Elo-R Competition Rating System

Aram Ebtekar

April 11, 2016

1 Introduction

This paper describes the Elo-R rating system, designed for use in programming competitions. The "R" in the name may stand for "Ranked", as it operates on game outcomes given as a rank-ordered list of players; or it may stand for "Robust", as the system is designed to be robust to outlier performances, like someone having a bad day in which their Internet connection died.

Competitions, in the form of sports, games and examinations, have been with us since antiquity. Whether the purpose is entertainment, training, or selection for a particular role, contestants and spectators alike are interested in estimating the relative skills of contestants. Skills are easiest to compare when there is a standard quantifiable objective, such as a score on a standardized test or a completion time in a race. However, most team sports, as well as games such as chess, have no such measure. Instead, a good player is simply one who can frequently win against other good players.

The Elo rating system assigns a quantitative measure of skill to each player. For example, an average player may be rated 1500, while a top player may exceed 2500. The scale is arbitrary, but can be used to rank players relative to one another: odds of one player beating another may be estimated in terms of the difference in their ratings. The Elo system, and some variants such as Glicko [**?**], provide useful formulas for updating ratings of players who play against one another 1v1, where there is a winner and a loser. These algorithms have some nice properties: they are fairly simple, fast, and only affect the ratings of participating players.

Now let's consider a more general setting in which a typical contest has much more than two participants. An arbitrary number of players compete simultaneously at a task. Rather than just a winner and a loser, the players are ranked 1st place, 2nd, 3rd, and so on. This description matches popular programming competition websites such as Codeforces [**?**] and TopCoder [**?**], each of which has tens of thousands of rated members from across the globe. Each publishes its own rating system, but without much theoretical justification to accompany the derivations.

By founding Elo-R upon a more rigorous probabilistic model, we achieve better properties in practice as well. Compared with the Codeforces system, Elo-R achieves faster convergence, more robustness against unusual performances, a more even spread of ratings, less inter-division boundary artifacts, and less inflation. Compared with the TopCoder system, Elo-R is more robust against unusual performances, and monotonic in the sense that performing better in a match will never result in a lower rating. The non-monotonicity of TopCoder has been studied in detail by [**?**]. Furthermore, Elo-R retains simplicity and efficiency on par with the other systems. In fact, I provide a very efficient parallel implementation that can simulate the entire history of Codeforces on a modest laptop within 30 minutes.

The organization is as follows: in section 2, we describe a formal Bayesian model for the competitions, and break down the rating update problem into two phases. Sections 3 and 4 describe each of these phases in turn, and supplement the derivations with some intuitive interpretations. Then in section 5, we discuss some ways to model uncertainty, analogously to how Glicko extended the original Elo system. In section 6, we discuss some properties of the Elo-R system in comparison with the Codeforces and TopCoder systems. Finally, section 7 presents the conclusions of this work.

2 Bayesian Model

Now we describe the setting formally. A series of competitive **rounds**, indexed by $t = 1, 2, 3, \ldots$, take place sequentially in time. The participants of a round t are a subset P_t of all the **players**, which are indexed by i. At time t, player i has latent **skill** $s_{i,t}$, which we seek to estimate from a history of observable **evidence** $e_1, \ldots, e_t$ produced by rounds $1, \ldots, t$.

In Bayesian fashion, we start with a prior belief distribution on $s_{i,t}$ immediately preceding round t, taking into account all of the rounds $1, 2, \ldots, t-1$. We briefly note that the prior distribution at round t is related to but slightly different from the posterior distribution at round $t-1$, as we'll discuss in Section TODO.

In any case, our main task is to combine the prior belief with the evidence e_t to obtain a posterior belief. This will serve as the basis for a post-round **rating**, defined to be the **maximum a posteriori** (MAP) estimate of $s_{i,t}$. That is,

$$r_{i,t} = \arg\max_{s_{i,t}} f(s_{i,t} \mid e_1, \ldots, e_t)$$

where $f(s_{i,t} \mid e_1, \ldots, e_t)$ is the probability density function (p.d.f.) corresponding to our posterior belief on player i's latent skill given the history of evidence.

Let's simplify the notation. Since the pre-round evidence $e_1, \ldots, e_{t-1}$ is common to both the prior and posterior beliefs, we treat them as understood and explicitly write only the new evidence e_t. This leaves t as the only round subscript, so we omit that as well. Thus, the post-round rating can be re-written more simply as

$$r_i = \arg\max_{s_i} f(s_i \mid e) = \arg\max_{s_i} f(s_i) \Pr(e \mid s_i)$$

where we used Bayes' rule to write the posterior p.d.f. as proportional to the product of the prior p.d.f. and the evidence probability.

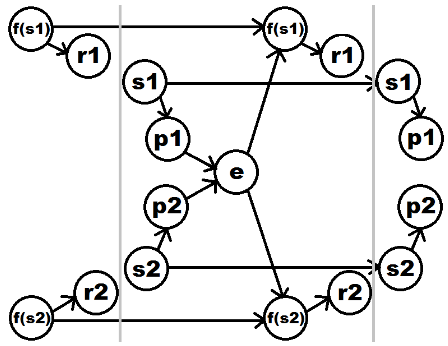
To model the evidence e, we assume a generative model of the round: each player randomly draws their performance p_i independently from a logistic distribution centered at s_i, with a variance that may depend on characteristics of the match t but not on i.

While the variables s_i and p_i are hidden, we assume that player i outranks player j (written $i \succ j$) in the round if, and only if, $p_i > p_j$, and that we observe the complete relative rankings. In other words, the observable evidence e consists of a total order on the set of participants.

Due to this important modeling assumption, the method is best-suited for competitions in which the most pertinent information is round rankings. Note that, in the programming contest setting, this means we discard specific information about player scores, which are difficult to model and depend heavily on the specifics of the problem set. This method would be ill-suited to model, for instance, track races, where a runner's absolute time is more informative than relative rankings.

On the other hand, it may be very well-suited to obstacle-course races, if each round consists of novel obstacles that make the absolute times hard to interpret.

Strictly speaking, ties have probability zero of occurring in our model. Assuming that ties are relatively infrequent in programming contests, they can be handled with a special modification that we'll describe later, essentially treating them as half a win and half a loss.



In order to absorb e into our posterior, a few approximations are in order. First, we assume the number of participants is so large that we're able to compute the performances exactly. Mathematically, this means $\Pr(e \mid p_i)$ is proportional to a delta "function" concentrated at the correct value of p_i. Hence,

$$\begin{aligned} f(s_i \mid e) &\propto f(s_i)\Pr(e \mid s_i) \\ &= f(s_i)\int \Pr(e \mid p_i) f(p_i \mid s_i)\, dp_i \\ &\propto f(s_i) f(p_i \mid s_i) \end{aligned}$$

where the constants of proportionality depend on e but not on s_i.

This suggests a natural two-phase update algorithm for each player i. In phase one, we estimate their performance p_i by its MAP given e. This estimate has very low error in the limit of very many participants, so we can take the MAP of p_i to be its true value. In phase two, we update the posterior according to the above expression, and compute the new rating r_i as the MAP of s_i given e.

3 Performance Estimation

Since we assume the posterior on p_i to be narrowly concentrated, the MAP estimate is as good as any. Thus, we seek to maximize

$$f(p_i \mid e) \propto f(p_i)\Pr(e \mid p_i)$$

In the prior distribution, $p_i = s_i + (p_i - s_i)$ is a sum of two independent random variables. s_i has the prior distribution, whose mode is the pre-match rating r_i'; let σ_i be its standard deviation. The other term $p_i - s_i$ is a zero-centered logistic random variable; let δ be its standard deviation.

The sum of two normal random variables is another normal variable whose mean is the sum of the component means. Inspired by this fact, we make another approximation: although neither s_i nor $p_i - s_i$ are normal, we treat their sum p_i as a logistic random variable centered at r'_i. By independence of its two components, p_i has variance $\delta_i^2 = \delta^2 + \sigma_i^2$.

We'll adopt the convention that any symbol that represents a standard deviation is multiplied by $\frac{\sqrt{3}}{\pi}$ when drawn with an upper-bar. For example, $\bar{\delta}_i = \frac{\sqrt{12}}{\pi}\delta_i$. In terms of $\bar{\delta}_i$, the logistic p.d.f. takes a convenient form:

$$f(p_i) \approx \frac{2e^{2(p_i - r'_i)/\bar{\delta}_i}}{\bar{\delta}_i \left(1 + e^{2(p_i - r'_i)/\bar{\delta}_i}\right)^2}$$

Since we are working in the limit of a large number of players, we have sufficient evidence to determine p_i even after ignoring relations like $j \succ k$ which don't include i. Thus, we can imagine e to be the evidence consisting of, for each $j \neq i$, whether $j \succ i$ or $j \prec i$. Taking p_i as fixed in the following probability expressions, using the logistic cumulative density function (c.d.f.), and ignoring constants of proportionality that depend on e but not on p_i:

$$\begin{aligned}
\Pr(e \mid p_i) &= \prod_{j \succ i} \Pr(p_j > p_i) \prod_{j \prec i} \Pr(p_j < p_i) \\
&\approx \prod_{j \succ i} \frac{1}{1 + e^{2(p_i - r'_j)/\bar{\delta}_j}} \prod_{j \prec i} \frac{e^{2(p_i - r'_j)/\bar{\delta}_j}}{1 + e^{2(p_i - r'_j)/\bar{\delta}_j}} \\
&\propto \frac{e^{2p_i \sum_{j \prec i} 1/\bar{\delta}_j}}{\prod_{j \neq i} 1 + e^{2(p_i - r'_j)/\bar{\delta}_j}}
\end{aligned}$$

Taking logarithms, there exist constants C and C' such that

$$\begin{aligned}
C + \ln f(p_i \mid e) &= C' + \ln f(p_i) + \ln \Pr(e \mid p_i) \\
&\approx \frac{2}{\bar{\delta}_i}(p_i - r'_i) - 2\ln\left(1 + e^{2(p_i - r'_i)/\bar{\delta}_i}\right) + 2p_i \sum_{j \prec i} \frac{1}{\bar{\delta}_j} - \sum_{j \neq i} \ln\left(1 + e^{2(p_i - r'_j)/\bar{\delta}_j}\right)
\end{aligned}$$

To maximize this expression, differentiate it w.r.t. p_i and set the result to zero:

$$\begin{aligned}
0 &= \frac{2}{\bar{\delta}_i}\left(1 - \frac{2e^{2(p_i - r'_i)/\bar{\delta}_i}}{1 + e^{2(p_i - r'_i)/\bar{\delta}_i}}\right) + \sum_{j \neq i} \frac{2}{\bar{\delta}_j}\left(\mathbb{I}(j \prec i) - \frac{e^{2(p_i - r'_j)/\bar{\delta}_j}}{1 + e^{2(p_i - r'_j)/\bar{\delta}_j}}\right) \\
&= \sum_{j \preceq i} \frac{2}{\bar{\delta}_j}\left(\frac{1}{1 + e^{2(p_i - r'_j)/\bar{\delta}_j}}\right) - \sum_{j \succeq i} \frac{2}{\bar{\delta}_j}\left(\frac{e^{2(p_i - r'_j)/\bar{\delta}_j}}{1 + e^{2(p_i - r'_j)/\bar{\delta}_j}}\right) \\
&= -\left(\sum_{j \preceq i} \frac{1}{\bar{\delta}_j}\left(\tanh\frac{p_i - r'_j}{\bar{\delta}_j} - 1\right) + \sum_{j \succeq i} \frac{1}{\bar{\delta}_j}\left(\tanh\frac{p_i - r'_j}{\bar{\delta}_j} + 1\right)\right)
\end{aligned}$$

By monotonicity of the tanh function, we can solve for p_i by a simple binary search. If faster convergence is desired, Newton's method can be used: since $\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x)$, a Newton step is very efficient, requiring no more hyperbolic function evaluations than a binary search step.

On the second line, the terms in parentheses can be thought of as a measure of surprise at the outcomes between i and j: they are the probability of the outcomes opposite to what actually occurred, when the performance of player i is fixed to p_i. In addition to the actual outcomes which come from e, the prior hallucinates two regularizing outcomes: one in which player i wins against itself, and one in which player i loses against itself. This regularization prevents the first- and last-place players from achieving $p_i = \pm\infty$. By choosing p_i such that the sum equals zero, we are effectively saying that the total surprise from wins should equal the total surprise from losses.

Here's another intuitive interpretation: if the δ_js are all equal, this amounts to finding the performance level p_i at which one's expected rank would match player i's actual rank, after accounting for the regularizing clones of player i. With unequal δ_js, this interpretation applies with a weighted ranking system in which player j counts $\frac{1}{\delta_j}$ times.

We briefly mention a way to handle ties: simply treat them as half a win and half a loss: since the above expression subtracts 1 for each win and adds 1 for each loss, averaging the two yields an offset of 0 for ties. It's a hack, but an elegant one!

4 Belief Update

With p_i in hand, we are now ready for the second phase of the update! We seek to maximize $f(s_i \mid e) \propto f(s_i) f(p_i \mid s_i)$. Ignoring for now the distinction between the prior at round t and the posterior at round $t-1$, we see that each round multiplies our belief p.d.f. by a logistic factor and a normalizing constant. Thus, our belief will always consist of a product of p.d.f.s.

Let's be a tad more general here and allow the belief to be any normalized product of normal and logistic p.d.f.s. The normal p.d.f.s can be composed into a single normal with mean and standard deviation (μ_0, τ_0). Let R_i be the set of rounds in which player i has participated, each contributing a logistic factor with parameters (μ_k, τ_k) for $k \in R_i \cap \{1, \ldots, t\}$. Naturally, the new factor from round t will have $\mu_t = p_{i,t}$ and $\tau_t = \delta$. Again defining $\bar{\tau}_k = \frac{\sqrt{12}}{\pi} \tau_k$,

$$
\begin{aligned}
f(s_i \mid e) &\propto e^{-(s_i-\mu_0)^2/\tau_0^2} \prod_k \frac{e^{2(s_i-\mu_k)/\bar{\tau}_k}}{\left(1 + e^{2(s_i-\mu_k)/\bar{\tau}_k}\right)^2} \\
&= e^{-(s_i-\mu_0)^2/\tau_0^2} \prod_k \frac{1}{\left(e^{-(s_i-\mu_k)/\bar{\tau}_k} + e^{(s_i-\mu_k)/\bar{\tau}_k}\right)^2} \\
&= e^{-(s_i-\mu_0)^2/\tau_0^2} \prod_k \frac{1}{\left(2 \cosh \frac{s_i-\mu_k}{\bar{\tau}_k}\right)^2}
\end{aligned}
$$

Hence, there exists a constant C such that

$$
C + \frac{1}{2} \ln f(s_i \mid e) = -\frac{(s_i - \mu_0)^2}{2\tau_0^2} - \sum_k \ln\left(\cosh \frac{s_i - \mu_k}{\bar{\tau}_k}\right)
$$

To maximize this expression, differentiate w.r.t. s_i and set the result to zero:

$$
0 = -\left(\frac{s_i - \mu_0}{\tau_0^2} + \sum_k \frac{1}{\bar{\tau}_k} \tanh \frac{s_i - \mu_k}{\bar{\tau}_k}\right)
$$

Just as in the performance computation, here we have a monotonic function of s_i. To compute the post-round rating $r_i = \arg\max_{s_i} f(s_i \mid e)$, we may use binary search or Newton's method.

This time, the intuitive interpretation comes from the pre-differentiated expression. By flipping the minus signs, we obtain a minimization objective with one term corresponding to each p.d.f. factor: the normal p.d.f. becomes a familiar L^2 penalty that pushes s_i towards μ_0. The logistic p.d.f.s are more interesting: they too are penalties that push s_i towards μ_k; however, instead of a simple squared error, we have the logarithm of a hyperbolic cosine!

At small scales (specifically, when $|s_i - \mu_k| \ll \tau_k$), it acts like an L^2 term. However, at large scales (when $|s_i - \mu_k| \gg \tau_k$), it acts like an L^1 term! Recall 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. The latter feature is due to the thicker tails of the logistic distribution, compared to the normal. In practice, it makes our rating algorithm robust to outlier performances that are far below or above the rest. This seems reasonable, as contest performances have empirically been seen to have thick tails, more like the logistic than the normal.

Note that when the rating is a weighted mean, it's possible to compute the posterior rating as a function of only the prior rating and the last round's performance. This is no longer possible when using L^1 or $\ln\cosh$ penalties. Naively, it appears this method must remember the entire history of past performances: certainly, the latest rating alone is insufficient. Nonetheless, we'll see in the next section that very high precision can be achieved while retaining only a bounded history length per player, so the memory and time complexities are only affected by a constant factor.

As a final matter, we estimate the standard deviation σ_i of the posterior distribution on s_i. Since this is intractable for general products of p.d.f.s, we make an approximation based on a formula that holds in the case of a product of normal p.d.f.s:

$$\frac{1}{\sigma_i^2} = \frac{1}{\tau_0^2} + \sum_k \frac{1}{\bar{\tau}_k^2}$$

5 Uncertainty

So far, we've shown how to go from prior to posterior belief for any given round. Two gaps remain: the prior belief on the skill of a player who has never competed, and the update from posterior in one round to prior in the next.

The "newbie prior" is fairly simple: for Codeforces, I chose a normal prior with mean 1500 and standard deviation 350. Since the normal has thin tails, a normal prior will discourage the granting of extreme ratings until sufficient evidence is gathered to justify it. To more strongly discourage premature assignment of extreme ratings, a smaller standard deviation can be used.

The remaining gap concerns the update that goes on between rounds, while a player is absent from competition. There are many ways to approach this, depending on what characteristics are desired of the rating system. Typically, we don't want to adjust the mean r of a player's skill distribution, but we may want to increase the uncertainty σ to account for the fact that a player may have improved or worsened during the time in which we haven't seen them compete. Thus, we might have σ^2 increase with time at a constant rate, as if the player's skill were being slowly and randomly perturbed by a process of Brownian motion. In this model, a long-inactive player's skill has to potential to change considerably upon their return. By combining Brownian motion with reporting the lower-bound estimate $r - 2\sigma$ instead of r, we can implement a rating decay system which favors active players and gradually diminishes inactive ones.

For Codeforces, we prefer to preserve the ratings of inactive players. Furthermore, we assume that players who compete frequently are developing much faster than those who are inactive. Thus, it makes sense to increase uncertainty on a per-round basis rather than on a continuous per-time basis. Only when player is competing, their pre-round σ^2 is increased by a noise term. We also assume the round performance has a constant uncertainty $\delta = 250$. Thus, a player's σ will consistently decrease with each round participation, eventually approaching a limit $\sigma*$ which depends on the noise variance η^2. I use $\sigma^* = 100$ and derived $\eta \approx 43.64$ by solving the fixpoint equation:

$$\frac{1}{\sigma^{*2}} = \frac{1}{\sigma^{*2} + \eta^2} + \frac{1}{\delta^2} \tag{1}$$

$$\text{Therefore, } \eta^2 = \frac{1}{1/\sigma^{*2} - 1/\delta^2} - \sigma^{*2} \tag{2}$$

Note that if the prior σ is set to σ^*, we obtain a system in which the added noise η exactly compensates for the new information δ, and so σ stays constant. However, I like to use a higher prior uncertainty and report $\mu - 2(\sigma - \sigma^*)$ as a player's public rating. Without Brownian time-based noise, this doesn't cause rating decay; instead, its effect is to encourage beginners who'll typically see their rating rise from the bottom rather than decline from the middle.

A simple way to increase the uncertainty from σ^2 to $\sigma^2 + \eta^2$ between rounds is to multiply the standard deviation of each factor in the posterior skill distribution by the same value $1 + \frac{\eta^2}{\sigma^2}$. Intuitively, we can think of each factor as a "measurement" of the current skill $s_{i,t}$. This noise operation, then, simply reduces our confidence in outdated measurements.

After participating in a large number of rounds, a player's oldest measurements will have such large τ_k that, for all reasonable skill levels s_i, $|s_i - \tau_k|$ will be very small. In this regime, a logistic factor with standard deviation τ_k behaves approximately like a normal distribution with standard deviation $\bar{\tau}_k$. Thus, old measurements can be fused into the normal prior and discarded without losing much accuracy. In this manner, we only need to retain a bounded number of the most retain logistic factors for each player.

The parameter and prior belief initializations are summarized in Algorithm 1, while the actual Elo-R update is summarized in Algorithm 2.

Algorithm 1 *init*()

```
σ* := 100
δ := 250
η² := 1/(1/σ*² − 1/δ²) − σ*²
for all players i do
    Initialize the list ⟨(μ_{i,-}, τ_{i,-})⟩ with one element (1500, 350)
    r_i := 1500
    σ_i := 350
    δ_i := √(σ_i² + η² + δ²)
end for
```

Algorithm 2 *update*()

for all match participants i **do**
  $p_i :=$ SOLVE $\sum_{j \preceq i} \frac{1}{\delta_j} \left(\tanh \frac{p_i - r_j}{\delta_j} - 1 \right) + \sum_{j \succeq i} \frac{1}{\delta_j} \left(\tanh \frac{p_i - r_j}{\delta_j} + 1 \right) = 0$
end for
for all match participants i **do**
  for all k **do**
    $\tau_{i,k} := \tau_{i,k} \sqrt{1 + \eta^2 / \sigma_i^2}$
  end for
  Append (p_i, δ) to the list $\langle (\mu_{i,\_}, \tau_{i,\_}) \rangle$ of belief factors
  if $\langle (\mu_{i,\_}, \tau_{i,\_}) \rangle$ contains too many elements **then**
    $\mu_{i,0} := \left(\tau_{i,0}^2 \mu_{i,1} + \tau_{i,1}^2 \mu_{i,0} \right) / \left(\tau_{i,0}^2 + \tau_{i,1}^2 \right)$
    $\tau_{i,0} := \tau_{i,0} \tau_{i,1} / \sqrt{\tau_{i,0}^2 + \tau_{i,1}^2}$
    Remove $(\mu_{i,1}, \tau_{i,1})$ from $\langle (\mu_{i,\_}, \tau_{i,\_}) \rangle$ and shift indices accordingly
  end if
  $r_i :=$ SOLVE $\frac{\mu_{i,0} - r_i}{\tau_{i,0}^2} + \sum_k \frac{1}{\tau_{i,k}} \tanh \frac{\mu_{i,k} - r_i}{\tau_{i,k}} = 0$
  $\sigma_i := 1 / \sqrt{\sum_k 1 / \tau_{i,k}^2}$
  $\delta_i := \sqrt{\sigma_i^2 + \eta^2 + \delta^2}$
end for

6 Properties (TODO: everything from here onwards needs serious rewriting)

First, we discuss some properties that Elo-R has in common with the published systems of Codeforces and TopCoder, as well as the classics Elo and Glicko. All of these systems propagate belief changes forward in time, never backward. This approach is simple, efficient, and has the benefit of never retroactively changing ratings from the past, nor the ratings of player who are not actively competing. Unfortunately, we do lose some accuracy and convergence speed compared to methods that propagate approximate updates more globally. Elo-R and Glicko converge to the right results a bit faster than the others, by including an uncertainty parameter that starts high for new players.

The two-phase approach of Elo-R is a bit unique, in that it's not memoryless (unless the memory is set to merge all the way down to a length of 1). Rating changes depend not only on the current rating and uncertainty, but on a list of recent performance values. Thus, we cannot make the same guarantees as Codeforces [**?**]. This is the price of a robust system: it's impossible to identify and eliminate outliers if we don't remember their values! Nevertheless, we have some analoguous guarantees:

- The rating is a monotonic function of the list of past performances. Thus, unlike on Topcoder [**?**], a situation will never arise where we would wish to have scored less.
- If you swap the order of two performances in the list, the rating goes up if the better performance moves forward in time, and down if the better performance moves backward. This follows from the fact that newer performances have smaller uncertainty, since uncertainties don't depend on the value of any performance.

- The performance p_i is measured in the same units as rating and has the property that, within a given contest, a higher ranking contestant always has higher p_i than a lower ranking one. In case of ties, the contestant with higher rating r_i also has (slightly) higher p_i.

The code and ratings of real Codeforces members as computed by Elo-R are available at https://github.com/EbTech/EloR. Original Codeforces ratings are at http://codeforces.com/ratings. One striking difference is massive inflation in the Codeforces system. Gennady Korotkevich, best known by his competitive programming handle "tourist", has been the reigning world champion for years. Toward the end of 2011, his rating reached a new ceiling of about 2700 according to both systems. However, as of this writing, his rating on Elo-R has increased by about 300 additional points, while on Codeforces it increased by almost 900. To get a sense of the magnitude of this change, 900 points is the difference between an average member and a Grandmaster! Indeed, most of the variance in the Codeforces system is concentrated at the top, with much smaller rating differences between beginner and intermediate members. This is caused by certain ad hoc elements of the system that are not founded on any rigorous model.

This paper will not evaluate the predictive accuracy of Elo-R; my experience with it suggests it does better than the other local methods listed above, but it is possible to do better with global methods. It's difficult to do a fair evaluation because it's not clear what exactly some of these models are trying to predict, besides the qualitative assertion that players with higher ratings should win more often. For example, Elo-R might be judged on the log-likelihood of observed match results, according to its own belief model. However, the joint likelihood is difficult to compute, and many of the other systems lack a corresponding belief model. One reasonable approach would be to approximate the likelihood as we did when estimating p_i, and use cross-validation to optimize the parameters of each rating system according to this criterion. Such evaluations will remain open to future investigation. Instead, let's focus on a unique feature of Elo-R that's absent in the Codeforces system, and arguably in TopCoder as well.

6.1 Random Trash

If all but the last M matches for each player are merged, and all binary searches are performed to an accuracy of ϵ, then a match with N players is processed in $O((N^2 + NM)\log(1/\epsilon))$ time. The ratings accumulate $O(\epsilon)$ numerical error per match, and likely a lot less in the long run due to statistical averaging. It's easy to see that the loops and summations are highly parallelizable.

While this equation is not memoryless, all τ_k corresponding to old performances will be very large, so their terms will behave like their linear approximations. Any sum of linear terms is itself linear, so we can merge the old terms. $p_{i,0}$ here is the weighted sum of the old performances, and the merged weight $1/\tau_0^2$ is the sum of the weights of these old performances. k now ranges only over indices of the new, unmerged, performances:

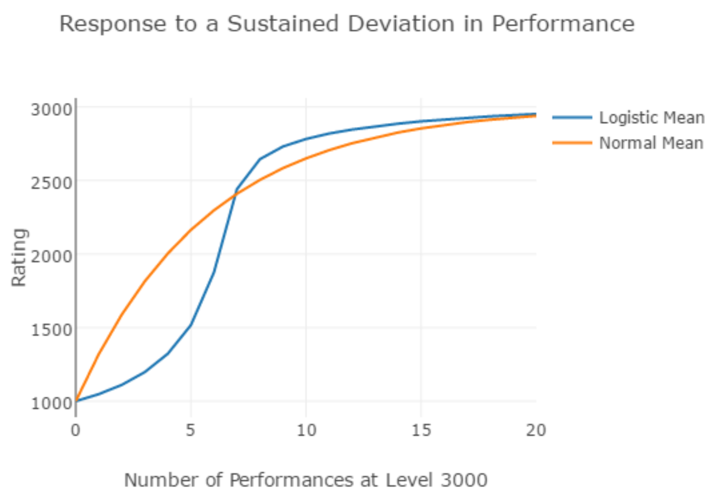
$$\frac{p_{i,0} - r_{i,M}}{\tau_0^2} + \sum_k \frac{1}{\tau_k} \tanh \frac{p_{i,k} - r_{i,M}}{\tau_k} = 0 \tag{3}$$

As a result, we only need to remember a small number of recent performances. Instead of growing without bound with the size of the history, the number of terms in the sum can be set to a small constant while merging the rest. If we decide to remember zero performances, it reverts to the linear memoryless method.

6.2 Robustness

Imagine a player who performs very consistently over a long period of time, repeatedly achieving $p_i = 1000$ until convergence. Now, perhaps as a result of attending an intensive training camp in Petrozavodsk, their skill changes dramatically. From this point on, they consistently achieve $p_i = 3000$.

How does each rating system respond to the first such surprise occurrence? Elo-R treats the new result as a fluke, an outlier that ought to be ignored. The player gains 48 points; as a result of the parameters we set, this is the maximum possible for an experienced player as $p_i \to \infty$. In practice, ratings may change by more than 48, as the maximum depends on existing fluctuations in their history; here we're looking at the extreme example of a player with a history of always performing at exactly $p_i = 1000$.



Had we tried to perform outlier reduction in a memoryless fashion, we would continue to increase the rating by 48 per match, oblivious to the possibility that the player truly did experience a sudden improvement. In Elo-R, the outlier status of a performance is treated as tentative. If later matches support the hypothesis of having improved, the rating will increase by an additional 63 points, followed by over 100 points in each of the third and following matches, as plotted by the blue curve above.

After six consecutive matches with $p_i = 3000$, the rating is 1875 and very unstable (even though σ_i is unchanged!). The system is no longer sure which to trust: the extensive history at level 1000, or the smaller number of recent matches at level 3000. Depending on what comes next, the player's rating can very quickly fall toward 1000 or rise toward 3000. However, note that in either case, the change will not overshoot, say to 5000, unless enough new evidence is accumulated at that level. As the $p_i = 3000$ streak continues, the seventh match on the blue curve jumps by a whopping 566 points. As the player's rating converges to 3000, the old $p_i = 1000$ data acquires outlier status, thus speeding convergence.

In contrast, while a system such as Codeforces does not compute p_i values in quite in the same way, we can obtain a good approximation by removing outlier reduction from Elo-R, effectively treating the performances to be averaged as normal instead of logistic measurements. This makes the system effectively memoryless, since it turns out that each match simply moves the rating about 16% closer to the new p_i value, independent of the history. With this change, we obtain the orange curve, which jumps a whopping 320 points at the very first performance. Indeed, there is no limit: if you could find players whose ratings are extremely high, and beat them even once, your rating would take arbitrarily large leaps.

Note that this is not quite true of TopCoder, which incorporates a hack that caps the maximum rating change: if TopCoder's update formula demands too large a change, the cap kicks in. In contrast, Elo-R's cap is a natural and smooth consequence of its update formula and is sensitive to whether a change is charting new territory, or merely confirming a plausible hypothesis. TopCoder does attempt to make the magnitude of its updates sensitive to the amount of fluctuation in a player's history, using a volatility measure, but this measure does not account for the direction of the changes, resulting in the non-monotonicity flaw mentioned above.

Notwithstanding arguments that a high rating ought to properly be earned over multiple matches rather than a single fluke, the other danger is that these observations also hold in reverse: one bad day on Codeforces can seriously damage one's rating and negate several rounds of steady progress. By using heavy-tailed logistic distributions everywhere, Elo-R understands that unusually high or low performances do occasionally occur, and one round in isolation is never a reliable signal.

Interestingly, despite the slow start, the blue curve ultimately converges faster than the orange one. Since Elo-R uses its memory to dynamically adapt its view of potential outliers, it overtakes the orange curve as soon as new evidence outweighs the old hypothesis!

7 Conclusions

This paper introduces the Elo-R rating system, which is in part a generalization of the two-player Glicko system, allowing an unbounded number of players. It assumes the players' performances, while potentially hard to quantify directly, can be ranked in a total order. As a natural consequence of some technical modeling assumptions, Elo-R is far more robust to atypical performances than any alternative known to the author.

Applications include many types of sports and video games, as well as programming contests, which presently rely on less rigorously derived models and hacks. The modeling assumptions are best suited to events where the players have minimal targeted interactions against one another, and instead compete individually to score better than rival players in an ongoing array of challenges. For instance, suppose we want to measure a person's skill in traversing obstacle courses, where the course design changes weekly. Completion times are only meaningful on a single course. However, if we treat each course as a match in Elo-R, it becomes possible to quantify and compare the skills of individuals, even if they have never completed the same course together.

Acknowledgements

I'm grateful to Daniel Sleator and Dougal Sutherland for discussions that contributed to the inspiration and preparation of this work. In addition, I'd like to thank Mike Mirzayanov for providing a useful case study by publishing the Codeforces rating algorithm.

References