An Elo-based rating system for TopCoder SRM

Fred Batty and Dmitry Kamenetsky
fbatty@gmail.com, dkamenetsky@gmail.com
March 3, 2020

Abstract
We present an Elo-based rating system for programming contests. We justify a definition of performance using the logarithm of a player’s rank. We apply the definition to rating TopCoder SRM. We improve the accuracy, guided by experimental results. We compare results with SRM ratings.

1 Introduction
SRM (single round matches) are regular programming contests organized by TopCoder [1] since 2001. TopCoder has developed its own rating system, which has been used throughout its 19 year history [2]. Various shortcomings of SRM ratings have been documented on TopCoder forums and elsewhere. Our purpose here is not to discuss these issues, but to provide a concrete proposal to remedy them. Players have sometimes asked: What would SRM ratings be if they were Elo-based? We would like to obtain a reasonable answer to this question.

There isn’t a standard method for applying Elo ratings to rounds of more than two players. We could consider a ranking of players as the set of results between each pair of players. However, such results are not independent from each others: a ranked result is the product of a single performance by each player. Instead, we will consider the ranking as a tournament. From desirable properties, we will deduce a formula for performance in ranked games.

We then use this formula to specifically rate SRM. Our goal is to more accurately predict the players’ performances after each round.

2 Performance
2.1 Rank performance
Let $RP(n,r)$ be the performance of a player ranked $r$ in a round of $n$ players. We consider the ranking as an elimination tournament, and count the number of wins.

If the tournament has $2^n$ players, the winner must win $n$ 1-1 rounds, so we have:

$$RP(2^n, 1) = n$$  \hspace{1cm} (1)

The player ranked 2 has one less win than rank 1:
Multiplying both \( r \) and \( n \) by any \( k > 0 \) does not change the number of wins:

\[
RP(n, r) = RP(kn, kr)
\] (3)

With these constraints, we find the only solution is:

\[
RP(n, r) = \log_2 n - \log_2 r
\] (4)

### 2.2 Expectations

We use the standard Elo formula for expectations [3]. A player with rating \( R_i \) is expected to outperform a player with rating \( R_j \) with probability:

\[
WP(R_i, R_j) = \frac{1}{1 + 10^{(R_j - R_i)/400}}
\] (5)

As with SRM ratings, the rating of new players is initially 1200.

### 2.3 Ties

In programming contests like SRM, ties generally reflect a limitation of the problem set or scoring, rather than an unexpected performance of the tied players. We would like ties to not affect the ratings.

We experimented with several accounting methods. We find the most accurate is to split the ties equally in actual and expected ranks, .5 in each. Slightly less accurate is to compute expected ranks regardless of ties, then split the tied ranks as is expected.

We now split the ties equally.

### 2.4 Relative performance

We have the results of a round, which may include multiple divisions and ties. We consider the results of each division separately. The result of a round is a list of scores \( S \), where \( s_i \) is the score obtained by player \( i \).

We compute rank and expected rank for each player \( i \):

\[
r_{\text{ties}} = \frac{1}{2} |\{ s_j \in S : j \neq i, s_j = s_i \}| \\
r = 1 + |\{ s \in S : s > s_i \}| + r_{\text{ties}}
\] (6)

\[
\hat{r} = 1 + \sum_{j : s_j \neq s_i} WP(R_j, R_i) + r_{\text{ties}}
\]

The relative performance of a player in the round is the difference of actual and expected rank performance:

\[
P = RP(n, r) - RP(n, \hat{r})
\] (7)

This can be written as:

\[
P(\hat{r}, r) = \log_2 \hat{r} - \log_2 r
\] (8)

The performance \( P \) equals a number of wins above or below expectations in a tournament of appropriately matched players.
2.5 Properties

Figure 1: Rank performance

Rank performance is convex (Figure 1). The sum of performances of a set of ranks is maximal if the ranks are distinct, and minimal if the ranks are uncertain or tied. Having split ties equally in actual and expected ranks, the expected ranks are at least as tied as the actual ranks. This ensures the sum of performances in a round is positive or zero. We have:

\[ \sum P \geq 0 \quad (9) \]

Figure 2: Relative performance

The rank is expected in logarithm (Figure 2). A player’s performance may average 0 in several ways:

\[ P(a, b) + P(b, c) + P(c, a) = 0 \]
\[ P(r, rx^k) = k \cdot P(r, rx) \quad (10) \]
\[ P(\hat{r}, r) = P(k\hat{r}, kr) \]

We will compute \( \Delta R \propto P \), preserving these properties.
2.6 Accuracy
We define the prediction error for expected and actual ranks \((\hat{r}, r)\):

\[
E(\hat{r}, r) = |P| = |\log_2 \hat{r} - \log_2 r|
\]  
(11)

Our primary accuracy metric is the average error for all participants in all rated rounds.

3 Proposed rating system
We have the performances of each player in an SRM. We would like to compute rating changes \(\Delta R\) which better predict future performances.

3.1 Initial factor
With a prior of 1 : 1, a performance \(P\) is outperforming expectations by a factor 1 : 2\(^P\). A rating difference \(\Delta R\) is expecting a better performance by a factor 1 : 10\(\Delta R/400\). We can convert the performances in rating units:

\[
\Delta R = \frac{400}{\log_2 10} P = K_0 \cdot P
\]

\(K_0 \approx 120\)  
(12)

If we expected the same performances the next round, and had no other information, this would be a reasonable \(\Delta R\). Thus we consider \(\Delta R \propto P\) as Elo-based or ‘Elo’.

3.2 Fixed K

![Figure 3: Fixed K](image)

Here we compute \(\Delta R = K \cdot P\), with K minimizing the error. We find the most accurate choice is \(K = 65\) (Figure 3). As we add factors in \(\Delta R\), we automatically adjust \(K\) to the most accurate.

3.3 Weight factor
A player’s prior rating should weigh in \(\Delta R\) according to the player’s experience. Let \(W\) be the weight, such that \(\Delta R \propto 1/W\), and \(NR\) the round number for the player, starting with 1. We experimented with
several possibilities, and find the best results with $W = \sqrt{NR}$. Figure 4 shows choices of $W = NR^\alpha$. Thus we choose $W = \sqrt{NR}$.

$$\Delta R = K \cdot \frac{P}{W}$$  \hspace{1cm} (13)

$K = 182$, the error is .7562.

### 3.4 Variance factor

A player’s performance has variance for various reasons, not necessarily predicting future performance. We compute the derivative of a player’s expected performance per change in rating:

$$P = -\log_2 \hat{r}$$

$$\frac{dP}{dR} = -\frac{1}{\ln 2} \frac{d\hat{r}}{dR}$$

$$\hat{r} = 1 + \sum_{j \neq i} w_j$$  \hspace{1cm} (14)

$$w_j = WP(R_j, R)$$

We write the expected rank 1 as the loss:win ratio relative to the player’s current rating:

$$\hat{r}(dR) = 10^{\frac{-dR}{400}} + \sum \frac{1}{1 + \frac{10(R + dR - R_j)}{400}}$$

$$\frac{d\hat{r}}{dR} = -\frac{\ln 10}{400} \left(1 + \sum w_j(1 - w_j)\right)$$

$$\frac{dP}{dR} = \frac{1}{K_0} \cdot \frac{1 + \sum w_j(1 - w_j)}{1 + \sum w_j}$$  \hspace{1cm} (15)

In units of performance, we have:

$$P' = \frac{1 + \sum w_j(1 - w_j)}{1 + \sum w_j}$$  \hspace{1cm} (16)

Extrapolating linearly, we can solve $P = 0$ with $\Delta R = K_0 \cdot \frac{P}{P'}$.

- $P' \leq 1$, so $\Delta R = K_0 \cdot P$ may be conservative.
- $P' \to 1$ when $\hat{r} \to 1$. No extrapolation is possible.
• $P' \to \frac{1}{n}$ when $\hat{r} \to n$. Here $P \neq 0$ has bits of precision, hence more likely predicts future performance.

We compute $\Delta R$ in a direction accounting for $P'$ and a multiplier $C$:

$$\Delta R \propto \frac{P}{1 + C \cdot P'}$$  

(17)

Figure 5: Choice of $C$

We find the most accurate $C \approx 4$ (Figure 5). Thus we choose $C = 4$.

We define the variance factor $V = 1 + C \cdot P'$. We now have:

$$\Delta R = \frac{K}{V} \cdot \frac{P}{W}$$  

(18)

$K = 415$, the error is $.7536$.

### 3.5 Maximum factor

An unexpected performance predicts future performance less reliably than a consistent performance. The ratings gain accuracy if we limit the magnitude of $P$ to a maximum $M$ using a sigmoid. A sigmoid preserves symmetry, and exactly linearity of performance around 0. We define the adjusted performance $PA$. We find the best results with:

$$PA = \frac{P}{1 + \frac{|P|}{M}}$$  

(19)

We find the most accurate $M \approx 6.75$ (Figure 6). Thus we choose $M = 6.75$. The rating change for each player is now:

$$\Delta R = \frac{K}{V} \cdot \frac{PA}{W}$$  

(20)

$K = 600$, the error is $.7513$.

### 3.6 Natural inflation

We have computed $\Delta R \propto P$ which make the ratings more accurate after a round. Each player is expected a performance 0, thus has an expected $\Delta R = 0$. The ratings are stable in expectation.

Because performances have a positive sum, more rating is won by the outperforming players than is lost by the underperforming players. The
FIGURE 6: Choice of M

ratings have net inflation. Because the players gain experience during a round, the players on average have better performances after the round. Thus inflation better predicts future performance than deflation. Because we minimized the prediction error, the average $\Delta R$ should approximately predict the next performances of participants relative to non-participants. We define this rate of inflation as natural inflation.

For comparison with SRM ratings, we consider natural inflation as stable. We refer to this Elo-based implementation as ‘Elo’ in our results.

‘Elo’: $K = 600; C = 4; M = 6.75$

3.7 Stability

We have $\Delta R \propto P$, predicting the players’ relative performances. Now we would like to estimate the performances over time, such that players with stable performances have stable ratings. To maintain relative accuracy, we will not adjust our current parameters.

3.8 Performance bonus

As long as $\Delta R \propto P$ exactly, the expected rating change of any player is zero. However, the expected performance in a round is a better performance than not participating. Players having practiced already have better performances before the round. Thus $\Delta R = 0$ in expectation predicts future performance less accurately than $\Delta R > 0$.

We adjust the expectation to expect inflation, as if the ratings increased. We choose a parameter $B = \Delta R$, then add the difference in expected performance to each player’s performance:

$$\Delta P = \frac{B}{K_0} \cdot P'$$  \hspace{1cm} (21)

We find $B \approx 6$ (Figure 7), and little accuracy can be gained from this parameter alone.

3.9 New players

So far we have a constant rating $R_0 = 1200$ for new players. However, the performance of new players is not constant. As the performance of
existing players improves, SRMs become more difficult. This raises the barrier to entry.

Before participating, potential players have opportunities to practice on recent rounds. Some may be experienced players coming from other platforms. Thus the performance of new players improves over time. Thus we adjust the initial rating for inflation.

- We choose a parameter $N$, the increase in $R_0$ per 100 rounds.
- After each round, adjust $R_0$ by $\frac{N}{100}$.
- Simultaneously, adjust $B$ for accuracy.

We find most accurately $N = 63, B = 27$ (Figure 8).

Thus, our parameters estimating a stable performance are:

'Elo2': $N = 63; B = 27$
4 Results

We first compare our ‘Elo’ implementation to SRM ratings. Table 1 shows the average computed ∆R, performances, and prediction error, using our definitions.

- The first row is our primary metric.
- The players by experience.
- Existing players, in each division.

| Players | ratings | ∆R | perf | err |
|---------|---------|----|------|-----|
| All     | 790127  | 19.5 | -20.8 | 0.224 | 0.298 | 0.7513 | 0.8301 |
| First round | 77818  | 65.3 | -186.2 | 0.391 | -0.373 | 0.8019 | 1.0766 |
| 2-7 rounds | 202858 | 32.5 | -25.1 | 0.285 | 0.244 | 0.6649 | 0.7105 |
| 8-24 rounds | 217893 | 12.2 | 10.9 | 0.239 | 0.511 | 0.7454 | 0.8204 |
| 25-74 rounds | 195019 | 5.1 | 4.5 | 0.171 | 0.395 | 0.7671 | 0.8263 |
| 75-199 rounds | 84825  | 0.8 | -0.7 | 0.050 | 0.284 | 0.8701 | 0.9093 |
| 200+ rounds | 11714   | -0.5 | -2.0 | -0.043 | 0.241 | 0.8965 | 0.9347 |
| Existing | 712309  | 14.5 | -2.7 | 0.206 | 0.372 | 0.7458 | 0.8032 |
| Division 1 | 388024 | 11.7 | -1.8 | 0.169 | 0.241 | 0.6930 | 0.7230 |
| Division 2 | 324285 | 17.8 | -3.7 | 0.251 | 0.528 | 0.8090 | 0.8992 |
| D1 H1   | 194004  | 34.1 | 52.1 | 0.636 | 0.796 | 0.9705 | 1.0406 |
| D1 H2   | 194020  | -10.7 | -55.8 | -0.298 | -0.314 | 0.4155 | 0.4054 |
| D2 H1   | 162141  | 58.7 | 55.3 | 0.881 | 1.348 | 1.1311 | 1.3831 |
| D2 H2   | 162144  | -23.2 | -62.7 | -0.379 | -0.291 | 0.4869 | 0.4153 |

Table 1: Player statistics, Elo : SRM

Because SRM ratings use a different definition of performance, we include results using independent metrics. Each round, we compute rank correlation statistics [4]:

- Kendall’s τ
- Spearman’s ρ

For each metric, we compute the fraction of rounds where ‘Elo’ better predicted the result than SRM ratings, splitting ties equally. Table 2 shows the percentages.

Tables 3, 4 and 5 compare our ‘Elo2’ and ‘Elo’ implementations.
| Rounds          | #   | τ   | ρ   | err  |
|----------------|-----|-----|-----|------|
| All            | 1950| 87.4| 87.1| 89.2 |
| Division 1     | 1196| 79.8| 79.6| 83.1 |
| Division 2     | 754 | 99.4| 98.9| 98.9 |
| 2-16 players   | 151 | 48.3| 48.7| 54.0 |
| 17-99 players  | 204 | 65.9| 65.4| 69.1 |
| 100-199 players| 337 | 86.5| 85.9| 91.5 |
| 200-399 players| 437 | 92.7| 92.7| 94.7 |
| 400-599 players| 310 | 95.5| 95.5| 96.1 |
| 600-799 players| 268 | 98.1| 98.1| 96.3 |
| 800+ players   | 242 | 99.2| 97.9| 98.3 |

Table 2: Round statistics, % Elo : SRM

| Players      | ratings | ΔR   | perf | err   |
|--------------|---------|------|------|-------|
| All          | 790127  | 25.7 | 0.217| 0.7497|
| First round  | 77818   | 90.6 | 0.426| 0.7997|
| 2-7 rounds   | 202858  | 40.4 | 0.280| 0.6649|
| 8-24 rounds  | 217893  | 15.6 | 0.217| 0.7442|
| 25-74 rounds | 195019  | 7.3  | 0.156| 0.7650|
| 75-199 rounds| 84825   | 2.4  | 0.050| 0.8662|
| 200+ rounds  | 11714   | 0.6  | -0.033| 0.8924|
| Existing     | 712309  | 18.6 | 0.194| 0.7443|
| Division 1   | 388024  | 14.9 | 0.163| 0.6915|
| Division 2   | 324285  | 22.9 | 0.232| 0.8074|
| D1 H1        | 194004  | 37.0 | 0.622| 0.9668|
| D1 H2        | 194020  | -0.2 | -0.038| 0.4163|
| D2 H1        | 162144  | 58.7 | 0.852| 1.1199|
| D2 H2        | 162144  | -17.3| -0.388| 0.4949|

Table 3: Player statistics, Elo2 : Elo

| Rounds          | #   | τ   | ρ   | err  |
|----------------|-----|-----|-----|------|
| All            | 1950| 58.4| 57.3| 64.6 |
| Division 1     | 1196| 59.5| 58.3| 63.8 |
| Division 2     | 754 | 56.6| 55.7| 65.8 |
| 2-16 players   | 151 | 52.6| 54.0| 59.3 |
| 17-99 players  | 204 | 56.6| 54.4| 60.3 |
| 100-199 players| 337 | 57.7| 56.8| 58.9 |
| 200-399 players| 437 | 62.2| 61.1| 65.7 |
| 400-599 players| 310 | 61.1| 61.0| 66.1 |
| 600-799 players| 268 | 59.0| 57.8| 69.8 |
| 800+ players   | 242 | 53.3| 50.4| 69.8 |

Table 4: Round statistics, % Elo2 : Elo
### Table 5: Statistics, each parameter

| Rating | err  | $\Delta R$ | R   |
|--------|------|------------|-----|
|        |      | $\mu$ | $\sigma$ | max | init | median | max |
| SRM    | 0.8301 | -20.8 | 130 | 900 | 1200 | 1043 | 3923 |
| initial| 0.7908 | 22.2  | 142 | 1202 | 1200 | 1301 | 4663 |
| K      | 0.7784 | 14.7  | 75  | 642 | 1200 | 1258 | 3684 |
| W      | 0.7562 | 18.5  | 102 | 1762 | 1200 | 1327 | 3734 |
| C      | 0.7536 | 19.6  | 103 | 2417 | 1200 | 1341 | 3684 |
| Elo    | 0.7513 | 19.5  | 102 | 1548 | 1200 | 1369 | 3709 |
| B      | 0.7511 | 21.5  | 104 | 1581 | 1200 | 1384 | 3801 |
| Elo2   | 0.7497 | 25.7  | 105 | 1591 | 1970 | 2095 | 4468 |

#### 5 Conclusion

Our ‘Elo’ implementation generally better predicts the players’ relative performances than SRM ratings. The ranks are also better predicted, with predictions improving with the number of players. Our ‘Elo2’ adjustments improve stability and slightly improve accuracy.

Our primary metric considers all the players’ performances in all SRM. The predictions are empirically accurate, on average, but not necessarily precise for any player or at any time.

We include source code and charts in appendix. Other results are posted on our web site [5].

#### 6 Acknowledgements

We would like to thank Ivan Kazmenko for reviewing this paper and helpful comments.

#### References

[1] Topcoder. [https://www.topcoder.com/about](https://www.topcoder.com/about).

[2] Topcoder: Algorithm competition rating system. [https://community.topcoder.com/tc?module=Static&d1=help&d2=ratings](https://community.topcoder.com/tc?module=Static&d1=help&d2=ratings).

[3] Wikipedia: Elo rating system. [https://en.wikipedia.org/wiki/Elo_rating_system](https://en.wikipedia.org/wiki/Elo_rating_system).

[4] Wikipedia: Rank Correlation. [https://en.wikipedia.org/wiki/Rank_correlation](https://en.wikipedia.org/wiki/Rank_correlation).

[5] [http://tc.eloranked.com](http://tc.eloranked.com).
Appendix A

//
// tc.eloranked.com/EloSRM.h
// (c) 2019-2020 Batty, Kamenetsky
//
#include <math.h>
#include <vector>
#include <algorithm>

namespace EloSRM {

const double EloScale = M_LN10 / 400;
const double K0 = 400 * M_LN2 / M_LN10;
const double R0 = 1200; // initial rating
double R1 = R0; // adjusted for inflation
const double N = .63; // per round
const double K = 600; // initial K
const double C = 4; // coeff P'
const double M = 6.75; // max perf
const double B = 27 / K0; // perf bonus

struct Player {
    int numRatings = 0;
    double rating = R1;
};

struct Result {
    Player* player;
    int div;
    double score;
    double deltaR;
};

inline double winP(double ri, double rj) {
    double s = (rj - ri) * EloScale;
    return 1 / (1 + exp(s));
}

void rateDivision(Result* results, int n) {
    #pragma omp parallel for
    for (int i = 0; i < n; i++) {
        Player* pi = results[i].player;
        double si = results[i].score;
        double ri = pi->rating;
        double erank = 1, arank = 1;
        double mu = 1, var = 1;
        for (int j = 0; j < n; j++) {
            double sj = results[j].score;
            double rj = results[j].player->rating;
            double wp = winP(rj, ri);
            mu += wp;
            var += wp * (1 - wp);
            if (si == sj) {
                erank += .5;
                arank += .5;
            } else {
                erank += wp;
                arank += si < sj;
            }
        }
        double err = log(erank / arank) * M_LOG2E;
        double err1 = var / mu;
        double perf = err + B * err1;
        double pa = perf * M / (M + abs(perf));
        double wf = sqrt(1 + pi->numRatings);
        double vf = 1 + C * err1;
        double w = wf * vf;
        results[i].deltaR = K * pa / w;
    }
}

void rateRound(std::vector<Result>& results) {
    int n = (int)results.size();
    std::sort(results.begin(), results.end(), [](auto& r0, auto& r1) { return r0.div < r1.div; });
    for (int i = 0; i < n; i++) {
        double[]
        for (int i = 0; i < n; i++) {
            Player* pi = results[i].player;
            pi->numRatings++;
            pi->rating += results[i].deltaR;
        }
    }
    R1 += N;
}
}
Appendix B

Figure 9: Stability