Using Item Response Theory To Rate (Not Only) Programmers

Michal Forišek*

April 15, 2009

Abstract. We show how Item Response Theory (IRT) can be used to define a new type of rating system, one that is especially suitable for programming competitions (and other types of competitions where difficulty of competitions varies between rounds). We show some useful theoretical properties of this rating system, including the ability to argue about hardness of past competition tasks, and about the precision of contestants' skill estimates. We also define an objective method of comparing different rating systems. In the final section of the paper we apply our methods on real competition data.

Keywords. item response theory, ranking, rating, programming competitions

1 Overview

In this section we provide an overview of topics relevant to this article:
– Research of scoring and ranking in programming competitions.
– Research in the area of rating systems.
– Item Response Theory.

1.1 Programming competitions

For a few years the International Olympiad in Informatics (IOI) community is concerned about the accuracy of the testing, scoring and ranking process. Several publications that research various aspects of this problem include (Cormack, 2006, Cormack et al., 2006, Forišek, 2004, Forišek, 2006, Opmanis, 2006, van Leeuwen, 2005, Verhoeff, 2006, Yakovenko, 2006).

A publication particularly relevant to the topic of this paper is (Kemkes et al., 2006) where Kemkes et al. use Item Response Theory to analyze scoring used at IOI 2005, and in particular the impact of the proportion of "easy" and "hard" test cases on the relevancy of the competition results. Based on the results of the analysis, new scoring methods with better discrimination are suggested.

(We would like to note that similar research has recently been conducted for other competitions as well, for example see (Gleason, 2008) for an analysis of two mathematical competitions.)

However, we would like to note that while these publications use IRT only passively, as a tool for analysis of tasks only. In parts of this paper, we will use IRT as an active tool – not only to rate tasks and participants, but also to make estimates and predictions.

*forisek@dcs.fmph.uniba.sk, Department of Informatics, Faculty of Mathematics, Physics and Informatics, Comenius University, Mlynská dolina, 842 48 Bratislava, Slovakia

1.2 Rating systems

The idea of a rating system has been studied for several decades. Competitiveness is a part of our human nature, and when there is competition, there is the need to rate and/or rank the competitors. Also, the need for rating is often encountered in educational systems, and many other areas.

In this context, *rating* means assigning a vector of properties to each subject (i.e., contestant), and *ranking* means arranging the subjects into a linear order according to some set of criteria. Usually, ranking is the goal, and rating represents possible means to achieve this goal.

The first rating systems were reward-based: A good performance was rewarded by granting the subject rating points. The main advantage of these rating systems was their simplicity. Due to this reason, such rating systems are still used in many popular sports, such as tennis and Formula 1.

These systems are usually designed so that their discrimination ability is highest among the top subjects, and rapidly decreases as the skill of the subjects decreases. Moreover, the reward values are usually designed ad-hoc, and the rating systems usually have little to no scientific validity.

One of the first areas to adopt a more scientific-based rating (and thus ranking) system was chess. The United States Chess Federation (USCF) was founded in 1939. Initially, USCF used the Harkness rating system (Harkness, 1967). This was a reward-based system determined by a table that listed the reward size as a simple function of the difference between the players' current ratings.

After discovering many inaccuracies caused by this system, a new system with a more solid statistical basis was designed by Arpád Elő, and first implemented in 1960. For details of this rating system see (Elo, 1978).

The Elo rating system was based on the following set of assumptions:

- The performance of a player in a game is a normally distributed random variable.
- For a fixed player, the mean value of his performance remains constant (or varies negligibly slowly).
- All players' performances have the same standard deviation.

The importance of the Elo model lies in bringing in the statistical approach. Even though subsequent research showed that each of these assumptions was flawed, and accordingly changes to the rating system were made, the currently used rating system in chess is still called the Elo rating system in Arpád Elő's honor.

An excellent overview of various rating systems in chess can be found in (Glickman, 1995). For a discussion of some problems of the currently used rating system, see (Sonas, 2002).

The next important step in the history of rating systems was the Glicko system (Glickman, 1999) that, in addition to calculating ratings, calculated also the rating deviation for each of the players – a value that measures the accuracy of the given player's rating. This system was later amended into the Glicko-2 system (Glickman,) that also computed the rating volatility – a value that measures how consistent a player's performances are. Glickman's rating systems were designed to handle situations where each event is a comparison of a pair of subjects.

In recent years, Glickman's models were generalized to handle situations when events involve more than two subjects. Notable advances in this direction include Microsoft's recently published TrueSkill^{TM} ranking system (Herbrich and Graepel, 2006, Herbrich et al., 2007, Dangauthier et al., 2008a), and TopCoder's rating algorithm (TopCoder Inc., 2008).

All these rating systems are incremental: given the previous estimates and new results, they compute a new set of estimates using Bayesian inference. Hence they are usually called Bayesian rating systems.

1.3 Item Response Theory

In practice we often encounter a situation when a variable we might be interested in can not be measured directly. For example, this situation is often encountered in *psychometrics*, when trying to measure aspects such as knowledge, abilities, attitudes, and personality traits.

The key approach to these situations is to model the measured attribute as a hidden, *latent variable*. In contrast to visible attributes, such as height and weight, these latent variables can not be observed or determined by direct measurement. However, these variables can be estimated from the results of appropriate tests.

The first and to date most common approach is currently known as the Classical test theory (CTT). As a gross oversimplification, we may state that in the CTT the test is scored, and the subject's score is used to estimate the latent ability. The main goal of CTT is to construct the tests in such a way that the *reliability* and *validity* of the test are maximized.

In recent years, CTT has been superseded by a more sophisticated approach, the Item Response Theory (IRT). The main difference is that IRT models include not only the latent variables we try to measure, but also *item parameters* (such as difficulty of a question in an IQ test). In general, IRT brings a greater flexibility and provides more sophisticated information than CTT did.

IRT is ideally suited for our setting, as in programming competitions the tasks indeed have various difficulty. We are interested both in determining the item parameters (i.e., discuss task difficulty) and in using this additional information to make better estimates of the contestants' abilities.

In this section we give an overview of the areas of IRT that are relevant to our article. For a more general overview of the areas related to IRT, we strongly recommend (Partchev, 2004). A reader interested in a deeper background in IRT is advised to pursue this topic further in the excellent monography (Baker and Kim, 2004).

The 2-parameter logistic model

First of all, we will assume that the latent ability we are interested in (e.g., the ability to solve programming competition tasks) is a scalar – i.e., that it can be described by a single real number. For each contestant c, we will denote their ability score as $\theta(c)$, or just θ if c is fixed.

As stated above, in IRT we take into consideration the individual test items. More precisely, we assume that each item comes with an inherent *item characteristic function*. This is a function that maps the subject's ability score θ to the probability that the subject answers the given item correctly.

We model programming tasks using the 2-parameter logistic model (2PL model). In this model, each item is described by two parameters: its difficulty b and its discrimination a. The item characteristic function in this model is given in (1).

$$Pr(\theta, a, b) = \frac{1}{1 + e^{-a(\theta - b)}} \tag{1}$$

Estimating parameters

One of the most common methods how to estimate the unknown parameters (be it task parameters, subjects' abilities or both) is the maximum likelihood method.

Let X be a random variable X with a parametrized probability distribution function f_y. We measured the random variable X and got the result x. The *likelihood* $L(y, x)$ of a given parameter value y is defined as the conditional probability $Pr[X = x|y]$. (Note that this is **not** the probability of y being the true parameter.) Our estimate of the unknown parameters will be the estimate for which the likelihood function is maximized.

For example, consider the case where a subject was given n tasks. For each of these tasks we know its 2PL parameters a_i and b_i. We also know whether the subject solved each of the tasks. Formally, let $s_i = 1$ if the i-th task was solved correctly, $s_i = 0$ otherwise. The values s_i are usually called the *response pattern*. Then the likelihood function of an ability estimate θ is the function:

$$L(\theta) = \prod_{i=1}^{n} Pr(\theta, a_i, b_i)^{s_i} \cdot (1 - Pr(\theta, a_i, b_i))^{1-s_i} \tag{2}$$

Fisher information and error of measurement

Whenever we observe a random variable, this observation gives us information that we can use to make a better estimate of the hidden parameters. This statistical version of information was first formalized by Sir Robert Fisher.

Intuitively, we can define information as the reciprocal of the precision with which the parameter could be estimated. Formally, let X be a random variable such that its probability distribution depends on a parameter θ. Let $L(\theta, x)$ be the likelihood function. Then the *score* V is the partial derivative with respect to θ of the natural logarithm of the likelihood function, and then the *Fisher information* is the expected variance of the score – or equivalently (as the expectation of the score is always zero) Fisher information is the expectation of the square of the score. Informally, this definiton corresponds to the steepness of the log-likelihood function in the vicinity of its maximum.

In the 2PL model, it can be computed that the Fisher information function of a single item with parameters a, b can be simplified to:

$$\mathcal{I}(\theta) = a^2 Pr(\theta, a, b)(1 - Pr(\theta, a, b)) \tag{3}$$

Given an ability estimate $\hat{\theta}$, the variance of this estimate can be estimated as the reciprocal of the test information function at that point:

$$Var(\hat{\theta}) = \frac{1}{\mathcal{I}(\hat{\theta})} \tag{4}$$

The standard error of measurement (SEM) is defined as the square root of the variance:

$$SEM(\hat{\theta}) = \sqrt{\frac{1}{\mathcal{I}(\hat{\theta})}} \tag{5}$$

In the case of the 2PL logistic model, we get

$$SEM(\hat{\theta}) = \sqrt{\frac{1}{\sum_{i=1}^{n} a_i^2 Pr(\theta, a_i, b_i)(1 - Pr(\theta, a_i, b_i))}} \tag{6}$$

2 Towards an IRT-based rating system

In this section we present our work directed towards designing a IRT-based rating system. First, we describe the setting for which we want to construct the system – in other words, the assumptions we make. These assumptions do hold for common programming competitions. We then discuss some properties any IRT-based rating system must have, and finally provide a brief description of our proof-of-concept implementation.

2.1 Assumptions

Our rating system is designed for competitions that consist of multiple rounds, each round consists of multiple items, and various items can have various degrees of difficulty.

We assume that the ability that determines the success in solving the items is a scalar.

Additionally, we assume that the setting is not antagonistic (contestants do not directly influence the performance of other contestants), and that there is no guessing – in other words, as ability decreases, the probability of solving a task converges to zero.

For the part of our research that is presented in this paper, we also assume that the abilities of subjects are invariant in time.

2.2 Necessary properties of IRT-based rating systems

The basic idea behind using IRT in a rating system is simple – the ability estimates and the task parameter estimates will be computed at the same time, as the maximum likelihood estimate given the observed response patterns.

However, the situation is not so easy in practice. We will now show that we need to enforce at least two additional restrictions.

The first issue is the symmetry of the item characteristic function. If we multiply all estimates (both for the abilities and for the task parameters) by -1, the likelihood of the estimate will not change. We need to break this symmetry somehow, and enforce that the positive direction represents higher ability level / task difficulty.

The second issue that needs to be addressed is the actual existence of the maximum likelihood estimate.

Consider the following simple example: Let $C = \{c_1, \ldots, c_{n+1}\}$ be the set of contestants in a round, and let $T = \{t_1, \ldots, t_n\}$ be a set of tasks in the round. Sey $s_{i,j} = [i > j]$, i.e., contestant i solved all tasks j for which $i > j$.

There is a clearly defined linear order on this set of contestants – for any pair of contestants, the set of tasks one of them solved is a strict superset of the set of tasks the other one solved. However, it can easily be proved that the maxima of the likelihood function are of the following form: For any integer x we can set $\theta(c_y) = -\infty$ for $y \leq x$ and $\theta(c_y) = \infty$ otherwise, set all $a(t_y) = 1$, set $b(t_y) = -\infty$ for $y < x$, $b(t_y) = \infty$ for $y > x$ and set $b(t_x)$ arbitrarily. This completely fails to reflect the linear order.

There is only one solution to both of these issues – we need to restrict the estimates to bounded intervals. Note that we are free to pick the exact bounds, as we are not influencing the results in any way by doing so, we are just defining the scale.

In our case with the 2PL model, we opted to restrict θ and b to the same interval $[-\beta, \beta]$, and to restrict a to the asymmetric interval $[-\alpha/10, \alpha]$. The rationale behind the restriction to a is to enforce that most tasks have positive a – i.e., the probability of solving the task increases with increasing ability.

(Note that in practice we can occassionally have tasks where the probability of solving the task actually slightly decreases with increasing ability. This is why we allow slightly negative values of α. On the other hand, if we are getting many tasks with $a < 0$, this is usually a sign that the ability we are measuring has nothing to do with the results.)

For the example presented above, after we enforce the restrictions, there is just one global maximum of the likelihood function – the abilities are uniformly distributed along $[-\beta, \beta]$, task difficulty parameters b are uniformly distributed between these, and all task discrimination parameters are α. This precisely corresponds to the linear order we described above.

2.3 Implementation details

Our proof-of-concept implementation used the values $\alpha = \beta = 10$. We compute the maximum likelihood estimate of all the parameters numerically. In order to reduce the running time, we used the observation that the parameters for each task can be estimated only from the (current estimate of the) subjects' abilities, i.e., independently from the other tasks. Hence we implemented a bootstrapping algorithm that alternately computes a new estimate for all the subjects' abilities and a new estimate for all the task parameters until sufficient convergence is reached.

3 Comparing rating systems

In situations where we have multiple rating systems, it is only natural to ask which of them is *better*. The word *better* can have multiple meanings, the most natural one (but by far not the only one) being "which of them estimates the true latent ability more precisely". In this section we give our answer to the question: Is there an objective method how to compare rating systems?

It is obviously impossible to compare rating systems directly, as we are not able to measure the latent ability.

However, there is a natural way out. Having a model and a set of rating estimates should enable us to predict the outcome of a future rated event. The more accurate predictions we can make, the more trustworthy the pair (rating system, prediction algorithm) is.

In other words, while we can not directly compare rating systems, we can compare rating systems accompanied by prediction algorithms.

Still, the previous paragraphs leave one open question: what exactly makes a prediction more accurate? The answer is not unique. Moreover, the answer to this question is actually what we should start with in practice. In the simplest setting with two-player antagonistic matches, the focus is usually on predicting the winner. (Or, more precisely, each player's probability of winning.)

The setting with multiple subjects allows for a much richer spectrum of possible goals. To name some: predicting a subject's score, a subject's placement, estimating their probability of placing among top K subjects, etc.

As one possible example, we will now focus on the last goal mentioned above: given a round with N known participants and an integer K, we want to predict the probabilities that each of them will place among the top K in the round.

3.1 Predicting success in a Bayesian rating system

In this section we will show an algorithm to predict the probabilities of placing in the top K using a Bayesian rating system. The presented algorithm is **not** our

original work – however, it is only informally known in the public domain, we we were not able to assign authorship to a particular author.

In the models used in the Bayesian rating systems the performance of each subject is modeled as a normally distributed random variable.

In this section we will assume that for each contestant i the rating system computed the estimate of the mean μ_i and the variance σ_i^2 of this random variable. (This is for example the case with the TopCoder ratings, where the subject's volatility computed by the rating system is an estimate of the standard deviation σ of this normal variable.)

This model is great when it comes to computing the expected placement of a subject – this is simply one plus the sum of probabilities that the other subject performs better.

However, the situation is much worse when we actually need to predict the probabilities. The only known algorithm in this case is a Monte Carlo randomized simulation: We simulate a sufficient number of rounds. For each round, we generate the actual values of all the performances, and sort them to find the top K subjects.

We would like to stress several major disadvantages of this prediction algorithm:

- Its convergence is slow – obviously the number of simulation steps necessary to achieve precision ε (with high probability) is at least linear in $1/\varepsilon$.
- It does not easily generalize for multiple round tournaments. It is easily seen that the time complexity necessary to predict the outcome of d consecutive rounds within some fixed ε precision grows exponentially with d.
- It can only predict relative order of the performances, not their actual values.

3.2 Predicting success in the IRT-based rating system

In this section we present the algorithm we designed to predict the probabilities of placing in the top K in our IRT-based rating system.

The main idea of our algorithm is that, similarly as in the previous case, we will simulate multiple rounds and take the average of the results.

First, we will start by generating the task parameters for the round. (In practice, the best way to do this is by randomly sampling a pool of known past tasks.) Given the set of tasks, we will use binary search to find the correct estimate for the threshold – i.e., the expected number of solved tasks needed to be in the top K.

Given some threshold, we can, for each of the subjects, compute the probability of solving at least that many tasks. The sum of these probabilities gives us the expected number of people that will cross this threshold. If this number is less than K, we need to lower the threshold, otherwise we need to raise it.

The needed subproblem (given a subject, a set of tasks and a threshold, compute the probability of reaching it) can be solved using dynamic programming – for each x and y, we compute the probability of solving exactly y out of the first x tasks.

Some of the advantages of this approach.

- Much faster convergence in practice – if we know the approximate task parameters beforehand, already the first round will give us a reasonable approximation of the answer.
- The approach generalizes to multiple round tournaments nicely.
- We can predict absolute performances. This can be useful e.g. to predict the number of contestants that will not be able to solve any tasks.

3.3 Using the standard error of measurement

At this point we would like to make one technical note.

For Bayesian rating systems handling newcomers poses a significant challenge. The existing rating systems usually use some kind of a provisional ad-hoc approach. For example, the TopCoder rating system assigns newcomers a slightly above-average rating of 1200, and the rating change formulas are set so that they enable large rating changes for the first few rounds.

The prediction algorithm as described above uses the ability estimates as the exact truth in order to make the predictions. A much better way is to also use the standard error of measurement that is provided by our chosen model. In this way, we can, for example, solve the newcomer problem in a clean and systematic way – for any newcomer, the number of attempted tasks is small, hence the information function returns a small value, hence the standard error of measurement will be high.

The prediction algorithm presented in 3.2 can be modified to include this information. Given a contestant c and a task t, we will compute the probability that c solves t using the assumption that c's actual ability is a normally distributed random variable with mean equal to our rating estimate, and standard deviation equal to the computed standard error of measurement. In this case, the predicted number of tasks c solves out of a set T of tasks can be expressed as:

$$PNT(c,T) = \frac{1}{SEM(c)\sqrt{2\pi}} \cdot \sum_{t \in T} \int_{-\infty}^{\infty} \exp\left(-\frac{(x-\theta(c))^2}{2SEM(c)^2}\right) \cdot \frac{1}{1+e^{-a(x-b)}}\, dx \quad (7)$$

Evaluating the predictions

Suppose that we predicted the vector of probabilities $(p_1, \ldots, p_N)$, and the actual outcome is $(s_1, \ldots, s_N)$ – where $s_i = 1$ if contestant i placed K-th or better, and $s_i = 0$ otherwise. How to measure how good the prediction was?

Our definition will be based on the following experiment: Imagine that we, in a sequence, throw N biased coins, where coin i will fall heads with probability p_i. In this experiment, we can easily compute the probability of getting the outcome $(s_1, \ldots, s_N)$: it is $\prod_{i=1}^{N} p_i^{s_i}(1-p_i)^{1-s_i}$.

Equivalently, this formula can be seen as the likelihood that the actual probabilities before the round were $(p_1, \ldots, p_N)$, given that the outcome was $(s_1, \ldots, s_N)$. And this is almost exactly how we'll define the quality of the prediction.

For computational reasons, we prefer to use the log-likelihood function in our definition instead. Given the results $(s_1, \ldots, s_N)$, we define the quality of a prediction $(p_1, \ldots, p_N)$ as:

$$Q(p_1, \ldots, p_N) = \sum_{i=1}^{N} s_i \log p_i + \sum_{i=1}^{n} (1-s_i) \log(1-p_i) \quad (8)$$

4 Evaluation on real life data

Data sets used for analysis

We tested our ideas on two separate data sets. One data set we used included 88 tasks used in two years of Slovak national programming competitions, and the scores of over 300 contestants solving them. All of these tasks used partial scoring on the scale 0 to 10, with any correct solution scoring at least 4, and any efficient solution scoring at least 7 points.

Our basic model was adapted to this setting as follows: For each task, we have three items. For each contestant, the first of these is set as solved iff she scored

ρ range	year 2006/07	year 2007/08
$[0.99, 1.00]$	9	12
$[0.98, 0.99)$	6	5
$[0.95, 0.98)$	9	9
$[0.90, 0.95)$	7	7
$[0.75, 0.90)$	12	8
$[0.00, 0.75)$	1	2
$[-1.00, 0.00)$	0	1

Table 1: Distribution of correlation coefficients between predicted and actual scores.

at least 3 points, the second iff she scored at least 6 points, and the third item is considered solved iff she scored at least 9 points.

The second data set included 560 tasks used in TopCoder competitions between 2006-05-09 and 2008-02-16, inclusive, and the performances of over 12 000 contestants on these tasks.

Sanity check

We used our proof-of-concept implementation of the rating system described in Section 2.3 to estimate the ratings of the contestants and the task parameters for the tasks in the first data set. A natural sanity check at this point is to examine whether the computed estimates give us a sufficient approximation of the real data. This check was performed as follows:

We divided the ability range into 30 equally large buckets. For each task, we divided the contestants that attempted to solve it into these buckets, based on their ability estimate θ. For each bucket i we now computed two values: The actual total score x_i its contestants achieved, and their expected total score y_i. The value x_i is simply computed by summing the corresponding input data, while y_i is computed based on the estimated parameters of the given task and on their individual ability estimates. Note that under ideal conditions the values x_i and y_i would be identical for all i, meaning that our model matches the original data perfectly.

Once we computed all x_i and y_i, we proceeded to compute the weighted correlation coefficient of the vectors (x_i) and (y_i), where the weight of each element was the number of contestants c_i in the corresponding bucket i.

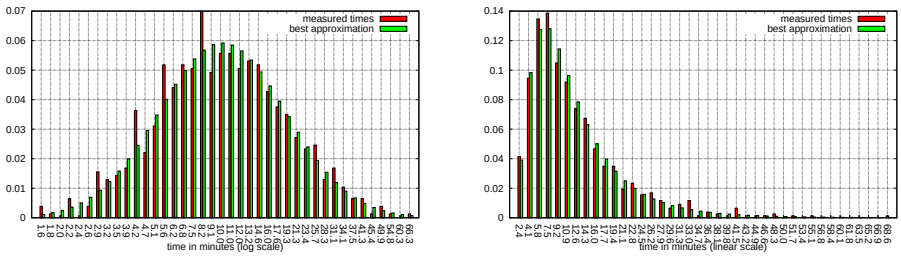
A summary of the resulting correlation coefficients is tabulated in Table 1. We see that for 64 out of 88 tasks ($\sim 73\%$) the correlation coefficient exceeds 0.9, and for half of these even exceeds 0.98, which is excellent. The three most significant outliers occurred in 2007/08, and all three were difficult tasks that only 10 contestants attempted to solve, and almost nobody did.

Solving time

After we processed the data from the TopCoder competition using our IRT-based rating system, we managed to discover that the random variable giving the solving time for a particular task usually has a log-normal distribution.

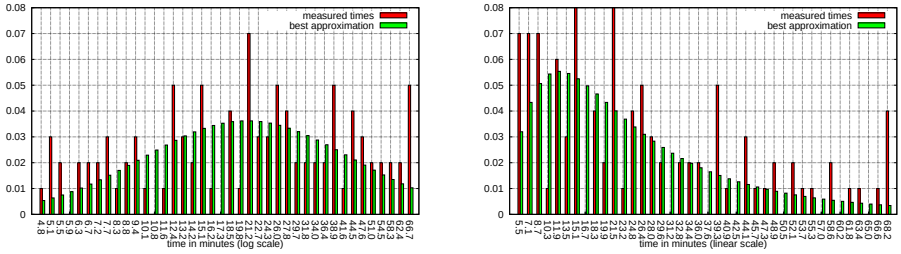
To verify this, we used the Jarque-Bera normality test (Bera and Jarque, 1980), on natural logarithms of solving times. Out of the 560 tasks in our data set, for 71 the sample size was obviously too small (at most 5 contestants succesfully solved the task).

Out of the remaining 489 tasks, for 221 the null hypotesis can be accepted at 99% confidency level, and for another 163 it can be accepted at 98% confidency level.



(a) log scale on x axis (b) linear scale on x axis

Figure 1: Solving times for PalindromeDecoding match the lognormal distribution at 99% confidency level. Sample size $N = 772$.

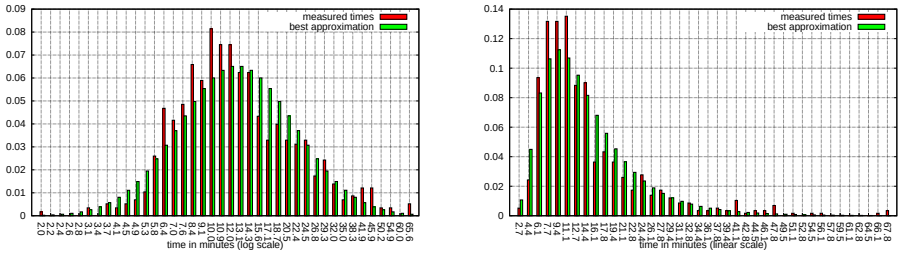


(a) log scale on x axis (b) linear scale on x axis

Figure 2: Solving times for Caketown match the lognormal distribution at 98% confidency level. Smaller sample size $N = 100$.

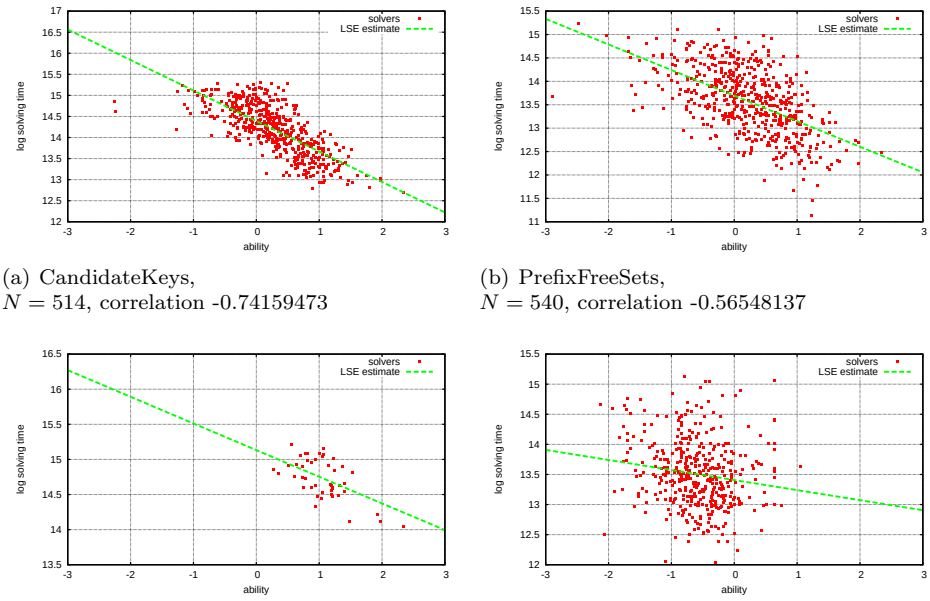
We verified the remaining 105 tasks by hand, and discovered that the main reasons for rejecting the lognormality hypotesis were mostly either small sample sizes, or it were too easy tasks where most of the solving times come from a narrow interval. Even for these cases the most probable lognormal distribution provides a reasonable approximation.

Furthermore, we note that for many (but sadly, by far not for all) tasks there is a significant correlation between the logarithm of the solving time and the ability estimate made by our rating system. Plots of the dependency for four random tasks are shown in Figure 4.



(a) log scale on x axis (b) linear scale on x axis

Figure 3: Solving times for RussianSpeedLimits do not match the lognormal distribution at 98% confidency level. Sample size $N = 577$. Lognormal distribution still offers a reasonable approximation.



(a) CandidateKeys,
$N = 514$, correlation -0.74159473

(b) PrefixFreeSets,
$N = 540$, correlation -0.56548137

(c) BasketballStrategy,
$N = 47$, correlation -0.62801324

(d) VoteRigging,
$N = 422$, correlation -0.15639571

Figure 4: Correlation between abilities and log solving times – and least squares linear approximation thereof – for four tasks.

Predicting advancement in the TopCoder Open 2008

We adapted our prediction algorithm to the TopCoder setting – by estimating the solving time using the observations presented above, we were able to compute the probability of exceeding a given score threshold. (We did ignore the challenge phase, and only computed the expected score of solving the three given tasks.)

Using this algorithm, we predicted the advancement probabilities for the first two online rounds of the TopCoder Open 2008. For the first round, the prediction given by our algorithm was slightly better as the one given by the TopCoder's own ratings and the Monte Carlo prediction algorithm (-812.26 vs. -818.85). For the second round the Monte Carlo method gave a slightly better prediction (-372.04 vs. -394.68).[1] Hence already our simplest IRT-based rating system was able to produce results comparable to the ones given by the existing rating system.

Additionally, our superior model allowed us to compute predictions that were not possible in the previously used model. Specifically, the tournament has a rule that one has to have a positive score in order to advance. Our prediction algorithm computed that in the first round the expected number of contestants with a positive score is 872.994, which is less than the 900 advancer spots. This is almost exactly what actually happened – the actual number of advancers was 864. For the second round the algorithm correctly predicted that all 300 advancer spots will be taken.

5 Conclusion

In this paper we presented an overview of our research in the area of rating algorithms. We describe a new type of a rating system we developed using Item Response Theory, define a formal way how to compare rating systems, and use it

[1] The presented numbers are values returned by the log-likelihood function (8).

to compare our rating system (adapted to a slighty different setting) to an existing Bayesian rating system.

Our rating system is more general than the existing models, in that it allows us to make predictions that were not possible in the existing models. Additionally, we believe that in settings with different tasks (such as programming competitions) an advanced version of the rating system will be able to use this additional information to give significantly better predictions than the existing Bayesian rating systems.

This is a new and promising area of research with many possible practical applications. Some points that surely deserve more attention include:
– A detailed analysis of the numerical aspects of the rating system (discussing convergence and its rate)
– Evaluation of this approach on data from other areas, such as sports.
– Modifying our approach to address settings in which abilities change over time.
– Using the computed data to argue about hardness of past competitions.

References

Baker, F. B. and Kim, S.-H. (2004). *Item Response Theory: Parameter Estimation Techniques*. CRC, 2 edition.

Bera, A. K. and Jarque, C. M. (1980). Efficient tests for normality, homoscedasticity and serial independence of regression residuals. *Economics Letters*, 6:255–259.

Cormack, G. (2006). Random Factors in IOI 2005 Test Case Scoring. *Informatics in Education*, 5:5–14.

Cormack, G., Munro, I., Vasiga, T., and Kemkes, G. (2006). Structure, Scoring and Purpose of Computing Competitions. *Informatics in Education*, 5:15–36.

Dangauthier, P., Herbrich, R., Minka, T., and Graepel, T. (2008a). TrueSkill Through Time: Revisiting the History of Chess. In *Advances in Neural Information Processing Systems*, volume 20, pages 931–938.

Dangauthier, P., Herbrich, R., Minka, T., and Graepel, T. (2008b). TrueSkill Through Time: Revisiting the History of Chess. In Platt, J., Koller, D., Singer, Y., and Roweis, S., editors, *Advances in Neural Information Processing Systems 20*, pages 337–344. MIT Press, Cambridge, MA.

Elo, A. (1978). *The rating of chessplayers, past and present*. Arco Publishing.

Forišek, M. (2004). On suitability of tasks for the IOI competition. Personal communication to the IOI General Assembly.

Forišek, M. (2006). On the Suitability of Programming Tasks for Automated Evaluation. *Informatics in Education*, 5:63–76.

Forišek, M. (2009). Vyhodnotenie reliability hodnotenia Olympiády v Informatike.

Gleason, J. (2008). An Evaluation of Mathematics Competitions Using Item Response Theory. *Notices of the ACM*, 55.

Glickman, M. E. Example of the Glicko-2 system.

Glickman, M. E. (1995). A Comprehensive Guide to Chess Ratings. *American Chess Journal*, 3:59–102.

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

Harkness, K. (1967). *Official Chess Handbook*. David McKay Company.

Herbrich, R. and Graepel, T. (2006). TrueSkill$^{\text{TM}}$: A Bayesian Skill Rating System. Technical report. MSR-TR-2006-80.

Herbrich, R., Minka, T., and Graepel, T. (2007). TrueSkill(TM): A Bayesian Skill Rating System. In *Advances in Neural Information Processing Systems*, volume 20, pages 569–576.

Kemkes, G., Vasiga, T., and Cormack, G. V. (2006). Objective Scoring for Computing Competition Tasks. In Dagiene, V. and Mittermeir, R., editors, *Information Technologies at School*.

Opmanis, M. (2006). Some Ways to Improve Olympiads in Informatics. *Informatics in Education*, 5:113–124.

Partchev, I. (2004). A visual guide to item response theory.

Sonas, J. (2002). The Sonas Rating Formula – Better than Elo?

TopCoder Inc. (2008). Algorithm Competition Rating System. Technical report.

van Leeuwen, W. T. (2005). A Critical Analysis of the IOI Grading Process with an Application of Algorithm Taxonomies. Master's thesis.

Verhoeff, T. (1990). Guidelines for Producing a Programming-Contest Problem Set. An unpublished personal note.

Verhoeff, T. (2006). The IOI is (not) a Science Olympiad. *Informatics in Education*, 5:147–159.

Yakovenko, B. (2006). 50% Rule Should Be Changed. Presented at Perspectives on Computer Science Competitions for (High School) Students.

Michal Forišek is currently finishing his PhD study at the Comenius University in Slovakia. He received a Master's degree in Computer Science from this university in 2004. He was the head of the problem committee for several international programming contests, including CEOI 2002 and most years of the Internet Problem Solving Contest (IPSC). In years 2006 to 2009 he serves as an elected member of the International Scientific Committee (ISC) of the International Olympiad in Informatics (IOI). His research interests range from theoretical computer science to education of mathematics, informatics, and algorithms.