin [2011] use the Bayesian Ranking Approximation algorithm to model *free-for-all* games as pairwise competitions. Another group of methods aims to extract individual ratings from team games [Huang et al. 2008; Li et al. 2018]. OptMach [Gong et al. 2020] uses an offline learning stage for relationship mining and predicting game outcomes, followed by an online optimization to form team matches. Ebtekar and Liu [2021] introduce a Bayesian rating system targeting contests with many participants.

QuickSkill [Zhang et al. 2022] uses a neural network and game-specific data to predict a player's rating after a small number of games. DeepSkill [Rezapour et al. 2024] uses machine learning and game-specific data to produce a real-time rating in an MMO team. Game-specific data was also used in other models as well [Chen et al. 2017; Menke and Martinez 2008].

Though the single-dimensional rating systems cannot model intransitivity, it can be learned from a collection of game results using a multi-dimensional system [Chen and Joachims 2016].

## 3 ESTIMATING PLAYER RATINGS

Matchmaking based on player skill requires properly estimating player's ratings. The formulations we describe below stem from the Elo and Glicko systems, but depart from them in the way that the resulting ratings are computed. We begin by describing our general formulation for updating player's ratings following a series of games (Section 3.1). Then, we describe a simple model that allows incremental updates (Section 3.2). Finally, we describe how our solution could be modified to ensure zero-sum updates, a desirable property that can improve the rating estimation (Section 3.3).

Though the matchmaking strategy we describe in Section 4 can be used with other rating models as well, the accuracy of our rating estimation model improves matchmaking and the opponent selection of matchmaking in turn contributes to improving the accuracy of the estimation.

### 3.1 Rating Updates

Our solution for updating a player's rating after a series of $m$ games can be applied to any given win probability function $f$. We follow a formulation similar to Glicko's update rule, such that

$$r'_a = r_a + K\,F'_a \qquad \text{where} \qquad F'_a = \sum_i^m \omega_i\, g(\phi_i)\big(s_i - f(a,i)\big) \tag{6}$$

with an additional weight term $\omega_i$ per game and an important difference where the $K$-factor is computed using the derivative of the win probability function $\frac{d\,f}{d\,r_a}$, such that

$$K = \frac{1}{D'_a} \qquad \text{where} \qquad D'_a = \sum_i^m \omega_i\, g(\phi_i)\frac{d}{d\,r_a} f(a,i)\ . \tag{7}$$

In our tests, we use Elo's win probability function $f = f_E$, but other functions can be used as well. We provide the derivative of $f_E$ in Appendix A.

Note that assuming $f$ is a linear function, the updated rating $r'_a$ results in $F'_a = 0$, when the probabilities $f$ are computed using $r'_a$, instead of $r_a$. Thus, $r'_a$ perfectly represents the weighted sum of the observed game outcomes of a linear $f$. Yet, $f$ is typically nonlinear. In this case, this update rule can be repeated multiple times until it converges. This corresponds to Newton's method, which often converges after a small number of iterations.

However, Newton's method is not guaranteed to converge, so it is a good idea to limit the change in rating by a certain threshold $\Delta r_{\max}$ and limiting $r'_a$ to the range $r_a \pm \Delta r_{\max}$. Though this safety threshold is certainly needed in practice, using a relatively large threshold ($\Delta r_{\max} = 350$ in our tests, which corresponds to a win probability change of about 88%), it is rarely used. Except for the first few games, when there is little data, most rating updates already fall within $\pm\Delta r_{\max}$.

The weights $\omega_i$ in Equations 6 and 7 are chosen to emphasize the contributions of more recent games over historical data. We achieve this by enforcing $\omega_{i-1} \le \omega_i$, starting with $\omega_m = 1$, and using

$$\omega_{i-1} = g(\alpha\,\phi_a^{(i)})\,\omega_i\ , \tag{8}$$

where $\alpha \ge 1$ is a system-level parameter that controls how responsive the rating prediction would be to the changes in the player's actual rating (we use $\alpha = 2$ in our tests) and $\phi_a^{(i)}$ is the player's rating deviation prior to game $i$. Notice that in addition to assigning higher weights to more recent games, this weighting scheme scales the contributions of past games based on the player's rating deviation. Thus, as the rating deviation approaches zero, all weights approach 1. If the player's rating deviation increases, the weights of the past games decrease accordingly and more recent games are emphasized.

Unlike Glicko, we do not attempt to predict the rating deviation $\phi$ with a formula that goes to zero. Instead, we numerically compute a running variance of the player's rating by weighting the contributions of the past games using the same weights as in Equation 8. We use a slightly modified version of West's algorithm [West 1979] by scaling past weights by $g(\alpha\,\phi_a)$, such that

$$W'_a = g(\alpha\,\phi_a)\,W_a + 1 \tag{9}$$

$$R'_a = R_a + (r'_a - R_a)/W'_a \tag{10}$$

$$V'_a = g(\alpha\,\phi_a)\,V_a + (r'_a - R_a)(r'_a - R'_a)\ , \tag{11}$$

where $W_a = \sum_i \omega_i$ is the running weighted sum of the variance weights, $R_a = \sum_i \omega_i r_a^{(i)}$ is the running weighted mean using the estimated rating $r_a^{(i)}$ after game $i$, and

$$\phi_a^2 = \frac{V_a}{W_a} \tag{12}$$

is the resulting weighted running variance. Prior to the first game of the player, the variables $R_a$ and $V_a$ are initialized to zero, and $W_a$ is initialized to a small value (just for avoiding division by zero for estimating the variance before the first game in Equation 12).

The underlying reason for this weighting model is the assumption that a player's actual rating can vary over time, as the players play more games and improve their skills. In some cases, even sudden and significant changes in actual rating may be expected, such as when the player discovers a new effective strategy. This weighing scheme automatically responds to quick changes in a player's estimated rating that increase the variance and thereby reduce the weights of past games.

### 3.2 Incremental Rating Updates

In practice, it is often desirable to update the player's rating after each game without having to store the information about all prior games. We accomplish this by approximating the sums $F'_a$ and $D'_a$ in Equations 6 and 7, which are computed by evaluating $f$ and its derivative using the current rating $r_a$. For computing these sums incrementally, we approximate $f(a, b)$ using $f(a^{(i)}, b)$ computed with $r_a^{(i)}$ and its derivative at $r_a^{(i)}$, such that

$$f(a, b) \approx f(a^{(i)}, b) + \left(r_a - r_a^{(i)}\right) \frac{d}{d\, r_a} f(a^{(i)}, b) \ . \tag{13}$$

This corresponds to a linear approximation of $f$, assuming that its derivative is constant. Let $F_a$ and $D_a$ be the running sums for the previous $m - 1$ games. The resulting update rule for the running sums after a game with each opponent $b$ then becomes

$$F'_a = g(\phi_b)\big(s - f(a, b)\big) + g(\alpha\, \phi_a)\big(F_a - \Delta r_a\, D_a\big) \tag{14}$$

$$D'_a = g(\phi_b) \frac{d}{d\, r_a} f(a, b) + g(\alpha\, \phi_a) D_a \tag{15}$$

where $\Delta r_a = r_a - r_a^{(m-1)}$ is the change in rating after the previous game. Initially, both running sums ($F_a$ and $D_a$) and $\Delta r_a$ are taken as zero.

By rearranging the terms in Equations 6 and 7, we can easily show that this update rule satisfies

$$F'_a - \Delta r'_a\, D'_a = 0 \qquad \text{where} \qquad \Delta r'_a = r'_a - r_a \ . \tag{16}$$

This means that $F_a - \Delta r_a\, D_a = 0$ was also satisfied after the previous rating update, so Equation 14 can be simplified by removing the second term, such that

$$F'_a = g(\phi_b)\big(s - f(a, b)\big) \tag{17}$$

and the resulting update rule can written as

$$r'_a = r_a + \frac{1}{D'_a}\, g(\phi_b)\big(s - f(a, b)\big) \ . \tag{18}$$

Note that in this form $F_a$ is entirely removed from the equation and we only need to store the running sum of the derivatives $D_a$ per player. This results in a highly efficient update rule and only requires storing $D_a$ along with the other running sums $W_a$, $R_a$, and $V_a$ per player, in addition to the estimated rating $r_a$.

**Algorithm 1:** Pseudocode of our incremental rating update for player $a$ after a game with opponent $b$ and resulting score $s$ (i.e. $s = 1$ if player $a$ wins and $s = 0$ if player $a$ loses).

```
1  function RatingUpdate(a, b, s)          // Function that updates a after a game with b
2      F_a  ← g(φ_b)(s − f(a, b))          // Equation 17, F_a is not stored.
3      D_a  ← g(φ_b) (d/dr_a) f(a, b) + g(α φ_a) D_a   // Equation 15 to update D_a
4      Δr   ← F_a / D_a                    // Δr is the rating change (not stored).
5      Δr   ← min(Δr_max, max(−Δr_max, Δr))   // Ensure that |Δr| is not too large.
6      if |Δr| = Δr_max then D_a ← F_a / Δr   // Correct D_a after limiting Δr by Δr_max.
7      UpdateStats(r_a, W_a, R_a, V_a, Δr)   // Use Δr to update the rating and stats.
8  function UpdateStats(r_a, W_a, R_a, V_a, Δr)   // Function that updates r_a and its statistics
9      r_a  ← r_a + s Δr                   // Equation 6 to update r_a
10     ω    ← g(α φ_a)                     // the weight current V_a/W_a (not stored)
11     W_a  ← ω W_a + 1                    // Equation 9 to update W_a
12     ΔR   ← r_a − R_a                    // ΔR is not stored.
13     R_a  ← R_a + ΔR/W_a                 // Equation 10 to update R_a
14     V_a  ← ω V_a + ΔR(r_a − R_a)        // Equation 11 to update V_a
```

Algorithm 1 shows the pseudocode of this incremental rating update process, evaluating the equations above. Since this corresponds to a Newton step, it is good practice to limit the rating update by $\Delta r_{\max}$. When $\Delta r'_a$ is modified by this limit, we should also set $D'_a = F'_a/\Delta r_a$, so that Equation 16 is still satisfied (as shown on line 6 of Algorithm 1). In addition, line 9 of Algorithm 1 includes an optional *scaling factor* parameter $s \leq 1$. We use $s = 1$, unless otherwise stated, but $s < 1$ can be used to improve numerical stability with extreme cases, as we show in Section 5.4.

### 3.3 Zero-Sum Rating Updates

The update rule described above is not zero-sum, meaning after two players play a game the increase in one player's rating can be smaller or larger than the decrease in the other player's rating. This is expected, because each player's rating update is impacted by the results of their prior games and opponents.

If zero-sum updates are desired, we can convert the update rules above to one that simultaneously considers both players and solves a simple least-squares optimization to determine a common rating change $\Delta r'$, such that

$$r'_a = r_a + \Delta r' \tag{19}$$

$$r'_b = r_b - \Delta r' \ . \tag{20}$$

The resulting optimization problem can be written as

$$\Delta r' = \underset{\Delta r'}{\operatorname{argmin}} \|\mathbf{D}\,\Delta r' - \mathbf{F}\|^2 \ , \qquad \text{where} \qquad \mathbf{D} = \begin{bmatrix} D'_a \\ -D'_b \end{bmatrix} \qquad \text{and} \qquad \mathbf{F} = \begin{bmatrix} F'_a \\ F'_b \end{bmatrix} \ . \tag{21}$$

This simple problem has a closed-form solution, such that

$$\Delta r' = \frac{\mathbf{D}^T\mathbf{F}}{\mathbf{D}^T\mathbf{D}} \ . \tag{22}$$

**Algorithm 2:** Pseudocode of our zero-sum rating update for two players $a$ and $b$ play a game with resulting score $s$ (i.e. $s = 1$ if player $a$ wins and $s = 0$ if player $a$ loses).

**function** `ZeroSumRatingUpdate`$(a, b, s)$ // Function that updates $a$ and $b$ after a game
  $F_a \leftarrow g(\phi_b)\big(s - f(a,b)\big)$ // Equation 17, $F_a$ is not stored.
  $F_b \leftarrow g(\phi_a)\big(f(a,b) - s\big)$ // Equation 17, $F_b$ is not stored.
  $D_a \leftarrow g(\phi_b)\frac{d}{d\,r_a} f(a,b) + g(\alpha\,\phi_a) D_a$ // Equation 15 to update $D_a$
  $D_b \leftarrow g(\phi_a)\frac{d}{d\,r_b} f(b,a) + g(\alpha\,\phi_b) D_b$ // Equation 15 to update $D_b$
  $\Delta r \leftarrow (D_a\,F_a - D_b\,F_b)/(D_a\,D_a + D_b\,D_b)$ // $\Delta r$ is not stored.
  $\Delta r \leftarrow \min(\Delta r_{\max}, \max(-\Delta r_{\max}, \Delta r))$ // Ensure that $|\Delta r|$ is not too large.
  **if** $|\Delta r| \neq 0$ **then**
    $D_a \leftarrow F_a/\Delta r$ // Correct $D_a$ to satisfy Equation 16.
    $D_b \leftarrow -F_b/\Delta r$ // Correct $D_b$ to satisfy Equation 16.
  **end**
  `UpdateStats`$(r_a, W_a, R_a, V_a, \Delta r)$ // Use $\Delta r$ to update player $a$.
  `UpdateStats`$(r_b, W_b, R_b, V_b, -\Delta r)$ // Use $\Delta r$ to update player $b$.

These zero-sum updates do not necessarily satisfy Equation 16, so correct computation requires storing $F_a$ per player and using Equation 14 to update it. However, the optimization aims to find $\Delta r'$ that minimizes the error in Equation 16 for both players. Therefore, in practice, it is safe to assume that Equation 16 is perfectly satisfied and omit storing $F_a$. Though this introduces a small amount of error, we found it to be insignificant in our tests. To compensate for this, we instead update the stored $D_a$ and $D_b$ values to satisfy Equation 16 for both players. Algorithm 2 shows the pseudocode of zero-sum updates that jointly changes the ratings of both players.

This zero-sum update rule results in more conservative changes in players' ratings. Such conservative updates make the ratings more stable, as they introduce smaller fluctuations to players' ratings over time after each game. In our tests, we found that zero-sum updates can slightly improve convergence, though this is not guaranteed. Nonetheless, zero-sum updates are recommended in practice because of their stability improvements by producing smaller rating changes.

## 4 MATCHMAKING

Matchmaking for games can involve various criteria. Here, we only concentrate on finding a desirable opponent for a player purely based on player ratings.

One important goal is to promote a win rate of 50% for all players, regardless of their ratings. Purely relying on probabilities, this would mean simply picking two players with identical ratings. In practice, however, there may not be an opponent with an identical rating in the pool for matchmaking. Also, purely relying on probabilities does not provide any guarantees on a player's experience. For example, due to various reasons, the actual win rate for a player within a window of games may significantly deviate from the expected value. While such deviations may be corrected later on with counter deviations that would balance the statistics, the experience of repeatedly losing games, for example, is likely to negatively impact a player's experience.

That is why, matchmaking should also consider a player's recent win record and try to maintain a win rate of 50% within a desirable window. We achieve this by evaluating the ideal opponent rating that would produce the desired expected win rate for a series of games, as we explain below.

### 4.1 Desired Opponent Rating for Matchmaking

Let $n$ be the number of previous games we consider for tracking the recent win rate of a player and $w$ be the number of games won by the player within this window (i.e. $w \leq n$). Using this data, we would like to promote an expected win rate of $\lambda$, such as $\lambda = \frac{1}{2}$, within a window of $2n+1$ games (including $n+1$ future games). If we pick a set of opponents against whom the player has a win probability of $p$ (based on the win probability function $f$) for the next $n+1$ games, the expected win rate $\lambda_p$ for the entire window of $2n+1$ games can be written as

$$\lambda_p = \frac{w + (n+1)p}{2n+1} \ . \tag{23}$$

By setting $\lambda = \lambda_p$ and rearranging the terms, the desired win probability $p$ for the next game can be calculated using

$$p = \frac{\lambda(2n+1) - w}{n+1} \ . \tag{24}$$

In the beginning, when the player has only played $m$ games, such that $m < n$, we use a window of $n+m+1$ games, resulting

$$p = \frac{\lambda(n+m+1) - w}{n+1} \ . \tag{25}$$

Notice that when $m = 0$ (and so $w = 0$), this equation results in $p = \lambda$.

Based on the above, the desired opponent rating $r_\star$ that would provide a win probability of $p = f(r_\star)$ for the player can be simply computed using the inverse of the win probability function

$$r_\star = f^{-1}(p) \ . \tag{26}$$

In practice, finding an opponent with this exact rating would be a challenge, so we must define an acceptable window for the opponent's rating around $r_\star$. Let $\Delta p$ define the probability window. The rating window for the opponent is then $f^{-1}(p \pm \Delta p)$. Note that using $\Delta p < 1/(2n+2)$ would ensure that $p - \Delta p > 0$ and $p + \Delta p < 1$.

### 4.2 Matchmaking for a Group of Players

When a group of players are connected to a game server for matchmaking, the server needs to find opponents for *all* players (not just one, as we considered above). The ideal solution involves for each player finding an opponent with a desired rating, such that the desired rating of the opponent also matches the player's rating. Though it is unlikely to find such an opponent for all players, this can be turned into an optimization problem that minimizes a loss function defined as the difference between the desired win probabilities and the assigned win probabilities after matchmaking.

We propose an alternative solution that is significantly simpler than solving this optimization problem. Our solution only considers the desired opponent ratings of all players, entirely ignoring their estimated ratings. This can be considered as treating players as if they have the same rating as their desired opponent rating. Our simple matchmaking algorithm for a group of players operates under this simplifying assumption, matching players with similar desired opponent ratings. Obviously, this does not lead to an optimal solution that minimizes a particular loss function, as mentioned above, but it can provide a simple and reasonable approximation in practice.

Algorithmically, this matchmaking can be performed by simply sorting the players based on their desired opponent ratings and pairing nearby players in this sorted list. After sorting, additional matchmaking criteria (besides rating) may be considered, instead of simply pairing consecutive players on the list. For example, it may be preferable to avoid repeatedly matching the same pair of players. In our tests, we statistically avoid such cases by simply shuffling the sorted list by swapping players within a small window.

## 5 EVALUATION

We jointly evaluate our matchmaking system together with our rating estimation method. We also provide comparisons to Elo (using $K = 24$, as recommended [Glickman 1995]), Glicko (assuming no time between games that would increase the rating deviations), and TrueSkill (using its default parameters) systems. We update player ratings after each game, as this is typical for online multiplayer games.

### 5.1 Experimental Setup

In our experiments, we rely on synthetic simulated data, as opposed to historical data collected from a series of games. There are three important reasons for this decision:

(1) We cannot modify the matchmaking of historical games after they have been played. Thus, historical data does not provide an effective mechanism for evaluating matchmaking.

(2) Historical data does not contain actual ratings of players. We can only have estimated ratings using different methods, but we cannot justifiably assert any actual ratings for players.

(3) Measuring accuracy with historical data is ineffective. This is because, in a correctly-matched game involving players with similar actual ratings, the players have similar probabilities to win, but the historical data only shows the outcomes of the games (i.e. who won), not these probabilities. Therefore, in the context of proper skill-based matchmaking, attempting to predict the outcomes of a game is often similar to predicting the result of a coin toss.

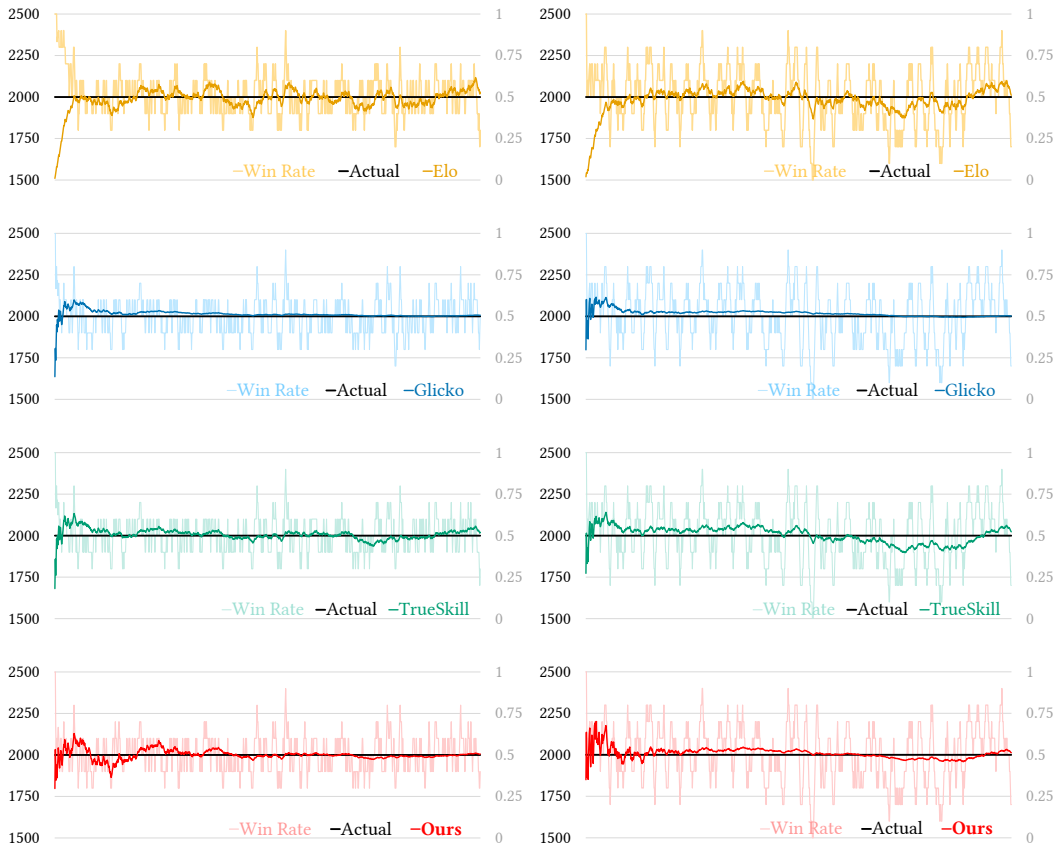
Therefore, our experiments use simulated games, where we can control matchmaking, know and modify the exact actual ratings of players, and properly evaluate the accuracy of different methods.

Our simulations are based on the same assumptions as the Elo system: each player has an actual rating and the outcome of a game can be determined using Elo's win probability function. Thus, we use a random number generator to determine the outcomes of games using players' actual ratings with Elo's formula.

### 5.2 Estimating a Single Player's Rating

Our first set of experiments assume that a new player enters a large pool of players for whom the estimated ratings have converged to their actual ratings. Thus, the only unknown in the system is this new player's rating, which we estimate using different methods. Unless otherwise stated, the opponents are generated by randomly picking a rating within the desired opponent rating window of the new player (using $n = 5$, which forms a window of 11 games, and $\Delta p = 1/(2n + 2) = 1/12$).

First, we consider a new player with a fixed actual rating. This is analogous to the case of a new player account in the system, but the player has already played the game (e.g. using a different account) long enough to achieve a fixed actual rating. Figure 1 shows example results, comparing our

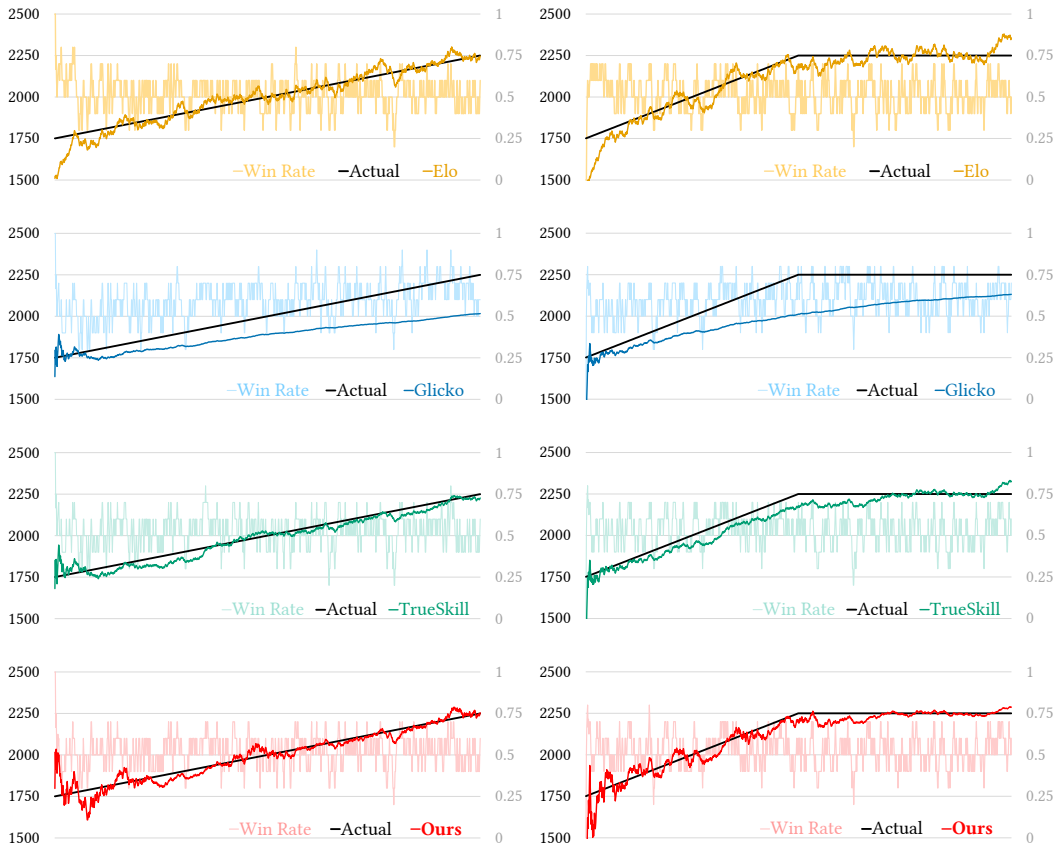



**(a)** *Fixed rating with our matchmaking* **(b)** *Fixed rating with random matchmaking*

**Fig. 1.** *The estimated rating of a new player with a fixed actual rating throughout their first 1000 games, comparing different rating estimation methods: (a) with our matchmaking for a window of 11 games and (b) without skill-based matchmaking by randomly assigning opponents in the rating range from 1750 to 2250. The lighter colors show the win rates of the past 10 games.*

incremental updates to Elo, Glicko, and TrueSkill, starting from the same initial rating estimation of $r_a = 1500$. Using our matchmaking to find opponents within a desired rating window (Figure 1a), all methods do a reasonable job of estimating the rating. Noticeably, Elo fails to converge due to its fixed update rate (i.e. the $K$-factor) and Glicko converges rapidly, while TrueSkill and our method closely estimate the actual rating with some noise due to randomized game outcomes. Also, using our matchmaking, all rating estimation methods achieve a win rate close to 50% most of the time, as can be seen in the win rate of the 10 most recent games, shown in these graphs.

To demonstrate the impact of proper matchmaking, in Figure 1b we show the same experiment but with randomly chosen opponents within a rating range of ±250 of the player's actual rating. This can be considered a reasonable range for picking opponents for this player, as the expected win rate for each match varies from 20% to 80%. Nonetheless, this leads to multiple long win/loss streaks of up to 10 games in this example. This increases the variance estimation of all methods (in comparison to using our matchmaking, as shown in Figure 1a), though this variance increase




**(a)** *Increasing actual rating* **(b)** *First increasing, then fixed rating*

**Fig. 2.** *The estimated rating of a new player throughout their first 1000 games, comparing different rating estimation methods with our matchmaking for a window of 11 games: (a) steadily increasing actual rating and (b) an increasing actual rating that stops increasing halfway through the games. The lighter colors show the win rates of the past 10 games.*

is less noticeable with Glicko, because its rating updates go to zero by design as more games are played without any time between them. Note that since rating estimation does not play a role in picking opponents, all rating estimation methods observe the same game outcomes.

More interesting examples are shown in Figure 2, one with a steadily increasing rating (Figure 2a) and one with a faster initial increase that stops halfway through the games (Figure 2b). In both cases, Elo provides a reasonable though noisy estimation. Glicko, however, fails to respond quickly enough, thereby resulting in a clear increase in win rate due to underestimation of the rating, even though our matchmaking tries to maintain a 50% win rate. Both TrueSkill and our rating estimation track the actual rating relatively closely and even begin to converge when the actual rating becomes fixed.

Figure 3 shows even more challenging cases. In the first one, the player's actual rating suddenly increases (e.g. when the player discovers an effective strategy). As can be seen in Figure 3a, Elo can quickly respond to such changes by making steady improvements based on its constant $K$-factor.


**(a)** *A sudden jump in actual rating* **(b)** *A gap in actual rating*

**Fig. 3.** *The estimated rating of a new player throughout their first 1000 games, comparing different rating estimation methods with our matchmaking for a window of 11 games: (a) a sudden increase in actual rating and (b) a gap in actual rating that is formed by the player intentionally throwing some games. The lighter colors show the win rates of the past 10 games.*

Glicko, however, fails to respond. Since Glicko's rating deviation decreases with every new game, the rating changes it can produce are limited (until the player stops playing and thereby increases its rating deviation). Even though the player has a win rate clearly above 50% after the actual rating change, Glicko's rating estimation lags. TrueSkill immediately reacts to the change in the actual rating, though it takes over 200 games for the estimated rating to reach the new actual rating in this example, during which the player enjoys an elevated win rate. Our method is slower to react to the change at first, but as the variance of the rating estimation increases, it begins to reduce the weights of prior games, and then quickly converges to the player's new actual rating.

Some players may prefer to play against easier opponents and intentionally lose games to reduce their estimated ratings. Such a scenario is presented in Figure 3b, where the player intentionally loses 100 games in a row and then begins to play as normal based on their actual rating. This creates a gap in the actual rating during those 100 games, to which Elo quickly responds. In fact, after intentionally losing only 26 games, Elo's rating estimation falls below 1500, where the player has

over 95% win chance with their actual rating. TrueSkill also responds to this gap quickly, taking 43 games to fall below 1500. Unlike Elo and TrueSkill, our method first reacts slowly, but then rapidly drops the estimated rating, and it takes over 60 games to reach below 1500. Glicko, however, barely reacts to this gap by slightly reducing the rating estimation after every game.

When the player begins to play at their actual rating, our method quickly responds and converges to it, reaching the actual rating only after 12 games in this example. Elo also responds to this change, though much slower than our method, taking 266 games, more than twice as long as the gap, to reach the actual rating. During this time, the player's win rate is highly elevated. Therefore, it is relatively easy for players to keep their rating estimations low with Elo. TrueSkill eventually reaches the actual rating as well, though it takes even longer than Elo. By the time Elo recovers, TrueSkill remains 200 points short, which corresponds to an error of up to about 25% in win probability estimation (see Equation 1 in the supplemental document for the conversion of rating errors to win probability errors). Glicko, as expected, fails to properly respond to this change as well by steadily underestimating the player's rating after this gap, providing a higher win rate than 50% despite the counter effects of our matchmaking system, frequently providing opponents with higher ratings than the estimated rating of the player by Glicko.

All these tests show that our method can accurately estimate a player's actual rating and properly respond to slight or drastic changes to it. Overall, it not only provides a more accurate estimation, but also achieves a desirable win rate with our matchmaking. Though our matchmaking method works well with any rating estimation method, it performs better with the more reliable rating estimations provided by our method.

We must also emphasize that these experiments do not tell the full story about Elo, Glicko, or TrueSkill. Elo's $K$ parameter can be increased to make it faster to respond to changes or decreased to make it less noisy. Adding time between games to increase Glicko's rating deviation would make it behave closer to TrueSkill. Similarly, TrueSkill can behave similar to Glicko in these examples by setting its dynamic variance parameter to zero. These are demonstrated in our supplemental document using alternative parameters. Nonetheless, not only do these methods require tuning their parameters, which is a significant challenge in practice, but also no single parameter exists that would make any of these methods outperform ours in all of these tests presented above.

### 5.3 Estimating Population Ratings

The advantages of our rating estimation can also be seen in the context of estimating ratings for an entire population. The tests here consider the scenario of a newly-released game with 1000 players. Rating estimations begin with assuming that all players have the same rating of 1500. At each round, all players are matched up with their opponents using our skill-based matchmaking method (Section 4.2). The sorted list of players is shuffled by swapping each player with a randomly selected player within a window of 50 players (i.e. 5% of the population size) to imitate imperfect matchmaking. Also, always matching up the same pairs of players with each other would not produce sufficient data for any rating estimation method. The results of the games are used for updating the rating estimations. We use zero-sum updates with our method (Algorithm 2).

Figure 4 shows the rating estimation results for a population with a Gaussian distribution of actual ratings in the range of 0 to 3000 (with a mean of 1500 and a standard deviation of 600), assuming that actual ratings (i.e. the relative strengths of players) remain constant. Notice that after only 100 rounds of games, our method provides a reasonable rating estimation for all players. Elo, Glicko,

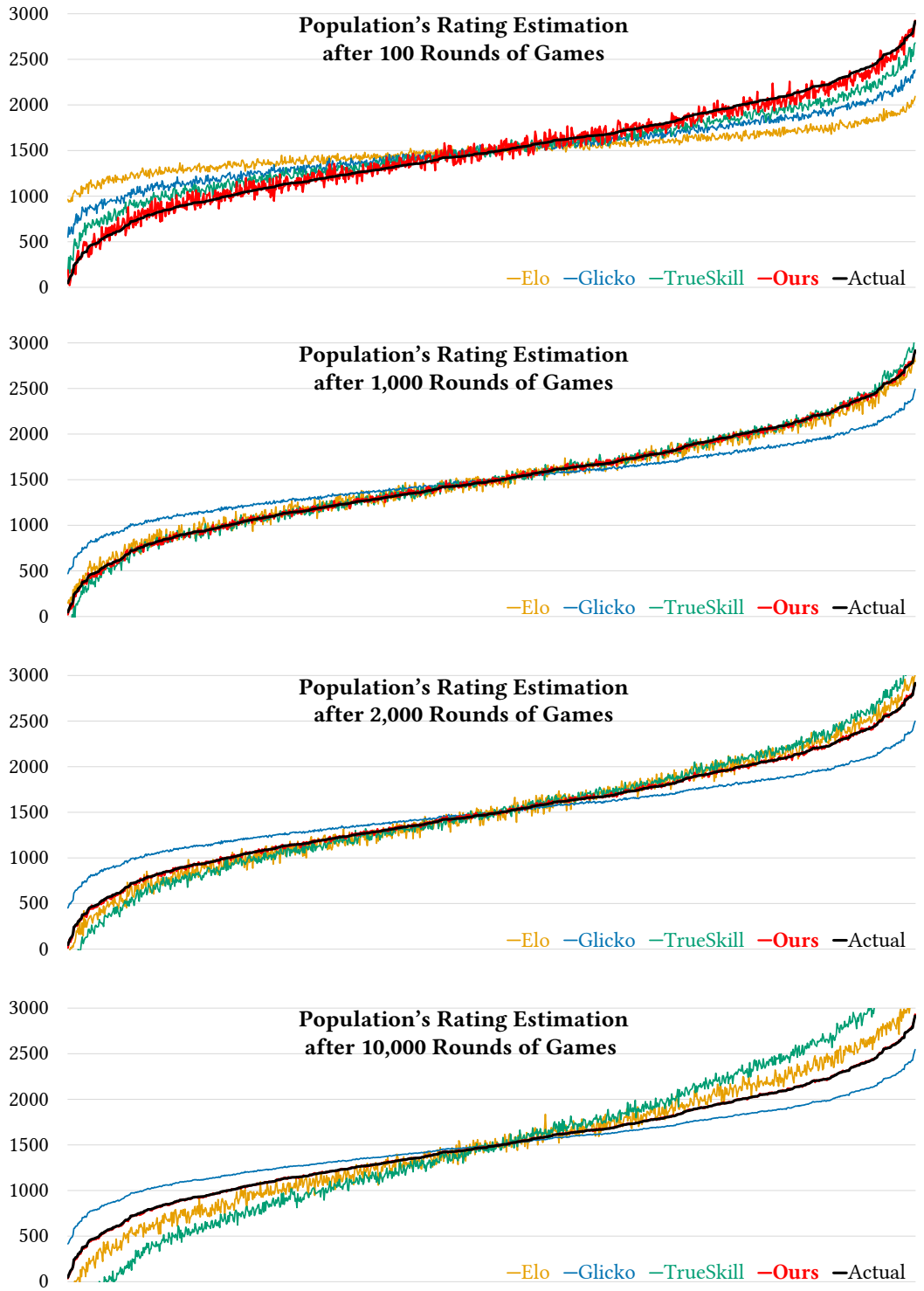



**Fig. 4.** *A population's rating estimation after one hundred, one thousand, and ten thousand rounds of games with different rating estimation methods. The horizontal axis is the player population.*

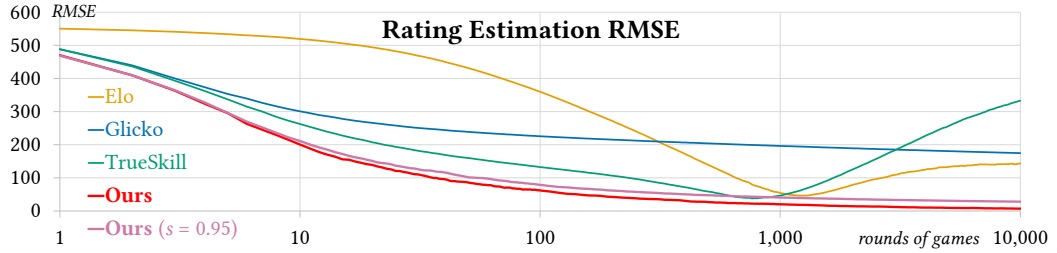



**Fig. 5.** *Root mean square error (RMSE) of different rating estimation methods for the same population in Figure 4 after each round of games on a log scale. This graph also includes a version of our method using a scale factor of $s = 0.95$, as explained in Section 5.4.*

and TrueSkill fail to match the range of the actual rating distribution, but all methods perform reasonably well at separating stronger players from weaker ones.

After 1,000 rounds, the rating estimation of our method is close to the actual ratings with a root mean square error (RMSE) of only 20, corresponding to a win probability error of less than 3%. At this point, Elo and TrueSkill also provide good rating estimations with RMSE of 54 and 46, corresponding to 8% and 7% win probability errors, respectively. Glicko does a good job of ordering the players based on ratings, but underestimates their actual rating differences.

The problems with Elo and TrueSkill begin to emerge after 1,000 rounds, as they continue to diverge, instead of converging to the actual ratings. At 2,000 rounds, both Elo and TrueSkill have higher RMSE of 79 and 125, corresponding to 10% and 17% win probability errors, respectively. This behavior can also be seen in Figure 5, showing the rating estimation RMSE after each round of games. Notice that, in this example, Elo and TrueSkill reach their minimum RMSE around 1,000 rounds and then begin to diverge. Glicko does not exhibit this divergent behavior only because we assume no time between games. As we show in our supplemental material, Glicko also suffers from the same problem when the rating deviations are permitted to increase between games.

With our method, RMSE falls below 200 (less than 25% error in win probability) only after 10 rounds (see Figure 5) and continues to decrease, reaching an RMSE of 7 (less than 1% error in win probability) at 10,000 rounds, where our rating values are barely visible behind the actual ratings in Figure 4.

The win rate of each player is shown in Figure 6 after 10, 100, and 1,000 rounds with our method. As expected, our matchmaking achieves close to a 50% win rate for most players in these tests with all rating estimation methods. However, achieving the desired win rate is a challenge near the extreme ends of the population's rating distribution, since no player exists who has a high chance of winning against the player with the highest rating. A similar circumstance also impacts the players near the lowest end of the rating spectrum. For a vast majority of players, however, our skill-based matchmaking reaches its goal of close to a 50% win rate in these tests.

We repeated these experiments using larger player populations (up to one million) and observed virtually identical results as the ones explained above, indicating that our conclusions extend to much larger player populations.

In our supplemental document we include the same experiment with 1000 players, but using alternative parameters for Elo, Glicko, and TrueSkill, showing that their behavior can be significantly impacted by their parameters. More specifically, with a larger $K$-factor, Elo can reduce the RMSE

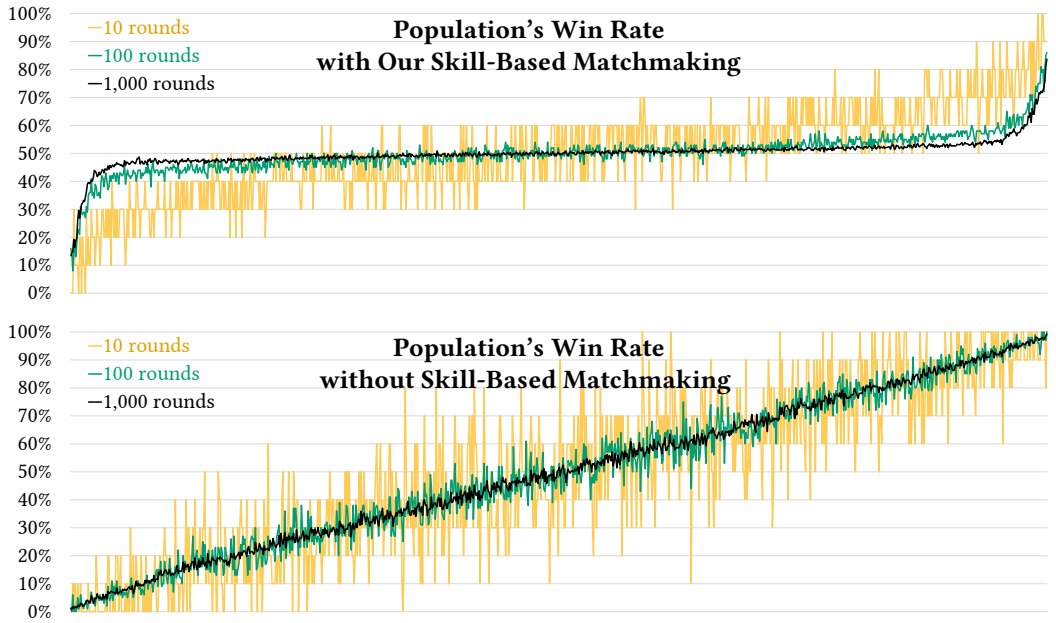



**Fig. 6.** *The win rate of all players (sorted by actual rating) after 10, 100, and 1000 rounds of games with and without our skill-based matchmaking. For the former, our rating estimation is used and the sorted players are shuffled by randomizing the placements within a window of 5% of players.*

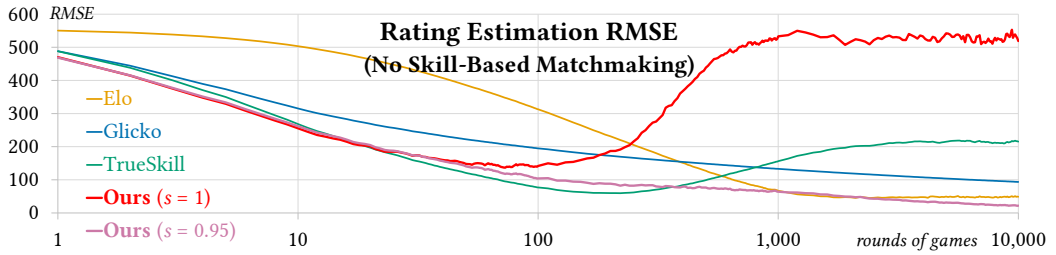



**Fig. 7.** *Root mean square error (RMSE) of rating estimation for the population after each round without skill-based matchmaking by randomly pairing players.*

faster, but then it begins diverging earlier. With the alternative parameters, Glicko and TrueSkill swap behaviors (i.e. Glicko begins to diverge and TrueSkill underestimates the rating range).

### 5.4 Estimating Population Ratings Without Skill-Based Matchmaking

To show the impact of our skill-based matchmaking, we repeat the same experiment in Figure 5 without skill-based matchmaking, by randomly pairing players for each round. The RMSE results of rating estimation for this experiment are shown in Figure 7.

Interestingly, all methods, except for ours, perform better without skill-based matchmaking in this test. In particular, Elo becomes stable after 1,000 rounds at around an RMSE of 50, though its RMSE decreases slower than the other methods until then.

Our rating estimation, on the other hand, becomes unstable without skill-based matchmaking. Though RMSE decreases for about 100 rounds, it then begins to increase and remains at a relatively high level. This is because the Newton iterations we use and the linear approximation of $f$ perform poorly in this case. We must emphasize that this is an extreme case and that our method remains stable in our tests even with highly poor skill-based matchmaking by shuffling the sorted players within a range of 30% of the population. Such an example is included in the supplemental document.

Fortunately, we can offer a simple solution to the stability problems of our method for such extreme cases, using the optional scaling factor $s$. As can be seen in Figure 7, our rating estimation becomes stable with $s = 0.95$ outperforming all other methods.

We include the RMSE curve of our method with $s = 0.95$ in Figure 5 as well to show that using $s < 1$ negatively impacts convergence, but not by a significant amount. Therefore, applications that cannot rely on skill-based matchmaking quality might favor $s < 1$.

## 6 CONCLUSION

We have presented a rating estimation method for competitive two-player games and methods for skill-based matchmaking that promote a target win rate for players. We show that our rating estimation can achieve better accuracy than prior methods and it can properly respond to sudden changes in a player's rating. Our results show that our skill-based matchmaking can achieve the desired win rate for most players in a population.

Though we have only explored two-player games in our formulation and evaluation, our method can be easily used for pre-arranged teams, where each team can be treated as a new player. However, we have not explored automated team selection from individual player ratings. An interesting future direction would be extending our approach to games with more than two players/teams.

## REFERENCES


Shuo Chen and Thorsten Joachims. 2016. Modeling Intransitivity in Matchup and Comparison Data. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining (WSDM '16)*. Association for Computing Machinery, New York, NY, USA, 227âĂŞ236. https://doi.org/10.1145/2835776.2835787

Zhengxing Chen, Yizhou Sun, Magy Seif El-Nasr, and Truong-Huy D. Nguyen. 2017. Player Skill Decomposition in Multiplayer Online Battle Arenas. *ArXiv* abs/1702.06253 (2017). https://api.semanticscholar.org/CorpusID:16985515

Aram Ebtekar and Paul Liu. 2021. Elo-MMR: A Rating System for Massive Multiplayer Competitions. In *Proceedings of the Web Conference 2021 (WWW '21)*. Association for Computing Machinery, New York, NY, USA, 1772âĂŞ1784. https://doi.org/10.1145/3442381.3450091

Arpad Elo. 1978. *The Rating of Chess Players Past and Present.* Arco.

Mark Glickman. 1995. A comprehensive guide to chess ratings. *Chess Journal* 3 (1995), 59–102.

Mark E. Glickman. 1999. Parameter estimation in large dynamic paired comparison experiments. *Journal of Applied Statistics* 48 (1999), 377–394.

Mark E. Glickman. 2001. Dynamic paired comparison models with stochastic variances. *Journal of Applied Statistics* 29 (2001), 673–689.

Mark E. Glickman and Albyn C. Jones. 1999. Rating the chess rating system. *Chance* 12, 2 (1999), 21–28.

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 *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '20)*. Association for Computing Machinery, New York, NY, USA, 2300âĂŞ2310. https://doi.org/10.1145/3394486.3403279

Ralf Herbrich, Tom Minka, and Thore Graepel. 2007. TrueSkill™: A Bayesian Skill Rating System. In *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference.* https://doi.org/10.7551/mitpress/7503.003.0076

Tzu-Kuo Huang, Chih-Jen Lin, and Ruby C. Weng. 2008. Ranking Individuals by Group Comparisons. *Journal of Machine Learning Research* 9, 72 (2008), 2187–2216. http://jmlr.org/papers/v9/huang08a.html

Stephanie Kovalchik. 2020. Extension of the Elo rating system to margin of victory. *International Journal of Forecasting* 36, 4 (2020), 1329–1341. https://doi.org/10.1016/j.ijforecast.2020.01.006

F.R. Kschischang, B.J. Frey, and H.-A. Loeliger. 2001. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory* 47, 2 (2001), 498–519. https://doi.org/10.1109/18.910572

Yao Li, Minhao Cheng, Kevin Fujii, Fushing Hsieh, and Cho-Jui Hsieh. 2018. Learning from Group Comparisons: Exploiting Higher Order Interactions. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2018/file/8208974663db80265e9bfe7b222dcb18-Paper.pdf

Joshua E. Menke and Tony R. Martinez. 2008. A BradleyâĂŞTerry Artificial Neural Network Model for Individual Ratings in Group Competitions. *Neural Comput. Appl.* 17, 2 (feb 2008), 175âĂŞ186. https://doi.org/10.1007/s00521-006-0080-8

Tom Minka, Ryan Cleven, and Yordan Zaykov. 2018. *TrueSkill 2: An improved Bayesian skill rating system.* Technical Report MSR-TR-2018-8. Microsoft. https://www.microsoft.com/en-us/research/publication/trueskill-2-improved-bayesian-skill-rating-system/

Ben Powell. 2023. Generalizing the Elo rating system for multiplayer games and races: why endurance is better than speed. *Journal of Quantitative Analysis in Sports* 19, 3 (2023), 223–243. https://doi.org/doi:10.1515/jqas-2023-0004

Mohammad Mahdi Rezapour, Afsaneh Fatemi, and Mohammad Ali Nematbakhsh. 2024. DeepSkill: A methodology for measuring teams' skills in massively multiplayer online games. *Multimedia Tools and Applications* 83, 10 (01 Mar 2024), 31049–31079. https://doi.org/10.1007/s11042-023-15796-x

Jeff Sonas. 2002. The Sonas rating formula - better than Elo? https://en.chessbase.com/post/the-sonas-rating-formula-better-than-elo

Ruby C. Weng and Chih-Jen Lin. 2011. A Bayesian Approximation Method for Online Ranking. *Journal of Machine Learning Research* 12, 9 (2011), 267–300. http://jmlr.org/papers/v12/weng11a.html

David H. West. 1979. Updating mean and variance estimates: an improved method. *Commun. ACM* 22 (1979), 532–535. https://api.semanticscholar.org/CorpusID:30671293

Chaoyun Zhang, Kai Wang, Hao Chen, Ge Fan, Yingjie Li, Lifang Wu, and Bingchao Zheng. 2022. QuickSkill: Novice Skill Estimation in Online Multiplayer Games. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management (CIKM '22)*. Association for Computing Machinery, New York, NY, USA, 3644âĂŞ3653. https://doi.org/10.1145/3511808.3557070

## A THE DERIVATIVE OF ELO'S WIN PROBABILITY FUNCTION

The derivative of Elo's win probability function $f_E$ in Equation 1 can be simplified as

$$\begin{aligned}\frac{d}{d\,r_a} f_E(a,b) &= q\, f_E(a,b)\big(1 - f_E(a,b)\big)\\ &= q\, f_E(a,b)\, f_E(b,a)\\ &= \frac{d}{d\,r_b} f_E(b,a)\\ &= -\frac{d}{d\,r_b} f_E(a,b)\ .\end{aligned}$$