Blackbox Analysis of Shuffle Algorithms

Michael McQuinn, Eric W. Davis Rozier
{mmcquinn, ewdr}@crhc.uiuc.edu

I. Introduction

Over the past three decades, music players have transformed from record players to Walkmans to Discmans to MP3 players. While this has happened, the amount of storage any one player could hold has grown tremendously from a single album to hundreds of albums. This fact alone has greatly altered how people access music.

Because of the continual increase in music storage capabilities on portable music players, much innovation has also occurred in this area. For example, user interfaces have increased in complexity from simple buttons to advanced touch screens on the current iPod Mini. Similarly, displays have changed from LEDs to multiline color displays.

A. Motivation

A growing problem with the increased storage capability is simply how users select which songs they would like to hear. No longer is a simple forward or backward button sufficient. Instead, users need the ability to easily and dynamically create playlists. Devices must become smarter and better understand the preferences of the users. One method to combat the problem of selection is to use random playlists. The idea is that if a playlist is truly random, then the user will get a good ”mix” of songs on a playlist and not need to manually select the next song each time.

Due to limited processing power and battery usage, sometime it is not practical to implement a fully psuedo-random number generator but instead use an approximating technique. When this happens, permutations of playlists can be lost or correlations between permutations can form. Our motivation for this work is to better understand the necessarily limited random number generators behind the most popular MP3 players of today. Also, we would like to develop an accurate Quality of Service (QoS) metric to determine how likely a shuffling algorithm will appear random to a user. This will enable us to more accurately analyze and compare the shuffling algorithms between different players.

More importantly, we would also like to analyze the utility of the randomness in a QoS fashion. A truly random stream, if it allows repeated songs, might be less optimal than an almost truly random stream which does not allow repeats.

II. Data Gathering

In order to analyze the shuffle algorithms, we must be able to efficiently gather playlist data. To do this, we designed an automated system to generate Dual Tone Multi Frequency (DTMF) songs which are loaded on the players and recognized by a PC running the multimon software package. [1] The data from multimon is then parsed into permutations, which are analyzed by the tests in Section III.

A. Tone Generation

Using the multimon program *gen*, we generated 10,000 unique songs consisting of a preamble, song number, and delimeter. See Figure 1. The unique song number represents one of songs on a MP3 players, while the preamble and delimeter help us parse the multimon output and regenerate the playlist.

Since all major portable audio players support different formats, we chose MP3 as the standard since currently they all support MP3 playback as a sort of defacto standard. Because of this, we converted all the DTMF songs to MP3s using *sox* and *lame* as seen in Figure 2.

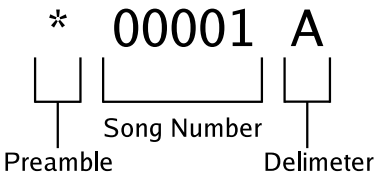

Fig. 1. Generated Tones Format

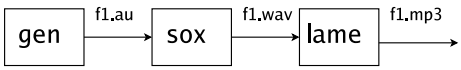

Fig. 2. Generation of a file f1.mp3

B. Tone Recognition

Once the songs have been generated, they are loaded onto the players by different means depending on the model:

- iRiver: By directly copying files onto the hard drive
- iPod Mini: Using iTunes for Windows XP [2] to load files
- iPod Shuffle: Using iTunes for Windows XP [2] to load files

For each test we ran, we reset completely the hard drive of each player and reloaded them with only the songs needed for that particular test. This helps reduce outside sources of error in our analysis, including effects of play count or rating. See Section **??** for information regarding Future Work on this subject.

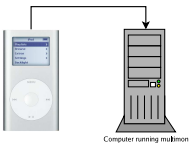

Fig. 3. Process Overview for tone recognition

III. Testing

In order to analyze the data generated by our MP3 players we conducted two primary tests. The first test was a test for the uniformity of the distribution of permutations, and the second was a test to see the level of autocorrelation between the permutations.

A. Test of Uniformity

To test the uniformity of the distribution of permutations we used the Kolmogorov-Smirnov [3] test, comparing the cumulative distribution of probability mass of the data with that of a uniform distribution. Ideally any given permutation will be equally as likely as any other, so that the player does not favor one randomization over another.

B. Test for Autocorrelation

Additionally we tested for independence in the sequence of permutations by calculation the lag_1 through lag_{10} autocorrelations [4]. This statistic should give us a feeling for any bias in the players that may cause certain patterns of permutations to occur more often than others.

C. Quality of Service

While we were unable to gather sufficient data, nor to devise a Quality of Service metric which we found fully suitable, initially we thought of testing for the average minimum distance between two songs in the same album. The idea being that perhaps within a given playlist their might be some bias towards a particular album, or dependence in the subsequent choices of which song to play next given the current song is from a given album. This method and our thoughts are contained within Section VI

IV. Results

A. iPod mini

We were able to collect a trace of 2180 complete permutations from the iPod before the battery died. Using this data we conducted a Kolmogorov-Smirnov test for uniformity. Figure 4 shows a histogram detailing the number of incidences of each permutation in our trace. The iPod had full coverage of all 120 possible permutations of five songs. The calculated test value D was found to be

$$D = 0.0214$$

for the iPod, thus we fail to reject our null hypothesis H_0 and conclude that we cannot detect a significant difference between our data and the uniform distribution. Figure 5 shows both the distribution of permutations in our data, and the ideal uniform distribution.

Testing for significant lag_1 through lag_{10} autocorrelation for the trace collected from the iPod revealed only one case where the null hypothesis of independance was rejected. That case was the lag_8 autocorrelation with starting point zero. For this test

$$Z_0 = -2.036241$$

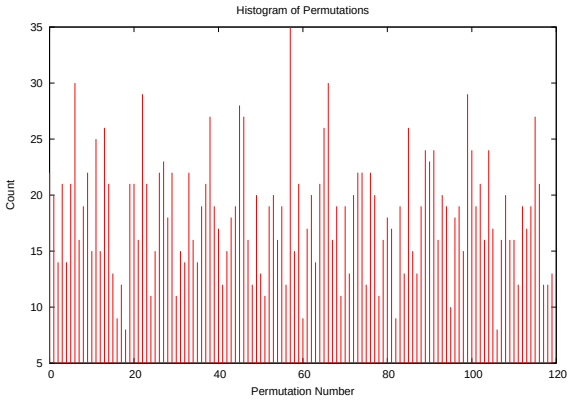

Fig. 4. Frequency of Permutations for iPod

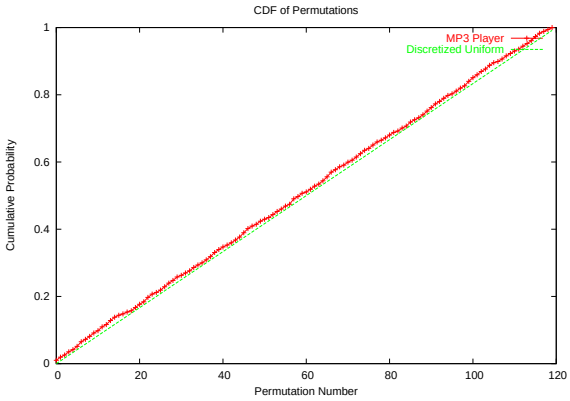

Fig. 5. CDF of iPod compared to Discretized Uniform for K-S Test

. Given our critical value

$$z_{0.025} = 1.96$$

we must reject the null hypothesis for independance in this case.

The iPod mini is in fact so faithful to avoiding autocorrelation that in several cases due to the random pairing of two permutations the last song played for one permutation in the trace is the same as the first song played in the next permutation in the trace. During analysis we counted the number of times this occured, finding 429 separate cases of such behavior.

B. iPod Shuffle

A total of 2533 randomly generated permutations were generated for the iPod Shuffle before the battery died. Using this data we conducted a Kolmogorov-Smirnov test for uniformity. Figure 6 shows a histogram detailing the number of incidences of each permutation in our trace. The iPod Shuffle did not have full coverage of all possible permutations only exhibiting incidences of 24 of the possible 120 permutations.

The calculated test value D was found to be

$$D = 0.162$$

for the iPod Shuffle, thus we reject our null hypothesis H_0 and conclude that there is a significant difference between our data and the uniform distribution. Figure 7 shows both the distribution of permutations in our data, and the ideal uniform distribution.

Of the 24 permutations we found in our trace, all ended in the second song. Comparing this data to some preliminary traces we collected using five song playlists and the iPod shuffle we found that it appears to be the case that for a playlist of size N, the iPod shuffle first generates a purely random shuffle and for subsequent playlists only shuffles the first $N-1$ songs ensuring the last song is never played twice in a row. More data is needed to further test this hypothesis. This could be an attempt to solve the problem found in the iPod mini which causes the same song to sometimes be played twice.

Testing for significant lag_1 through lag_{10} autocorrelation for the trace collected from the iPod Shuffle revealed many cases for each lag_x where the null hypothesis of independence was rejected. This was most likely due in part to the incomplete coverage of our permutation space by the iPod Shuffle.

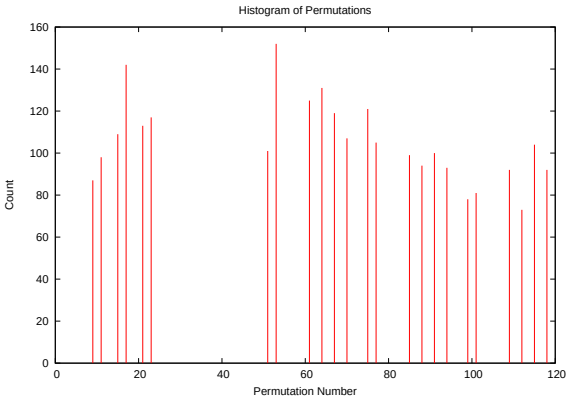

Fig. 6. Frequency of Permutations for iShuffle

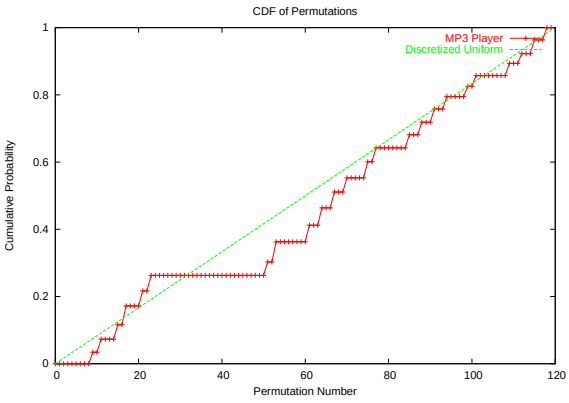

Fig. 7. CDF of iShuffle compared to Discretized Uniform for K-S Test

The test results for these cases are ommitted for brevity but a summary is given in Figure 8.

Having realized these facts about the iPod Shuffle's algorithm we ran the tests for uniformity again, this time comparing only the explored state space of the permutations. Figure 9 shows a histogram detailing the number of incidences of each permutation in our trace. The iPod Shuffle had full covereage of all 24 possible permutations.

The calculated test value D was found to be

$$D = 0.176$$

for the iPod Shuffle, given the critical value of 0.249031

Lag_1 Autocorrelation	1/1 cases
Lag_2 Autocorrelation	2/2 cases
Lag_3 Autocorrelation	3/3 cases
Lag_4 Autocorrelation	4/4 cases
Lag_5 Autocorrelation	4/5 cases
Lag_6 Autocorrelation	5/6 cases
Lag_7 Autocorrelation	5/7 cases
Lag_8 Autocorrelation	7/8 cases
Lag_9 Autocorrelation	8/9 cases
Lag_{10} Autocorrelation	5/10 cases

Fig. 8. Summary of iPod Shuffle autocorrelation results.

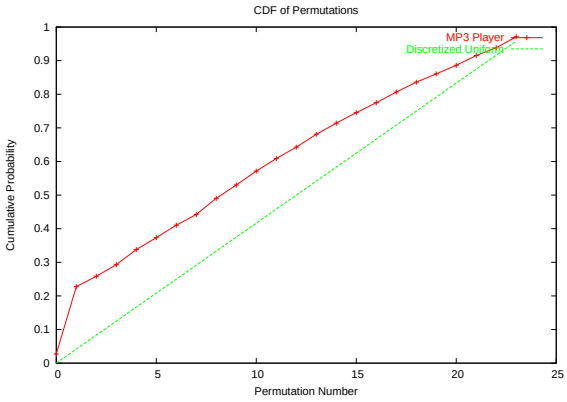

Fig. 10. CDF of iShuffle compared to Discretized Uniform for K-S Test

for a sample size of 24 we fail to reject our null hypothesis H_0 and conclude that we cannot detect a significant difference between our data and the uniform distribution. Figure 10 shows both the distribution of permutations in our data, and the ideal uniform distribution.

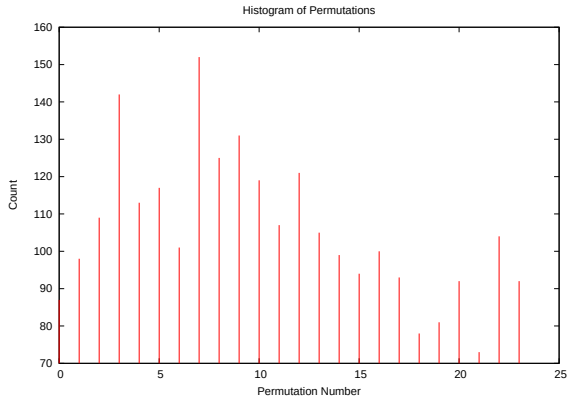

Fig. 9. Frequency of Permutations for iShuffle

C. iRiver

Data was collected in the form of 87 randomly generated permutations for the iRiver before it became obvious that the generation of permutations was done deterministically. Using this data we conducted a Kolmogorov-Smirnov test for uniformity. Figure 11 shows a histogram detailing the number of incidences of each permutation in our trace. The iRiver did not have full coverage of all possible permutations only exhibiting incidences of 3 of the possible 120 permutations.

The calculated test value D was found to be

$$D = 0.825$$

for the iRiver, thus we reject our null hypothesis H_0 and conclude that there is a significant difference between our data and the uniform distribution. Figure 12 shows both the distribution of permutations in our data, and the ideal uniform distribution.

By observing the iRiver's behavior and collecting playlists of maximal size we were able to determine the algorithm by which the iRiver creates a playlist of a given size N, and thus explain why only three playlists are observed in our data. The iRiver stores internally four shuffled playlists of size 10,000 songs. In order to extract a playlist of abitrary size N the iRiver simply selects all songs in a given maximal playlist P_x which are also in the playlist of size N and plays them in the order they are found in playlist P_x. Thus each $P_x, 1 \leq x \leq 4$ maps to a playlist $P_{N,x}$. In the case of five songs the mapping

$P_1 \rightarrow P_{5,1}$ is identical to the mapping $P_4 \rightarrow P_{5,4}$ producing the same playlist:

$$(1, 4, 5, 3, 2)$$

Although the number of permutations seems a bit low, it is worthwhile to note that as the iRiver always starts with the same song, thus eliminating the problem of playing the same song twice which occurs when using the iPod mini. While this may not be the optimal solution for the problem at hand, given that for small playlists like those presented her, a user is apt to start to realize he or she always hears the same song first, and only hears three unique orderings of the songs.

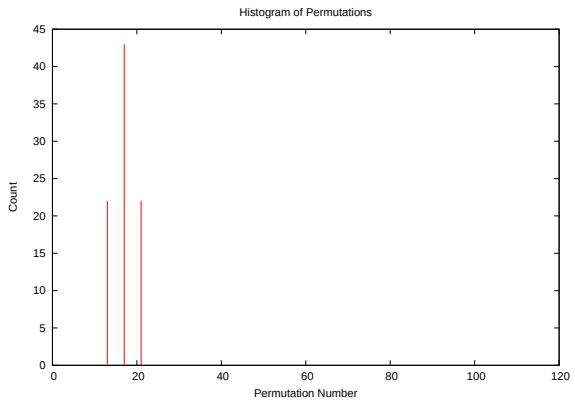

Fig. 11. Frequency of Permutations for iRiver

Testing for significant lag_1 through lag_{10} autocorrelation for the trace collected from the iRiver revealed significant levels of autocorrelation in each and every possible case. This is far from surprising considering that the permutations were chosen deterministically and played in the same order over and over again: $P_{5,1}, P_{5,2}, P_{5,3}, P_{5,4}, P_{5,1}, P_{5,2}, P_{5,3}, P_{5,4}, \ldots$

V. Conclusions

While the iPod mini was the only player which generated a uniform and complete coverage of the entire

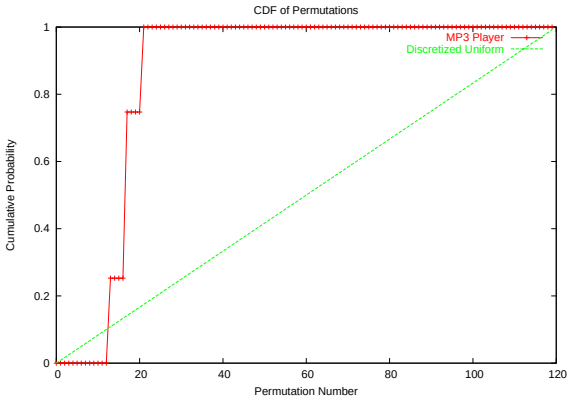

Fig. 12. CDF of iRiver compared to Discretized Uniform for K-S Test

permutation space, it was also the only player to suffer from the possibility of song repetition. Given that the iPod mini we tested was released before the iPod Shuffle, it is quite possible that Apple has since changed it's algorithm to the one utilized by the iPod Shuffle in response to complaints about repeated songs. Such an incident would probably only have to happen once before a typical user would begin complaining about QoS delivered by the product.

The iPod Shuffle comes in second in overall randomness of it's algorithm. While it does not have complete coverage of the 120 permutations of five songs.

VI. Future Work

Although this work shows interesting results about some of the most popular MP3 players on the market, it is lacking in many ways. Because of this, we plan to do much work in the future to help better understand the shuffling capabilities of these players. Much of the future work encompasses understanding the interworkings of the iPod Shuffle and iPod Mini, but some work is needed to better understand the iRiver as well.

- Determining an accurate and fitting Quality of Service metric for shuffling

- Predictive analysis to prove we can generate an iRiver playlist of any size
- Determining the best set of permutations for the iRiver according to our QOS metric
- Determining whether play count or rating affects the permutation uniformity for the iPod Shuffle and iPod Mini
- Verify and understand why the iPod Shuffle only generates (n-1) permutations of shuffles, where n is the size of the play list.
- Perform more experiments with different play list size to guide understanding of the shuffling algorithms
- Determine whether the iTunes playcount and user rating affects the shuffling permutations for the iPod Shuffle and iPod Mini

VII. Acknowledgements

Thanks to:

- Michael Ihde (ihde@crhc.uiuc.edu) for much help with setting up the multimon software packages and general enthusiams for doing cool things.

References

[1] T. Sailer. Linux radio transmission decoder. [Online]. Available: http://www.baycom.org/ tom/ham/linux/multimon.html

[2] A. Corporation. [Online]. Available: http://www.apple.com/itunes/

[3] W. D. K. A. M. Law, *Simulation Modeling and Analysis*. Burr Ridge, Illinois: McGraw Hill, 2005.

[4] B. L. N. J. Banks, J. S. Carson and D. Nicol, *Discrete-Event System Simulation*. Upper Saddle River, New Jersey 07458: Prentice Hall, 2005.