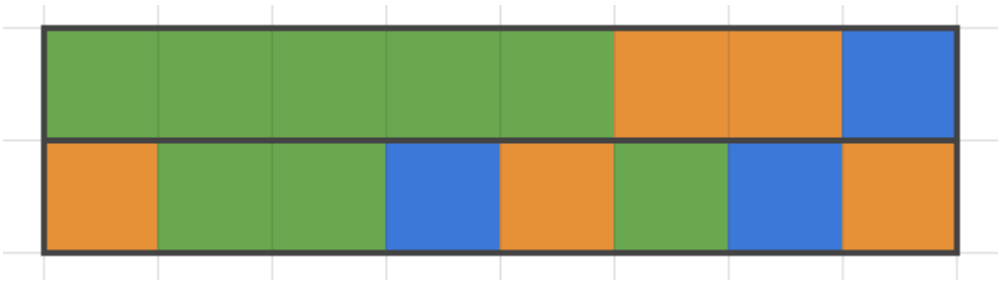
When Random Isn't Good Enough - An Analysis Into Playlist Shuffling

We're all thinking about it. Spotify doesn't shuffle playlists randomly, does it? That one song's been hiding away in your playlist for god knows how long, but Spotify still somehow finds time to play that one Ed Sheeran song that you feel guilty listening to. You'll be glad to know that your intuition is correct: Spotify doesn't actually shuffle randomly.[1] So that begs the question: Why? Additionally, what actually is the best way to shuffle a playlist? That's the question I want to discuss, and answer in this essay.

Why can't we just shuffle randomly?

Let's suppose you're in Spotify's position, shuffling songs in a playlist. When you get tasked to shuffle those songs, what would you do?
Your first instinct is probably something along the lines of... actually shuffling randomly. Let's say we're doing an algorithm that attaches a number to each song in the playlist and every time you shuffle, you're given a random number that hasn't already been played before. Of course, it's only common sense to assume actually shuffling is the best option - but you'll quickly be stricken with the Gambler's Fallacy. If you get 3 or more songs back to back from the same artist in a 50 song playlist full of varied genres and artists, it definitely won't feel random. I'll put it into a visual.



Which one of these rows looks more random to you? The second one definitely looks more random. But if you're actually doing completely random chance, both of these options are just as likely as each other.

This means our algorithm can't be truly random, because to the average person, shuffling randomly doesn't actually *feel* random.

---

[1] https://community.spotify.com/t5/Your-Library/shuffle-playlist-seems-to-play-the-same-songs/td-p/5507802/redirect_from_archived_page/true

What makes a good shuffle?

Okay, so we've decided that shuffling randomly isn't great for people because of the fact that our brains will do anything to look for patterns, even if there isn't any.

Because of the Gambler's Fallacy, we know that recognising any patterns in a sequence gives the illusion that the sequence isn't random. Therefore, our ideal shuffle should prevent any recognisable patterns or at least prevent streaks of similar songs. It also should be a form of shuffling that's still somewhat fair to all songs, so that every song still manages to be heard in your playlist. Also, we should prevent duplicate songs, especially because our shuffles aren't necessarily that good at being statistically random.

Spotify's bad solution to the problem

Let's try comparing to Spotify's shuffling algorithm, to measure these new metrics against. Spotify does something similar to random chance, but they additionally weight songs by certain factors they have, such as whether you've streamed or skipped them more often than not, if they're newer in your playlist, or if you like to think conspiratorially: towards more popular artists.
This algorithm might also refuse to allow artists to play several times in a row, which all of these things added together give off the vibe of "totally random!" to the average person, while constantly being fed songs they enjoy.

It prevents most recognisable patterns while shuffling by limiting how many similar songs are in a row, and prevents duplicates, so for those metrics it gets an easy pass. Now, you might think (if you've paid attention) the algorithm is rigged against certain songs, so how can it be good at shuffling? If you've asked this question, you've stumbled upon the biggest complaint and failure of the Spotify algorithm.

One of the biggest complaints about the way Spotify shuffles music is that it buries certain songs further down in the playlist, and overplays other songs by shuffling them closer to the top of the playlist. To prove this, I'll try simulating the Spotify algorithm and calculating things with the results.

Using Lua code, I'm going to try to simulate shuffling a playlist 100 times[2] and record the position each song is at, and group them by each set of 10 songs in the shuffle. e.g , the first 10 songs are grouped together and so on.

---

[2] With additional parameters: 50 songs, made by 10 artists all given weights from 1 to 10, where a weight of 10 means that it's 2 times more likely to be chosen than a weight of 1, and making sure that there aren't any of the same artists within 3 songs. The rating of each song stays constant between shuffles.

Rating	Group 1	Group 2	Group 3	Group 4	Group 5	Total
1	56	69	70	76	129	400
2	60	66	72	88	114	400
3	72	71	71	83	103	400
4	113	122	148	146	171	700
5	44	51	67	82	56	300
6	77	79	81	76	87	400
7	100	92	89	67	52	400
8	158	161	136	122	123	700
9	193	179	164	170	94	800
10	127	110	102	90	71	500

Now, it's relatively clear that there's an association between the rating and the group of 10 songs it's in, but that's not what we need to work out.
There's also a weighting system that'll be important for later, but that'll be explained when we get there.

This complaint about Spotify forcing certain songs lower down in a shuffle becomes especially clear when we factor in that realistically, most people won't listen to every song that someone has in a playlist. For the sake of results, I'll say people listen to about 20 songs per listening session. To figure out if any song is effectively being "buried", we can try to calculate the amount of times we have to listen to the first 20 songs of a shuffle (in a listening session) before listening to every song in the playlist. If our results show that we have to do tons upon tons of listening sessions before getting to every song, we can conclude the shuffle isn't good enough to listen to every song.

Coupon Collector's Problem

I like to think this question brings to mind the Coupon Collector's problem. [3]
This problem asks a close enough question to what we're asking - It asks: If you buy an item containing a coupon, and there's a variety of coupons you could get, all with **equal** odds of being obtained, how many items would you have to buy to get all the coupons out of that variety? Let's try and understand the Coupon's Collector's problem to see if there's something there that can help us.

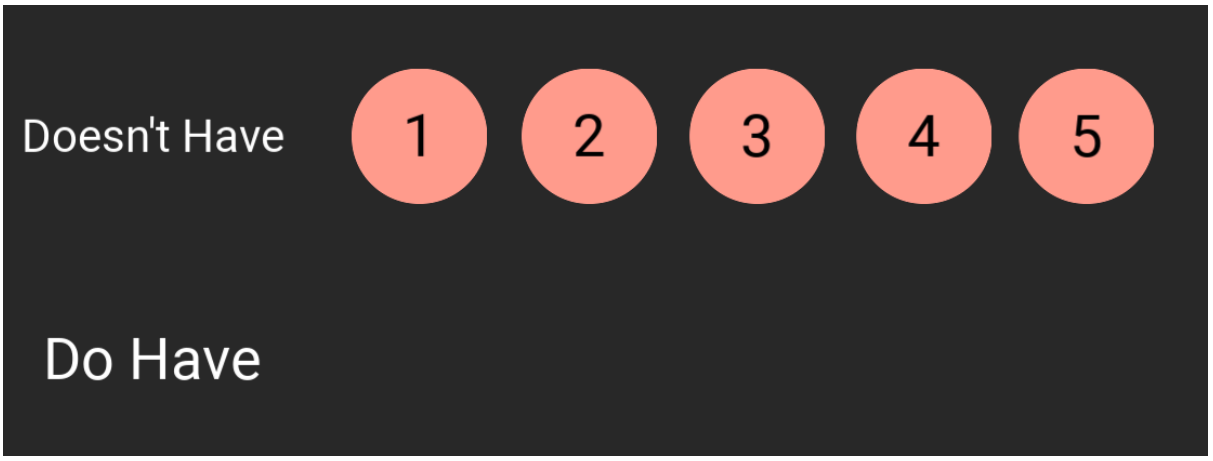
[3] https://www.youtube.com/watch?v=khoRV9BFdJE helped me understand, if you want another explanation.

Let's imagine the scenario where we have a playlist with 5 songs, where the odds of getting any specific song is all the same. We want to find the expected value of the number of draws to listen to all 5 songs at the start of the playlist.

As a refresher, the expected value represents the predicted outcome of a probabilistic event. The expected value for the number of trials before a success is 1/p, e.g, expecting a dice roll having to flip 6 times before reaching a 6, because

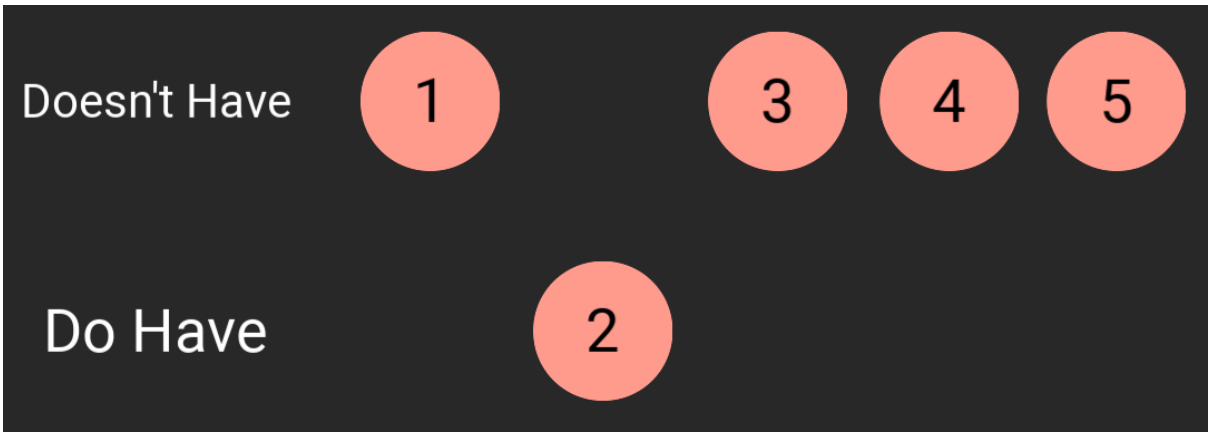
$$E[x] = \frac{1}{p} = \frac{1}{1/6} = 6$$

Where E[x] represents the expected value.
Okay, let's go back to our example with our 5 songs. I'll list off each step visually so it's easier to understand.



So when we start off, we haven't listened to any song, we have no choice but to listen to something new, so

$$\mathrm{P}(\text{new song}) = 1.$$



Let's continue, once we've picked a song, and replace it back into the playlist, we have a 1/5 chance of listening to 2 again, but realistically we'll listen to a new song, so

$$\mathrm{P}(\text{new song}) = 4/5.$$

We can repeat this until we run out of songs, and generally we can state

$$P(new\ song) = \frac{5-i}{5}$$

Where i is the amount of songs we've already listened to.
Even more generally, we can say for n songs,

$$P(new\ song) = \frac{n-i}{n}$$

We can now use the expected value formula from earlier to get an expected amount of playlist listens for any new song, where i is the songs you've heard before.

$$E[new\ song] = \frac{1}{p} = \frac{1}{n-i/n} = \frac{n}{n-i}$$

For example, if we'd already listened to 3 songs in the playlist, but we wanted to listen to the 4th one, we'd expect to listen to the playlist 2.5 times until a 4th new song appears first.

$$E[4^{th}\ new\ song] = \frac{5}{5-3} = \frac{5}{2}$$

Therefore, dragging this to its natural conclusion, E[n] would represent the expected value of listening to each song first, which would be the sum of E[new song] for every song.

$$E[n] = \frac{n}{n} + \frac{n}{n-1} + \frac{n}{n-2} + \ldots + \frac{n}{1}$$

Let's clean things up a little.

$$E[n] = n(\frac{1}{n} + \frac{1}{n-1} + \frac{1}{n-2} + \ldots + \frac{1}{1})$$

And let's just swap around some of the terms in this sequence.

$$E[n] = n(1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n})$$

And there we have it! Usually, that sequence in the brackets is turned into the harmonic series of N as a shorthand. Unfortunately, this would be amazing if our songs were equally weighted, but in our scenario it's rigged against songs with lower ratings, and we're only listening to the first 20 songs in each shuffle.

Now, when it comes to the Coupon Collector's Problem, where the probability is not equal, there's a formula that approximates the expected value, which looks like:

$$E \approx \sum_{i=1}^{S} \frac{1}{p_i}$$

Don't think too hard about how to derive this, because that's definitely something we don't have time for, but we know how to use it: where E is the expected value of the trials, S is the number of items, and p is the probability that song i appears in a specific position. This approximation only works where the probability of all the songs appearing is somewhat realistic, and there aren't some super rare songs that only ever appear once in a blue moon.

We also have to factor in the fact we're not listening to the entire playlist - only the first 20 songs, which definitely messes our probabilities up. We'll have to simplify the problem further by pretending that the position of a song is independent, even though because of our "no artists can have 3 songs in a row" rule, that won't be true.

Let p = probability that song i appears in a specific position k. Assume that the positions are independent.

$$P(not\ in\ k) = 1 - p$$

$$P(not\ in\ 20\ positions) = (1 - p)^{20}$$

$$P(appears\ at\ least\ once\ in\ 20\ positions) = 1 - (1 - p)^{20}$$

Now, we can estimate the expected value of listening to the first 20 songs, by replacing p in the approximation formula with the probability we found above.

$$E \approx \sum_{i=1}^{S} \frac{1}{1-(1-p)^{20}}$$

If you still remember that table we had all the way above, we can finally use the figures we had from up there to help get an estimation for the expected value.

Rating	Group 1	Group 2	Group 3	Group 4	Group 5	Total
1	56	69	70	76	129	400
2	60	66	72	88	114	400
3	72	71	71	83	103	400
4	113	122	148	146	171	700
5	44	51	67	82	56	300
6	77	79	81	76	87	400
7	100	92	89	67	52	400
8	158	161	136	122	123	700
9	193	179	164	170	94	800
10	127	110	102	90	71	500

S refers to our number of songs,which is 50 for our playlist. Additionally, the table also shows the total number of songs over the 100 shuffles for each rating, which allows us to break down how many songs have each rating, e.g 4 songs have rating 1, and were shuffled 100 times to get a total of 400.

We also need to use our weighting system from before to calculate the probability of a song being picked in the playlist, where each weight means how much more likely that rating is to be picked than a rating 1 song (e.g, a rating 10 is 2x more likely to be picked than a rating 1).

Rating	Weights
1	1
2	1.1
3	1.2
4	1.3
5	1.4
6	1.6
7	1.7
8	1.8
9	1.9
10	2

The probability of a song of a certain rating being picked can be determined using the total weight of all the songs in the playlist, which can be figured out by calculating how many songs in the 50 song playlist had what ratings, since the ratings stayed constant for each song in this playlist.

$\sum weights = 77.5$

$$P(song\ of\ rating\ a\ being\ picked) = \frac{weight\ a}{total\ weight}$$

We'll use examples for songs of rating 1.

$$P(song\ of\ rating\ 1\ being\ picked) = \frac{1}{77.5}$$

Let's insert this into our formula for the probability a specific song is picked in the first 20 songs.

$$1 - (1 - \frac{1}{77.5})^{20} \approx 0.22875 \approx 23\%\ of\ being\ picked$$

When we substitute this into the summation from earlier, and repeat with every rating, we get an equation like

$$E[amount\ of\ listening\ sessions] \approx 4 \times \frac{1}{1-(1-\frac{1}{77.5})^{20}} + 4 \times \frac{1}{1-(1-\frac{1.1}{77.5})^{20}} + 4 \times \frac{1}{1-(1-\frac{1.2}{77.5})^{20}} \ldots$$

The summation goes on beyond the page, so I'm refusing to type the whole thing. This effectively gives us an approximation that gets us the expected value of sessions needed to listen to every song in a playlist. If you were wondering what the 4 does, it's because for ratings 1,2,3 there's 4 songs in our 50 song playlist that contains those ratings. Evaluating this expression results in

$$E[amount\ of\ listening\ sessions] \approx 157$$

Which means for a 50 song playlist, you'd have to listen to the first 20 songs up to 157 times to listen to every single song in the playlist at least once. From that alone, we can pretty safely conclude that Spotify's shuffling algorithm is majorly flawed. Obviously, this model is also flawed, we haven't factored in the constraints that prevent artists from getting several songs in a row, or maybe that Spotify makes it even less likely for certain songs to appear in a playlist, but those factors would likely add even more listening sessions onto the expected value.

To put this in a bit more of a perspective,the average song in 2020 was about 3 minutes and 17 seconds long.[4] Using our expected value, and the fact you have to listen to 20 songs per listening session, that puts the amount of time you'd have to listen to music before you actually get to every song in your playlist… at 171 hours and 50 minutes. On an unrelated note, you can watch the entire show "Breaking Bad" 2 times in a row in 128 hours.

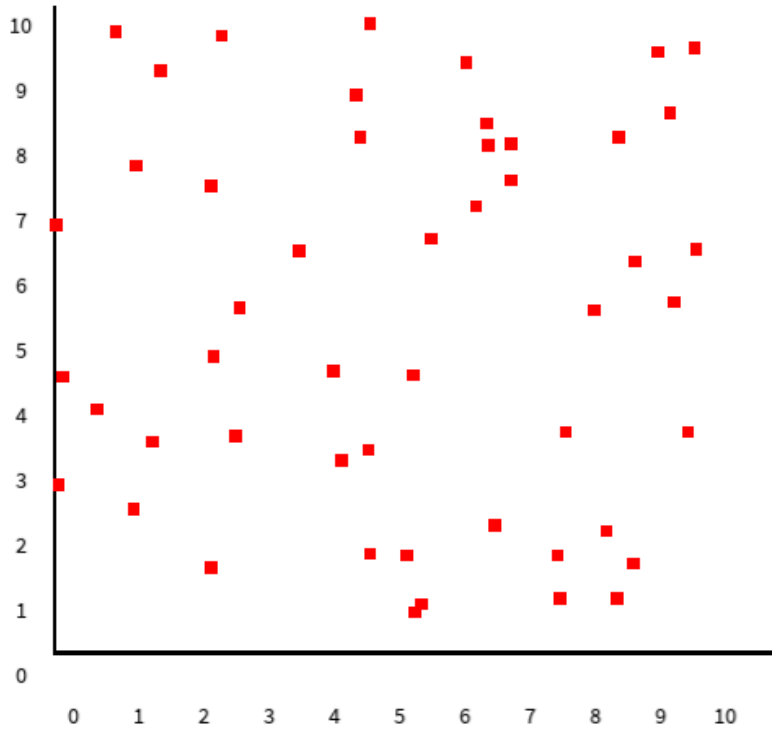
[4]https://www.statista.com/chart/26546/mean-song-duration-of-currently-streamable-songs-by-year-of-release/

What else can we do?

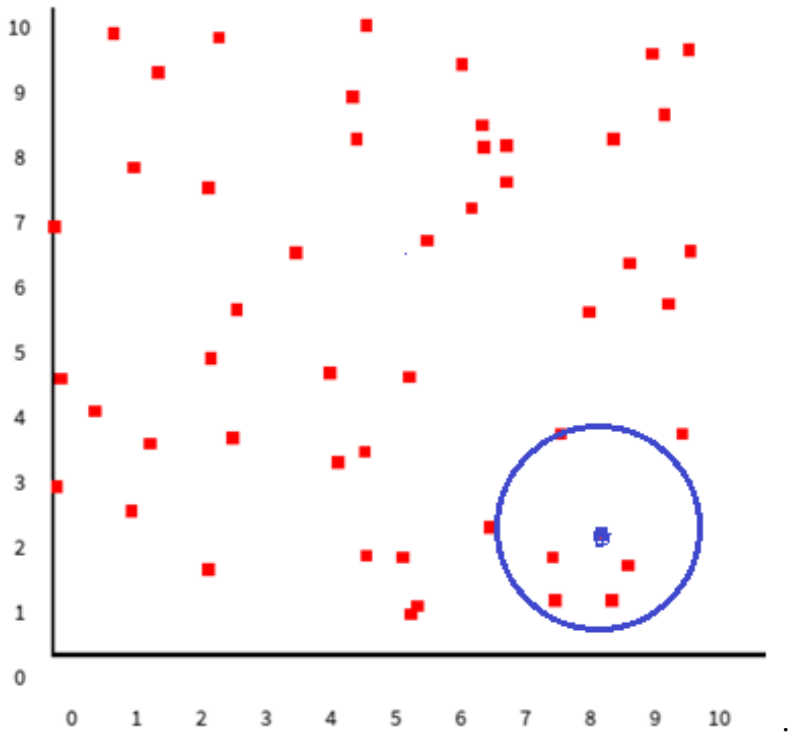
You might've heard all of this yapping about Spotify and their method of shuffling, and you thought to yourself: "I think I'd do better." Firstly, I'd say something about arrogance, but honestly, if we want the algorithm that shuffles the most randomly for human beings, we surely must have better.

Introducing: an untitled algorithm that may or may not actually shuffle songs properly! I'll try to explain what it is and if you can recognise an algorithm similar to it, point it out because I've definitely just reinvented the wheel with this.

Let's express a playlist in terms of each song, and give each song 2 factors. Those 2 factors could mean literally anything, from the speed of a song to how close does the song sound to Elvis Presley. We'll plot each song based on those factors. It doesn't have to be just 2 factors, and in practice it definitely wouldn't be just 2 factors, but trying to visualise this with 3, 4 or 5 dimensions becomes exponentially complicated.



Now, realistically the songs would appear in clusters rather than a completely random arrangement, but that's food for thought later.
Plotting all of these songs onto a graph means we can measure similarity by how close each song is to any of the others. Let's say we pick a random song out of the 50, and select the 5 most similar songs out of that 50. We'll visualise this with a circle with a radius of whatever the distance is from that original point to the 5th most similar point relative to that original point.



The random song (colored in blue) will be first in our shuffle, so it cannot ever be picked again. We will also exclude any of the unpicked songs in the circle as they're too similar to the song we picked. Then we just pick a song that's unpicked and outside the circle, and repeat. If we only have 6 remaining songs, that means that every song left would be picked, so we'll just shuffle those last 6 songs randomly, as normal.

Using this method, we can prevent any duplicate songs appearing in the shuffle, and prevent any streaks of similar songs appearing at once, but we're not really sure on if it's fair to every song. We need to test whether or not the algorithm is random for every song.

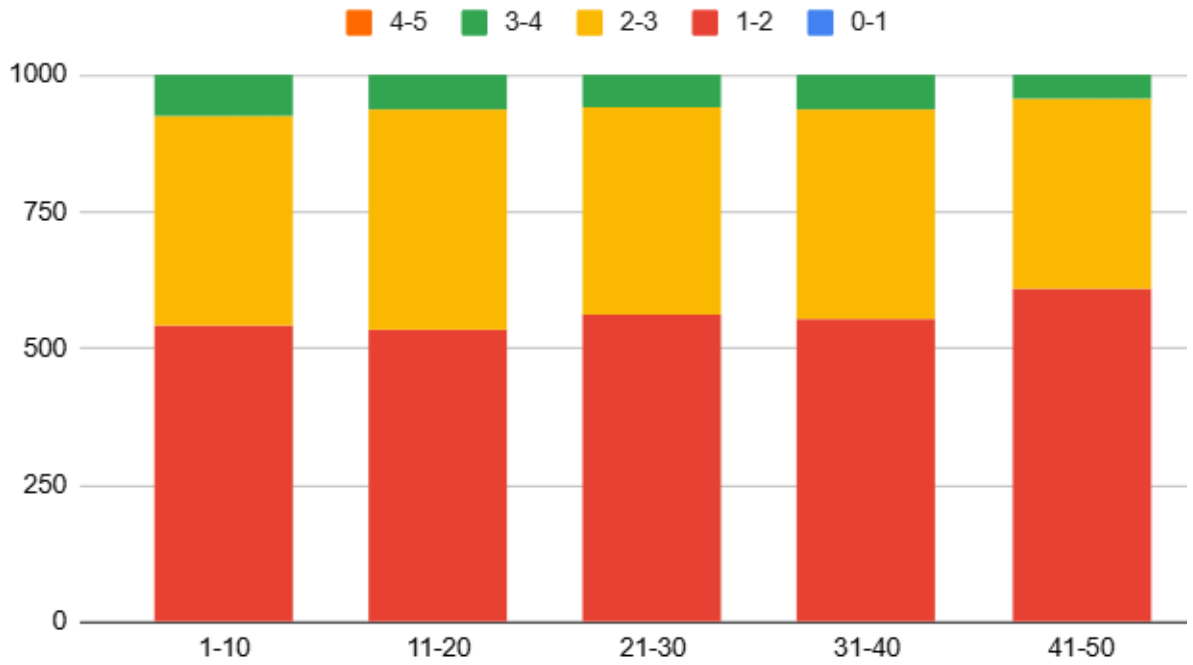
We can do this by throwing this algorithm into a Lua program, and generating a table by how similar a song is to other songs (by measuring the distance between a song and the 5th most closely related song to it) compared to how many times they appear in the first 20 times of a shuffle, and shuffling 100 times, we get the table:

Pos Group	0-1	1-2	2-3	3-4	4-5	5-6	6-7	7-8	8-9	9-10	>10
1-10	0	541	386	73	0	0	0	0	0	0	0
11-20	0	534	403	63	0	0	0	0	0	0	0
21-30	0	563	378	59	0	0	0	0	0	0	0
31-40	0	553	386	61	0	0	0	0	0	0	0
41-50	0	609	347	44	0	0	0	0	0	0	0

Now, you might be a little shocked to see half the table show up with 0's as a result, but if you think about it, it isn't really surprising, because the odds of a song where

the 5th closely related song is really far away is incredibly unlikely and basically means your graph would have 45 different songs really far away from 1 song.

We can try and measure how relevant the similarity of a song is to the position it is in a group by simply making a bar chart and inspecting our results.[5]



Using a bar chart gives us this result for each range of values. You can see that the proportion of 1-2 songs appearing in each set is rather similar. We can basically say that the similarity of a song won't affect where it appears in a playlist, but more similar songs generally just appear more in a playlist, just because there's more of them.

The most glaring problem with this algorithm is that since your data would often be clustered together, there's a decent chance that you might get songs that are pretty notably similar to each other but outside of the "exclude 5th most similar song" rule in the algorithm, which leads to the same Gambler's Fallacy issue as a pure random algorithm has. You could try to improve on this by excluding songs that are less than a specific level of similarity, but then that has a possibility of weird fringe cases cropping up and making it not random for those weird fringe cases.

[5] I initially wrote a whole section trying to use the Chi^2 distribution to verify this, only to realise that the fact that there's so many songs that are naturally at the 1-2 range means there's likely an association and the Chi^2 distribution would be a formality. That was a bit sad to realise.

Conclusion

Overall, there's a lot of other methods for shuffling a playlist that I didn't get to cover, mostly because many of them don't prioritise randomness.[6] It's worth mentioning that in a majority of scenarios that you don't really value randomness - like a playlist in a gym or a radio station which probably wants the most similar songs playing together to try and prevent people from getting surprised at a mood switch that's totally illogical.

When it comes to thinking about randomness, it's really difficult to hack into the brain to see what's perceived as random for people, and I think that's something that's cool about us. We're human, so we don't just think perfectly every time.

[6] There was a method of shuffling that utilised some sort of pathfinding algorithm to try and make as smooth of a transition as possible between songs. That was cool.