pp!

Sean A. Adamson

Last updated: October 26, 2020

1 Introduction

It is well-known that the current *osu!* performance indicator, *pp* (or *performance points*), severely misrepresents player skill on many beatmaps. Several attempts to identify those beatmaps which overestimate skill have been made [1], [2], with these so-called *farm maps* dominating best performance leaderboards for the majority of players.

The system in its current form [3] divides a map into different skill classes—*aim*, *speed*, and *accuracy*—and uses certain performance measures (accuracy, maximum combo, and number of missed notes) achieved by a player, along with a beatmap difficulty rating, to award a value to each skill class, eventually combining them into the overall *pp* awarded for setting a score. At the time of its inception, imbalances in this system could easily be overlooked due to its simplicity. As the algorithm has developed, the details of each stage of calculation have naturally become more convoluted and arbitrary. Moreover, the interplay between the calculation of difficulty rating and *pp* awarded on a map is somewhat unnatural. Careful rebalances of the system are becoming a significant undertaking, with their infrequency causing stagnation in the metagame from the perspective of many players.

In this paper, motivated by intuitive principles, we explore an alternative statistical framework within which balance changes may be made. Our proposed system exhibits multiple advantages over that currently used, including that completely automatic rebalances may be pushed at any time (while freedom to make major balance adjustments is preserved), any number of arbitrary performance measures may be easily taken into account, and application to game modes other than *osu!standard* is possible.

Our contributions. We propose a new method (referred to as *pp!*) to determine the performance required of a score. This method is statistical in nature, and makes use only of data presently stored by the *osu!* game servers.

- Our system directly replaces the role of skill classes with performance measures (such as accuracy, combo, and miss count), awarding *pp!* to each based on their estimated probability distributions, scaled by a difficulty rating. These difficulty ratings are calculated based on statistics of each performance measure, using techniques closely related to graph centrality.
- We provide explicit examples of appropriate difficulty rating algorithms, while a large class of alternative algorithms could also be used to comprise major balance adjustments.

- As opposed to the current system, *pp!* is **free from arbitrarily chosen performance functions**, instead utilising a statistical approach alongside simple and intuitive principles. Because of this, the system is able to reward performance on beatmaps to which it is otherwise difficult to assign good measures of skill.
- In our system, automatic rebalances can easily be pushed at any time; farm maps from the previous iteration of the system will be innately weakened in the next. This fosters a **constantly evolving metagame**, in line with *peppy*'s suggestion of more frequent balance changes, caused by the system attempting to reach a balanced state as players submit scores on beatmaps which have their difficulty temporarily overestimated.
- Our system is **easily retrofitted with new performance measures** if, in the future, additional useful data pertaining to scores is stored on the *osu!* game servers.
- In principle, our system **can be applied to any game mode**.

Related works. The current implementation of the *pp* system can be found in the repository [3]. Xu, Da, and Wang [4] describe a method to calculate relative weightings from a *fuzzy preference relation*, from which we derive our primary difficulty calculation algorithm. An alternative difficulty algorithm is based on the *PageRank* algorithm of Brin and Page [5].

Organisation of the paper. We first discuss calculation of the performance of scores given the *difficulties* of their beatmaps in section 2, using the accuracy performance metric as an example. In section 3, we show a process to determine these difficulties statistically. We then discuss a straightforward generalisation to include additional performance measures (such as combo and miss count) in section 4. Some remarks about the implementation of our method are then given in section 5. In section 6, we conclude with some possible disadvantages of our system, and propose possible solutions to each of these shortcomings.

2 Performance calculation

In this section, we describe our method of determining the performance for a score, given that we have already quantified the difficulty of its corresponding beatmap. We begin by giving an intuitive notion of what is meant by *difficulty* in this context, deferring its quantitative mathematical construction until section 3. For simplicity, we will only consider a single performance measure (namely *accuracy*), however, the generalisation to many performance metrics is intuitive and will be described in section 4.

2.1 Defining beatmap difficulty

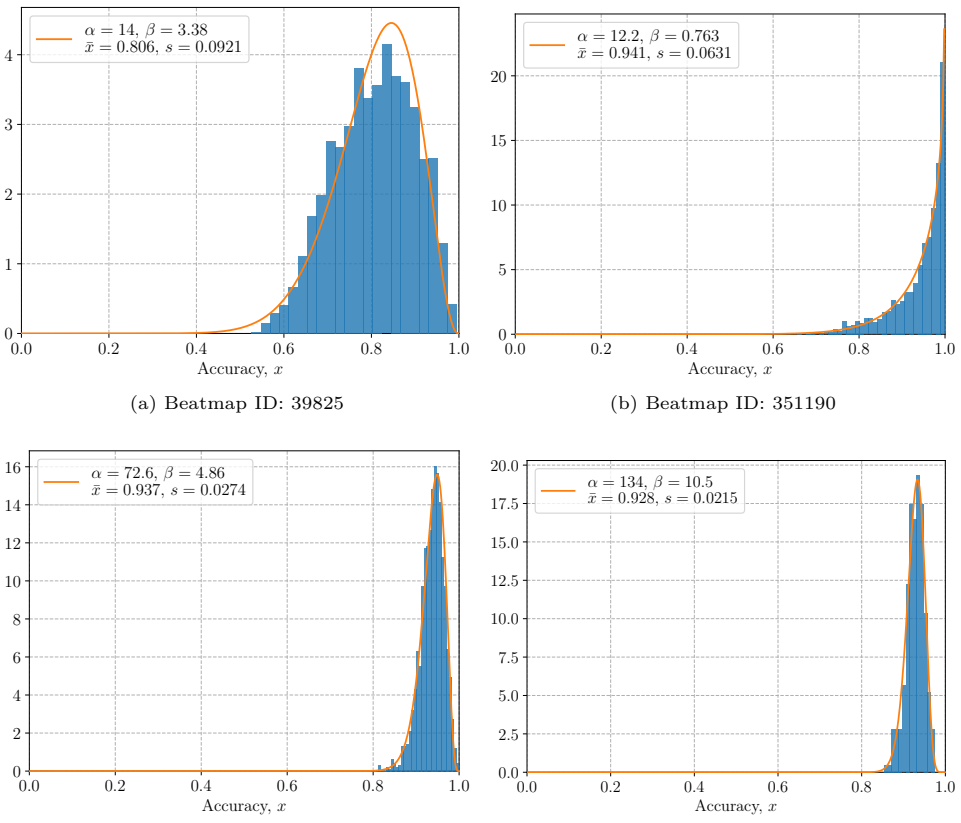
We denote the *difficulty* of a beatmap by a real number $d > 0$, where larger values correspond to harder beatmaps. The suitable notion of difficulty for our purposes is a representation of "the hardness of a beatmap, as perceived by a hypothetical average player". In other words, it is supposed to represent how difficult a randomly chosen player would be expected to perceive a beatmap.

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. Estimating these shape parameters using the method of moments, we can instead parametrise this distribution in terms of only its sample mean $\bar{x}$ and sample variance s^2 by the substitutions

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

Hence, it is simple to extract a Beta distribution which models the accuracy distribution of any beatmap. One could also use a more sophisticated estimation technique such as *maximum likelihood* estimation, however, the additional processing required for this must be weighed against the finer estimation it provides. Some example accuracy distributions and their corresponding Beta distributions are depicted in fig. 1, indicating that our model using Beta distributions is indeed a reasonable one.



(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 estimated 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]`.

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

1. The *pp!* awarded for a score with median 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 as a function of accuracy should be proportional to F.

These assertions uniquely determine that the accuracy *pp!* awarded should be given by

$$2d \cdot F(x). \tag{2}$$

We will still combine player scores together in the same way *pp* does: as an (unnormalised) weighted average. Because of this, we can equally well multiply the *pp!* function for all beatmaps by any fixed positive constant without affecting player or score *pp!* values. Hence, by choosing to introduce a factor of $1/2$ in eq. (2), we will unambiguously define our *pp!* function π as

$$\pi(x) = d \cdot F(x). \tag{3}$$

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

3 Beatmap difficulty

This section is dedicated to a procedure for qualitatively determining a suitable beatmap difficulty, as defined intuitively in section 2.1. To do this, we first construct a directed graph of beatmaps, with edges (where they exist) representing the relative imbalance in difficulty of a pair of maps, according to a desired performance metric (which we will again take to be accuracy in this section for simplicity). We will then discuss how game modifiers can be handled within our framework, and conditions which can be applied to reduce the number of edge weight computations required. Finally in this section, we will describe some explicit examples of graph algorithms which can be used to model beatmap difficulties.

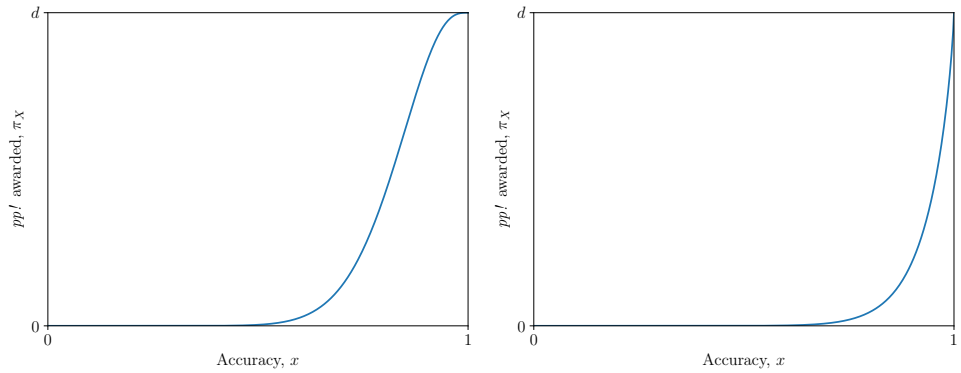
3.1 Comparison graph

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.

3.1.1 Edge weights

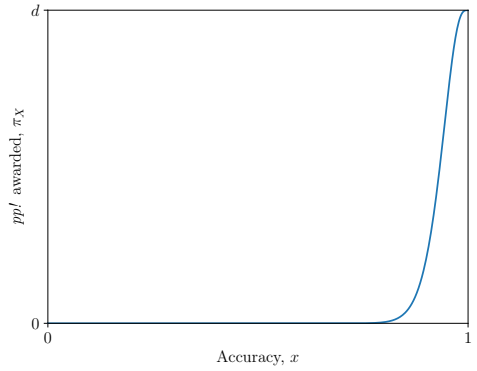
We assign weights to the directed edges in E to represent each difficulty comparison by defining a function $t\colon E \to \mathbb{R}$ as follows.

Let $(x, y) \in E$ and let Q and R be the sets of players who have submitted scores on beatmaps x and y respectively. Define the common set of n players $P \equiv Q \cap R$ and let $(a_i)_{i=1}^n$ and $(b_i)_{i=1}^n$ be lists of accuracy values for all scores which are submitted by players in P on beatmaps x and y respectively, where a player is represented in both lists by the same index. Now define accuracy differences for each player $(c_i)_{i=1}^n = (a_i - b_i)_{i=1}^n$. We also choose to also assign to each of these

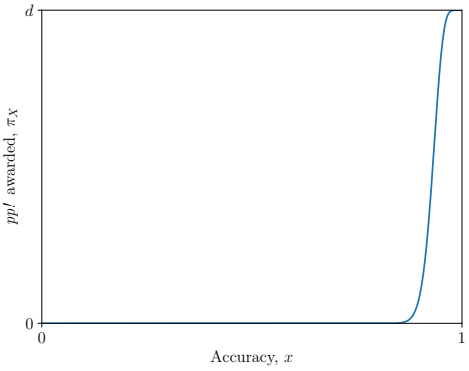


(a) Beatmap ID: 39825

(b) Beatmap ID: 351190



(c) Beatmap ID: 960875



(d) Beatmap ID: 1192895

Figure 2: A selection of *pp!* functions as in eq. (3), corresponding to the PDFs shown in fig. 1. The maximum available *pp!* for a beatmap is its difficulty rating d.

accuracy differences a reliability value, given by a function decaying in the time elapsed between the submission of their corresponding scores. Let s_i and t_i be the submission times, and the submission time difference $\delta_i = |s_i - t_i|$. One choice for the weight function (having also the desirable property that it smoothly attains zero after finite time) is given by

$$w_i = \begin{cases} \exp\left[-\frac{2\delta_i}{\tau - \delta_i}\right] & \text{if } \delta_i < \tau, \\ 0 & \text{otherwise,} \end{cases} \tag{4}$$

where τ is some time constant over which players improve at the game (a reasonable value is $\tau = 36\,\text{weeks}$). This function is depicted graphically in fig. 3.

In all the following, it is not necessary that the weights be normalised. The weighted mean and (corrected) weighted standard deviation of the accuracy differences are respectively

$$\bar{c} = \frac{\sum_{i=1}^{n} w_i c_i}{V_1}, \tag{5a}$$

$$\sigma_c = \frac{\sum_{i=1}^{n} w_i (c_i - \bar{c})^2}{V_1 - V_2/V_1}, \tag{5b}$$

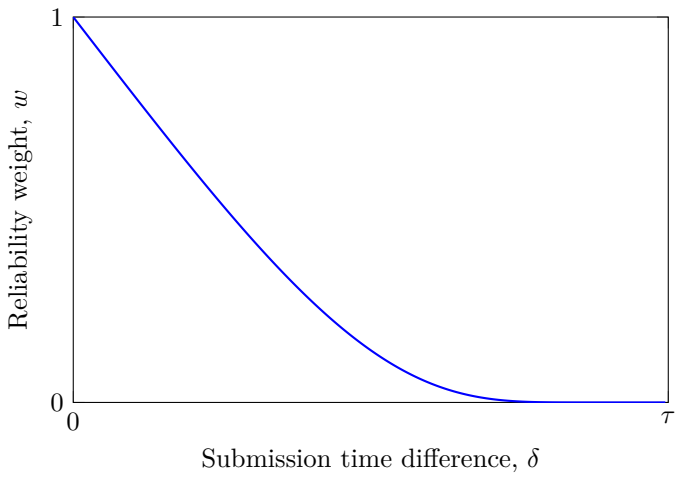


Figure 3: The reliability weight we assign to performance measure differences for scores submitted by a particular player. Scores submitted at the same time by a player have the maximum reliability of 1, while scores with greater time separations have lower reliability. The chosen function is decreasing, convex, and decays smoothly to 0 reliability at a threshold time difference τ which can be chosen appropriately.

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

$$\sigma_{\bar{c}} = \frac{\sqrt{V_2}}{V_1}\sigma_c. \tag{6}$$

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

$$t\colon (x, y) \mapsto \frac{\bar{c}}{\sigma_{\bar{c}}}. \tag{7}$$

This can be thought of as a reliability-weighted version of the t-statistic that is used in dependent t-tests for paired samples.

3.1.2 Game modifiers

In the edge weighting process of the previous section 3.1.1, no mention of game modifiers (or *mods*) was made. For each *osu!standard* beatmap, there exist a further 35 possible combinations of mods that may be applied (from the individual mods `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 $|E| \sim |V|^2$, the initial computation of all edge weights would be significantly more impractical. A partial solution is provided by restricting which (map, mod) pairs we allow edges to be formed between. We will only permit edges to be formed between two vertices in the following cases:

1. The vertices do not both have mods applied.
2. The vertices belong to the same *beatmapset*.

Let us denote the number of different allowed combinations of mods (including no mods) applied by M. Since in practice the number of edges formed due to item 2 is

very small compared to that of item 1, this effectively results in the final comparison graph being constructed as the (non-disjoint) union of M smaller graphs.

While we cannot improve the quadratic scaling of our edge count through the restrictions made, let us examine the constant factor by which the edge count is reduced. If n is the number of unique beatmaps (excluding mods), then the number of edges without restrictions is approximately $(Mn)^2/2$. With the restriction of item 1, the edge count becomes approximately $(M - \frac{1}{2})n^2$. Therefore, the number of edges required is reduced by a factor of approximately $M^2/(2M - 1)$. For relatively large M, this means a reduction by a factor of approximately $M/2$, equal to 18 in the case of *osu!standard*.

3.1.3 Edge conditions

Rather than the comparison graph being complete, it is sensible to form only edges with a large enough sample of shared players. Specifically, we form edges only if the number of common players shared between the two beatmaps (with strictly positive reliability weights) satisfies $n \geq N$ for some threshold $N \geq 2$. A reasonable value is $N = 30$, however, as *pp!* does not carry the demand for confidence usually present in statistical tests, experiments to determine the efficacy of *pp!* with lower N could also be performed.

While restricting to $N \gg 1$ makes the most sense statistically, it has the disadvantage of always forbidding edges where one of the beatmaps has very few scores submitted (such that the number of players shared between vertices is then necessarily small). This can of course naturally occur among the very hardest maps, that few players are able to pass. We defer further discussion of this issue until section 5.4, where we will provide two possible solutions.

3.2 Difficulty values

In this section, we exhibit a method (derived fully in appendix A) for finding beatmap difficulties from the comparison graph we constructed in the previous section 3.1, based on the preference relation algorithm of [4]. We also provide some alternative examples of algorithms which could be used for difficulty calculation, based on the *PageRank* algorithm [5].

Define the matrix $\boldsymbol{T} = (t_{ij})$ by

$$\boldsymbol{T} = \boldsymbol{A} - \boldsymbol{A}^{\mathsf{T}}, \tag{8}$$

where $\boldsymbol{A} = (a_{ij})$ is the (weighted) adjacency matrix of our comparison graph $G = (V, E)$. We use the convention that a nonzero element a_{ij} of an adjacency matrix represents the weight of an edge from j to i. Note that our definition of $\boldsymbol{T}$ is quite natural in the sense that, for our comparison graph, flipping the direction of an edge is equivalent to instead flipping the sign of its weight. Hence, $\boldsymbol{T}$ is an adjacency-like matrix (usually called a *skew-adjacency* matrix) encoding this property through its skew-symmetry: if an edge exists between two vertices i and j, we may always treat that the edge is formed in the direction $i \leftarrow j$ with weight t_{ij}. It is for this reason that we will sometimes use the notation $t_{i \leftarrow j}$ for graph weights. The vector of difficulties which we wish to determine is denoted by $\boldsymbol{d}$, and its elements $d_i > 0$ include also the difficulties of beatmaps with mods applied (which are also vertices in our graph as in section 3.1.2). These difficulty values are defined to satisfy the normalisation criterion $\sum_i d_i = d^*|V|$, such that the mean difficulty is d^* (which can be chosen appropriately).

3.2.1 Preference-based algorithm

Let $m = \max_{i,j} t_{ij}$, let $n = |V|$ be the order of our comparison graph, 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

$$m_{ij} = \begin{cases} k_i & i = j, \\ -1 & i \leftrightarrow j, \\ 0 & \text{otherwise}, \end{cases} \tag{9a}$$

$$v_i = \frac{nd^*}{m(n-1)} \sum_{\substack{j \\ i \leftrightarrow j}} t_{i \leftarrow j}. \tag{9b}$$

In practice, this linear system will be very large and, despite its sparsity, may prove to be very resource intensive to solve. Fortunately, it is possible to solve eq. (9) by iteration. Denoting by $d_i(s)$ the priorities at step s and setting the initial values $d_i(0) = d^*$, we iterate

$$d_i(s+1) = \frac{1}{k_i} \left[\frac{nd^*}{m(n-1)} \sum_{\substack{j \\ i \leftrightarrow j}} t_{i \leftarrow j} + \sum_{\substack{j \\ i \leftrightarrow j}} d_j(s) \right] \tag{10}$$

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

An intuition for the means by which the algorithm assigns difficulty values to each vertex becomes clear upon examining eq. (10). Starting from an initially uniform distribution of difficulties, at each step we assign to each vertex the average difficulty value of its neighbours. We then allow difficulty to *flow* in and out of the vertices according to the edge weights (with some global scaling applied to ensure the normalisation of difficulties is conserved). The average such flow into a vertex over its incident edges is added to its new difficulty value. The final solution is given once a steady state is reached, provided convergence to such a state is possible.

3.2.2 PageRank-based algorithms

PageRank is a graph centrality algorithm originally designed for ranking web pages to be returned by search engines. The most important pages are placed at the top of a search query. Written using our notation, where we are interested in beatmap difficulty rather than the importance of web pages and use our normalisation convention $\sum_i d_i = nd^*$, it works by iterating

$$d_i(s+1) = (1-\alpha)d^* + \alpha \sum_j \frac{a_{i \leftarrow j}}{D^+(j)} d_j(s) + \alpha d^* \sum_{\substack{j \\ \deg^+(j)=0}} d_j(s), \tag{11}$$

where $D^+(j) = \sum_i a_{i \leftarrow j}$ is the sum of the outgoing weights of j, and $\alpha \in (0,1)$ is a parameter known as the *damping factor* (typically set to around 0.85).

We can also modify the PageRank algorithm of eq. (11) to perhaps suit better our difficulty rating purposes:

- We replace the sum of outgoing weights $D^+(j)$ of vertex j in the second term with the sum of incoming weights $D^-(i) = \sum_j a_{i \leftarrow j}$ of vertex i. This places a normalisation on the flow of difficulty received by a vertex from its in-neighbours, rather than each vertex dividing up the difficulty it provides to its out-neighbours; the difficulty of a beatmap does not scale with its popularity.
- We ignore the final term, which exists to make sink vertices act as uniform sources to all vertices in the graph. In the web surfing context this is the assumption that, upon arriving at a page with no outgoing links, a surfer will jump to a page from the web at random.

These adjustments result in our modified PageRank iteration

$$d_i(s+1) = (1-\alpha)d^* + \frac{\alpha}{D^-(i)} \sum_j a_{i \leftarrow j} d_j(s), \tag{12}$$

where any divergent terms with $D^-(i) = 0$ are taken to vanish. Difficulty values resulting from this modified algorithm are no longer inherently normalised, and so the normalisation $\sum_{i=1}^n d_i(s) = nd^*$ must be ensured at each step of iteration.

For PageRank-based algorithms, we must ensure all edge weights of our comparison graph are non-negative, such that the resulting PageRank matrix is guaranteed to have a unique largest real eigenvalue. We can trivially bring our comparison graph to this form by flipping the direction of all edges whose weights are negative, and flipping the sign of these weights.

4 Other performance measures

Thus far, we have considered only accuracy as a measure of player performance on beatmaps. While this could be reasonable for those game modes where skill is tied very closely to accuracy, we would ideally like to be able to use other performance measures. In particular, since a large component of the *osu!standard* mode has to do with cursor aim, accuracy alone may not be enough to capture a player's performance on a beatmap. In this section, we describe a generalisation of the techniques exhibited so far to multiple performance measures.

We can of course use exactly the same techniques as in section 3 to find beatmap difficulty values based on other performance measures, such as miss count and combo. Edges are formed with as many weights as performance measures used, and only when any edge formation conditions imposed (see section 3.1.3) hold for all measures. The general strategy we take to find the *pp!* awarded for each performance measure extends that of section 2.2 to an arbitrary measure. Let X denote a particular performance measure for scores set on a particular beatmap which has difficulty d_X. The process to find the performance measure *pp!* for a score as a function of the performance measure value attained is simply:

1. Identify a class of probability distributions which model the performance measure X for every map.
2. Estimate the shape parameters of the distribution for the particular map considered.

3. Award *pp!* for X according to the function

$$\pi_X(x) = d_X F_X(x), \tag{13}$$

where F_X is the cumulative distribution function for the probability distribution with the estimated parameters.

Note that, for both difficulty calculation and performance calculation, we require that greater values of a performance measure indicate a better performance. This is important, for example, if we wish to use miss count as a measure; we should instead use the proportion of notes *not* missed.

We define the overall *pp!* awarded as the product of the *pp!* for each individual performance measure. For example, if A is accuracy, C is combo, and H is the proportion of notes hit, then the overall *pp!* function for a map with respective difficulties d_A, d_C, and d_H is given by

$$\begin{aligned} \pi(a, c, h) &= \pi_A(a) \cdot \pi_C(c) \cdot \pi_H(h) \\ &= d_A F_A(a) \cdot d_C F_C(c) \cdot d_H F_H(h) \\ &= d \cdot F(a, c, h), \end{aligned} \tag{14}$$

where for the final equality we define an overall difficulty

$$d = d_A d_C d_H, \tag{15}$$

and the corresponding function F by

$$F(a, c, h) = F_A(a) \cdot F_C(c) \cdot F_H(h). \tag{16}$$

We see from eq. (16) that our definition for overall *pp!* as the product of the *pp!* for each performance measure is natural. This is because, in the ideal case where the performance measures considered are mutually independent, F is precisely their joint cumulative distribution function. For this reason, in order to best model performance, we should aim to choose a set of performance measures that are as close to being mutually independent as possible. In general, for a random vector of N performance measures $\boldsymbol{X} = (X_1, \ldots, X_N)$, the overall difficulty rating of a map is given by

$$d = \prod_{i=1}^{N} d_i \tag{17}$$

and the overall cumulative distribution function by

$$F(\boldsymbol{x}) = \prod_{i=1}^{N} F_i(x_i), \tag{18}$$

resulting in an overall *pp!* function

$$\pi(\boldsymbol{x}) = d \cdot F(\boldsymbol{x}). \tag{19}$$

Using only data that is currently recorded on the *osu!* servers, a reasonable set of performance measures for use with *osu!standard* is

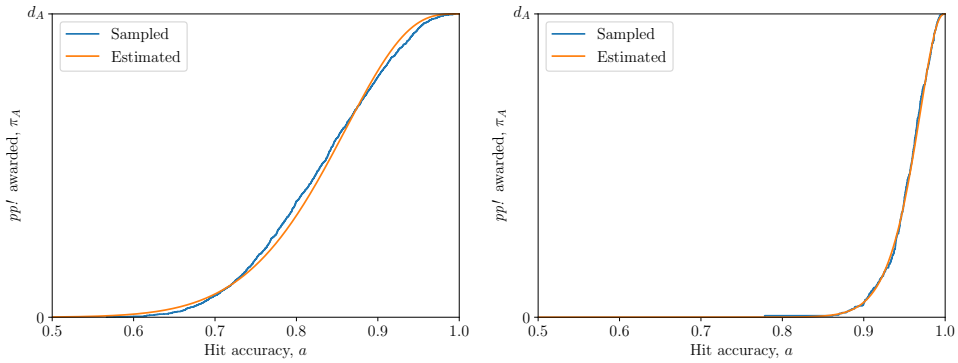
- **Accuracy** calculated only for notes that were not missed.

- **Combo**; the maximum combo attained over the course of a map.
- **Miss count** in the form of the **proportion of notes *not* missed**.

Note that we choose not to include missed notes in the accuracy measure, for the sake of independence from the miss count measure. We now suggest distributions to model each of these performance measures.

4.1 Accuracy (excluding misses)

We model the "hit accuracy" (accuracy on notes not missed) by the same distribution as was used for regular accuracy in section 2.2. For these beta distributions, we again estimate the shape parameters from eq. (1) or perform the more expensive maximum likelihood estimation. Explicitly, we assume that hit accuracy $A \sim \text{Beta}(\alpha, \beta)$ where the shape parameters α and β are to be appropriately estimated from the scores submitted on a beatmap with particular mods enabled. Figure 4 compares sampled hit accuracy data to our estimated distributions on some example beatmaps.



(a) Beatmap ID: 39825

(b) Beatmap ID: 960875

Figure 4: Estimation of the *hit accuracy* (accuracy excluding misses) cumulative distribution functions, scaled by difficulty to give *pp!* functions, for some example beatmaps (without mods enabled). Figure 4a shows the beatmap `IOSYS - Marisa wa Taihen na Kanbu de Tomatte Ikimashita (DJPop) [Love]`, while fig. 4b shows `DECO27 - Ghost Rule (Awaken) [Mayday]`.

4.2 Combo

Maximum combo is the least reliably modelled performance measure of those we consider, as sampled data tends to be grouped around sections of beatmaps are more difficult in terms of cursor aim. Nonetheless, we will attempt to find a loose model for combo to be a component of our *pp!*, as it has traditionally been a measure that players are familiar with contributing towards the *pp* they are awarded for scores.

We suggest to model the combo C on a map as a geometric distribution $C \sim \text{Geo}(p)$ with support $\{0, 1, \dots\}$, where we can estimate the success probability parameter by the maximum likelihood estimate

$$p = \frac{1}{1 + \bar{c}}. \tag{20}$$

This geometric distribution models the number of notes hit until a miss occurs. Strictly, it is only an appropriate model in situations where the following conditions hold:

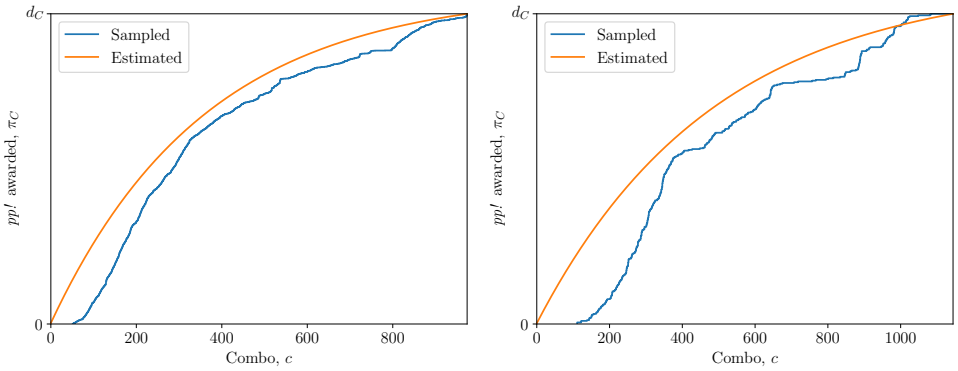
- Whether or not a note is missed is independent of other notes.
- The probability of a miss, p, is the same for all notes.

Neither of these conditions hold for our case, however, as we are searching for only a loose model of combo (which will allow it to be approximately accounted for in the awarding of *pp!*) we choose to use this model regardless.

We note that a geometric distribution is a discrete probability distribution supported on an infinite set, but there are only finitely many notes in any beatmap. Because of this, although not strictly necessary, we choose to truncate the geometric CDF to be supported on $\{0, \ldots, c_{\max}\}$, where $c_{\max}$ is the maximum combo attainable on the beatmap, by appropriately scaling it to attain a value 1 at $c_{\max}$. That is, we make the modification

$$F_C(c) \leftarrow \frac{F_C(c)}{F_C(c_{\max})}. \tag{21}$$

This modified F_C is no longer exactly the CDF for a geometric distribution, however, it is a valid CDF close to that of a geometric distribution, and we nonetheless choose it to model the distribution of combo. Real examples using this modified CDF are shown in fig. 5.



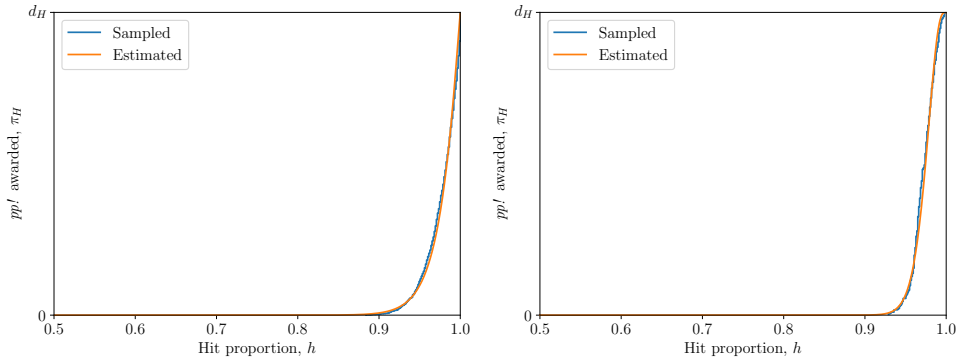
(a) Beatmap ID: 39825

(b) Beatmap ID: 960875

Figure 5: Modified estimation of the *maximum combo* cumulative distribution functions, scaled by difficulty to give *pp!* functions, for some example beatmaps (without mods enabled).

4.3 Miss count (hit proportion)

As mentioned previously, we will take into account the number of notes missed on a map by equivalently considering the proportion of all notes hit, which we will call "hit proportion". As with accuracy and hit accuracy, we model hit proportion by $H \sim \text{Beta}(\alpha, \beta)$, again estimating the shape parameters α and β by the method of moments or maximum likelihood estimation. Estimation for example beatmaps are shown in fig. 6.



(a) Beatmap ID: 39825

(b) Beatmap ID: 960875

Figure 6: Estimation of the *hit proportion* (proportion of notes not missed) cumulative distribution functions, scaled by difficulty to give *pp!* functions, for some example beatmaps (without mods enabled).

5 Notes on implementation

We now give some details relevant to a practical implementation of our *pp!* system. Performing automated rebalances of the *pp!* system is first described, wherein maps whose distributions have changed significantly have their difficulty values and distribution parameters recalculated. We then examine the addition of new beatmaps to the system, as would occur when maps undergo the ranking procedure. Next, we tackle the related issue of scaling difficulty and *pp!*; we may wish for difficulty ratings and awarded *pp!* values to be presented to users in a form resembling in magnitude those already present in the current *pp* system, for the sake of easier readability. We then suggest possible solutions to the problem of certain beatmaps having too few scores submitted from which to gather meaningful statistics. We conclude the section with a list of short further remarks on implementation.

5.1 Automated rebalancing

In any *osu!* performance rating system, there will exist beatmaps whose assigned difficulty rating strays from the difficulty as perceived by players. For our *pp!* system, such cases will mostly arise in the form of certain beatmaps not having being played by a representative set of players appropriate for the true difficulty of the beatmap. This could, for example, occur on maps which currently have their difficulty underestimated by the *pp* system, or perhaps on newly ranked maps (see section 5.2). An advantage of our system is that, as beatmaps with overestimated difficulty are identified and "farmed" by the playerbase, the inflated popularity such maps gain will result in better data being contributed for those maps. This in turn means that, if the difficulty of such maps was then recalculated, they would be awarded a difficulty rating more representative of their true difficulty. Of course, one could also make changes to the *pp!* algorithm itself, resulting in more major changes to the way *pp!* is awarded, but this will not be the focus of this section. To avoid confusion, we will refer to the automated process described above as a *refresh*, and reserve the term *rebalance* for major changes to the algorithm itself.

A refresh consists of multiple stages:

1. Updating weights of edges in the comparison graph, and adding new edges as appropriate.
2. Adding new edges to the comparison graph as appropriate.
3. Recalculating beatmap difficulty ratings from the comparison graph.
4. Recalculating the shape parameters for performance measure distributions.
5. Recalculating *pp!* for all scores from the new difficulties and distributions.

Although a full update of all graph edges is possible (as would have to be done for an initial release), for efficiency we suggest that only maps whose performance measure distributions have changed significantly be considered in item 1. For those maps, only their corresponding vertices in the comparison graph need have their incident edges updated or added to.

To identify maps to be included in item 1 of a refresh, one method is to perform a two-tailed one-sample t-test on each performance measure, with null hypothesis that the current mean is equal to the sample mean from the previous refresh. The test statistic is

$$\frac{\bar{x} - \mu_0}{s/\sqrt{n}}, \tag{22}$$

where $\bar{x}$ is the current sample mean, s is the current sample standard deviation, n is the number of samples, and μ_0 is the sample mean from the time of the previous refresh. If the p-value for any of the tests is smaller than some significance level (typically 0.1, 0.05, or 0.01), then the vertex should have its edges updated.

5.2 Adding beatmaps

In principle, adding new beatmaps is as simple as adding corresponding vertices to the comparison graph, forming edges as in section 3.1, and estimating the shape parameters of the performance measure distributions. In order for a new map to be assigned a reliable difficulty rating, it must be connected to other vertices. Additionally, enough scores must have been submitted in order to give reliable performance measure statistics. For these reasons, we suggest that *qualified* beatmaps are required to have accumulated enough scores to form a chosen number of edges before they attain the *ranked* status. Upon transitioning from qualified to ranked status, submitted scores should not be deleted; whether or not they are then awarded *pp!* after the map is ranked is matter up to debate.

Before a map is transitioned to the ranked category, its difficulty rating must be obtained from the comparison graph (see the algorithms of section 3.2). In order to reduce the number of times the difficulty rating algorithm must be executed, we suggest that maps be given the ranked status in daily batches. Thanks to our chosen normalisation of difficulties $\sum_i d_i = nd^*$ (which scales linearly with the comparison graph order n) we need not make adjustments to difficulty values found for beatmaps added since the last refresh.

5.3 Scaling difficulty and performance

Due to our normalisation, the mean beatmap difficulty value is d^*. Using our overall *pp!* function eq. (14), we thus have that *pp!* values for individual scores are typically smaller than d^*. In order to produce user friendly values for difficulty

ratings and *pp!* values, we set d^* appropriately, and introduce a constant multiplier $M > 0$ chosen such that *pp!* closely matches current *pp* values. All *pp!* awarded for scores are multiplied by M. That is, the overall *pp!* function in eq. (19) for a map is instead

$$\pi(\boldsymbol{x}) = Md \cdot F(\boldsymbol{x}). \tag{23}$$

The constant M represents twice the desired median *pp!* awarded to scores submitted on beatmaps of the mean difficulty rating d^* (including those with human-passable mods enabled). We suggest setting $d^* = 5$, and as an initial guess trying $M = 100$.

5.4 Rare scores

In section 3.1.3, to ensure more reliable statistical comparisons, we suggested the restriction of edge formation between comparison graph vertices to when the number of common players between two vertices (with strictly positive reliability weights) is at least some threshold value N. However, this restriction also presents a disadvantage: it necessarily excludes those maps from forming any edges that are so extremely difficult that only a handful of (fewer than N) players are able to pass. There are a few main solutions to this problem, which we outline here beginning with the most straightforward.

Threshold reduction. The simplest solution is to reduce the threshold N to as low as feasible, say in the region $2 \leq N \leq 9$. While making individual statistical comparisons between vertices with such low sample sizes will lead to larger fluctuations in the difficulty ratings of affected maps, the problem may be smoothed out somewhat by the large number of incident edges to these vertices which is a consequence of them being passed by very experienced players.

A slightly more complicated option is to reduce the threshold as above, but only for comparisons that include at least one beatmap with mods enabled. The logic here is that beatmaps allowed into the ranked category should, with no mods enabled, almost always be passable by a non-negligible number of players. This extra condition may, however, be an unnecessary one: under such logic, almost all beatmaps with no mods enabled should have a large number of submitted scores, and thus bypass the regular large threshold value regardless.

No-fail scores. The most sophisticated solution comes in the form of accepting certain scores submitted using the `NF` mod to be included in vertex comparisons. As `NF` scores do not award *pp!* equal to that of regular scores, there could be less incentive for players to submit scores played to the best of their ability using this mod, and thus we must be careful to avoid possible exploitation of out statistical system in this case.

In a pair of vertices to be compared, if a vertex does not have enough scores submitted to exceed the threshold of N shared players, we allow `NF` scores (in descending order of either score or performance measure) to be included. We limit the amount of `NF` scores included to the least of:

- The top fraction (say 10%) of the total number of `NF` scores submitted on the map, rounded up to the nearest integer.
- The amount of `NF` scores necessary such that the shared player count attains the threshold N.

In this way, we avoid the possible exploit of a group of players spamming submission of low performance scores on a difficult beatmap, in order to boost the *pp!* awarded to the few players who are able to pass the map.

Rare scores list. In the case that a beatmap difficulty still cannot be defined, perhaps scores submitted for such vertices should be displayed in a list on each player's profile. A *rare score* leaderboard could also be created, which globally ranks players by the number of rare scores they have accumulated. Such features would complement either of the two previous solutions to rare scores, and would also allow lower skill players (who cannot achieve a first-place rank on any map) to showcase some interesting scores they have submitted as part of their player profiles.

5.5 Further remarks

Here we list some short further remarks on *pp!* implementation.

- It is important that scores with the `NF` mod enabled are not be awarded the full *pp!* amount, so that players are incentivised to submit pass scores, which contribute towards comparison graph statistics.
- Certain mods should be counted as equivalent for the purposes of comparison graph vertices and performance measure distribution estimation.
  - The mods `NC` and `DT` are equivalent.
  - Beatmaps with mods `SD` or `PF` enabled are counted as equivalent to those without these mods enabled.
- A statistical system such as ours could more fairly award *pp!* to atypical maps which have thus far eluded the ranking criteria for maps. Scores on such maps with leaderboards, such as those in the *loved* category, could be awarded *pp!* fairly if so desired.

6 Discussion

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

1. Upon initial deployment, there will be a turbulent period for beatmap difficulties (and the *pp!* awarded on each map) as players adjust to a system which rewards players on maps which are not part of the relatively small pool of beatmaps which previously awarded excessive amounts of *pp*. In the new system, there will still exist beatmaps that are considered over-weighted, however, exceptionally over-weighted beatmaps should only be present in the period straight after initial deployment, before any subsequent refreshes have occurred. Conversely, current farm maps will initially award too little *pp!*, as many players are currently more conscious of their performance on these beatmaps than others.

   Solution. After some initial refreshes have taken place, these undesirable effects will become much less prominent, as the accuracy distributions on

all exceptional beatmaps. Indeed, this could even be viewed as a long-term advantage.

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

   Solution. This effect is lessened by the inclusion of combo as a performance measure, as already described in section 4. It could be completely resolved (although this is not necessary) if it were possible to redefine accuracy as an *effective accuracy*, which discounts physically impossible to hit parts of a beatmap from the accuracy calculation.

3. It may not be possible to define difficulties for extremely rare scores, such as those submitted on beatmaps only passable by a handful of players.

   Solution. Reasonable solutions were already discussed in section 5.4.

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 beatmaps without a leaderboard (and inside the beatmap editor).

Acknowledgements

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

A Preference-based difficulty algorithm

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

$$\boldsymbol{T} = \boldsymbol{A} - \boldsymbol{A}^{\mathsf{T}}. \tag{24}$$

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

$$r_{ij} = \frac{1}{2}\left(1 + \frac{1}{m} t_{ij}\right). \tag{25}$$

Note that, in terms of preference, flipping the direction of an edge is equivalent to instead flipping the sign of its weight. It is for this reason that we may sometimes write $t_{i \leftarrow j}$ for t_{ij}, and assume that i is always the *target* vertex of its incident edges.

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

$$\bar{r}_{ij} = \begin{cases} r_{ij} & i \leftrightarrow j, \\ \frac{1}{2} + \frac{n-1}{2}(d_i - d_j) & \text{otherwise.} \end{cases} \tag{26}$$

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

$$\begin{aligned} nd_i &= \frac{2}{n-1}\left(\sum_j \bar{r}_{ij} - \frac{1}{2}\right) = \frac{2}{n-1}\sum_{\substack{j \\ i\neq j}} \bar{r}_{ij} \\ &= \frac{2}{n-1}\sum_{\substack{j \\ i\leftrightarrow j}} r_{ij} + \sum_{\substack{j \\ i\not\leftrightarrow j \\ i\neq j}} \left[\frac{1}{n-1} + d_i - d_j\right] \\ &= \frac{2}{n-1}\sum_{\substack{j \\ i\leftrightarrow j}} r_{ij} + (n - k_i - 1)\left(\frac{1}{n-1} + d_i\right) - \sum_{\substack{j \\ i\not\leftrightarrow j \\ i\neq j}} d_j \end{aligned} \tag{27}$$

Rearranging this equation, we have

$$(k_i+1)d_i + \sum_{\substack{j \\ i\not\leftrightarrow j \\ i\neq j}} d_j = \frac{n-k_i-1}{n-1} + \frac{2}{n-1}\sum_{\substack{j \\ i\leftrightarrow j}} r_{ij}. \tag{28}$$

Using eq. (25) to write this in terms of $t_{i\leftarrow j}$ gives the system of n linear equations

$$(k_i+1)d_i + \sum_{\substack{j \\ i\not\leftrightarrow j \\ i\neq j}} d_j = 1 + \frac{1}{m(n-1)}\sum_{\substack{j \\ i\leftrightarrow j}} t_{i\leftarrow j} \tag{29}$$

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

$$k_i d_i - \sum_{\substack{j \\ i\leftrightarrow j}} d_j = \frac{1}{m(n-1)}\sum_{\substack{j \\ i\leftrightarrow j}} t_{i\leftarrow j}. \tag{30}$$

We may wish to instead use the normalisation $\frac{1}{n}\sum_{j=1}^n d_j = d^*$, so that the mean priority is d^*. In this case, the equation is instead

$$k_i d_i - \sum_{\substack{j \\ i\leftrightarrow j}} d_j = \frac{nd^*}{m(n-1)}\sum_{\substack{j \\ i\leftrightarrow j}} t_{i\leftarrow j}. \tag{31}$$

Defining $\boldsymbol{M} = (m_{ij})$ and $\boldsymbol{v} = (v_i)_{i=1}^n$ as in eq. (9), this is simply $\boldsymbol{M}\boldsymbol{d} = \boldsymbol{v}$.

We can also efficiently solve eq. (31) by iteration. Denoting by $d_i(s)$ the priorities at step s and setting the initial values $d_i(0) = d^*$, we iterate

$$d_i(s+1) = \frac{1}{k_i}\left[\frac{nd^*}{m(n-1)}\sum_{\substack{j \\ i\leftrightarrow j}} t_{i\leftarrow j} + \sum_{\substack{j \\ i\leftrightarrow j}} d_j(s)\right] \tag{32}$$

until the (scaled) total change between iterations $\delta(s) = \frac{1}{nd^*}\sum_{i=1}^n |d_i(s) - d_i(s-1)|$ is seen to satisfy $\delta(s) < \varepsilon$ for some choice of small ε.

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] T. Müller. (2016). "osu-performance," [Online]. Available: https://github.com/ppy/osu-performance (visited on 11/19/2019).

[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.

[5] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer Networks and ISDN Systems*, vol. 30, no. 1, pp. 107–117, Apr. 1998, ISSN: 0169-7552. DOI: 10.1016/S0169-7552(98)00110-X.