# An Industrial-Strength Audio Search Algorithm

## *Abstract*

*We have developed and commercially deployed a flexible audio search engine. The algorithm is noise and distortion resistant, computationally efficient, and massively scalable, capable of quickly identifying a short segment of music captured through a cellphone microphone in the presence of foreground voices and other dominant noise, and through voice codec compression, out of a database of over a million tracks. The algorithm uses a combinatorially hashed time-frequency constellation analysis of the audio, yielding unusual properties such as transparency, in which multiple tracks mixed together may each be identified. Furthermore, for applications such as radio monitoring, search times on the order of a few milliseconds per query are attained, even on a massive music database.*

## 1 Introduction

Shazam Entertainment, Ltd. was started in 2000 with the idea of providing a service that could connect people to music by recognizing music in the environment by using their mobile phones to recognize the music directly. The algorithm had to be able to recognize a short audio sample of music that had been broadcast, mixed with heavy ambient noise, subject to reverb and other processing, captured by a little cellphone microphone, subjected to voice codec compression, and network dropouts, all before arriving at our servers. The algorithm also had to perform the recognition quickly over a large database of music with over 1M tracks, and furthermore have a low number of false positives while having a high recognition rate.

This was a hard problem, and at the time there were no algorithms known to us that could satisfy all these constraints. We eventually developed our own technique that met all the operational constraints.

The Shazam algorithm can be used in many applications besides just music recognition over a mobile phone. Due to the ability to dig deep into noise we can identify music hidden behind a loud voiceover. On the other hand, the algorithm is also very fast and can be used for copyright monitoring at a search speed of about 1000 times realtime, thus enabling a modest server to monitor significantly many media streams. The algorithm is also suitable for content-based cueing and indexing for library and archival uses.

## 2 Basic principle of operation

The recognition algorithm is described.

### *2.1 Combinatorial Hashing*

Each audio file is "fingerprinted," a process in which reproducible hash tokens are extracted. A time-frequency analysis is performed, marking the coordinates of local

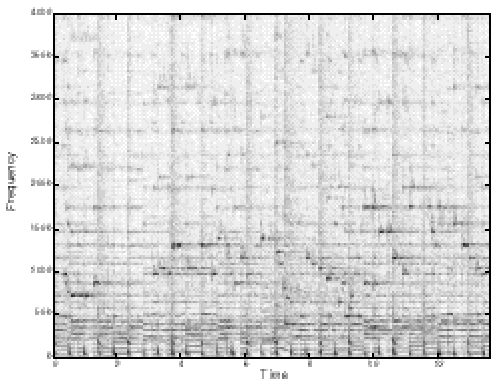

Fig. 1A - Spectrogram

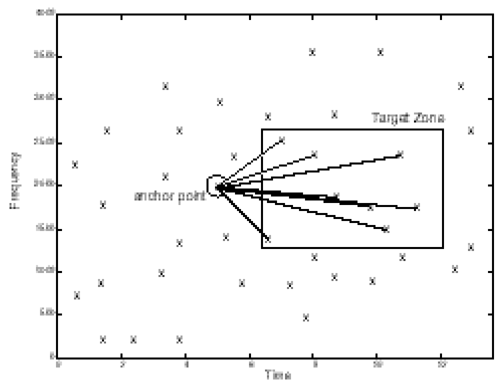

Fig. 1C - Combinatorial Hash Generation

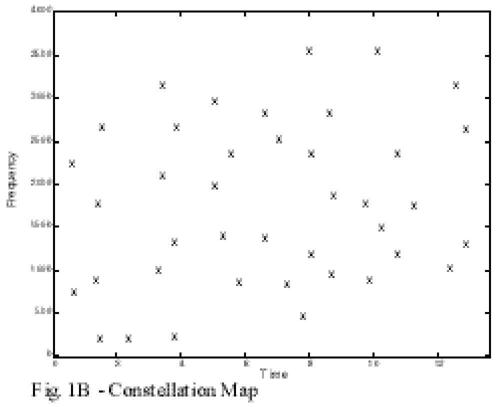

Fig. 1B - Constellation Map

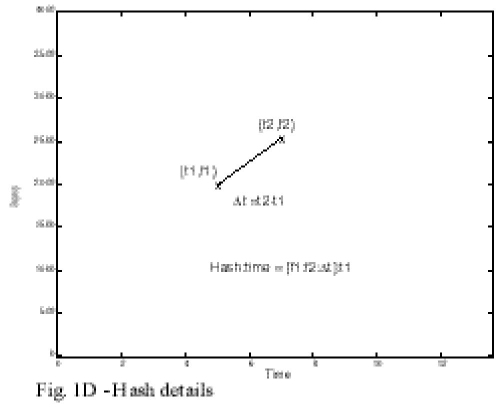

Fig. 1D - Hash details

maxima of a spectrogram (Figures 1A and 1B), thus reducing an audio file down to a relatively sparse set of time-frequency pairs. This reduces the search problem to one similar to astronavigation, in which a small patch of time-frequency constellation points must be quickly located within a large universe of points in a strip-chart universe with dimensions of bandlimited frequency versus nearly a billion seconds in the database. The peaks are chosen using a criterion to ensure that the density of chosen local peaks is within certain desired bounds so that the time-frequency strip for the audio file has reasonably uniform coverage. The peaks in each time-frequency locality are also chosen according amplitude, with the justification that the highest amplitude peaks are most likely to survive the distortions listed above.

Hashes are formed from the constellation map, in which pairs of time-frequency points are combinatorially associated. Anchor points are chosen, each anchor point having a target zone associated with it. Each anchor point is sequentially paired with points within its target zone, each pair yielding two frequency components plus the time difference between the points (Figure 1C and 1D). These hashes are quite reproducible, even in the presence of noise and voice codec compression. Furthermore, each hash can be represented as a 32-bit unsigned integer. Each hash is also associated with the time offset from the beginning of the respective file to its anchor point.

To create a database index, the above operation is carried out on each track in a database to generate a list of hashes and their associated offset times. Track IDs may also be appended to the small data structs, yielding an aggregate 64-bit struct, 32 bits for the hash and 32 bits for the time offset and track ID. To facilitate fast processing, the 64-bit structs are sorted according to hash value.

The number of hashes per second is approximately equal to the density of constellation points per second times the fan-out factor into the target zone. For example, if each constellation point is taken to be an anchor point, and if the target zone has a fan-out of size F=10, then the number of hashes is approximately equal to F=10 times the number of constellation points extracted from the file. By limiting the number of points chosen in each target zone, we seek to limit the combinatorial explosion of pairs. The fan-out factor leads directly to a cost factor in terms of storage space.
By forming pairs instead of searching for matches against individual constellation points we gain a tremendous acceleration in the search process. For example, if each frequency component is 10 bits, and the □t component is also 10 bits, then matching a pair of points yields 30 bits of extra information, versus only 10 for a single point. Then the specificity of the hash would be about a million times greater, and thus the speed of processing is similarly accelerated. On the other hand, due to the combinatorial generation of hashes, assuming symmetric density and fan-out for both database and sample hash generation, there are $F^2$ times as many token combinations to search for, thus the total speedup is a factor of about $1000000/F^2$, or about 10000.

Note that the combinatorial hashing squares the probability of point survival, i.e. if p is the probability of a spectrogram peak surviving the journey from the original source material to the captured sample recording, then the probability of a hash from a pair of points surviving is approximately $p^2$. This reduction in hash survivability is a tradeoff against the tremendous amount of speedup provided. The reduced probability of individual hash survival is mitigated by the combinatorial generation of a greater number of hashes than original constellation points. For example, if F=10, then the probability of at least one hash surviving for a given anchor point would be the joint probability of the anchor point and at least one target point in its target zone surviving. If we simplistically assume IID probability p of survival for all points involved, then the probability of at least one hash surviving per anchor point is $p*[1-(1-p)^F]$. For reasonably large values of F, e.g. F>10, and reasonable values of p, e.g. p>0.1, we have approximately

$$p \approx p*[1-(1-p)^F]$$

so we are actually not much worse off than before.

We see that by using combinatorial hashing, we have traded off approximately 10 times the storage space for approximately 10000 times improvement in speed, and a small loss in probability of signal detection.

Different fan-out and density factors may be chosen for different signal conditions. For relatively clean audio, e.g. for radio monitoring applications, F may be chosen to be modestly small and the density can also be chosen to be low, versus for the somewhat more challenging mobile phone consumer application. The difference in processing requirements can thus span many orders of magnitude.

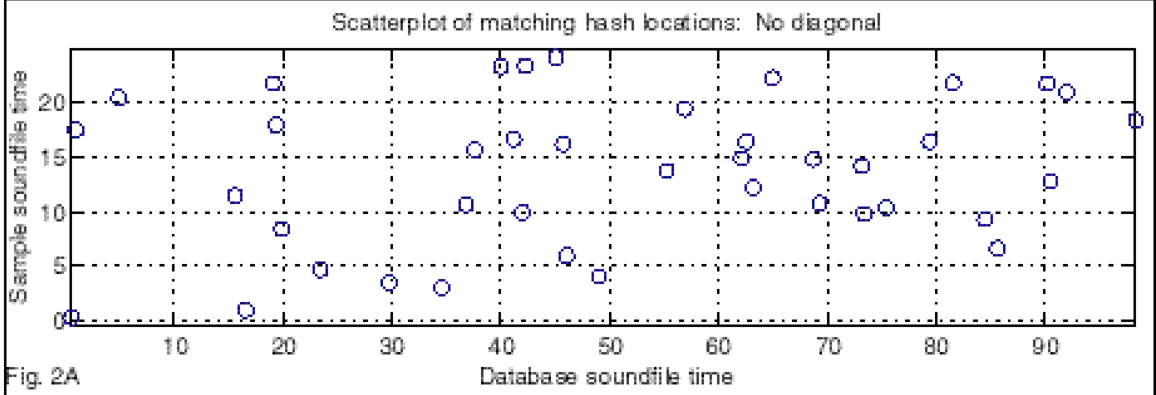

Fig. 2A

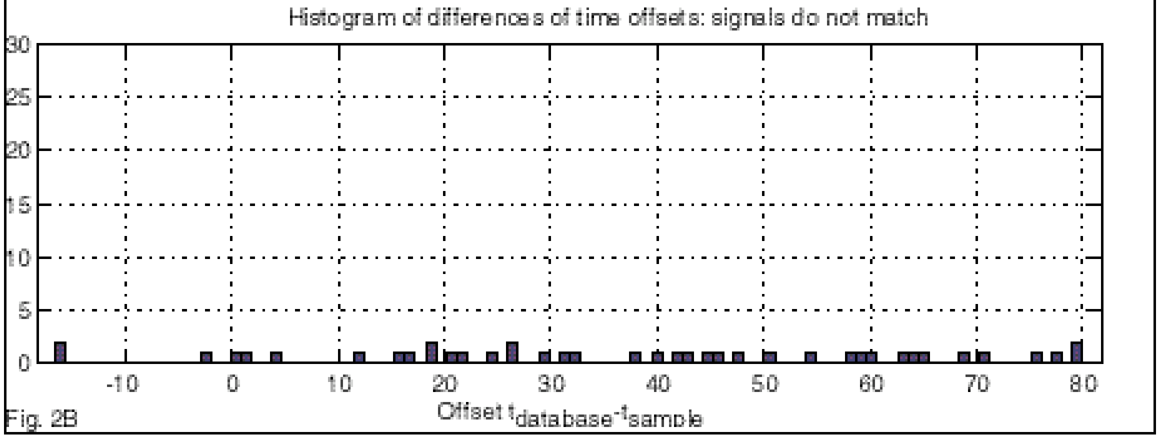

Fig. 2B

## 2.2 Searching

To perform a search, the above fingerprinting step is performed on a captured sample sound file to generate a set of hash+offset records. Each hash from the sample is iteratively used to search in the database for matching hashes. For each matching hash found in the database, the corresponding offset times from the beginning of each file are associated into time pairs. The time pairs are distributed into bins according to the track ID associated with the matching database hash.

After each sample hash has been used to search in the database to form matching time pairs, the bins are scanned for matches. Within each bin the set of time pairs represents a scatterplot of association between the sample and database sound files. If the files match, matching features should occur at similar relative offsets from the beginning of the file, i.e. a sequence of hashes in one file should also occur in the matching file with the same relative time sequence. The problem of deciding whether a match has been found reduces to detecting a significant cluster of points forming a diagonal line within the scatterplot. Various techniques could be used to perform the detection, for example a Hough transform or other robust regression technique. Such techniques are overly general, computationally expensive, and susceptible to outliers.

Due to the rigid constraints of the problem, the following technique solves the problem in approximately N*log(N) time, where N is the number of points appearing on the scatterplot. For the purposes of this discussion, we may assume that the slope of the

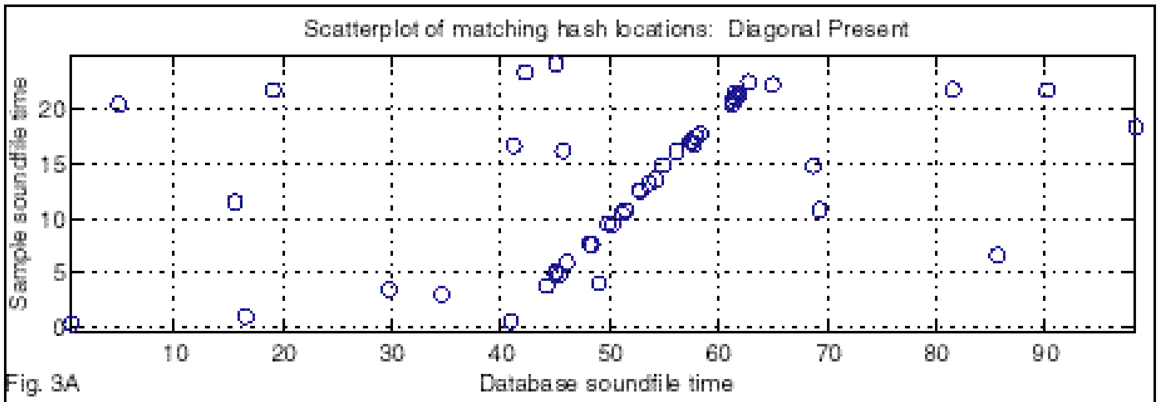

Fig. 3A

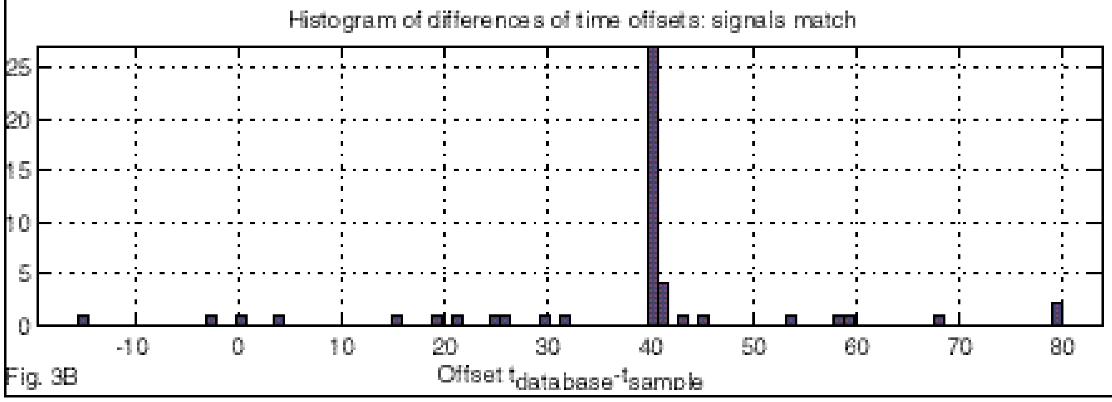

Fig. 3B

diagonal line is 1.0. Then corresponding times of matching features between matching files have the relationship

$$t' = t + \text{offset},$$

where t' is the time coordinate of the feature in the matching database soundfile and t is the time coordinate of the corresponding feature in the sample soundfile to be identified. For each $(t_k', t_k)$ coordinate in the scatterplot, we calculate

$$\Delta t_k = t_k' - t_k.$$

Then we calculate a histogram and scan for a peak. This may be done by sorting the set of $\Delta t_k$ values and quickly scanning for a cluster of values. The scatterplots are usually very sparse, due to the specificity of the hashes due to the combinatorial method of generation as discussed above. Since the number of time pairs in each bin is small, the scanning process takes on the order of microseconds per bin, or less. The score of the match is the number of matching points in the histogram peak. The presence of a statistically significant cluster indicates a match. Figure 2A illustrates a scatterplot of database time versus sample time for a track that does not match the sample. There are a few chance associations, but no linear correspondence appears. Figure 3A shows a case where a significant number of matching time pairs appear on a diagonal line. Figures 2B and 3B show the histograms of the $\Delta t_k$ values corresponding to Figures 2A and 3B.

This bin scanning process is repeated for each track in the database until a significant match is found.

Note that the matching and scanning phases do not make any special assumption about the format of the hashes. In fact, the hashes only need to have the properties of having sufficient entropy to avoid too many spurious matches to occur, as well as being reproducible. In the scanning phase the main thing that matters is for the matching hashes to be temporally aligned.

## 3 Performance

### 3.1 Noise resistance

The algorithm performs well at levels of noise down below –6 dB SNR (for white noise), and even –9 dB SNR for structured noise, such as voiceovers. We can correctly identify music in the presence of voices, traffic noise, dropout, and even other music. To give an idea of the power of this technique, from a heavily corrupted 15 second sample, a statistically significant match can be determined with only about 1-2% of the generated hash tokens actually surviving and contributing to the offset cluster. Even moderately-to-highly distorted audio samples can easily have separations in excess of 10 standard deviations from the threshold. Furthermore, a property of the scatterplot histogramming technique is that discontinuities are transparent, allowing immunity to dropouts and masking due to interference.

One somewhat surprising result is that even with a large database, we can correctly identify each of several tracks mixed together, including multiple versions of the same piece, a property we call "transparency".

### 3.2 Speed

For a database of about 20 thousand tracks implemented on a PC, the search time is on the order of 5-500 milliseconds, depending on parameters settings and application. The service can find a matching track for a heavily corrupted audio sample within a few hundred milliseconds of core search time. With "radio quality" audio, we can find a match in less than 10 milliseconds, with a likely optimization goal reaching down to 1 millisecond per query.

### 3.3 Specificity and False Positives

The algorithm was designed specifically to target recognition of sound files that are already present in the database. It is not expected to generalize to live recordings. That said, we have anecdotally discovered several artists in concert who apparently either have extremely accurate and reproducible timing (with millisecond precision), or are more plausibly lip synching.

The algorithm is conversely very sensitive to which particular version of a track has been sampled. Given a multitude of different performances of the same song by an artist, the algorithm can pick the correct one even if they are virtually indistinguishable by the human ear.

We occasionally get reports of false positives. Often times we find that the algorithm was not actually wrong since it had picked up an example of "sampling," or plagiarism. Still, we must make a choice about how to set the threshold for deciding the minimum score necessary to decide whether a match is significant. There is a classic tradeoff between true hits and false positives, and thus the maximum allowable percentage of false positives is a design parameter that is chosen to suit the application.

## *3.4 Applications*

The technology is flexible and has many uses.

### 3.4.1 Consumer mobile music recognition service

We have deployed the algorithm to scale in our commercial music recognition service, with over 1.7M tracks in the database. The service is currently live in Germany and the UK, with several hundred thousand users, and will soon be available in additional countries in Europe, Asia, and the USA. The user experience is as follows: The user hears music playing in the environment. She calls up our service using her mobile phone and samples up to 15 seconds of audio. An identification is performed on the sample at our server, then the track title and artist are sent back to the user via SMS text messaging. The information is also made available on a web site, where the user may register and log in with her mobile phone number and password. At the web site, or on a smart phone, the user may view her tagged track list and buy the CD. The user may also download the ringtone corresponding to the tagged track, if it is available. The user may also send a 30-second clip of the song to a friend. Other services, such as purchasing an MP3 download may become available soon.

### 3.4.2 Radio and media monitoring application

We are turning our attention to media monitoring, in which we can tune the algorithm to take advantage of the high (i.e. above 0 dB) SNR for increased speed performance, allowing us to perform copyright monitoring for royalty distribution. We could monitor every major radio station in the US in realtime with a modest number of PCs (~10), thus providing full-accountability monitoring.

## 4 Acknowledgements

Thanks to Julius O. Smith, III for letting me bounce around random and impractical ideas, for providing the fundamental insight that this must be a solvable problem due to the existence proof that we humans obviously are able to recognize audio (probably without resorting to quantum computation and the like). Thanks also to Daniel P. W. Ellis for getting me oriented in the field of audio recognition. Thanks to Chris Barton, and Philip Inghelbrecht, as well Dhiraj Mukherjee, for going out on a limb to envision and push this crazy idea before it became possible. And thanks to my wife Katja for letting me do yet another startup, against her better judgement.