# ORDO v1.0
# Ratings for chess and other games*

Miguel A. Ballicora†

Ordo is a program designed to calculate ratings of individual chess engines (or players). It has a similar concept than the Elo rating[a], but with a different model and algorithm. Ordo keeps consistency among ratings because it calculates them considering all results at once. In that respect, it behaves similarly to BayesElo[b]. Ordo is distributed under the GPL license and binaries are available for GNU/Linux, Windows®, and OS X. In addition, the sources are portable and could be easily compiled for other systems.

[a] `http://en.wikipedia.org/wiki/Elo_rating_system`
[b] `http://remi.coulom.free.fr/Bayesian-Elo/`

### Precompiled Files

In this distribution, you may find versions for GNU/Linux (32 and 64 bits) or Windows® (64 and 32 bits). For convenience, you can rename the proper file for your system to **ordo** (GNU/Linux) or **ordo.exe** (Windows®). As an input example, a publicly available file `games.pgn` is included[1]. A batch file (`ordo_example.bat`) is included in the Windows® distribution[2]. It is a quick and great start for users of that operating system.

### GNU/Linux compilation and installation

After unzipping the contents, you can type

```
make
make install
```

or in Ubuntu

```
sudo make install
```

### Usage

The input should be a file that adheres to the PGN standard[3]. Based on the results in that file, Ordo automatically calculates a ranking . The output can be a plain text file and/or a *comma separated*


†e-mail: mballicora (at gmail dot com)
[1] Taken from the recently discontinued *Ingo Bauer's* IPON rating list
[2] Kindly prepared by *Adam Hair*
[3] `http://en.wikipedia.org/wiki/Portable_Game_Notation`

*value*[4] (`.csv`) file. The `.csv` file is an interesting option because it can be opened/imported by most spreadsheet programs. Once imported, the user can choose to format the output externally. The simplest way to use Ordo is typing in the command line:

```
ordo -p games.pgn
```

which will take the results from `games.pgn` and output the text ranking on the screen. If you want to save the results in a file `ratings.txt`, you can run:

```
ordo -p games.pgn -o ratings.txt
```

By default, the average rating of all the individuals is 2300. If you want a different overall average, you can use the switch `-a` to set it. For instance to have and average of 2500, you can do:

```
ordo -a 2500 -p games.pgn -o ratings.txt
```

or if you want the results in `.csv` format, use the switch `-c`.

```
ordo -a 2500 -p games.pgn -c rating.csv
```

If you want both, you can use:

```
ordo -a 2500 -p games.pgn -o ratings.txt -c rating.csv
```

### Anchor

In addition, `-A` will fix the rating of a given player as a reference (*anchor*) for the whole pool of players.

```
ordo -a 2800 -A "Deep Shredder 12" -p games.pgn -o ratings.txt
```

That will calculate the ratings from `games.pgn`, save it in `ratings.txt`, and *anchor* the engine *Deep Shredder 12* to a rating of 2800. Names that contain spaces should be surrounded by quote marks as in this example.

### White advantage

The switch `-w` sets the rating advantage for having white pieces in chess. Alternatively, the (highly recommended) switch `-W` lets Ordo calculate it automatically. With this switch we can complete the above example:

```
ordo -a 2800 -A "Deep Shredder 12" -p games.pgn -o ratings.txt -W
```

If the user knows that the *white advantage* is most likely a certain value that could vary within a certain range, this uncertainty could be given by the switch `-u`. Therefore, a combination of switches `-w <value>` and `-u <deviation>` may provide a prior information to Ordo to calculate the white advantage. When a high number of games is played, this prior information will be less and less relevant. Ordo assumes a Gaussian distribution centered in `<value>` with a standard `<deviation>`.

[4] `http://en.wikipedia.org/wiki/Comma-separated_values`

### Simulation and errors

The switch `-s <n>` instructs Ordo to perform `<n>` simulations, virtually *replaying* the games `<n>` times. The results will be randomly re-assigned for each game according to the probabilities calculated from the ratings. After running the simulations, and based on all those different results, Ordo calculates standard deviations (*errors*) for the ratings. For this purpose, an optional switch is `-F value`, where `value` is the % *confidence level* (The default is 95.0, which is roughly equivalent to ± 2 standard deviations). The errors displayed are relative to the pool average. However, if one of the players is *anchored*, the rest of the errors will be relative to that *anchor*. In this case, the anchor error will be zero since it is the point of reference. To get the errors for rating differences between a given pair of players, the switch `-e file.csv` should be added. It will generate an error matrix saved in `file.csv`.

To run these simulations, a minimum reasonable number is about `-s 100`. Take into account that the more simulations, the longer it takes to complete the runs. The errors calculated will be more accurate, but more than 1000 simulations is probably not needed. This is an example to use these switches:

```
ordo -a 2800 -A "Deep Shredder 12" -p games.pgn -o ratings.txt -W -s1000 -e errors.csv
```

It is important to emphasize that the errors displayed in the output are always against the reference (anchor). For example, if the anchor is Engine X (Deep Shredder 12 in the example above) set at 2800, and Engine Y is 2900 with an error of 20, then the interpretation is that the difference between Y and X is 100 ± 20.

As mentioned above, when no engine is set as anchor the *hidden* reference is the average of the pool. For instance, if the average is set to 2500 (default is 2300) and the rating output for Engine X is 2850 ± 20, the difference between Engine X and the average of the pool is 350 ± 20. That is how the output should be interpreted. It is incorrect to use this error to estimate relative values against other engines. For that purpose, the switch `-e` needs to be provided to obtain a matrix with every single error for every engine-engine match ups.

If an anchor (reference) is provided, but the user wants the errors to be relative to the average of the pool, the switch `-V` should be added in the command line. This is what other rating software has as default.

```
ordo -a 2800 -A "Deep Shredder 12" -p games.pgn -o ratings.txt -W -s1000 -e errors.csv
-V
```

In this case, you will see that the rating of `Deep Shredder 12` will not have an error of zero.

### Parallel calculation of simulations

If the switch `-n <value>` is used, Ordo will use `<value>` number of processors in parallel for the simulations. This may be a significant speed-up.

### Superiority confidence

If simulations have been run, using the switch `-C` will output a matrix with the confidence for superiority (CFS) between each of the players. Each of the numbers is an answer to the question

*"What is the maximum confidence I can set the test to show that player x is not inferior to player y and still obtain the same positive answer?"*. The matrix file is in *comma separated values* format, and it could be opened by any spreadsheet program if it was saved with the *.csv extension. In addition, if the user provides the switch `-J`, the CFS values between the player and the next one in the ranking will be displayed in the output.

### Draw rate (for equal opponents)

By default, Ordo considers that the draw rate for evenly matched players is 50%. Internally, it calculates the draw for matches in which a player is stronger than the other. This parameter does not change the rating results, but it will affect the errors calculated after simulations. Two switches can control this parameter. First, `-d` sets the draw rate (which is assumed to be constant throughout the database). Alternatively, the (highly recommended) switch `-D` lets Ordo calculate it automatically. It makes sense if the user wants to calculate more accurate errors, or just for informative purposes. For instance:

```
ordo -a 2800 -A "Deep Shredder 12" -p games.pgn -o ratings.txt -W -s1000 -e errors.csv
-V -D
```

Will calculate the draw rate and outputs it at the end of

```
ratings.txt
```

(or the screen, if the switch `-o` is omitted). Similarly to the calculation of the white advantage, the user can provide prior information for the draw rate. A combination of switches `-d <value>` and `-k <deviation>` will do that. Ordo assumes a Gaussian distribution centered in `<value>` with a standard `<deviation>`. When as the number of games increases, this information will have less and less impact on the final result.

### Ignore draws (`-X` switch)

This switch internally ignores all draws from the database as they have not been played. This is only present for experimentation, not for a serious rating calculation.

### Minimum games

In certain cases, the user may not want to include certain players with very few games played in the rating. For that reason, the switch `-t <value>` provides to the program a threshold of minimum games played for a participant to be included in the final list. The games are still included for calculation.

### Perfect winners and losers

Players who won all games (*perfect winners*) or lost all of them (*perfect losers*) create problems in the rating calculation. It is impossible to estimate those rating accurately because winning all or losing all correspond to a $+\infty$ or $-\infty$ rating, respectively. In addition, the calculation slows down considerably because of the impossibility to converge. Ordo removes these players automatically during the calculation, and place them back after convergence has been reached. The rating

assigned to them is a minimum ("floor") for *perfect winners* and a maximum ("ceiling") for *perfect losers*. This is indicated by a $>$ or $<$ symbol in the output text. These limits are established by calculating the rating they would have had if one of the games was a draw. For example, if player had a performance of 10/10, a proper rating estimation lays between $(+\infty)$ and the one corresponding to a performance of 9.5/10. A nice side effect of this technique is that distinguishes players with perfect score that had different type of opposition or played different number of games. It is not the same to have been undefeated for three games than twenty.

### Group connections and pathological data

Sometimes, a data set contains players or groups/pools of players that did not play enough games against the rest. These *isolated* groups produce meaningless ratings when compared to the general pool. The `-g` switch saves a report of how many groups are in this situation. The information in this report may guide the user to properly link those groups with extra games. Doing so will stabilize the whole ranking. When the data set is "ill" connected, Ordo will attempt to run by purging *perfect winners* and *perfect losers*. Their ceiling or floor rating will be estimated at the end (see above). However, a warning will be diplayed. When purging those players is not enough to guarantee a proper connection, a second warning will be issued. But, this time the program will stop and exit with an error code (i.e. non-zero). To force the calculation even in these conditions, the switch `-G` should be used. Be careful, this could be slow and the algorithm may not converge.

### Multiple anchors

When several players are known to have very accurate ratings, it is possible to assigned fixed values to them. In that case, they will behave like multiple anchors. An example will be:

```
ordo -p games.pgn <optional switches> -m anchors.csv
```

where `anchors.csv` is a file that contains lines like this

```
"Gull 1.1", 2350.0
"Glaurung 2.2 JA", 2170
"Crafty 23.1 JA", 2000
```

telling Ordo to fix Gull 1.1, Glaurung 2.2 JA, and Crafty 23.1 JA to 2350, 2170, and 2000, respectively. The name of the anchors should be present in `games.pgn`.

### Match up information

The switch `-j` will output to a file information about all different matches that have been played. It shows the rating difference (Diff) between those particular players and the standard deviation (SD) for that difference. These values come from the simulations performed with the switch `-s`, so everything is taken into account, not only the information about a particular match. In addition, there is a column with the *confidence* that would be needed in order to be able to claim superiority based on Diff and SD. The column is CFS, confidence for superiority, which plays the same role as the *likelihood of superiority*[5]. A fragment of the output is:

[5] `https://chessprogramming.wikispaces.com/Match+Statistics`

```
 3) Critter 1.4 SSE42        2562 :   2400 (+1467,=772,-161),  77.2 %

    vs.                           :  games (    +,   =,   -),   (%) :   Diff,  SD, CFS (%)
    Houdini 2.0 STD               :    100 (   23,  54,  23),  50.0 :    -35,   9,    0.0
    Komodo 4 SSE42                :    100 (   33,  44,  23),  55.0 :     +6,  11,   71.0
    Deep Rybka 4.1 SSE42          :    100 (   28,  48,  24),  52.0 :    +23,  10,   99.0
    Stockfish 2.1.1 JA            :    100 (   25,  66,   9),  58.0 :    +44,   8,  100.0
```

In this example, we can say that Critter 1.4 SSE42 is superior to Deep Rybka 4.1 SSE42 with a 99% confidence. We can only say that it is better than Komodo 4 SSE42 with a 71% confidence. The reason is because the rating difference is 6 and the standard deviation is 11.

### Loose anchors with prior information (`-y`)

Ordo offers an alternative approach to calculate ratings with previous knowledge from the user (using Bayesian concepts). With the switch `-y`, the user can provide a file with a list of players whose ratings will float around an estimated value. Those players will work as *loose anchors* in the list. This strategy is useful when the data is scarce and, as a consequence, wild swings could appear in the ratings. This is what happens at the beginning of a new rating list or tournament. Ordo accepts an estimated rating for a player, but takes into account how uncertain that value is. In other words, the user also has to provide the standard error for the estimated value. That means that the value will be 68% of the time between $\pm$ the uncertainty value provided. It is assumed that the estimated rating will follow Gaussian distribution. In Bayesian terms, that constitute the prior distribution for the rating of that particular player. For instance, if one line of the file provided with the switch `-y` contains

```
"Houdini 3", 3200, 50
```

That means Houdini's initial rating is 3200 with an uncertainty of 50. With this approach the user should have the best educated guess possible, otherwise, the ranking will suffer. Using information from a previous well established rating lists can add stability to the new list and, as games are added, the contribution of the "previous information" will fade away.

### Relative anchors (`-r`)

Another problem in some engine tournaments is that version upgrades enter with no previous ratings. However, we know in certain situations that the new versions cannot have very different ratings from the previous one. Therefore, the user can make a good educated guess about the rating of the new version. For instance, if you know that the new version is within 20 points of the previous one you can use the `-r` switch to provide a file with lines like this:

```
"Bouquet 1.8a", "Bouquet 1.8", 0, 20
```

That means version 1.8a came after 1.8 and it is estimated to have the same rating (0) with an uncertainty of 20. With different versions, you can have different lines. An example with Stockfish may be:

```
"Stockfish 160913", "Stockfish 4", 0, 20
"Stockfish 4", "Stockfish 250413", 0, 50
```

```
"Stockfish 250413", "Stockfish 120413", 0, 20
"Stockfish 120413", "Stockfish 250313", 0, 20
```

This constitute different *relative anchors*. When two versions are radically different, you can say nothing and they will be treated as different engines, or for instance

```
"Komodo 1063", "Komodo 4534", 0, 1000
```

The first is a complete rewrite with a parallel search. Thus, the uncertainty of 1000 reflects this fact and make both versions virtually disconnected. If you want to include more specific info, you could say

```
"Komodo 1063", "Komodo 4534", 160, 100
```

Here, 160 is the estimation of how much improvement you have by going from 1 core to 16 and 100 represents how uncertain that is.

### Switches

The list of the switches provided are:

```
usage: ordo [-OPTION]
 -h          print this help
 -H          print just the switches
 -v          print version number and exit
 -L          display the license information
 -q          quiet mode (no screen progress updates)
 -Q          quiet mode (no screen progress except simulation count)
 -a <avg>    set rating for the pool average
 -A <player> anchor: rating given by '-a' is fixed for <player>, if provided
 -V          errors relative to pool average, not to the anchor
 -m <file>   multiple anchors: file contains rows of "AnchorName",AnchorRating
 -y <file>   loose anchors: file contains rows of "Player",Rating,Uncertainty
 -r <file>   relations: rows of "PlayerA","PlayerB",delta_rating,uncertainty
 -R          remove older player versions (given by -r) from the output
 -w <value>  white advantage value (default=0.0)
 -u <value>  white advantage uncertainty value (default=0.0)
 -W          white advantage will be automatically adjusted
 -d <value>  draw rate value % (default=50.0)
 -k <value>  draw rate uncertainty value % (default=0.0 %)
 -D          draw rate value will be automatically adjusted
 -z <value>  scaling: set rating for winning expectancy of 76% (default=202)
 -T          display winning expectancy table
 -p <file>   input file in PGN format
 -c <file>   output file (comma separated value format)
 -o <file>   output file (text format), goes to the screen if not present
 -E          output in Elostat format (rating.dat, programs.dat & general.dat)
 -g <file>   output file with group connection info (no rating output on screen)
 -G          force program to run and ignore warnings for isolated groups
 -j <file>   output file with head to head information
 -s  #       perform # simulations to calculate errors
 -e <file>   save an error matrix, if -s was used
 -C <file>   save a matrix (.csv) with confidence for superiority (-s was used)
 -J          add an output column with confidence for superiority (next player)
 -F <value>  confidence (%) to estimate error margins. Default is 95.0
 -X          ignore draws
 -t <value>  threshold of minimum games played for a participant to be included
 -N <value>  number of decimals in output, minimum is 0 (default=1)
 -M          force maximum-likelihood estimation to obtain ratings
```

```
-n <value>  number of processors for parallel calculation of simulations
```

### Memory Limits

Currently, the program can handle almost un unlimited number of games and players. It is only limited by the memory of the system.

### Exit code

When Ordo ran successfully, it will exit with a code = 0. When problems arose (insufficient memory, database not well connected, empty input, wrong parameters, etc.), Ordo will return a number that is guaranteed to be non-zero. This could be used in scripts to know whether the process reached its goal or not. For instance, the following script in bash (linux) will catch if processing games.pgn was correct or not.

```
#!/bin/sh

./ordo -p games.pgn
exit_code=$?

if [ $exit_code = 0 ]; then
    echo Ordo run properly
else
    echo Ordo returned with error: $exit_code
fi
```

### Ordoprep

A tool is available in another distribution[6] to shrink the PGN file. The output will contain only the results of the games. In addition, it could discard players that won all games, or lost all games. Other switches allow the exclusion of players that do not have a minimum performance or played too few games.

Typical usage is:

```
ordoprep -p raw.pgn -o shrunk.pgn
```

Which saves in `shrunk.pgn` a pgn file with only the results. You can add switches like this:

```
ordoprep -p raw.pgn -o shrunk.pgn -d -m 5 -g 20
```

where `-d` tells Ordoprep to discard players with 100% or 0% performance, `-m 5` will exclude players who did not reach a 5% performance, and `-g 20` will exclude players with less than 20 games. After all this, `shrunk.pgn` could be used as input for Ordo

[6] `https://github.com/michiguel/Ordoprep/releases`

### Model for rating calculation

The model assumes that differences in strength are analogous to differences in levels of energy (Fig. 1). A lower (more stable) level of energy would represent a stronger player. The analogy is that a valley is better at attracting water than a mountain top. In physics and chemistry, a particle or a molecule that can be in two different states can be predicted to be in one or the other with a certain probability.

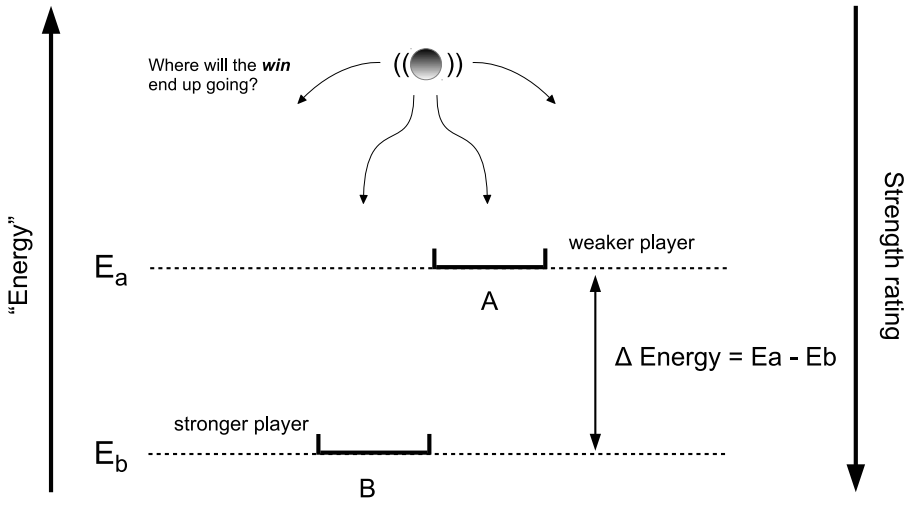


Figure 1: Energetic levels as strength levels

The probability to be found at each level is proportional to the *Boltzmann factor*[7] $e^{-\beta E_i}$. If $N_a$ is the number of particles in level A, and $N_b$ is the number of particles in level B, their ratio will be:

$$\frac{N_a}{N_b} = \frac{e^{-\beta E_a}}{e^{-\beta E_b}} = e^{-\beta(E_a - E_b)} \tag{1}$$

$\beta$ is a constant of the system. The analogy is that we treat the probabilities of a *win* to land in level A or B as the probability of a particle to be in A or B. Therefore, after reordering equation 1, the *fraction of wins* ($f_{b,a}$) of player B in a match vs. A will be:

$$f_{b,a} = \frac{N_b}{N_a + N_b} = \frac{1}{1 + e^{-\beta(E_a - E_b)}} \tag{2}$$

if we define strength rating $R$ as the negative value of *energy*, then, $R_a = -E_a$. For convenience, we flip the scales with the purpose that higher ratings are represented with higher values (Fig. 2), and the the *fraction of wins* ($f_{b,a}$) of player B in a match vs. A will be represented by eq. 3.

$$f_{b,a} = \frac{1}{1 + e^{-\beta(R_b - R_a)}} \tag{3}$$

[7] `https://en.wikipedia.org/wiki/Boltzmann_factor`

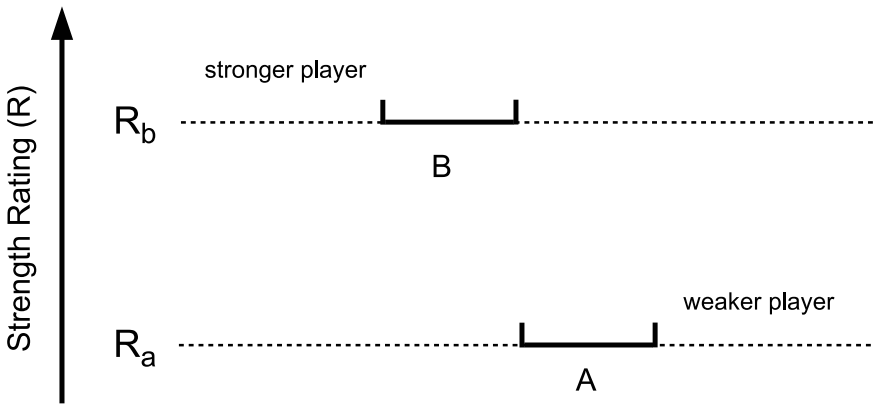


Figure 2: Rating scale

This equation has the same form as the logistic function[8]. With this equation we can calculate the predicted fraction of wins between two players. The predicted performance $P_x$, or number of wins of player $x$ among a pool of other players will be the summation of each of the predicted fractions $f$ for each game.

$$P_x = f_{x,opp(1)} + f_{x,opp(2)} + ... + f_{x,opp(n)} = \sum_{i=1}^{n} f_{x,opp(i)} \tag{4}$$

where $n$ is the total number of games played by $x$ and $opp(i)$ is the opponent it faced in the game $i$. Then:

$$P_x = \sum_{i=1}^{n} \frac{1}{1 + e^{-\beta(R_x - R_{opp(i)})}} \tag{5}$$

The most likely strength rating values ($R$) for each player are ones that satisfy that each *predicted* performance $P_x$ equals the respective *observed* performance ($O_x$) of player $x$ (actual number of games won by $x$). Therefore, the goal is to find $R$ values so the following unfitness ($U$) score equals zero, where $m$ is the total number of players, and $j$ is each individual player.

$$U = \sum_{j=1}^{m} (P_j - O_j)^2 \tag{6}$$

Finding an adequate procedure to minimize $U$ until reaches zero is critical for a proper convergence towards the optimal solution. The way Ordo fits it is in discrete steps (similar to *hill climbing*[9]), and making those steps smaller and smaller once the convergence was reached. However, those steps are constrained to certain values to avoid big swings during the calculation. After many different tests, this procedure was found to be safe and fast.

[8] `http://en.wikipedia.org/wiki/Logistic_function`

[9] `http://en.wikipedia.org/wiki/Hill_climbing`

### Scale

Chess players are accustomed to the Elo rating. Traditionally, it has been based on a normal (Gaussian) distribution, which is the one that the World Chess Federation (FIDE) still uses[10]. Here, the default value of $\beta$ was chosen to resembles the Elo scale. For that reason, the rating difference when the winning expectancy is 76% has been set to 202 rating points. This parameter could be modified with the switch `-z`, and the overall scale can be displayed with switch `-T`.

The model is valid if the strength assigned to the individual players is additive like energy. If we know the strength differences between A→B and B→C, we should be able to calculate A→C as A→B + B→C. Then, this should accurately predict the results of a match between A and C. Empirical observations seem to suggests that those estimations are reasonable, at least within a certain range.

Certain theoretical assumptions have be done to account the existence of draws. One of the is that the actual draw rate remains similar throughout the rating scale. Empirically, this is a reasonable approximation for most cases.

### White advantage calculation

The rationale to calculate the white advantage ($W_{adv}$) is that the expected outcome for white should be as close as possible to the actual white performance. In other words, the number of points obtained by white ($W_p$) should be the same as the number of points expected to be obtained by white ($W_e$).

$$E = (W_p - W_e)^2 \tag{7}$$

Therefore, the optimum $W_{adv}$ is the one that minimizes $E$, which is the overall error squared in equation 7.

$$W_e = \sum_{i=1}^{n} Expectancy(RW_i + W_{adv}, RB_i) \tag{8}$$

Here, $n$ is the total number of games, $RW_i$ and $RB_i$ are the ratings (in game $i$) of white and black, respectively. *Expectancy* is actually equation 3.

$$W_e = \sum_{i=1}^{n} \frac{1}{1 + e^{-\beta(RW_i + W_{adv} - RB_i)}} \tag{9}$$

Then, combining 7 and 9

$$E = \left( W_p - \sum_{i=1}^{n} \frac{1}{1 + e^{-\beta(RW_i + W_{adv} - RB_i)}} \right)^2 \tag{10}$$

[10] `http://en.wikipedia.org/wiki/Elo_rating_system`

$W_{adv}$ is calculated iteratively, until $E$ is minimized. This calculation assumes that $W_{adv}$ is relatively constant throughout the database. Once $W_{adv}$ is obtained, the ratings are re-calculated. The procedure continues until the numbers stabilize.

### Draw rate model

To estimate the probability of a draw in a single game the model from Fig. 2 needs to be expanded to have an extra "draw state" (Fig. 3).

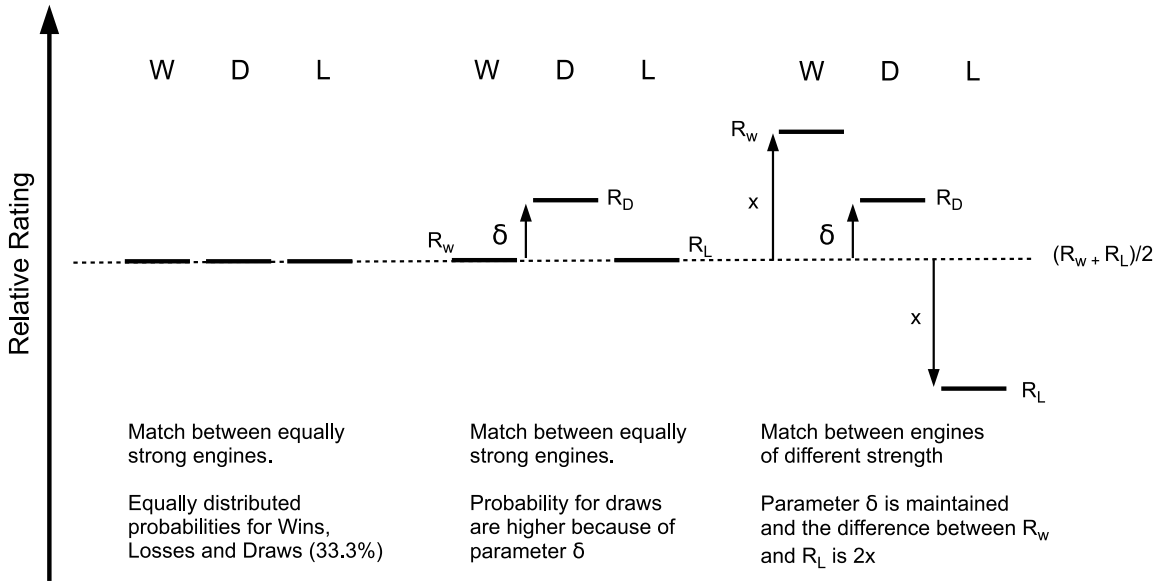


Figure 3: Rating scale introducing an extra state for draws

The draw rate does not affect the rating calculation, or the performance for each player in the simulations. However, it affects the relative distribution of *wins*, *losses*, and *draws* simulated, which has an influence on the errors calculated. Therefore, to have a more realistic simulation and an accurate estimation of the errors we need to predict the probability for a draw. But, the draw rate is not uniform, as it depends on the rating differences between the opponents. Thus, draw rate depends on two parameters, $D_{eq}$ (draw rate when the two opponents are of equal strength) and $\Delta R$. Ordo assumes that $D_{eq}$ is relatively constant throughout the database. If we know $D_{eq}$, the following equation

$$E = \sum_{m=1}^{M} (D_m - N_m \, D_{exp}(\Delta R_m + W_{adv}, D_{eq}))^2 \tag{11}$$

will give $E$ as the overall error in the estimation of $D_{eq}$. Here, $m$ is the match number, $M$ is the total number of matches, $N_m$ is the number of games played in each match $m$, $\Delta R_m$ is the rating difference in that particular match, $D_m$ is the number of draws observed, and $W_{adv}$ is the *white advantage*. $D_{exp}$ is a function that gives the draw rate expected given a certain $\Delta R$ and $D_{eq}$. Note that here a match is considered any series of games between two opponents with the same colors. In other words, they are any set of games with the same opponents and conditions. With this

equation, $D_{exp}$ is calculated iteratively until $E$ is minimized. To apply this algorithm we need the function $D_{exp}$. In the following section we show how to calculate the draw rate when opponents are of equal strength and later from a given $p$ and $D_{eq}$. From $\Delta R$, the performance expected ($p$) can be directly calculated.

### Draw rate between opponents of equal strength

We can model the draw rate by introducing an extra *draw state* (Fig. 3). This is a derivation of the equation that relates draw rate ($D$) and $\delta$.

$$1 = W + D + L \tag{12}$$

Here, $W$, $D$, and $L$ are the respective win, draw, and loss rates. Since the opponents are of equal strength, W equals L.

$$1 = 2\,W + D \tag{13}$$

Based on the assumptions that the probabilities of the different levels are proportional to the *Boltzmann factor*[11] $e^{-\beta E_i}$, the following ratio can be established ($R_i = -E_i$, higher ratings mean lower "energy levels").

$$\frac{D}{W} = \frac{e^{\beta R_D}}{e^{\beta R_W}} = e^{\beta(R_D - R_W)} = e^{\beta\delta} \tag{14}$$

Replacing into eq. 13

$$1 = e^{\beta\delta} W + 2W \tag{15}$$

$$W = \frac{1}{e^{\beta\delta} + 2} \tag{16}$$

Combining with eq. 13 we obtained $D_{eq}$, which is the draw rate when both players are equally strong. This value depends on $\delta$.

$$D = D_{eq} = 1 - \frac{2}{e^{\beta\delta} + 2} = \frac{e^{\beta\delta}}{e^{\beta\delta} + 2} \tag{17}$$

### Draw rate from $p$ (performance) and $D_{eq}$

Performance ($p$) is the ratio of the total points obtained by a player in a given number of games. It is defined by this simple relationship.

$$p = W + D/2; \quad W = p - D/2 \tag{18}$$

[11] `https://en.wikipedia.org/wiki/Boltzmann_factor`

To define $D_{eq}$, we are going to assume it is constant, regardless of the absolute strength of each individual. We then have three possible states, W (win), D (draw), and L (loss), in which the state D is separated by $\delta$ from the average of the levels W and L. In this scenario, and reordering eq. 17 we have:

$$\frac{1 - D_{eq}}{2D_{eq}} = e^{-\beta\delta} \tag{19}$$

For convenience we will call $e^{-\beta\delta} = \phi$ then

$$\frac{1 - D_{eq}}{2D_{eq}} = \phi \tag{20}$$

$D_{eq}$ is the rate when $R_W$ and $R_L$ are at the same level. If $R_W$ and $R_L$ change, and $\delta$ remains at the same distance from the average of $R_W$ and $R_L$, the equations that relate the probabilities for each state are:

$$R_{avg} = \frac{R_W - R_L}{2};\ \ x = R_W - R_{avg} = R_{avg} - R_L \tag{21}$$

$$W/D = e^{\beta(x-\delta)} = e^{\beta x}e^{-\beta\delta} \tag{22}$$

$$D/L = e^{\beta(x+\delta)} = e^{\beta x}/e^{-\beta\delta} \tag{23}$$

For convenience, if we call $e^{-\beta\delta} = \phi$ as we did before we get

$$W/D = e^{\beta x}\phi;\quad D/L = e^{\beta x}/\phi \tag{24}$$

therefore

$$\frac{W}{D}\frac{L}{D} = \phi^2;\quad L = \frac{\phi^2 D^2}{W} \tag{25}$$

combining this equation with eq. 12 and reordering:

$$0 = W^2 + DW - W + \phi^2 D^2 \tag{26}$$

replacing W with eq. 18 we obtain

$$0 = (p - D/2)^2 + D(p - D/2) - (p - D/2) + \phi^2 D^2 \tag{27}$$

expanding, simplifying, and reordering leads to

$$0 = (4\phi^2 - 1)D^2 + 2D + 4(p^2 - p) \tag{28}$$

replacing with eq. 20

$$0 = \left( \left( \frac{1 - D_{eq}}{D_{eq}} \right)^2 - 1 \right) D^2 + 2D + 4(p^2 - p) \tag{29}$$

Solving this quadratic equation, we obtain the predicted draw rate ($D$) between two given opponents, as long as we know the predicted performance ($p$) and the draw rate between equally matched opponents ($D_{eq}$). This is used to plug it in eq. 11.

### Draw rate and win rate relationship

Reordering eq. 26 we obtain

$$D^2 = \phi^{-2} W(1 - W - D) \tag{30}$$

Note that this relationship is equivalent to the basic assumption used by Davidson[12] to develop his draw model

$$D = \nu \sqrt{W\ L} \tag{31}$$

Here, $\nu = \phi^{-2}$ and $L = 1 - W - D$. Shawul and Coulom showed that this relationship is superior for chess engines when compared to other alternatives[13]. Replacing $\phi$ in eq. 30 with eq. 20 we obtain

$$D^2 = \left( \frac{2D_{eq}}{1 - D_{eq}} \right)^2 W(1 - W - D) \tag{32}$$

Equation 32 is the one used by Ordo to obtain the draw rate for any pair of opponents as a function of win probability ($W$) and draw rate for equal opponents ($D_{eq}$).

### Draw rate calculation

The rationale to calculate the draw rate for equal opponents ($D_{eq}$) is that the expected outcome of number of draws showuld be as close as possible to the actual number of draws in the database. In other words, the number of draws observed ($D_{obs}$) should be the same as the number of draws expected ($D_{exp}$).

$$E = (D_{obs} - D_{exp})^2 \tag{33}$$

Therefore, the optimum $D_{eq}$ is the one that minimizes $E$, which is the overall error squared in equation 33.

$$D_{exp} = \sum_{i=1}^{n} D_i \tag{34}$$

[12] Equation 2.5 in `http://stat.fsu.edu/techreports/M169.pdf`

[13] `https://dl.dropboxusercontent.com/u/55295461/elopapers/elopapers/ChessOutcomes.pdf`

Here, $n$ is the total number of games, and $D_i$ is the probability of a draw for game $i$. From equation 29, $D_i$ could be solved as

$$D_i = \frac{-1 + \sqrt{1 - 4(p_i^2 - p_i)(4(\frac{1-D_{eq}}{2D_{eq}})^2 - 1)}}{4(\frac{1-D_{eq}}{2D_{eq}})^2 - 1} \tag{35}$$

where $p_i$ is the expected performance for white for each game, and could be calculated from equation 3 as

$$p_i = \frac{1}{1 + e^{-\beta(RW_i + W_{adv} - RB_i)}} \tag{36}$$

$RW_i$ and $RB_i$ are the ratings (in game $i$) of white and black, respectively. Once $D_{eq}$ is estimated, $p_i$ and $D_i$ are calculated (equations 35 and 36) for each game to obtain $D_{exp}$ and $E$ (equations 33 and 34). Optimum value of $D_{eq}$ is the one that minimizes $E$ and it is calculated iteratively. This calculation assumes that $D_{eq}$ is relatively constant throughout the database. Once $D_{eq}$ is obtained, the ratings are re-calculated as it is done with $W_{adv}$. The procedure continues until the numbers stabilize.

### Rating calculation with prior information

When user provides Ordo with either *loose anchors*, *relative anchors*, *white advantage* uncertainty, or a *draw rate* uncertainty the calculation is performed by a *maximum-likelihood estimation*. In those cases, for each game the probability for the given outcome (W, D, or L) is calculated and the logarithm of this value is added and accumulated. This will constitute an unfitness score that will need to be minimized. In addition, to this score, the logarithm of the probabilities for each loose anchor, relative anchor, white advantage, and draw rate are accumulated. An overall minimization brings optimum values for the ratings of each player and each of the above mentioned parameters. Note that adding the logarithm of each of the probabilities is analogous to multiplying the probabilities.

### Forcing maximum likelihood

Another option to force Ordo to perform a *maximum-likelihood estimation* to calculate the ratings is by providing the switch `-M`. This option is generally a bit slower and probably not necessary since the output should be nearly identical with perfect convergence, but it is a good feature for comparison an debugging.

### Acknowledgments

*Adam Hair* has extensively tested and suggested valuable ideas.

### License