Assignment 2: Audio Identification

Introduction

For this assignment that task was to implement and test an audio identification system. To approach this task, an attempt was made to reimplement the Shazam algorithm [1] which approaches the task by building audio fingerprints for each track in the database based on their frequency components. For each query track, an audio fingerprint is built in the same way, and this is compared against the fingerprints in the database in the hope of finding a match.

Implementation

Fingerprint Building:

Time Frequency Representation

In order to build an audio fingerprint, the audio data is first converted into a time-frequency spectrogram representation using the python 'librosa' module. The effect of using several different representations was investigated, with the Short-Time Fourier Transform (STFT) spectrogram being found to produce the best results. Optimal parameters were also investigated, and the results are shown later on.

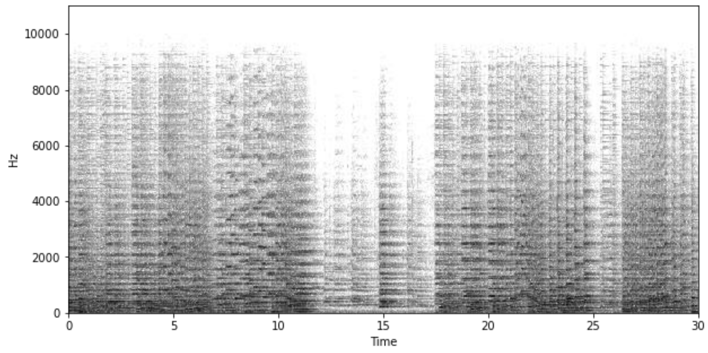
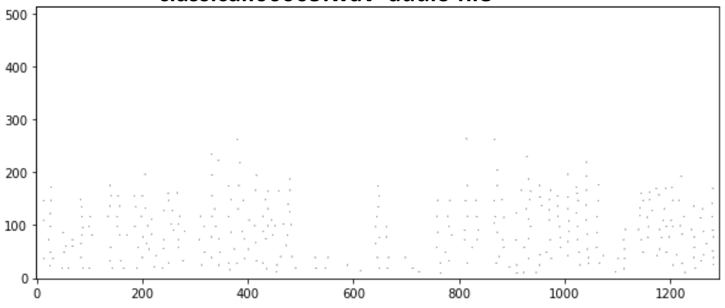


Figure 1- STFT Spectrogram Representation for 'classical.00005.wav' audio file

Peak Picking

Once a time-frequency representation of the audio data was produced, only the strongest frequencies are kept as 'peaks'. These were found using the 'peak_local_max' function from the 'skimage' module, which defines a peak as an amplitude value which is greatest in a local 'neighbourhood' around it, according to a 'minimum distance' parameter. The function returned coordinate points at which the found peaks occurred. Peak points for each audio sample can be used to distinguish tracks according to their specific time-frequency components. If multiple tracks are found to have the same peak points there is a chance they may be the same piece of music. The best values for the parameters 'minimum distance' and 'threshold', i.e., the value peaks must be above, were investigated and shown in the 'accuracy vs. speed' section.

Figure 2- Peak coordinates for 'classical.00005.wav' audio file



Fingerprint Hashing

Audio fingerprints for each track in the database were built from the found peak coordinates. Each peak coordinate was taken as an anchor point, and according to a 'fan value' the peaks were sorted through and those within the distance of the 'fan value' were used to create a hash.

Hashes were created by taking the frequency value of the anchor point (freq1), the frequency value of the selected peak (freq2) and the time difference between them (t_delta). These were converted to a string, and concatenated, and using the 'hashlib' module, converted to a hash value. The hash values produced were appended to a hash list corresponding to the audio file, alongside the time offset from the beginning of the database file to its anchor point.

$$Hash\ generated = (hash\ encrypted\ string(freq1, freq2, t_{delta}), time\ offset)$$

Audio Identification:

Match Hashes

To identify a query audio track, the first step is to obtain hash values for the query track. These hashes are then compared, one by one, against the hashes of each track in the database. If matching hashes are found, the corresponding database track is saved with the number of matching hashes that were found. Once all the database tracks have been sorted through a list has been obtained showing the number of matching hashes per database track. Wang [2] finds that if a large number of matching hashes are found between the query track and a database track, often the correct track is identified.

Align Matches

Wang [2] suggests that further step of aligning the matches in time can help to improve the accuracy of the returned song. The idea is that the matching hashes found must be found at the same relative position within the query track and the database track. To investigate this, the time offset from the beginning of the database file to its anchor point was saved alongside each hash. For the real track, all the identified offsets for each matching hash will be approximately the same value, assuming the query track is being played and sampled at the sample speed as the database track. Therefore, in order to check the time alignments for each matching hash, the difference between the offsets is found by:

$$difference = database\ offset\ from\ original\ track - sample\ offset\ from\ recording$$

For the true matching track, the difference value will always be positive.

For this assignment, the time alignment problem was attempted by counting up the number of number of duplicate time offset difference values for each database track found with matching hashes. Then, the database tracks found to have the most matching hashes that are time aligned are returned as probable song matches.

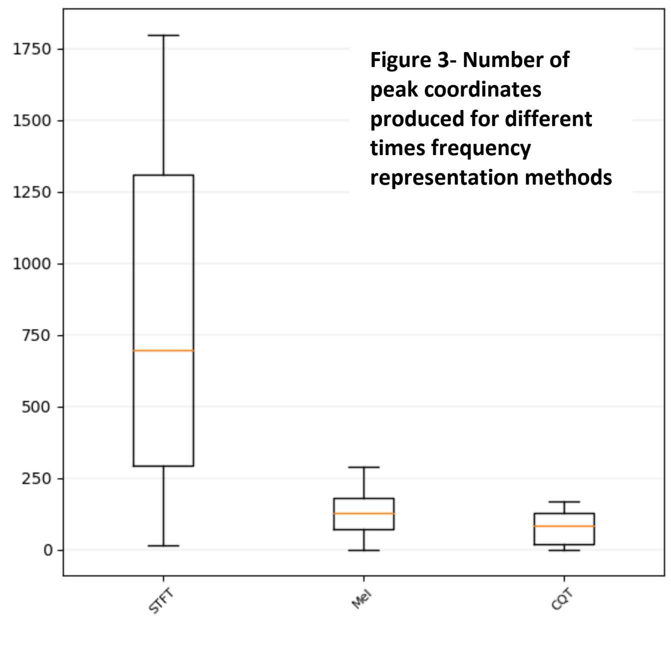


Figure 3- Number of peak coordinates produced for different times frequency representation methods

Accuracy vs. Speed

In order to find the best parameters at each stage, the system was tested with different parameter values for 20 queries (10 classical, 10 pop) against the full database of songs. The effect of query time and result accuracy was measured. For this system, accuracy was prioritised over speed, as a query execution is not necessarily time sensitive, and improving the system's robustness to noise was deemed more useful. However, query time was recorded to investigate the effect of the parameter values against time, and also to try and ensure that the chosen parameter values would not lead to extensively long query timings.

The accuracy results produced from changing parameter values were evaluated according to the Mean Average Precision (MAP) value of the queried data, and the accuracy of the top three results, i.e.:

$$MAP, \overline{P} = \frac{1}{J}\sum_{j=1}^{J}\overline{P_{Q_j}}$$

Where:

- {Q$_1$, …, Q$_j$} is defined as the set of query documents
- $\overline{P_{Q_j}}$ is the average precision for a given query

$$Returned\ Accuracy = \frac{\sum_{j=1}^{J} R}{J}$$

Where:

- $R = \begin{cases} 1, & if\ Q_j\ is\ in\ top\ three\ ranked\ results \\ 0, & otherwise \end{cases}$
- J is the number of query documents

Time Frequency Representations and Parameters

Different spectrogram representations were investigated using default parameters, and a maximum peak length of 2000 so that query time was practical:

Representation	Average Query Time (s)	MAP	Returned Accuracy
STFT	55.99760845	0.20142449	0.25
Mel	53.98121033	0.165507866	0.15
CQT	53.7673689	0.108865847	0.1

An STFT representation was found to give the best accuracy, therefore it was taken forward. This seemed to be due to the fact that the STFT spectrogram produced the largest number of peak coordinate (constellation) points, as can be seen in Figure 3.

STFT parameter values for window length and hop size were investigated. It was found that an increase in frame size, led to an increase in query time (Figure 4), and also an increase in accuracy (Figure 5). Both of these are likely due to the fact that a larger frame size produced more constellation peaks. It was also observed that a longer hop size produces the same effect for query time, but the opposite effect for accuracy. Based on the experimental results found, the best parameters for STFT window length and hop size found for this assignment were chosen to be a window length of 2048, and a hop size of 512.

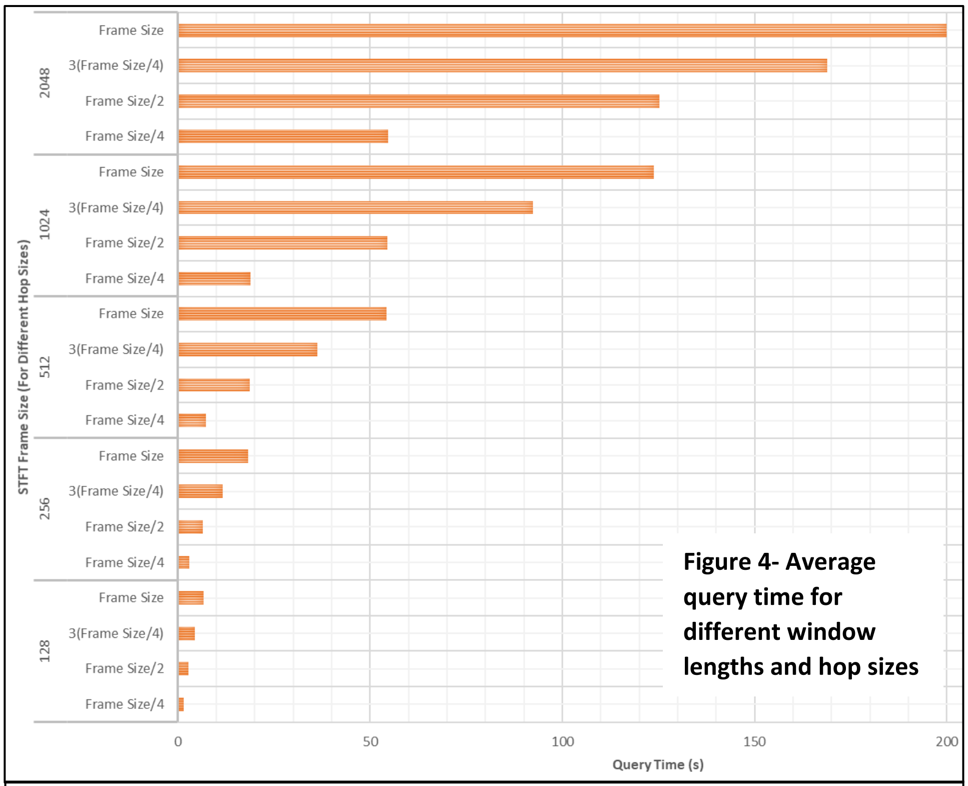


Figure 4- Average query time for different window lengths and hop sizes

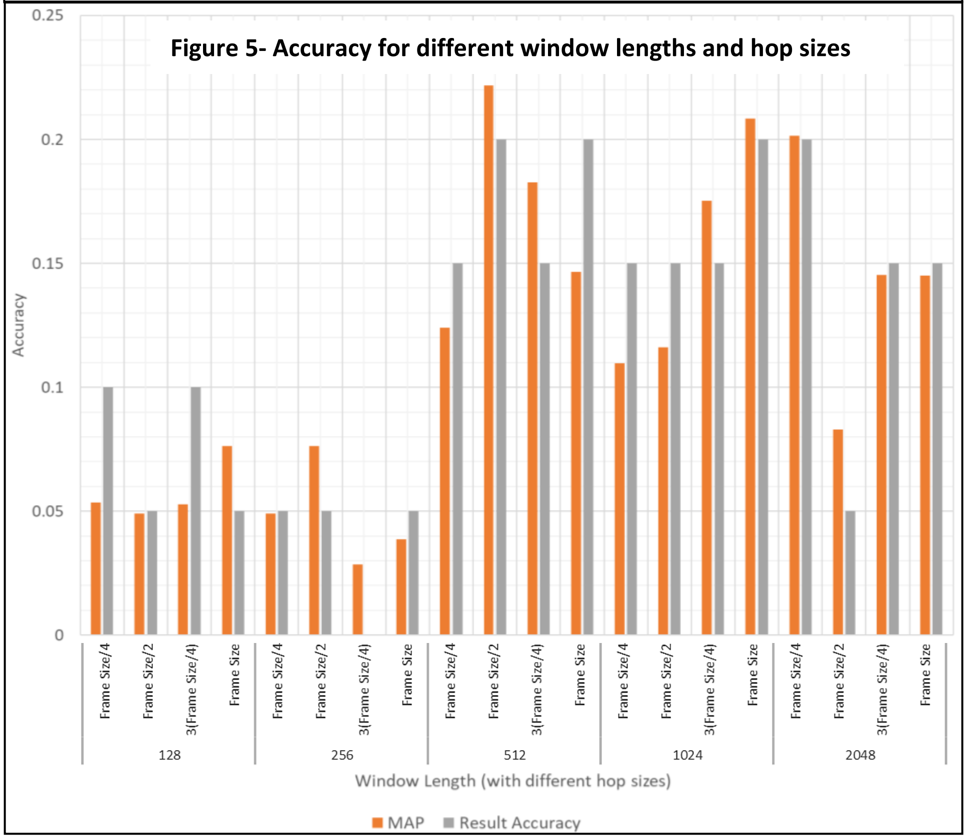


Figure 5- Accuracy for different window lengths and hop sizes

Peak Picking Parameters

In order to investigate the effect of different parameters for peak picking, varying values were tried for the 'min_distance' (the minimum distance by which peaks are separated by), and the 'threshold_rel' (minimum intensity of peaks) parameters of the 'peak_local_max' function used to return the peak coordinates.

It was seen (Figure 6) that increasing the threshold, and also increasing the minimum distance reduced the number of peak coordinates produced. Again, this had the effect of increasing the average query time, however not necessarily improving the accuracy scores and many more incorrect hashes were found with an increase of peak coordinates.

It was found that the values 'min_distance' = x, and 'threshold_rel' = x, were found to produce the best accuracy results of x, with an average query time of x.

Fingerprint Hashing Parameters

When generating the fingerprint hashes that were used to identify a track, the two parameters that could be considered were the 'max hash time delta' value which was used to limit the number of hashes created, and therefore improve query time. According to this parameter, hashes were only created if the 't_delta' value fell below this value,

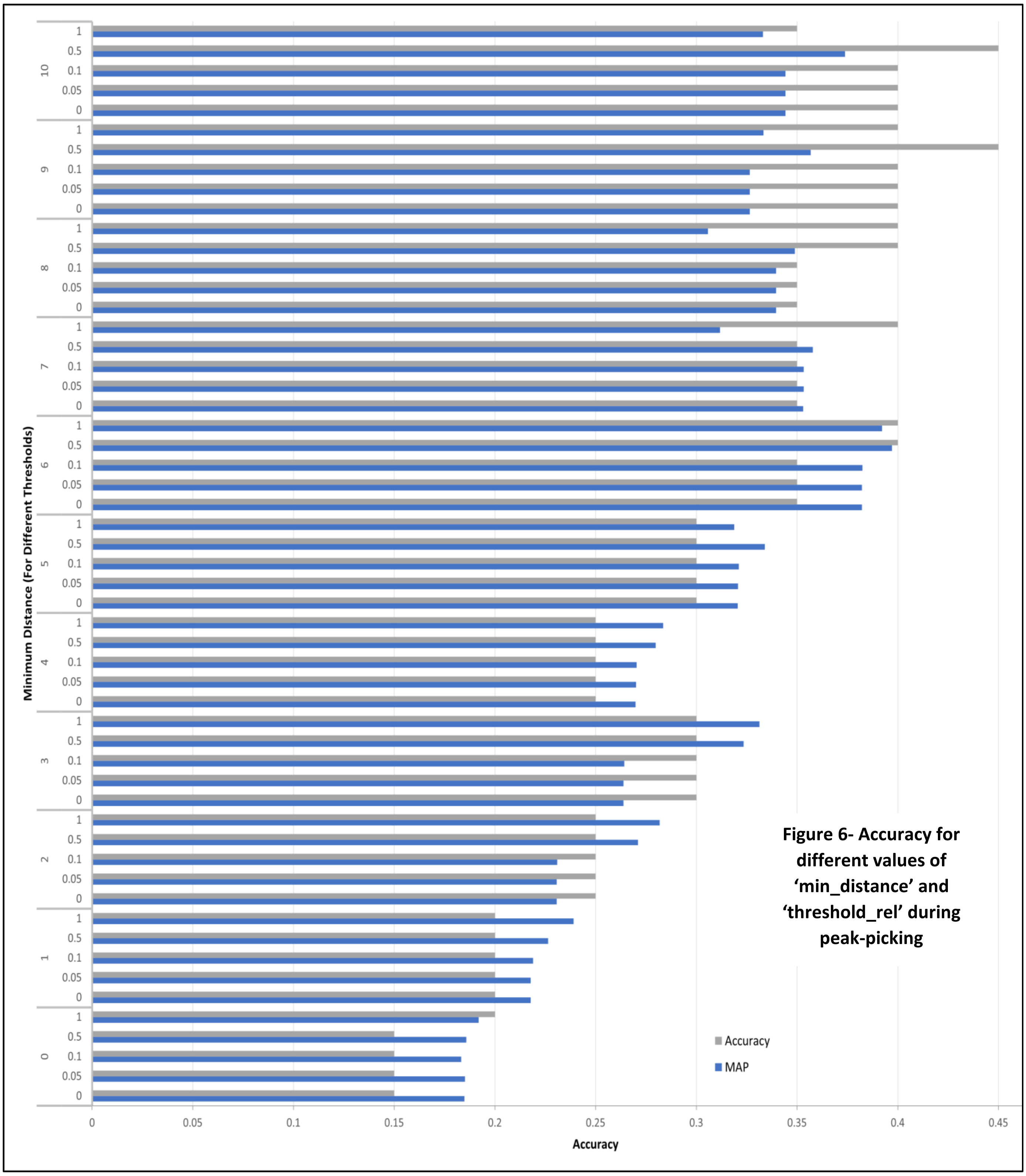


Figure 6- Accuracy for different values of 'min_distance' and 'threshold_rel' during peak-picking

i.e. the peak coordinates were not further apart in time than the chosen value.

The second parameter that could be considered when generating hashes was the 'fan_value' which was used to control the number of peak coordinates that were 'fanned' out from the anchor point.

The effects of varying these parameters were investigated (Figure 7), and it was found that a high 'fan_value' improved accuracy scores noticeably. The 'max hash time delta' parameter was found to have the largest influence on query times, as increasing this value increased the number of hashes generated and therefore search times. If this value was too small, accuracy was reduced, therefore a medium value was preferred.

The optimum values for these parameters were found to be 'max hash time delta' = 500, and ‘fan_value’ = 25.

Results

The system was tested using snippets from the database tracks (no noise), and also against a query database of tracks with noise.

The results were found to be:

Dataset	No Noise Data	Noisy Data
Average Query Time	129.0379502	71.54406668
MAP	0.985	0.561597082
Accuracy	0.95	0.55

Conclusions

The system was found to work very well against data without noise. However, once noise is introduced to the query data the accuracy of the system is reduced significantly. The query time was also longer with the absence of noise as most of the testing was done on a noisy system so the parameters were optimised for that. In the absence of noise more peaks will be found and therefore the query time is longer.

The query time taken is also longer than would be desired. Over the course of the assignment it was found that improving accuracy often resulted in an increased query time as to improve accuracy more data was needed, but this increased search times.

To improve this system I believe the key would be to investigate whether the peak picking could be improved. Perhaps more effort should be made to guarantee an even distribute of peak coordinate points to improve the robustness of the system to noise. If the accuracy could be improved by this method, then ideally less peak coordinated would be needed which would also reduced query time.

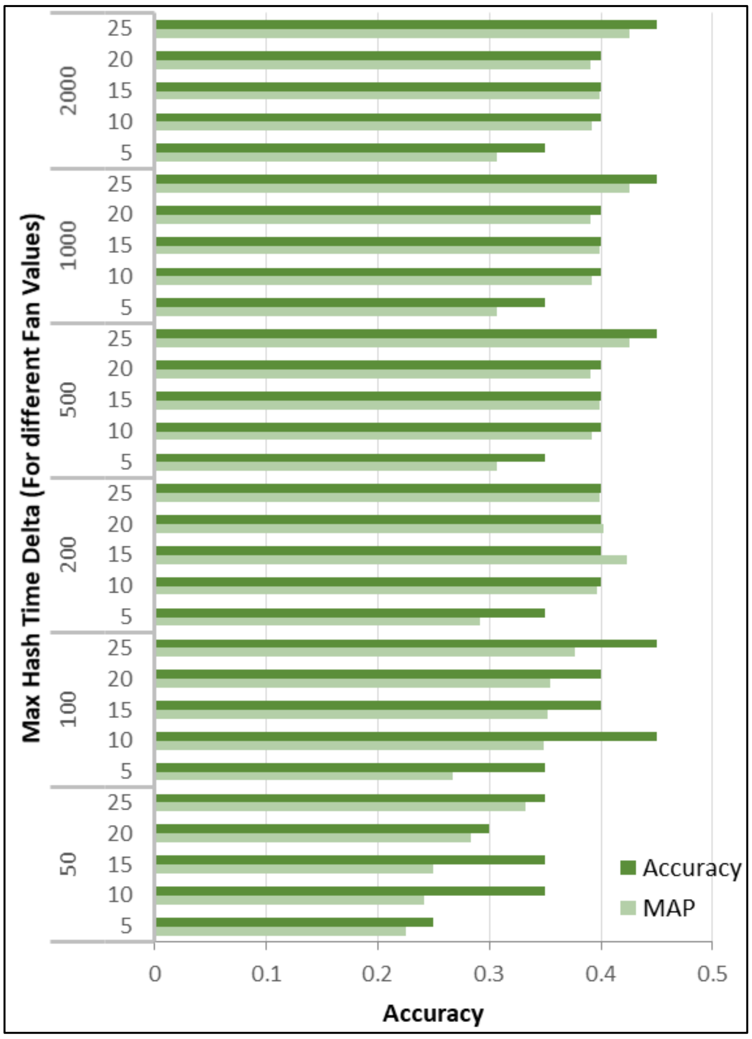


Figure76- Accuracy for different values of ‘max_hash_time_delta’ and ‘fan_value’ during peak-picking

Overall the system was found to be accurate in the absence of noise, but slow. In the presence of noise the system was 56% accurate and a bit faster.

References

[1] A. L.-C. Wang, “An Industrial-Strength Audio Search Algorithm,” Shazam Entertainment, Ltd,, London, 2003, http://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf.

[2] A. Wang, “The Shazam music recognition service,” *Communications of the ACM,* vol. 49, no. 8, https://doi.org/10.1145/1145287.1145312, 2006.