Documentation of the GGP Rating System

Martin Günther

February 20, 2009

Abstract

This document describes the current implementation of the GGP rating system.

1 Data structures

First, some words on the used data structures (ref. table 1). The data structures used for player, game and match have their intuitive meaning, but there is a new data structure called "match set". A match set simply is a collection of matches, all played in the same year, round and day of a competition, and all on the same game. All past competitions have featured such match sets, and explicit handling of match sets makes it possible to process all matches of such a match set in one batch.

Another unusual entry is the "coefficients" property of a game, which will be explained in section 2.1. For now, it suffices to think of it as the information that the linear regression rating stores about a game.

2 Linear regression rating

The central idea of linear regression rating is summarized in an email by Jim Clune:

> Here's some notation:
>
> | | |
> |---|---|
> | $g(m)$: | game associated with match m |
> | $s(r, m)$: | score of role r in match m |
> | $p(r, m)$: | player assigned to role r in match m |
> | $q(p)$: | rating of player p |
>
> It seems to me that we want to be able to express $E[s(r, m)]$ (the expected value of the score of role r in match m) in terms of $q(p(r', m))$ for each role r' in the game. One way to achieve this might be to assume $E[s(r, m)]$ is a linear function of these variables and perform least-squares linear regression. For a game with roles $= \{r_1, r_2, r_3\}$, this yields relationships such as:

$$E[s(r_1, m)] = c_0 + c_1 * q(p(r_1, m)) + c_2 * q(p(r_2, m)) + c_3 * q(p(r_3, m))$$

class	property	description
player	rating	a single number reflecting the player's current rating (e. g., 1000.0)
game	name	the game name (e. g., chess)
	roles	the roles of the game (e. g., black, white)
	coefficients	linear-regression-specific game info
match	match ID	the ID of the match (e. g., Match.3390056123)
	players	the participants of the match *(→ player)*
	scores	the scores $(0 \dots 100)$ achieved by each player
	match set	the match set that this match is part of *(→ match set)*
match set	match set ID	the ID of this match set (e. g., MatchSet.385572910)
	year, round, day, match set number	self-explanatory (e. g., 2007, 3, 2, 10)
	game	the played game (by definition, this is identical for all played matches of a match set) *(→ game)*

Table 1: Data structures

where c_0, c_1, c_2, c_3 are game-specific constants computed by the linear regression.

We could then calculate the expected outcome for each player in a given match and compare it with their actual outcome. If expected and actual outcomes are the same, the rating remains unchanged. Otherwise, the rating is adjusted up or down proportional to the difference between the actual outcome and the expected outcome.

- Jim

The following sections will show how to update the game-specific coefficients c_0, c_1, c_2, c_3 (section 2.1), how to calculate the expected scores of a player (section 2.2) and how to update the player ratings (section 2.3).

2.1 Updating the game information

The linear regression rating algorithm keeps so-called "game information" about each played game. This game information is just a matrix of linear regression coefficients and reflects the relationship between the role a player plays, the ratings of all players in the game and the expected score of the player in a certain game.

As stated above, linear regression rating assumes that the expected score of a player playing role r_1 can be approximated by a linear function of the form:

$$E[s(r_1, m)] = c_0 + c_1 * q(p(r_1, m)) + c_2 * q(p(r_2, m)) + c_3 * q(p(r_3, m)) \quad (1)$$

The coefficients $c_0, \ldots, c_3$ are updated immediately after each played match set for the whole batch of matches. (The coefficients are different for each role, so the game information for a 3-player game such as this would consist of $3 \cdot 4 = 12$ coefficients.)

If, in a given match set, most high-rated players scored worse (on average) than most low-rated players, the coefficient associated with the target role (c_1 in the example above) can become negative. If this is the case, the linear regression is repeated without the target role, and the target role's coefficient is set to zero. There are no constraints on the other coefficients.

If there are already known coefficients from former matches, the weighted average of the old and new coefficients is stored.

2.2 Calculating the expected scores

In the next step, the expected score of each player is calculated using formula 1. If the calculated value is less than 0 or greater than 100, the actual expected value is set to 0 or 100, respectively. After that, the total expected scores of each player are calculated by summing up the player's expected scores in all matches that the player played in that match set, so that the total expected score can exceed 100.

2.3 Updating the player ratings

In a third step, the player rankings are updated according to the following formula:

$$q'(p) \leftarrow q(p) + l * (a(p) - e(p)) \tag{2}$$

where $q(p)$ and $q'(p)$ are the current and updated rating of player p, l is the learning rate (cf. section 2.4), $a(p)$ is the actual and $e(p)$ the expected score. Update also takes place after each completed match set, so the player ratings are adjusted as quick as possible in order to provide more accurate information to the next game information updating step.

Player's ratings are carried over between competitions. New player's ratings are initialized with a constant value of 1000.

2.4 Constant vs. dynamic learning rate

The choice of the learning rate has a big impact on the actual results of the algorithm. If the learning rate is too large, the player ratings become erratic and don't converge. If the learning rate is too small, they take too long to converge. Experiments were done with a dynamic learning rate: starting off with a high learning rate and reducing it over time. However, it was decided to use a constant learning rate of 1.0 for the final version of the rating system.

3 Experiments

3.1 Constant vs. Dynamic Learning Rate

The results for one run of both algorithms are shown in figures 1 and 2. Figure 3 shows the actual scores for comparison.

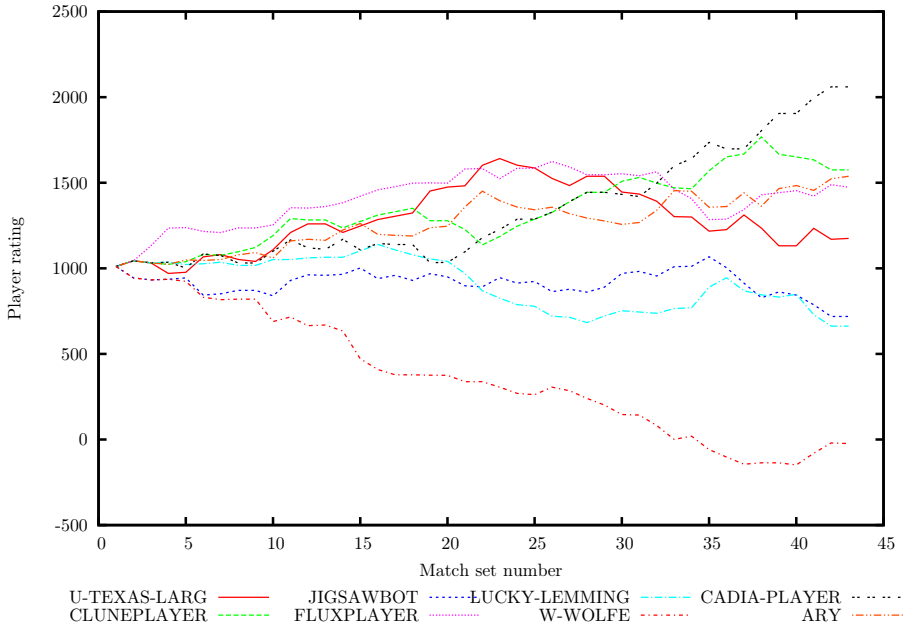

Figure 1: Linear regression ratings with a *constant* learning rate (Competition 2007 Preliminaries, constant learning rate = 1.0)

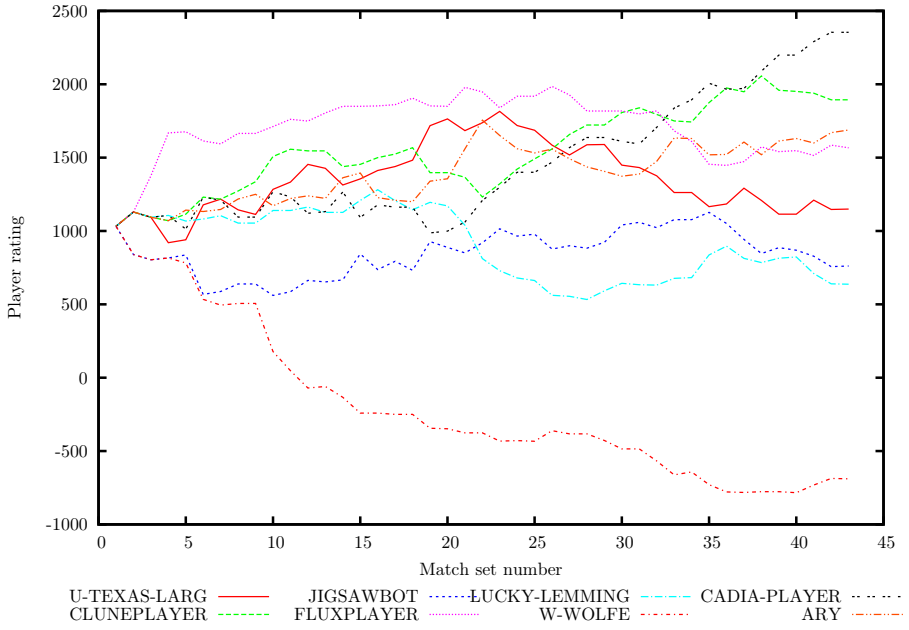

Figure 2: Linear regression ratings with a *dynamic* learning rate (Competition 2007 Preliminaries, dynamic learning rate = 6.0...1.6, linear descent)

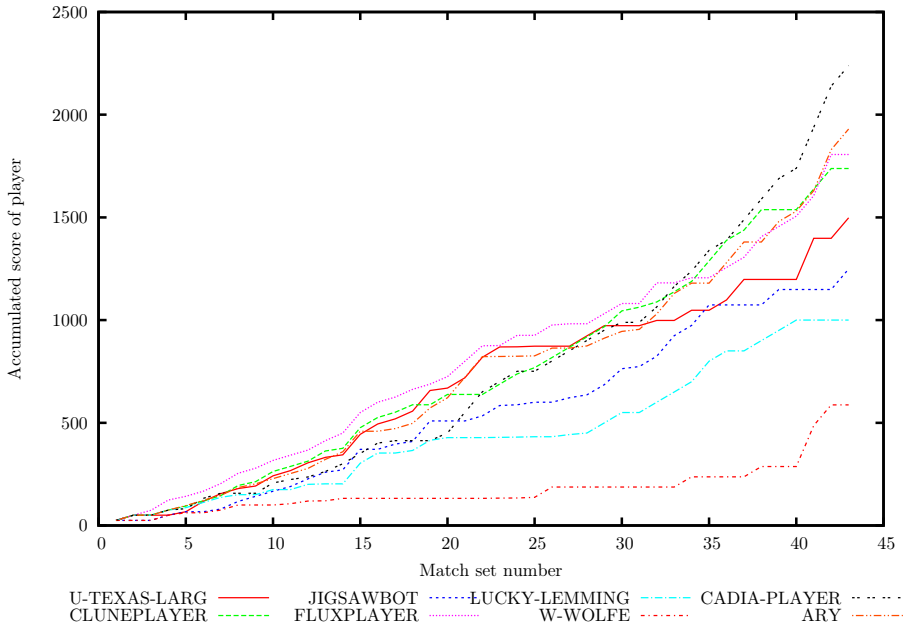

Figure 3: Weighted direct scores (Competition 2007 Preliminaries; using round weights 0.25, 0.5, 0.5 and 1.0)

3.2 Re-Run

As a quick "sanity check", both algorithms were re-run on the same data set, using the final player ratings of the first run to initialize the player ratings of the second run. The results are shown in figures 4 and 5. As expected, the re-run produced results in the same range as the original one. The ratings are not constant, which can be partly explained by the fact that the players were under development during the competition, and so their playing strength did not remain constant. A good example is Cadia-Player, which performed only moderately well during the first half of the competition, and so its rating is corrected downward during the re-run. In the second half of the competition, its playing strength greatly improved, and so the rating rises again.

3.3 Calculated Weights

To demonstrate the properties of the algorithm, some of the calculated weights will be analyzed.

The results of the 8 matches of tictactoe-large[1] are shown in table 2. The first two columns are the ratings of both players, the third column shows the actual outcome and the last column contains the expected outcome, calculated via the coefficients in table 3.

Tictactoe-large is an example where the algorithm performs well. The prediction is statistically very significant ($p = 0.0051$), and the calculated weights are in the expected range. However, there are match sets which do not give

[1] Competition 2007 preliminaries, round 3, day 1 – or short 2007-R3D1

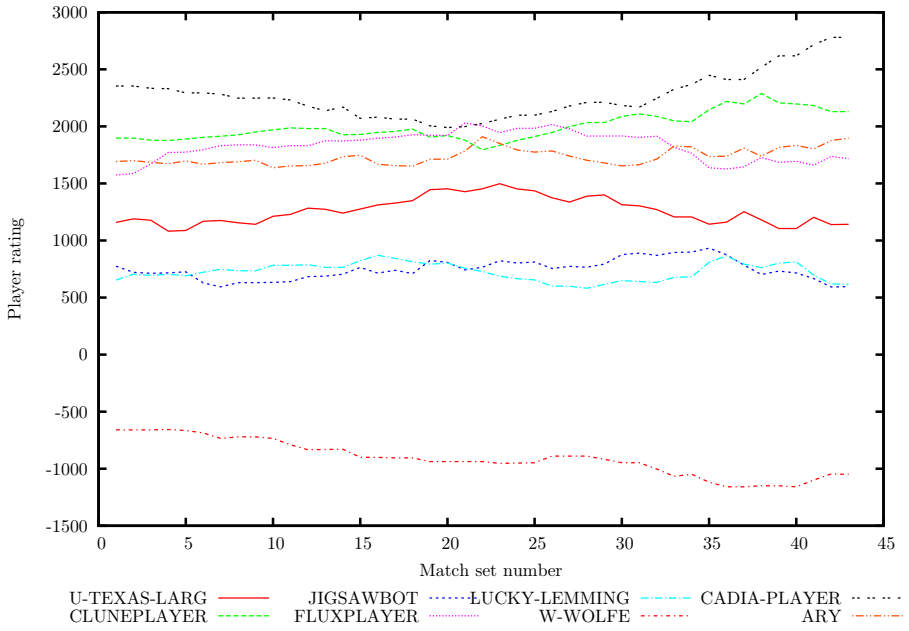

Figure 4: Re-Run of linear regression ratings with a *constant* learning rate

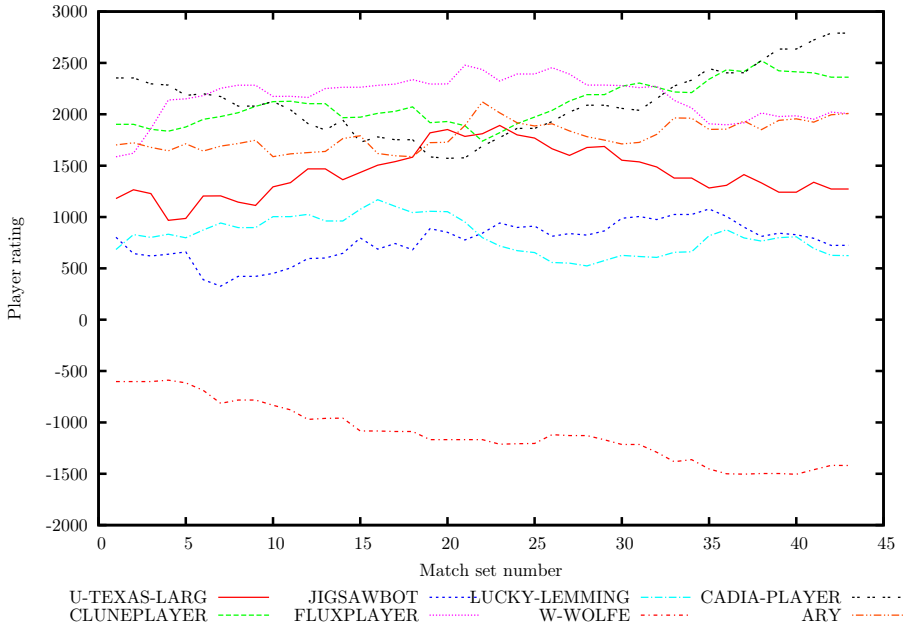

Figure 5: Re-Run of linear regression ratings with a *dynamic* learning rate

Table 2: Player ratings, real and predicted outcomes of tictactoe-large

Rating 1	**Rating 2**	**Outcome**	**Outcome (pred.)**
1455.0136	1166.2216	50.0	56.1203
1875.7368	1127.6833	100.0	65.8533
-729.6037	1519.9328	0.0	2.3262
2007.6933	837.7279	50.0	74.7947
1166.2216	1455.0136	50.0	43.8797
1127.6833	1875.7368	0.0	34.1467
1519.9328	-729.6037	100.0	97.6738
837.7279	2007.6933	50.0	25.2053

Table 3: Coefficients for role 1 of tictactoe-large, $p = 0.0051$ (linear correlation coefficient probability)

	Best Estimate	**SD**	**Coeff. [%]**
c_0	50.000	26.5921	53.1841
c_1	0.0212	0.0129	60.8856
c_2	-0.0212	0.0129	-60.8856

satisfying results, like Amazons (2007-R4D1). Although the higher-rated player won in seven of the eight cases (table 4), the algorithm assigns a negative coefficient to both roles (table 5). The reason for this is probably that the difference in ratings is huge in the "offending" match (number 7). Since least-squares minimization is used, the linear regression algorithm prefers many smaller errors to one huge error and picks its coefficients accordingly. As already mentioned in section 2.1, these results will not be used, and instead a linear regression without the target role will be performed.

Table 4: Player ratings, real and predicted outcomes of amazons

Rating 1	**Rating 2**	**Outcome**	**Outcome (pred.)**
1515.4385	710.5769	100.0	76.1431
1210.5041	2291.2757	0.0	22.6747
1939.6671	-732.956	100.0	123.8856
828.5942	1600.0015	0.0	49.8975
710.5769	1515.4385	0.0	53.7589
2291.2757	1210.5041	100.0	52.7323
-732.956	1939.6671	100.0	49.5567
1600.0015	828.5942	100.0	71.3512

Table 5: Coefficients for role 1 of amazons, $p = 0.0541$

	Best Estimate	**SD**	**Coeff. [%]**
c_0	112.4939	54.0653	48.0607
c_1	-0.0075	0.0247	-330.895
c_2	-0.0353	0.0247	-69.9282

4 Implementation Issues

The implementation reads most of the attributes in table 1 directly from the match XML files, available from the game master. Specifically, these are: **match id**, **roles**, **players**, and **scores**.

Unfortunately, the following attributes are not available in machine-readable form, but only via the web interface: **game name**, **year**, **round**, and **day**.

Additionally, the concept of a **match set** has been introduced here, which is not supported by the game master. Also, there is no direct way to access the underlying match XML files. Thus, the present process for acquiring the necessary data is a little awkward and consists of the following steps:

1. mirror the whole games.stanford.edu website
2. manually move all XML files into a directory
3. copy-and-paste game name, year, round and day from the website into a CSV file, called match_index.csv, in the same directory.

In summary, the amount of manual work could be reduced significantly if

- the match XML files would be provided directly by the game master (e. g., in the form of URIs), and
- the missing data, most importantly the game name, would also be included in the XML format.