ShaRk Diffcalc

Komi

November 1, 2021

1 Introduction

(WIP) This writeup aims to formally define the objectives and models used in the latest iteration of the ShaRk proposal for SR/PP rework.

2 Diffcalc Definition

2.1 General paradigm overview

The main idea of the proposed diffcalc is to assigned individual difficulty values to each note, according to the combination of different types of difficulty computed through individual modules. These modules are:

- Rice-dependant
  - Density
  - Manipulability/mashability
  - Strain/Motion
- LN-dependant
  - Inverse
  - Release
  - Hold Strain
  - LNness

2.2 Modules

2.2.1 Density

The *Density* module aims to calculate the amount of *mental stress* (not physical) caused by the amount of existing notes in the *playfield* around any given note.

2.2.1.1 Parameters

- ***bin_size**:* Size of the window used to compute the note density around any given note.

2.2.1.2 Algorithm

```
output=array(length(ListOfHitObjects))

for each HitObject h in ListOfHitObjects:
    c = count of notes around the note in a [-500ms, 500ms] range

    output[Hitobject]=c

return output
```

2.2.1.3 Known Issues

This module has currently identified issues.

- **Use of rigid window threshold**
  - **Problem:** The use of a rigid threshold window causes some maps not count density as desired. This is specially visible on loved maps with rates, where the average computed density doesn't scale linearly with the rate of the map (i.e. see Kuroneko Manyuuki on different rates).
  - **Possible fix:** with the same rigid threshold, use a weighted approach to calculating the notes, by collapsing [-500ms, 500ms] range (may need to use a larger window after changes) into the [-3,3] range of a $\mu = 0$, $\sigma = 1$ normal distribution (bell curve).

2.2.2 Manipulability

The *Manipulability* module aims to assess how easy it is to play any given note based on the manipulability/mashability of the pattern surrounding it (that is, if the pattern can be combo'ed and/or played with good accuracy by playing it as if it was a simpler pattern that requires less or more trivial hand movements).

2.2.2.1 Parameters

- ***bin_size:*** Size of the window used to compute the manipulability of a given note.

2.2.2.2 Algorithm

```
output=array(length(ListOfHitObjects))
column_counts=array(1,1,1,1)

for each HitObject h in ListOfHitObjects:

    Compute how many notes are in each column inside the window
    for each HitObject h1 in the range (h.timestamp-bin_size/2, h.timestamp+bin_size/2)
        column_counts[h1]+=1

    left_hand_count=column_counts[0]+column_counts[1]
    right_hand_count=column_counts[2]+column_counts[3]

    How evenly distributed notes are in the left hand
    left=min(column_counts[:2])/max(column_counts[:2])
    How evenly distributed notes are in the right hand
    right=min(column_counts[2:])/max(column_counts[2:])
    How evenly distributed notes are between left hands
    both=min(left_hand_count,right_hand_count)/max(left_hand_count,right_hand_count)

    output[h]=average(left,right,both)

return output
```

2.2.2.3 Known Issues

This module has currently identified issues.

- **False Positives**
  - **Problem:** The use of the simple divisions like

    `min(column_counts[:2])/max(column_counts[:2])`

    means that patterns like one-column longjacks are treated as almost "impossible to manipulate", which is probably not what would be wanted.
  - **Possible fix:**
- **False Negatives**
  - **Problem:** The use of the simple divisions like

    `min(column_counts[:2])/max(column_counts[:2])`

    means that patterns which have evenly distributed notecounts but aren't easily manipulable, such as the ones in 1, get high manipulability values while they shouldn't.
  - **Possible fix:**

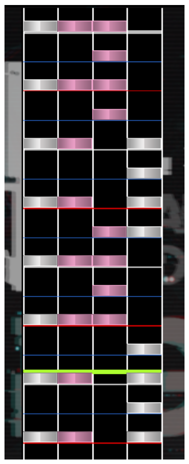


Figure 1: Example of a "false negative"-causing pattern in the Manipulability module (detected as manipulable despite not being so). Both hands would receive a Manipulability value of 1, despite right hand not being possible to manipulate/mash.

2.2.3 Strain

The *Strain* module aims to assess the *physical stress* of each note based on the four pairs formed by it and the following note in each of the columns.

2.2.3.1 Parameters

- ***jack_weight:*** Strain multiplier assigned to *jack* movements (same column between the two notes). Current value: 2.5
- ***onehand_weight:*** Strain multiplier assigned to *one hand* movements (same hand, different column). Current value: 1.4
- ***twohand_weight:*** Strain multiplier assigned to *two hand* movements (any of the columns of the other hand) Current value: 0.7

2.2.3.2 Functions

- ***base_difficulty:*** Base strain difficulty assigned for consecutive note pairs, depending on distance between notes. Computed as

$$base\_strain(i,j) = \frac{100}{timestamp(j) - timestamp(i)} \tag{1}$$

  where i is the note being assigned a strain difficulty and j is the next note in a column c.

- ***grace_suppression:*** Function computes the degree of suppression of the base strain difficulty, to mitigate the effects of graces/large chordjacks in the computations. Its form is a modified sigmoid function defined as

$$suppress(i,j) = \frac{1}{1 + e^{6-0.1*(timestamp(j)-timestamp(i))}} \tag{2}$$

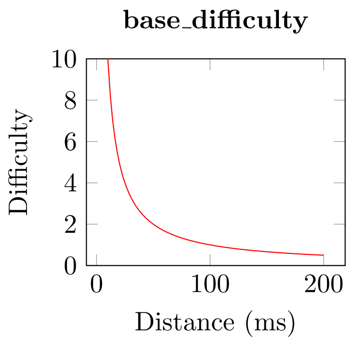
  where i is the note being assigned a strain difficulty and j is the next note in a column c.

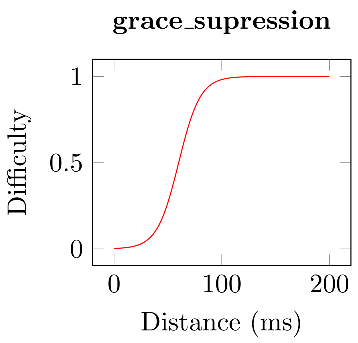
- ***strain_difficulty:*** Final strain difficulty based on movement type (jack, onehanded, twohanded) and distance. It is computed as

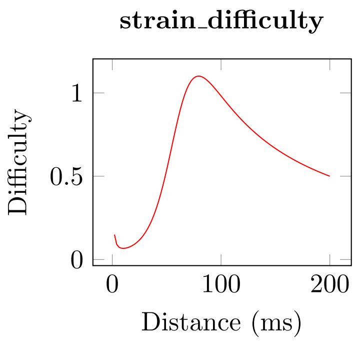
$$strain(i,j) = w * base\_strain(i,j) * suppress(i,j) \tag{3}$$

  where i is the note being assigned a strain difficulty, j is the next note in a column c, and w is the assigned weight ($onehand\_weight$ or $twohand\_weight$) depending on the columns of i and j. Please note that **jacks are not suppressed**, and their final $strain\_difficulty$ is only computed as

$$strain(i,j) = jack\_weight * base\_strain(i,j) \tag{4}$$







2.2.3.3 Algorithm

```
output=array(length(ListOfHitObjects))

for each HitObject h in ListOfHitObjects:

    strain=0

    for each Column c in (0,1,2,3):

        h1 = next HitObject after h in column c
        h2 = previous HitObject before h1 in column c

        Jacks are not suppressed
        if same column (h,h1):
            strain=strain+jack_weight*base_strain(h,h1)

        suppress(h,h1) allows to suppress graces, while suppress(h2,h1)
        allows avoiding giving weight to notes in other columns
        that jack with a note that is producing a chord with h
        else if same hand (h,h1):
            strain=strain+oh_weight*base_strain(h,h1)*suppress(h,h1)*suppress(h2,h1)
        else if different hand (h,h1):
            strain=strain+th_weight*base_strain(h,h1)*suppress(h,h1)*suppress(h2,h1)

    output[h]=strain

return output
```

2.2.3.4 Known Issues

This module has currently identified issues.

- **Dense Jacks overbuffed**
  - **Problem:** Because each note is treated semi-independently, consecutive jacks gets considerably buffed. For example, with the current weight values, a [12][12] jack will get assigned a total difficulty of $4 * \frac{100}{distance}$. On the other hand, a [1][1] jack, despite being virtually the same hand/finger movement, is assigned a total strain difficulty of $2 * \frac{100}{distance}$, half the difficulty as a double jack.
  - **Possible fix:**
- **Certain Jumpstream maps overbuffed**
  - **Problem:** Some jumpstream maps seem to be overbuffed at the moment; see Peter on Crack (Jole), Freedom Dive, Takecore of Yourself, Cyber Induction. Cause may be One Handed patterning, but it is still unknown.
  - **Possible fix:**