pp!

Sean Adamson

Last updated: October 8, 2020

Contents

1 Introduction

It is well-known that the current *osu!* performance metric, *pp*, suffers from severe misrepresentation of player skill on many beatmaps. Several attempts to identify those beatmaps which overestimate skill have been made [1], [2]. In this work, we present a new method (called *pp!*) to determine the performance of an *osu!* score. This method makes use only of data presently stored by the *osu!* game servers. We first discuss calculation of the performance of a score, given its accuracy and *beatmap difficulty* (which we define). A process to determine this beatmap difficulty entirely statistically is then shown. We then discuss the role of other performance metrics (such as maximum combo) in our system. Some remarks about the implementation of this method are then given. Finally, we discuss some of the main selling points of the method, as well as its disadvantages. We propose possible solutions to each of these shortcomings.

2 Determining performance

2.1 Defining beatmap difficulty

The *difficulty* of a beatmap is denoted by a number in the interval $[0, 1]$, where larger values correspond to harder beatmaps. It is intended to represent "the hardness of a beatmap, as would be assigned by the hypothetical average player". In other words, it is supposed to represent how difficult a randomly chosen player would perceive a beatmap on average. The exact details of one possible way to determine such beatmap difficulties quantitatively are given in section 3.

2.2 Performance from accuracy and difficulty

We begin by assuming that the accuracy of scores set on a particular beatmap, denoted by random variable X, is distributed as $X \sim \text{Beta}(\alpha, \beta)$ for some shape parameters α and β. It is possible to instead parametrize this distribution in terms of only its mean μ and variance σ^2 by the substitutions

$$\alpha = \mu \left(\frac{\mu(1-\mu)}{\sigma^2} - 1 \right), \quad \beta = (1-\mu) \left(\frac{\mu(1-\mu)}{\sigma^2} - 1 \right). \tag{1}$$

Hence, since μ and σ^2 can be estimated from existing submitted scores, it is simple to extract the Beta distribution which best models the accuracy distribution of any beatmap, by replacing μ and σ^2 by the sample mean $\bar{x}$ and sample variance s^2 respectively in eq. (1). Some example accuracy distributions, and their corresponding Beta distributions, are depicted in fig. 1.

We now wish to calculate the *pp!* that will be assigned to a particular score. Let $F_X(x)$ be the CDF of the random variable X evaluated at accuracy x, and let d denote the beatmap difficulty. We make the following assertions:

1. The *pp!* awarded for a score with average accuracy should be equal to the beatmap difficulty d.
2. The *pp!* awarded for a score with 0 accuracy should be equal to 0.
3. The *pp!* awarded π_X as a function of accuracy should be proportional to F_X.

This uniquely determines the *pp!* function to be

$$\pi_X(x) = \frac{d}{F_X(\bar{x})} F_X(x). \tag{2}$$

Example *pp!* functions are plotted in fig. 2.

3 Determining beatmap difficulty

This section is dedicated to a procedure for qualitatively determining the difficulty, as defined heuristically in section 2.1, of all beatmaps.

3.1 Comparison graph

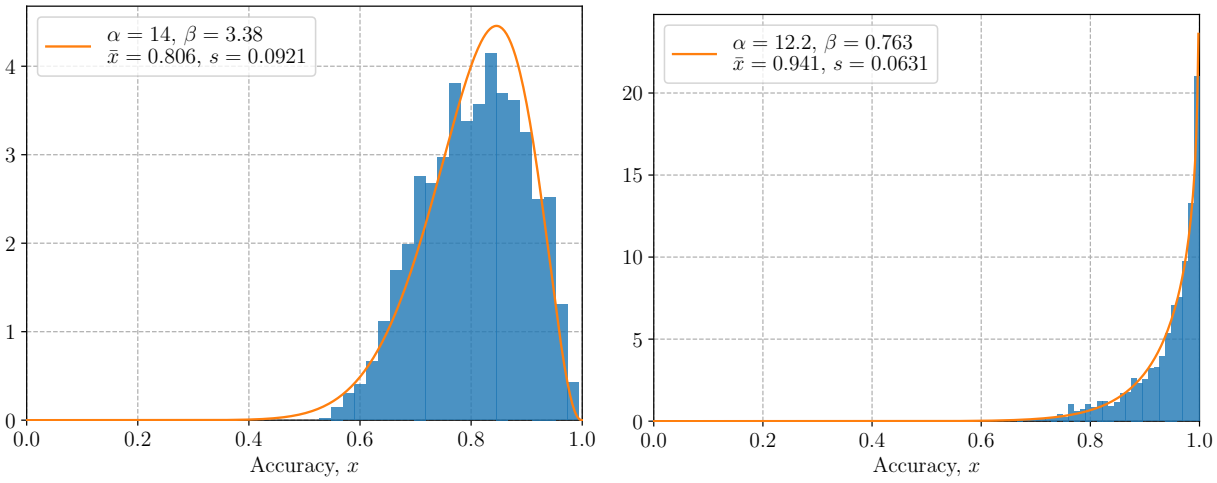
We construct a weighted oriented graph $G = (V, E)$, where the vertices V represent all beatmaps and the directed edges E represent ordered pairs of beatmaps whose difficulties we will attempt to directly compare. We then discuss the inclusion of *game modifiers* in our comparison graph.

3.1.1 Edge weights

We assign weights to the directed edges in E to represent the magnitude of each difficulty comparison by defining a function $t\colon E \to \mathbb{R}$ as follows.

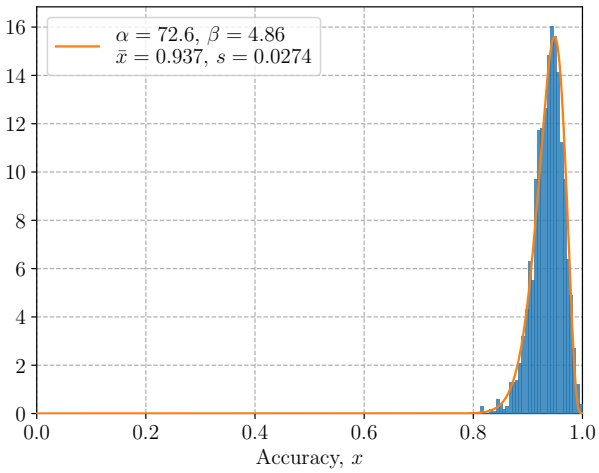
Let $(x, y) \in E$ and let P and Q be the sets of users who have submitted scores on beatmaps x and y respectively. Define the common set of n users $U \equiv P \cap Q$, and let $(a_i)_{i=1}^n$ and $(b_i)_{i=1}^n$ be accuracy vectors for all scores which are submitted by users in U on beatmaps x and y respectively, and where in both vectors the same index represents the player who set the score. Now define accuracy differences for each player $(c_i)_{i=1}^n = (a_i - b_i)_{i=1}^n$. We choose to weight the importance of each of these differences by a function of time elapsed between their submissions. One such choice of weight is an exponential decay

$$w_i = 2^{-|s_i - t_i|/\tau}, \tag{3}$$

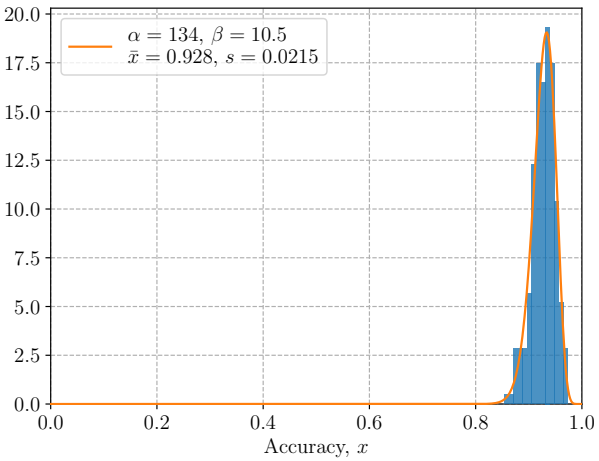


(a) Beatmap ID: 39825

(b) Beatmap ID: 351190



(c) Beatmap ID: 960875



(d) Beatmap ID: 1192895

Figure 1: A selection of accuracy distributions. Histograms shown in blue use data from the top 10,000 players (according to *pp*) as of 2019-08-01. Beta PDFs are shown in orange, with shape parameters α and β calculated from the same data using eq. (1). Figure 1a shows `IOSYS - Marisa wa Taihen na Kanbu de Tomatte Ikimashita (DJPop) [Love]`, fig. 1b shows `cYsmix feat. Emmy - Tear Rain (jonathanlfj) [Normal]`, fig. 1c shows `DECO27 - Ghost Rule (Awaken) [Mayday]`, and fig. 1d shows `The Koxx - A FOOL MOON NIGHT (Astar) [Silverboxer's Supernova]`.

where s_i and t_i are the submission times, and τ is some time constant over which users improve at the game (a reasonable value is $\tau = 8$ weeks). Define the weighted mean and (corrected) weighted standard deviation of the accuracy differences

$$\bar{c} = \frac{\sum_{i=1}^{n} w_i c_i}{V_1}, \tag{4a}$$

$$\sigma_c = \frac{\sum_{i=1}^{n} w_i (c_i - \bar{c})^2}{V_1 - V_2/V_1}, \tag{4b}$$

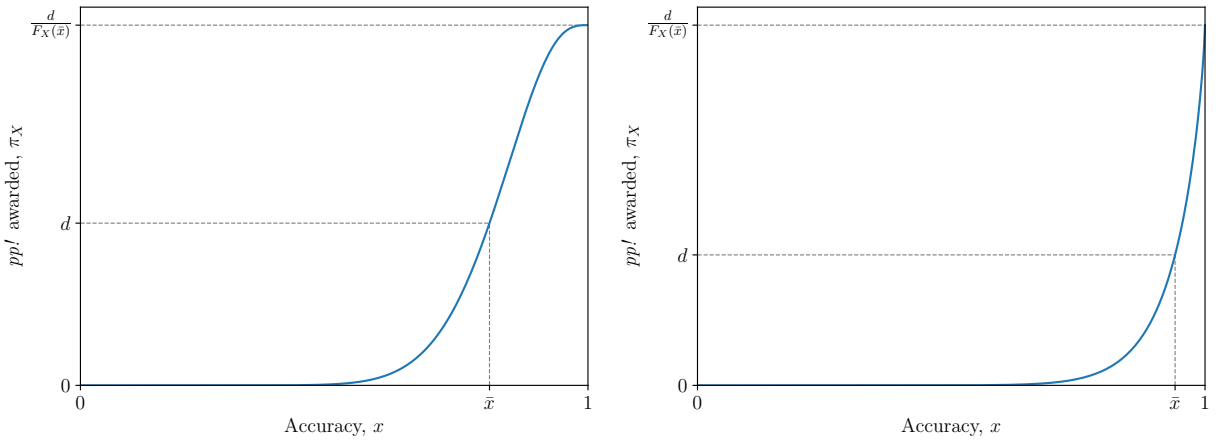
where $V_1 = \sum_{i=1}^{n} w_i$ and $V_2 = \sum_{i=1}^{n} w_i^2$. The standard error of the weighted mean is calculated as

$$\sigma_{\bar{c}} = \frac{\sqrt{V_2}}{V_1} \sigma_c. \tag{5}$$

Finally, our edge weights are defined by normalization to units of standard error as

$$t \colon (x, y) \mapsto \frac{\bar{c}}{\sigma_{\bar{c}}}. \tag{6}$$

In the above, it is sensible to form the edge (x, y) only if the number of common players shared between the two beatmaps satisfies $n \geq N$ for some threshold $N > 1$ ($N = 50$ is a reasonable value [3], however, we also recommend testing values in the range $N = 30$ or even lower).



(a) Beatmap ID: 39825

(b) Beatmap ID: 351190

(c) Beatmap ID: 960875

(d) Beatmap ID: 1192895

Figure 2: A selection of *pp!* functions as in eq. (2), corresponding to the PDFs shown in fig. 1. The maximum available *pp!* for a beatmap of difficulty d is $d/F_X(\bar{x})$. An accuracy of $\bar{x}$ corresponds to a *pp!* of d being awarded.

3.1.2 Game modifiers

Notice that in the above process, no mention of *game modifiers* was made. For each *osu!standard* beatmap, there exist a further 35 possible combinations of game modifiers that may be applied (from the individual modifiers EZ, HD, HR, DT/NC, HT, and FL). If we were to treat each of these as separate beatmaps, the order of our comparison graph would increase by a factor of 36. In this case, since $|E| \sim |V|^2$, computations become significantly more impractical. A solution to this is provided by the additional structure of *beatmapsets* on the set of beatmaps: **we only permit edges including game modifiers to be formed between vertex pairs having the same beatmapset**. The same logic is applied to other game modes.

3.2 Beatmap difficulty vector

In this section, we provide a modified version of a method given in [4] for finding the priority vector for an incomplete fuzzy additive preference relation. Here, the priority vector (which we will denote $\boldsymbol{d}$) is a vector of beatmap difficulties which we wish to determine. Define the matrix $\boldsymbol{T} = (t_{ij})$ by

$$\boldsymbol{T} = \boldsymbol{A} - \boldsymbol{A}^{\mathsf{T}}, \tag{7}$$

where $\boldsymbol{A} = (a_{ij})$ is the adjacency matrix of our comparison graph $G = (V, E)$. Let $m = \max_{i,j} t_{ij}$ and $n = |V|$. Let the existence of an undirected edge (an edge in either direction) between vertices i and j be denoted $i \leftrightarrow j$, and let the total degree of vertex i be denoted $k_i = \deg^-(i) + \deg^+(i)$. It can be shown (see appendix A) that the linear system $\boldsymbol{M}\boldsymbol{d} = \boldsymbol{v}$ to solve for the beatmap difficulties $\boldsymbol{d} = (d_i)_{i=1}^n$

is given by specifying $\boldsymbol{M} = (m_{ij})$ and $\boldsymbol{v} = (v_i)_{i=1}^n$ to be

$$m_{ij} = \begin{cases} k_i & i = j, \\ -1 & i \leftrightarrow j, \\ 0 & \text{otherwise}, \end{cases} \tag{8a}$$

$$v_i = \frac{1}{m(n-1)} \sum_{\substack{j \\ i \leftrightarrow j}} t_{ij}. \tag{8b}$$

4 Other performance metrics

It is worth addressing the fact that accuracy is the only performance metric we have used thus far to develop *pp!* While this is perfectly reasonable for those game modes where skill is tied very closely to accuracy, it may surprise the reader that we apply this same approach to *osu!standard*. One might argue that, since a large component of *osu!standard* has to do with cursor aim, accuracy alone may not be enough to capture a user's performance on a beatmap; we may wish to augment accuracy using another performance metric.

4.1 Combo as a performance metric

The only other direct performance metric that is currently recorded in the *osu!* game servers is the *maximum combo* of a score. Since it is difficult to make use of maximum combo directly in determining performance, there are a few possibilities of how to proceed:

1. Ignore maximum combo and assume that the dependence of accuracy on misses is significant enough to incorporate the aim component of *osu!standard*.

2. Assume that the accuracy and maximum combo distributions of a beatmap (which have random variables X and C respectively) depend on each other weakly enough that we can reasonably generalize the *pp!* function in eq. (2) to take the form of a joint distribution,

$$\pi_{X,C}(x, c) = \pi_X(x) \cdot G_C(c) = \frac{d}{F_X(\bar{x})} F_X(x) \cdot G_C(c), \tag{9}$$

   where G_C is some cumulative distribution function relating to maximum combo. There are two obvious choices for this function:

   (a) The function G_C is chosen to be the cumulative distribution function of maximum combos for the particular beatmap.

   (b) The *pp!* awarded is chosen to scale linearly with maximum combo, so

$$G_C(c) = \frac{c}{h_C}, \tag{10}$$

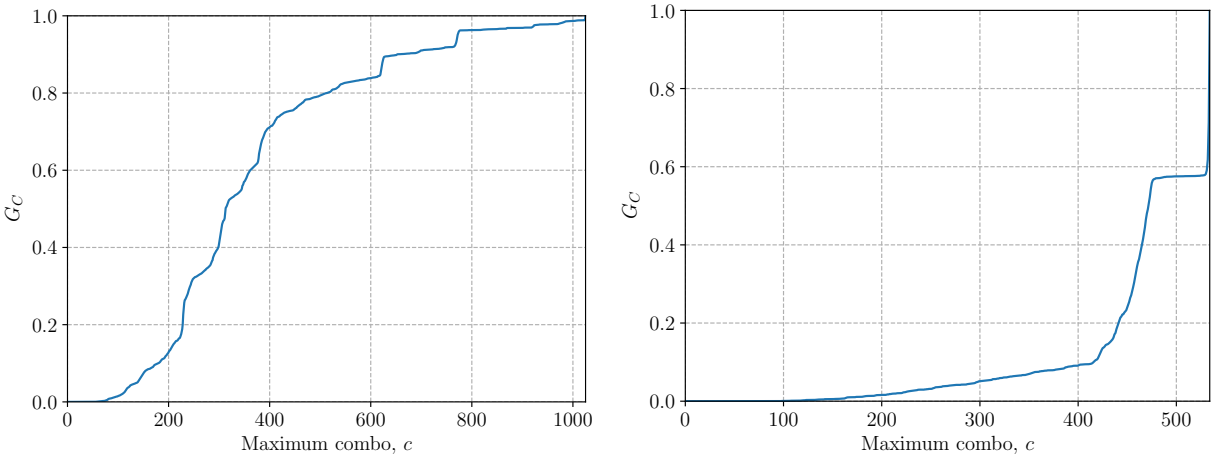
   where h_C is the highest achievable maximum combo on the beatmap (a *full combo*).

Item 2a has the advantage that the aim component of difficulty at different points during the beatmap is built-in (see fig. 3), however, the other options are simpler to implement and require fewer computational resources. In fact, item 2b is a simplification of item 2a where every beatmap is assumed to have uniform aiming difficulty throughout. Note also that, in *osu!standard*, the method of item 2a is sensitive to combo lost due to missed slider-ends; on beatmaps where users typically only miss a few slider-ends, this causes a jump in *pp!* awarded for scores with full combo compared to those with close to full combo. An extreme example can be seen on the right in fig. 3b. Whether this is desired behaviour is a subjective choice.

5 Notes on implementation

5.1 Adding beatmaps

Adding a new beatmap is as simple as adding its corresponding vertices to the comparison graph, and forming the relevant edges as in section 3.1. It is important to note that in order for a new beatmap



(a) Beatmap ID: 104229

(b) Beatmap ID: 1642274

Figure 3: Example cumulative distribution functions for maximum combo, using data from the top 10,000 players (according to *pp*) as of 2019-09-01. These functions are used to calculate the *pp!* awarded via eq. (9). Observe that variation in aiming difficulty throughout each beatmap manifests itself as a varying gradient, with larger gradients representing more difficult to hit notes. Using eq. (10) for G_C instead is equivalent to the assumption that aiming difficulty is uniform throughout each map. Figure 3a shows `Team Nekokan - Can't Defeat Airman (Blue Dragon) [Holy Shit! It's Airman!!]` and fig. 3b shows `Vickeblanka - Black Rover (TV Size) (Sotarks) [Extra]`.

to be assigned a difficulty, it must be contained in at least one edge along with a beatmap that has already been assigned a difficulty. For this reason, we suggest that *qualified* beatmaps are required to have accumulated enough scores to form such an edge (or preferably multiple such edges for a more stable difficulty) before they attain *ranked* status, whereupon importantly their leaderboards are **not** reset.

5.2 Updating beatmaps

From time to time, the accuracy distribution of a particular beatmap may have changed enough over time to merit the recalculation of its accuracy (and maximum combo) distribution, along with the weights of all its containing edges. One method for deciding whether to perform such a *refresh* is to perform a two-tailed one-sample t-test with null hypothesis that the mean accuracy has remained unchanged since the last refresh (or since when the beatmap was first added in the case of no prior refresh). The test statistic is

$$\frac{\bar{x} - \mu_0}{s/\sqrt{n}}, \tag{11}$$

where $\bar{x}$ is the current sample mean accuracy, s is the current sample standard deviation, n is the current number of samples, and μ_0 is the sample mean accuracy from the time of the previous refresh. If the p-value of the test is smaller than some α (which would need to be decided, but we suggest trying $\alpha = 0.01$ initially), then a refresh should be performed. We expect this to be most frequently the case with newer beatmaps, and also any beatmap whose difficulty is deemed to be underestimated by many users (a temporary state which the refresh procedure is designed correct for). When a refresh is performed on a vertex and its containing edges, *pp!* values awarded for all associated scores can be updated to reflect this latest information.

5.3 Beatmap difficulty normalization

The method described in section 3.2 results in a beatmap difficulty vector $\boldsymbol{d} = (d_i)_{i=1}^n$, normalized such that $\sum_{i=1}^n d_i = 1$. We may choose to renormalize the difficulties as $\boldsymbol{d'} = M_d \boldsymbol{d}$, where $M_d > 0$ is some constant, so then $\sum_{i=1}^n d'_i = M_d$. This M_d could be chosen such that the beatmap difficulties match the existing *star ratings* of beatmaps as closely as possible. In the event that k new beatmaps are added to V, and then the unit-sum beatmap difficulties recalculated to be $(d_i)_{i=1}^{n+k}$, we would like to preserve the values of existing difficulties and *pp!* awarded. This is achieved by always using the renormalized difficulties $\boldsymbol{d'}$ in calculating *pp!* using eq. (2) (or eq. (9)), and simply updating the difficulty multiplier by the assignment

$$M_d \leftarrow \frac{M_d}{1 - \sum_{i=n+1}^{n+k} d_i} \tag{12}$$

once the new beatmaps have been added. We can also make the *pp!* awarded for scores most closely resemble that of the current *pp* system by choosing a further constant multiplier, $M_p > 0$, that is applied to the *pp!* function of eq. (2) (or eq. (9)).

5.4 Rare scores

In choosing to only make a comparison between two beatmaps when the number of common players n is at least some threshold value N (see end of section 3.1.1), we implicitly assumed that it was possible for at least N players to pass every beatmap without game modifiers applied. This is, of course, not necessarily the case when the vertex being considered is a beatmap with game modifiers. In order to deal with this situation, we choose to relax the threshold condition to its minimum possible value of $n \geq 2$ for vertices representing beatmaps with any of the potentially difficulty-increasing game modifiers applied (EZ, HD, HR, DT/NC, FL, or corresponding modifiers for other game modes). In practice, relaxing the threshold to a slightly greater value, say $n \geq 4$, may be more effective to safeguard against fluctuations that could otherwise be introduced when calculating edge weights in eq. (6). For any vertex with $n < N$, a *refresh* (see section 5.2) of the accuracy (and maximum combo) distribution, as well as corresponding edge weights, should always be performed before *pp!* is awarded for newly submitted scores.

It is unfortunate that we must sacrifice some reliability in the *pp!* awarded for extremely rare scores in doing this. In the case that a beatmap difficulty still cannot be defined for a particular vertex which has at least one score submitted, perhaps such scores should be showcased in a list on each user's profile. An *impressive score* leaderboard could perhaps be created which ranks users by the number of rare scores they have accumulated.

5.5 Further remarks

- We believe that no *pp!* should be awarded to scores with the NF game modifier, as it offers no real modification to a beatmap compared to the extra computational cost of including it in our comparison graph (see section 3.1.2).
- Using sparse matrices in all computations would be very beneficial to memory efficiency.
- It is of great importance to properly index the database of submitted scores as `(beatmap_id, user_id, enabled_mods)`, for the purpose of constructing the comparison graph.

6 Discussion

We conclude by briefly discussing some of the advantages and disadvantages of the *pp!* system presented. For each disadvantage, we propose a possible (though in some cases perhaps not ideal) solution.

6.1 Advantages

1. As opposed to the current system, *pp!* is **free from arbitrarily chosen measures of performance**, instead utilizing a statistical approach. Because of this, the system is able to reward performance on beatmaps to which it is otherwise difficult to assign good measures of skill.
2. In principle, the *pp!* system **can be applied to any game mode** (with or without combo being accounted for, as discussed in section 4.1). Proper examination of existing accuracy data for *osu!taiko* and *osu!catch*, not only restricted to top ranked players, is required to assess the appropriateness of the Beta distribution assumption.
3. Fosters a **constantly evolving *pp!* meta**, caused by the system attempting to reach a balanced state as users submit scores on beatmaps which have their difficulty temporarily overestimated, or a significant bias in the skill of users with submitted scores, at a particular time. This is in line with *ppy*'s suggestion of more frequent balance changes, while also requiring less human effort.
4. The system is **easily retrofitted with new performance metrics** if, in the future, additional useful data pertaining to scores is stored on the *osu!* game servers.

6.2 Disadvantages (and proposed solutions)

1. Upon initial deployment of *pp!*, there will be a turbulent period for beatmap difficulty values and the amount of *pp!* awarded on each beatmap, as users adjust from being awarded large amounts of *pp* on a relatively small pool of *farm* beatmaps. During this period, there will still be beatmaps that are considered *over-weighted* in terms of the amount of *pp!* they award; users will see visible fluctuations in the *pp!* awarded for their scores over time, even after they have been submitted. For example, current farming beatmaps will initially award too little *pp!*, as many people are currently more conscious of their performance on these beatmaps than others.

   Solution Over time, these undesirable effects should become less and less significant, as the accuracy distributions on all beatmaps stabilise by being played fairly. Indeed, this can be viewed as a long-term advantage (see section 6.1).

2. Certain pathological beatmaps on which even a perfect score does not yield 100% accuracy, e.g. beatmaps with *2B* components (essentially only *Aspire* beatmaps among the ranked category), have their difficulties overestimated, leading to too much *pp!* being awarded.

   Solution The effect is lessened if combo is included using the simplified manor described in item 2b of section 4.1, or is completely resolved if it is possible to redefine accuracy as an *effective accuracy*, which discounts physically impossible notes from the accuracy calculation. Otherwise, these beatmaps would have to be treated manually on a case-by-case basis.

3. Extremely rare scores, such as those set by only a handful of users able to pass a beatmap with certain mods applied, will have a relatively large uncertainty associated to the *pp!* they award.

   Solution Possible solutions were discussed in section 5.4, and involved reducing the number of submitted scores required to form edges in our comparison graph where game modifiers are involved, or even creating a separate leaderboard which counts these impressive scores for each user.

4. The calculation of difficulties in this system is only applicable to beatmaps for which scores may be submitted (those with leaderboards).

   Solution Beatmap difficulties derived using *pp!* will be scaled to closely resemble the currently used *star rating* (see section 5.3), so simply use the existing *star rating* method for all beatmaps without a leaderboard.

Acknowledgements

We would like to thank Dylan Kennett for many useful discussions.

A Derivation of the linear system

The following derivation requires familiarity with the first four pages of [4]. Let $\boldsymbol{A} = (a_{ij})$ be the adjacency matrix of our comparison graph $G = (V, E)$, and define the matrix $\boldsymbol{T} = (t_{ij})$ by

$$\boldsymbol{T} = \boldsymbol{A} - \boldsymbol{A}^{\mathsf{T}}. \tag{13}$$

Let $m = \max_{i,j} t_{ij}$ and let $n = |V|$. Let $\boldsymbol{d} = (d_i)_{i=1}^{n}$ be the priority vector for which we wish to solve, which satisfies the normalisation $\sum_{i=1}^{n} d_i = 1$. The most natural choice of incomplete fuzzy additive preference relation $\boldsymbol{R} = (r_{ij})$ to represent our comparisons is

$$r_{ij} = \frac{1}{2}\left(1 + \frac{1}{m} t_{ij}\right). \tag{14}$$

Let the existence of an undirected edge (an edge in either direction) between the vertices corresponding to i and j be denoted $i \leftrightarrow j$. Following [4], we construct from $\boldsymbol{R}$ its *fitting relation* $\bar{\boldsymbol{R}} = (\bar{r}_{ij})$, where

$$\bar{r}_{ij} = \begin{cases} r_{ij} & i \leftrightarrow j, \\ \frac{1}{2} + \frac{n-1}{2}(d_i - d_j) & \text{otherwise.} \end{cases} \tag{15}$$

Denoting the total degree of vertex i by $k_i = \deg^-(i) + \deg^+(i)$, and applying [4, Theorem 3] to $\bar{\boldsymbol{R}}$,

$$\begin{aligned} nd_i &= \frac{2}{n-1}\left(\sum_j \bar{r}_{ij} - \frac{1}{2}\right) = \frac{2}{n-1}\sum_{\substack{j \\ i\neq j}} \bar{r}_{ij} \\ &= \frac{2}{n-1}\sum_{\substack{j \\ i\leftrightarrow j}} r_{ij} + \sum_{\substack{j \\ i\not\leftrightarrow j \\ i\neq j}} \left[\frac{1}{n-1} + d_i - d_j\right] \\ &= \frac{2}{n-1}\sum_{\substack{j \\ i\leftrightarrow j}} r_{ij} + (n-k_i-1)\left(\frac{1}{n-1} + d_i\right) - \sum_{\substack{j \\ i\not\leftrightarrow j \\ i\neq j}} d_j \end{aligned} \tag{16}$$

Rearranging this equation, we can write

$$(k_i+1)d_i + \sum_{\substack{j \\ i\not\leftrightarrow j \\ i\neq j}} d_j = \frac{n-k_i-1}{n-1} + \frac{2}{n-1}\sum_{\substack{j \\ i\leftrightarrow j}} r_{ij}. \tag{17}$$

Using eq. (14) to write this in terms of t_{ij} gives the system of n linear equations

$$(k_i+1)d_i + \sum_{\substack{j \\ i\not\leftrightarrow j \\ i\neq j}} d_j = 1 + \frac{1}{m(n-1)}\sum_{\substack{j \\ i\leftrightarrow j}} t_{ij} \tag{18}$$

to solve for all d_i. Using the normalisation $\sum_{j=1}^n d_j = 1$, we can rewrite the left summation to give

$$k_i d_i - \sum_{\substack{j \\ i\leftrightarrow j}} d_j = \frac{1}{m(n-1)}\sum_{\substack{j \\ i\leftrightarrow j}} t_{ij} \tag{19}$$

Defining $\boldsymbol{M} = (m_{ij})$ and $\boldsymbol{v} = (v_i)_{i=1}^n$ as in eq. (8), this is simply $\boldsymbol{Md} = \boldsymbol{v}$.

We can also efficiently solve eq. (19) by iteration. Denoting by $d_i(s)$ the priorities at step s and setting the initial values $d_i(0) = \frac{1}{n}$, and ensuring the normalisation $\sum_{i=1}^n d_i(s) = 1$ at each step, we iterate

$$d_i(s+1) = \frac{1}{k_i}\left[\frac{1}{m(n-1)}\sum_{\substack{j \\ i\leftrightarrow j}} t_{ij} + \sum_{\substack{j \\ i\leftrightarrow j}} d_j(s)\right] \tag{20}$$

until the total change between iterations $\delta(s) = \sum_{i=1}^n |d_i(s) - d_i(s-1)|$ is seen to satisfy $\delta(s) < \varepsilon$ for some choice of small ε (a value $\varepsilon = 10^{-4}$ was seen to be sufficiently small during testing).

An intuition for the means by which our algorithm assigns priorities to each vertex of a graph becomes clear upon examining eq. (20). Starting from an initially uniform distribution of priorities, at each step we assign to each vertex the average priority of its neighbours. We then allow priority to *flow* in and out of the vertices according to the edge weights (with some global scaling applied to ensure the normalisation of priorities is conserved). The average such flow into a vertex over its incident edges is added to its new priority value. The final solution is given once a steady state is reached, provided the process converges to such a state.

References

[1] A. Kalmanovich. (2017). "osu-pps," [Online]. Available: https://grumd.github.io/osu-pps (visited on 09/07/2019).

[2] HyperBCS. (2017). "osu! Farmers Market," [Online]. Available: https://pp.ofmc.me/ (visited on 09/07/2019).

[3] M. Bland, *An introduction to medical statistics*. Oxford University Press, 2015, pp. 141–142.

[4] Y. Xu, Q. Da, and H. Wang, "A note on group decision-making procedure based on incomplete reciprocal relations," *Soft Computing*, vol. 15, no. 7, pp. 1289–1300, Jul. 2011, ISSN: 1433-7479. DOI: 10.1007/s00500-010-0662-3.