MoC - Master of Celebration

Automatic Playlist Generation

Version 1.0
August 26, 2003

Dominik Schnitzer[1]

This paper describes an implementation of the Music Similarity Measure of Jean-Julien Aucouturier and Francois Pachet described in their paper ***Music Similarity Measures: What's the Use?*** *The proposed algorithm to compute a timbre distance between different pieces of music was used to create an automatic playlist generation tool - MoC. Programming it was the main task of my interim at the IMKAI Institute of the University of Vienna. Read on for the results and implementation details.*

[1]<dominik@schnitzer.at>

Contents

1 Overview

MoC aims to be an automatic playlist generation tool for your digital music collection. Basically MoC analyzes the collection using Jean-Julien Aucouturier's and Francois Pacet's Music Similarity Measure described in [1], which enables computation of timbre distances between songs. After analysis of a music collection, MoC is able to automatically generate playlists by searching for the most similar songs for a given query song. The query song is selected by the user, the playlist and thus the search for similar songs done automatically by MoC. MoC doesn't include any song meta data in it's search, it works on raw audio data and analysis of it.

MoC runs under GNU/Linux and is written in C++. MoC is my first experience in experimenting with music similarity measures. It was created as an interim work at the IMKAI Institute of the University of Vienna [2].

2 Implementation

As mentioned before, MoC basically implements the ideas presented in [1]. In addition to the steps presented in the mentioned paper, extra processing like normalizing the music had to be done, to get good results with the similarity measure. MoC has two operation modes: Analysis and Query mode. Songs are analyzed and added to the MoC repository in Analyze mode, in Query mode playlists are generated. A workflow diagram is presented for every mode, which is then described step by step. Text in brackets generally points to files in the MoC program sources, to make finding the referenced functions easier.

2.1 Requirements

The following libraries are required to successfully build MoC on your Linux machine:

- GStreamer Libraries, Version 0.6.0, `http://www.gstreamer.net/`
  These libraries do the MP3 decoding and down-sampling of the audio data, so basically they are required to get the music files in the right shape for analysis.

- FFTW Library, Version 3.0, `http://www.fftw.org/`
  This is the fastest library to get an FFT analysis done.

- GTKMM Libraries, Version 2.2, `http://gtkmm.sf.net/`
  Used for the tiny Graphical User Interface, for generating playlists.

- Besides: A fairly new GCC Compiler (Version 3.2) will help.

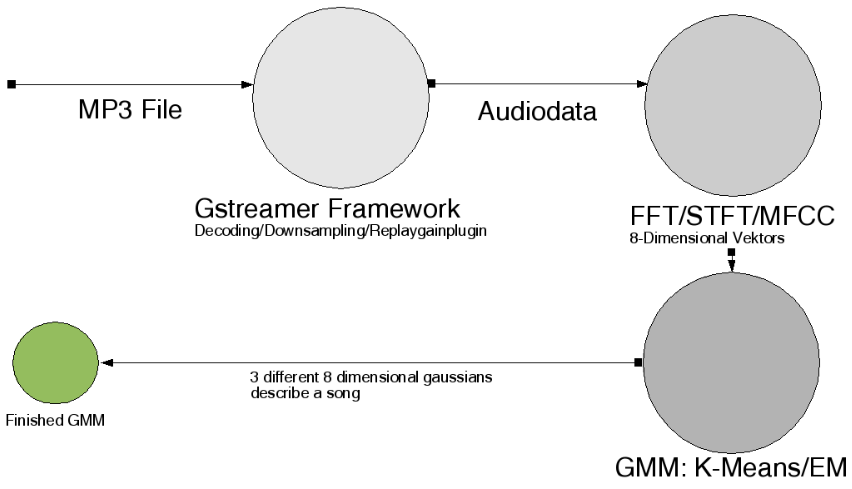


Figure 1: Audio Analysis in MoC

2.2 Analysis

Figure 1 sketches the whole process of sound analysis, taking place in MoC.
But before real timbre analysis of a music piece, following preprocessing steps have to be taken:

- *Decode the compressed music file.* Currently MoC can only handle MP3 audio files. It uses the GStreamer audio framework to decode the MPEG data. It will be quite easy to extend the supported file-types of MoC, by just using the appropriate GSteamer plug-in. (`src/audiofilein.cc`)
- *Down-sample it to mono, 22050Hz, 16Bit.* After decoding, audio is down-sampled with a GStreamer plug-in to one channel and a predefined frequency. [Music Similarity Measures: What's the Use] doesn't mention this step, but for the sake of processing speed and usability of the program, downsampling seemed absolutely necessary. (`src/audiofilein.cc`)
- *Normalize the audio data.* This step had to be taken into account (with all it's bad side effects), since my reference and test music collection, like everybody else's, varied greatly in sound volume. In MoC a freely available algorithm to normalize music was chosen: normalization with the replaygain algorithm. Replaygain takes various psychoacoustic thoughts into account, and is in a C and Matlab reference implementation available here [3]. Since it takes some psychoacoustic thoughts into account, it should at least result in a qualitative quite good normalization.

MoC uses a freely available replaygain C-implementation. (`src/gstreplaygain.c`, `src/gain_analysis.c`)

After these preprocessing steps real audio signal analysis starts:

- *Apply a STFT on the audio data.* It is suggested in [1] that a Short Time Fourier Transformation should be done for every 50ms of audio. MoC uses a 2048 samples windowsize, resulting in a 90ms analysis window. Since MoC overlaps 1/2 of the processed window in every step, the effective STFT resolution actually boils down to 50ms again.
  The FFT analysis is done with help of the high speed FFTW Library, and uses double precision. (`src/stft.cc`, `src/moc.cc::get_gmm()`)
- *Timbre extraction using MFFCs.* In this step the data vectors resulting from the STFT are used to find the Mel Frequency Cepstrum Coefficients for every window. The MFCCs for each frame then describe it's local timbre. Only the first 8 MFCCs are computed. MoC uses Malcom Slaney's Matlab MFCC implementation [4] as a reference for it's C++ one. After computation of the MFCCs, the timbre analysis data of a typical 3 minute song would consist of about 32.000 coefficients (4000 analysis windows of 8 dimensional MFCC analysis vectors). (`src/mfcc.cc`)
- *Fit the timbre data a Gaussian Mixture Model.* In this step it is tried to fit the timbre analysis data vectors of the previous step into a Gaussian Mixture Model consisting of three, 8-dimensional Gaussian distributions (a GMM with M=3). To fit the data into the three gaussians, as proposed, classic Expectation Maximization (EM) is used for finding the optimum cluster. In this implementation EM with a maximum of 800 iterations is performed. EM is initialized by a standard K-Means clusterer.
  After fitting the datapoints into the GMM only 51 numbers characterize a song - the parameters of three gaussians plus a weighting coefficient for each gaussian, specifying the importance/weight of it in the GMM.
  (`src/moc.cc::clusterby_kmeans()|clusterby_em()`)

After analysis, the during the song timbre analysis gathered data, is stored in the $HOME/.moc/ directory. Each song indexed under it's MD5 checksum. By using the file's MD5 checksum as an index, duplicates are avoided and results of analysis can be tracked down manually easy, by finding out an mp3 file's MD5 sum (i.e. using the `md5` command) and looking for the according file in the $HOME/.moc/ directory. (`src/gmm.cc::save_model()`)
The contents of a typical analysis file looks like this:

```
/music/Nena - Irgendwie, Irgendwo, Irgendwann (New Version).mp3
GMM: Mixture-Models 3 Gauss-Dimensions 8
MD5: 6a78ce7216ae8aeb38a80e3309b12b92
```

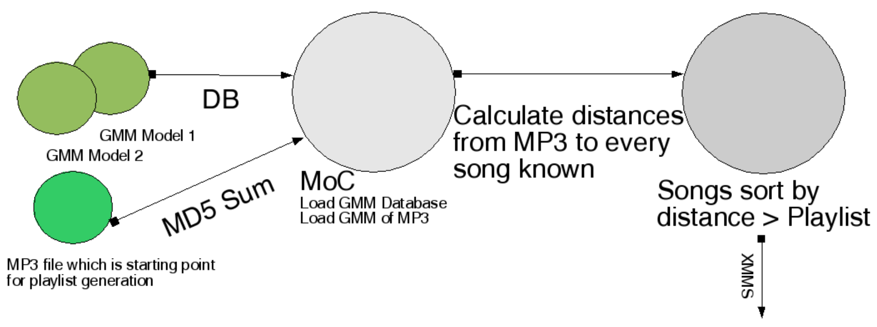

DB
GMM Model 1
GMM Model 2
MD5 Sum
MP3 file which is starting point for playlist generation
MoC
Load GMM Database
Load GMM of MP3
Calculate distances from MP3 to every song known
Songs sort by distance > Playlist
XMMS

Figure 2: Playlist generation in MoC

```
Covariances:
Model 0: 101.641 0.332 0.153 0.141 0.157 0.090 0.103 0.096
Model 1: 1.320 0.137 0.194 0.092 0.050 0.057 0.040 0.043
Model 2: 0.490 0.098 0.091 0.099 0.055 0.042 0.042 0.039
Means:
Model 0: -21.014 1.365 -0.072 0.055 -0.226 0.082 -0.001 0.247
Model 1: -14.825 2.144 0.203 0.436 0.181 0.163 0.051 0.096
Model 2: -12.612 1.445 0.282 0.354 0.187 0.227 -0.039 0.117
Priors:
Model 0: 0.07316254277
Model 1: 0.3204650521
Model 2: 0.6063724051
```

2.2.1 Usage: Adding MP3s

After building and installing MoC on your machine, you'll have to add some MP3s to MoCs repository, to generate playlists in the next step. Issue at the command prompt:

```
moc -a *.mp3
```

This will add all files in the current directory to MoC. It will take some time. Adding one MP3 takes about 1 minute on a 600 MHz machine. After adding some more MP3s, first playlists can be generated, by using the query mode, which is described in the next chapter. All analysis information gathered through time is placed in $HOME/.moc/.

2.3 Query

The Query mode of MoC is shown in *Figure 2*. It is used to generate playlists using one song chosen by the user (the query song) as base song for generation. Before playlist generation can take place the following initialization steps have to be done:

- *Load the song database.* During MoC initialization all previously computed song models are loaded into memory. (`src/moc.cc::init_database()`)
- *Lookup the Query song.* After database initialization the given query song is looked up in the database. Lookup is done by calculating the MD5 sum of the MP3 file and searching for the checksum in the database. (`src/main.cc`)

Now that all required song information and song models are available and playlist generation can proceed.

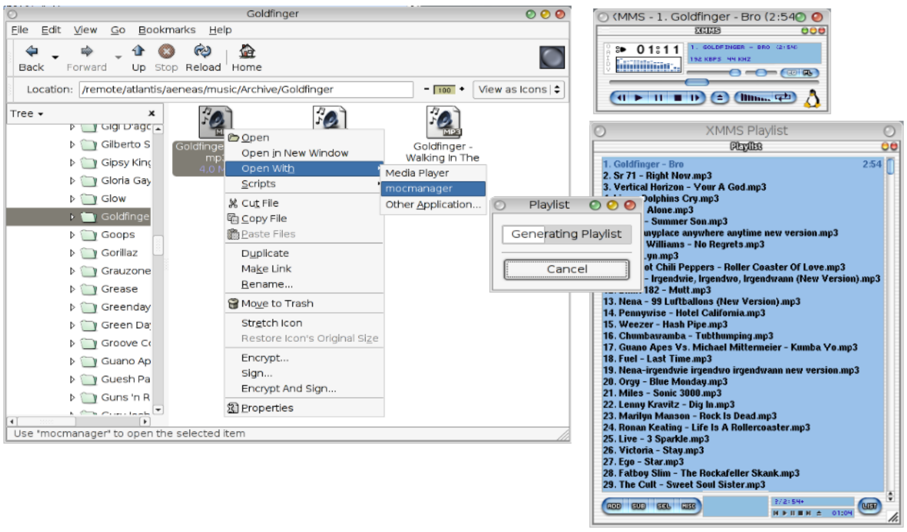
- *Compute all song distances the query song.* To compute distances from one song to another a simple technique was chosen: For every GMM in the database, 2000 datapoints are sampled using random numbers based on the gaussian parameters of the song models. Then the likelihood of the newly sampled datapoints, given the query song model, is computed. This basically gives a probability how similar a GMM is to another - this value is neither normalized nor symmetric. Unfortunately [1] doesn't say how to normalize the calculated pseudo distances. At the time of writing Aucouturier and Pachet published a new paper on their website [5], explicitly mentioning the normalization process, but the given formula isn't very clear on this topic too.
  This is what MoC uses for timbre distance calculation between two songs a and b. (`src/gmm.cc`)

Timbre Distance Computation in MoC

$$D(a,b) = \left(\left| 1 - \frac{P(a,b)}{P(a,a)} \right| + \left| 1 - \frac{P(b,a)}{P(b,b)} \right| \right) * \frac{1}{2} \qquad (1)$$

D(a,b)	timbre distance between song a and b
P(a,b)	sum of the probabilities, that a datapoint, which was sampled with the GMM of b, is created by the GMM of a.

- *Playlist output.* After all distances from the query song to the songs in the database are computed, this list is reverse-sorted by distance. The first N songs (N is chosen by the user) of this list are the songs, which are closest to the query song. Those songs form the final playlist and are written to the terminal/a file or loaded in the MP3 player, depending on what MoC front-end is used. (`src/main.cc`)



```
1. Goldfinger - Bro                                   2:54
2. Sr 71 - Right Now.mp3
3. Vertical Horizon - Your A God.mp3
Dolphins Cry.mp3
Alone.mp3
- Summer Son.mp3
nyplace anywhere anytime new version.mp3
Williams - No Regrets.mp3
.yn.mp3
ot Chili Peppers - Roller Coaster Of Love.mp3
- Irgendwie, Irgendwo, Irgendwann (New Version).mp3
182 - Mutt.mp3
13. Nena - 99 Luftballons (New Version).mp3
14. Pennywise - Hotel California.mp3
15. Weezer - Hash Pipe.mp3
16. Chumbawamba - Tubthumping.mp3
17. Guano Apes Vs. Michael Mittermeier - Kumba Yo.mp3
18. Fuel - Last Time.mp3
19. Nena-irgendwie irgendwo irgendwann new version.mp3
20. Orgy - Blue Monday.mp3
21. Miles - Sonic 3000.mp3
22. Lenny Kravitz - Dig In.mp3
23. Marilyn Manson - Rock Is Dead.mp3
24. Ronan Keating - Life Is A Rollercoaster.mp3
25. Live - 3 Sparkle.mp3
26. Victoria - Stay.mp3
27. Ego - Star.mp3
28. Fatboy Slim - The Rockafeller Skank.mp3
29. The Cult - Sweet Soul Sister.mp3
```

Figure 3: Using `mocmanager` in the Gnome Nautilus File-manager.

2.3.1 Usage: Playlist Generation

After having added some MP3s to MoC's repertoire, it's possible to generate first playlists. The command:

```
moc -q 10 '/music/Die Aerzte - Ein Lied fuer dich.mp3' > play.m3u
```

generates a 10 elements playlist, starting with the song *Die Aerzte - Ein Lied fuer dich.* After about 30 seconds (for my 1700 titles large collection) the following playlist was generated:

```
/music/Die Aerzte - Ein Lied fuer dich.mp3
/music/Die Aerzte - Meine Freunde.mp3
/music/U2 - Elevation (Tomb Raider Mix).mp3
/music/Ac Dc - Hells Bells.mp3
/music/Die Toten Hosen - Paradies.mp3
/music/Die Aerzte - Angeber.mp3
/music/Die Aerzte - Schunder-Song.mp3
/music/Puddle Of Mud - She Hates Me.mp3
/music/Die Aerzte - Langweilig.mp3
/music/Rem - Its The End Of The World As We Know It.mp3
```

The above command pipes all output of MoC into a file called `play.m3u`, which can be played by modern MP3 players, like xmms:

```
xmms play.m3u
```

MoC in it's current release also includes a gtk2 front-end, which can be used to integrate MoC in your preferred file-manager. The binary is called `mocmanager` and has the same command line syntax as the classic `moc` program, so MP3 files can be easily linked, to be processed with `mocmanager`. The GUI version of MoC has a progress bar, indicating the playlist generation progress and directly opens the results in your favorite MP3 player.

3 Evaluation

MoCs project web-page is: `http://www.schnitzer.at/dominik/moc/`
Besides the latest MoC version and more documents about it, the web-page also hosts a copy of the test data, 1700 song analysis results, used in this paper.

3.1 Sample Playlists

A Good Playlist This playlist is an example for a very good result. As seen, base for the query was the song *Tori Amos - China*. In the 20 elements long result 5 songs of *Tori Amos* are found and - except song 5 - all sound very similar: They have less or low background music and the artist tries to sing very beautiful (i.e. *Evita*, *Diana Krall*, *Ella Fitzgerald*). Surprisingly even the schmaltzy *N'sync* song fits into the playlist.

```
1  /music/Tori Amos - China.mp3
2  /music/Tori Amos - Silent All These Years.mp3
3  /music/Tori Amos - Happy Phantom.mp3
4  /music/Tori Amos - Famous Blue.mp3
5  /music/Ferris Mc - Take Your Chance (Viel Zu Spaet).mp3
6  /music/Tony Bennett - I Left My Heart In San Francisco.mp3
7  /music/Spooks - Karma Hotel.mp3
8  /music/Usher - You Remind Me of a Girl (Isis Remix).mp3
9  /music/Marilyn Monroe - I Wanna Be Loved By You.mp3
10 /music/Ella Fitzgerald - Night in Tunisia.mp3
11 /music/N'sync - This I Promise You.mp3
12 /music/Diana Krall - The Night We Called It A Day.mp3
13 /music/John Debney - A New Hope (The Emperors New Groove).mp3
14 /music/Evita - Don't Cry For Me Argentina .mp3
15 /music/R. Kelly - The World's Greatest.mp3
16 /music/Tori Amos - Never Seen Blue.mp3
17 /music/Tom Jones - Whats New Pussy Cat.mp3
18 /music/Deine Lakaien - 2nd Sun.mp3
19 /music/Bob Marley - Waiting In Vain.mp3
20 /music/Profyle - Liar.mp3
```

A Normal Playlist In this average query result using the punk rock song *The Offspring - Self Esteem* as base song, about 40% of the songs on the playlist don't fit on it: The first 7 songs are well chosen, but *Shania Twain*, the country queen, or *Aretha Franklin* and all songs at position 15+ are bad choices - Interesting to listen to, but with quite a few inappropriate songs on the playlist, it's a not so appealing experience if one is in the mood for a certain music style.

```
1  /music/Offspring - Self Esteem.mp3
2  /music/Die Aerzte - Mnner Sind Schweine.mp3
```

```
3  /music/The Cult - Edie (Ciaobaby).mp3
4  /music/Die Aerzte - Die traurige Ballade von Susi Spakowski.mp3
5  /music/Bon Jovi - Bad Medicine.mp3
6  /music/Apollo 440 - Aint Talking About Dub.mp3
7  /music/Jbo/Jbo - Hose Runter.mp3
8  /music/Shania Twain - That Don't Impress Me Much.mp3
9  /music/Him - Join Me.mp3
10 /music/Mad Caddies - Youth Gone Wild.mp3
11 /music/Die Aerzte - Mach Die Augen Zu.mp3
12 /music/Die Aerzte - Geh mit mir.mp3
13 /music/Aretha Franklin - Respect.mp3
14 /music/Die Aerzte - Ignorama.mp3
15 /music/Bruce Springsteen - Born In The Usa.mp3
16 /music/Fettes Brot - Schade Schokolade.mp3
17 /music/Los Del Rio - Macarena.mp3
18 /music/Dj Tomekk - 1,2,3 Rhymes Galore.mp3
19 /music/Gerd Show - Der Steuersong (Ketchup Song Rmx).mp3
20 /music/Corrs - Runaway.mp3
```

Bad Playlists Currently, and this partly has it's origin in normalization of music before analysis, especially disco music with heavy drums and electronic sounds gives very poor playlists. An example playlist, where just one interesting result was found (song 5) illustrates the problem:

```
1  /music/Djs At Work - Someday (Vocal Edit).mp3
2  /music/Susan Vega - My Name Is Luca.mp3
3  /music/Corrs - Breathless.mp3
4  /music/Michael Sembello - Maniac.mp3
5  /music/Sophie B. Hawkins - Right Beside You.mp3
6  /music/Tori Amos - Do It Again.mp3
   ...
```

3.2 Problems and Ideas for better Results

As shown in the previous section, sometimes weird playlists are computed using the implemented music similarity measure, playlists where anybody could tell, that songs on it just dont fit together.

Apart from the fact that really good working similarity measure isn't found yet, in the case of the implemented timbre similarity measure, sometimes bad performance could also origin from the circumstance that a song can't be represented well by 3 GMMs. Maybe using more GMMs for this very song would improve results dramatically. Maybe using a 2 GMM model would be perfect. Dynamically choosing the number of GMMs which represent a song best, would be ideal. Basically the problem in this case would be finding the optimum number of clusters for the data-vectors to be analyzed. Unfortunately determining the optimum number of clusters is not trivial.

Another problem occurring in the current implementation, not directly related to this similarity measure, but with playlist generation in general, is best shown in *Figure 4*. The problem shown comes from the fact that the playlists in MoC are all calculated with the query/start song in mind, so they should all sound in a way similar to this song, but among each other timbre distances could be very large. This issue could be resolved if

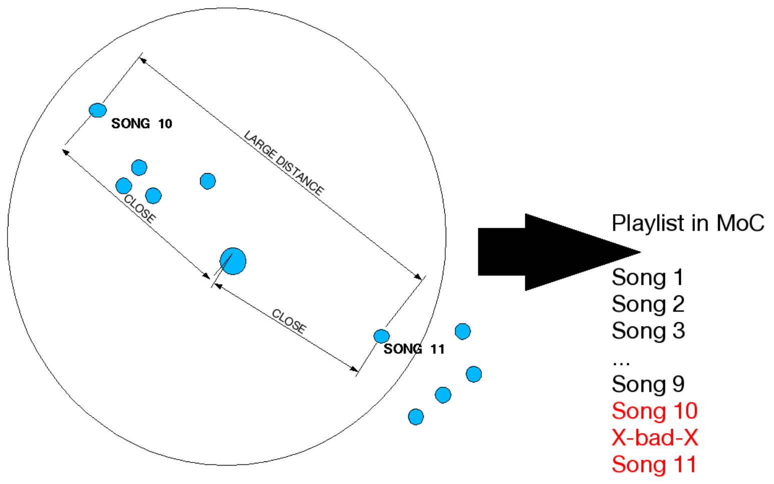


Figure 4: A Typical Playlist Generation Problem in MoC

the songs in MoC are put on a 2 dimensional grid. To archive this, distances from every song to each other would have to be calculated, which is very time-consuming.

3.3 Further Improvements

Apart from the already suggested improvements which could improve the resulting playlists by just relying on a timbre analysis, the following ideas could be merged into MoC to further improve computed playlists:

1. Use song meta-information like artist/band or genre to spice up the results upon user preference. This way it could be avoided that just songs of one artist are put on the list. This requires a well maintained music collection or a way to gather song information online.

2. Include another similarity measure like automatic genre recognition. Internally MoC could include a predefined genre tree of/how genres fit/absolutely do not fit together. Using this additional information genres which absolutely don't fit together (like the Bob Marley's drums and some Disco beats) can be separated clearly.

3. To speed up analysis, it could also be tried to operate directly on MPEG data, which is the common format of music to be analyzed nowadays. The computational heavy part of decoding and (parts of) audio analysis would be unnecessary, if analysis could be done directly on the MPEG audio data. This idea is also mentioned in a follow-up paper of Aucouturier and Pachet [5] and is closer researched in this paper [6].

4 Conclusion

MoC in it's current stage (Version 0.1.1) is not yet ready for Jon Doe' living room and his daily playlist generation. But it provides a good base and playground for further experiments and improvements in the research area of automatic playlist generation and music similarity.

References

[1] Aucouturier, J.J. & Pachet, F. Music Similarity Measures: What's The Use?. In proceedings of the 3rd International Symposium on Music Information Retrieval (ISMIR) 2002 Paris, France.

[2] IMKAI Institute/University of Vienna, Austria
Web: `http://www.ai.univie.ac.at/`

[3] ReplayGain Volume Normalization Group
Web: `http://www.replaygain.org/`

[4] Slaney, M. Mel Frequency Cepstrum Coefficient Analysis. Matlab Implementation, 1998, Interval Research Corporation.

[5] Aucouturier, J.J. & Pachet, F. Finding Songs That Sound The Same. In proceedings of the 1st IEEE Benelux Workshop on Model based Processing and Coding of Audio (MPCA) 2002 Leuven, Belgium.

[6] Tzanetakis, G. and Cook, P. Sound Anaylsis using MPEG Compressed Audio. In Proc. IEE International Conference on Acoustics, Speech and Signal Processing, Istanbul 2000