Realtime Chord Recognition of Musical Sound: a System Using Common Lisp Music

Takuya FUJISHIMA *
CCRMA, Stanford University
Stanford CA 94305, USA
fujishim@ccrma.stanford.edu

Abstract

This paper describes a realtime software system which recognizes musical chords from input sound signals. I designed an algorithm and implemented it in Common Lisp Music/Realtime environment. The system runs on Silicon Graphics workstations and on the Intel PC platform. Experiments verified that the system succeeded in correctly identifying chords even on orchestral sounds.
Keywords: music, sound, chord, machine, recognition, realtime, numerical, AI

1 Introduction

Musical chord recognition is a process of identifying specific pitch simultaneities that combine to produce the functional harmonic palette of western music[1]. Traditionally, it is approached as a polyphonic transcription task to identify individual notes [1] [2] [3], which is then followed by a symbolic inference stage to determine the chord (Fig.1) [4]. This approach suffers from recognition errors at the first stage. The errors result from noises, and from overlapping harmonics of notes in the spectrum of the input audio signal. These inevitable errors makes the chord recognition task diffcult.

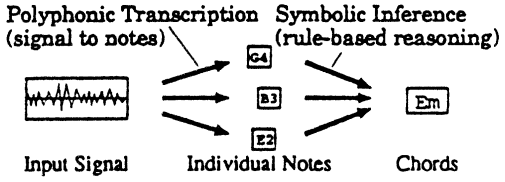


Figure 1: Traditional Approach

Recently, Leman proposed an alternative approach [5]. His "Simple Auditory Model" (SAM for short) avoids the unreliable note identification process by adopting numerical representation and manipulation of information throughout the process, to realize a robust recognition system. The key of SAM is the use of an intensity map of twelve semitone pitch classes. Derived from a spectrum through straightforward numerical processing, the map preserves more information and achieves more robustness than note names do.

Based on the framework of SAM, I designed a chord recognition algorithm, which is described in the following section.

2 Algorithm

The overview of my chord recognition algorithm is shown in Fig. 2.

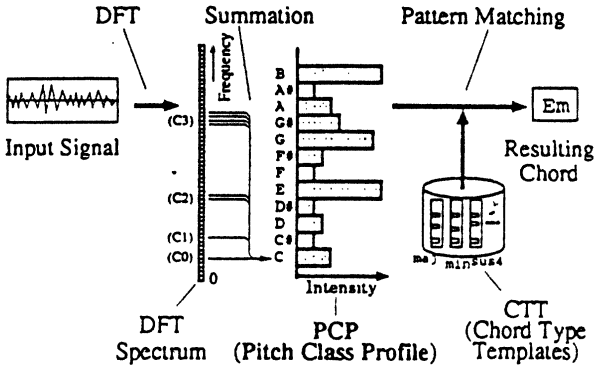


Figure 2: Chord Recognition Algorithm Overview

This algorithm first transforms an input sound to a Discrete Fourier Transform (DFT) spectrum, from which a Pitch Class Profile (PCP) is derived. Then it does pattern matching on the PCP to determine the chord type and root. Two major heuristics are introduced to improve the overall performance. Details follow.

2.1 DFT Spectrum

First, this algorithm transforms a fragment of the input sound stream to a DFT spectrum. Let f_s be the sampling frequency and $x(n)$ be a fragment

*Currently at YAMAHA Corp. Electronic Musical Instruments Division, 10-1 Nakazawa, Hamamatsu 430 Japan

[1]Human listeners perceive chords even in a single melodic line. This aspect is beyond the scope of this research.

of the input sound signal with N samples, where $n = 0, 1, \cdots N-1$ is the index. Then the DFT spectrum $X(k)$ is given for $k = 0, 1, ..., N-1$ as:

$$X(k) = \sum_{n=0}^{N-1} e^{-2\pi i k n/N} \cdot x(n)$$

Note that $X(0), \cdots, X(N/2-1)$, or the first half, gives the entire spectrum, as $x(n)$ is real.

2.2 Pitch Class Profile

From $X(k)$, this algorithm generates a Pitch Class Profile (PCP). A PCP is similar to the intensity map in Leman's SAM. It is a twelve dimension vector which represents the intensities of the twelve semitone pitch classes. For instance, the first PCP element shows how strong C is in total.

Let $PCP(p)$ be a PCP. It is defined for the index $p = 0, 1, \cdots, 11$ as:

$$PCP(p) = \sum_{l \text{ s.t. } M(l)=p} ||X(l)||^2$$

where $M(l)$ is a table which maps a spectrum bin index to the PCP index. $M(l)$ can initially be defined as:

$$M(l) = \begin{cases} -1 & \text{for } l = 0 \\ \text{round}(12 \log_2((f_s \cdot \frac{l}{N})/f_{ref})) \bmod 12 & \text{for } l = 1, 2, \cdots, N/2-1 \end{cases}$$

where f_{ref} is the reference frequency that falls into $PCP(0)$. The term $(f_s \cdot \frac{l}{N})$ represents the frequency of the spectrum bin $X(l)$. Note that $PCP(p)$ and $M(l)$ are actually further modified here for later use (see 2.4).

2.3 Pattern Matching

For each PCP obtained, this algorithm does pattern matching using the Chord Type Templates (CTT). The CTT's represent the knowledge of musical chords. $CTT_c(p)$ is set to 1 if the chord type c at root C has the pitch class p in it, and 0 otherwise. For instance, the CTT for "M7" is $(1,0,0,0,1,0,0,1,0,0,0,1)$.

I chose 27 chords (Table 1) in 2 groups, based on if a chord stays within an octave(S1) or not(S2).

Two matching methods are compared: the nearest neighbor and the weighted sum[6]. Each calculates scores for all the chord types, and for all the roots rotating the PCP, to choose the one with the best score.

Nearest Neighbor Method

The score to be minimized for a chord type c is given as:

$$\text{Score}_{nearest,c} = \sum_{p=0}^{11} (T_c(p) - PCP(p))^2$$

Table 1: Chord Types Chosen

group	chord types
S1	G_{single}, G_{dim}, G_{dim7}, $G_{m7^{-5}}$, G_{dimM7}, G_m, G_{m7}, G_{mM7}, G_7, G, G_{M7}, G_{aug}, G_{aug7}, G_{augM7}, G_{sus4}, $G_{sus4\ 7}$
S2	G_{m9}, $G_{7^{-9}}$, G_9, $G_{7^{+9}}$, F_m/G, F_{mM7}/G, F/G, F_{M7}/G, F_{dimM7}/G, F_{augM7}/G, G_{M9}

G is put as a placeholder for readability.

where $T_c(p)$ is a typical PCP tuned by hand, starting from the original $CTT_c(p)$.

Weighted Sum Method

The score to be maximized is given as:

$$\text{Score}_{weighted,c} = \sum_{p=0}^{11} W_c(p) \cdot PCP(p)$$

where $W_c(p)$ is the weight vector tuned by hand, again starting from $CTT_c(p)$. Through the tuning, different chord types get different weights, so as to reflect the number of the notes in the chord type and the probability of the emergence of the chord type. Also, negative weights are put for better separation among similar chord types.

2.4 Heuristics

To improve the overall performance of this algorithm, I introduce the following two heuristics.

1) **PCP Smoothing over time:** Assuming that a chord usually lasts for several PCP frames, this algorithm does smoothing over past PCP's by an averaging operation to reduce the noise.

2) **Chord Change Sensing:** A chord change is usually sudden, when the PCP vector changes its direction in the twelve dimension space. Therefore, we can sense chord changes by monitoring the direction of the PCP vector.

Other minor heuristics include:

- PCP preprocessing, including non-linear scaling
- Elimination of less important regions[7] in $M(l)$
- Use of an DFT window
- Silence detection to avoid unnecessary analysis
- Attack detection to avoid noisy PCP's

for which details are not mentioned further here.

3 Implementation

Common Lisp Music/Realtime extension (CLM/R for short) is used to implement this algorithm. CLM is a flexible set of Common-Lisp-based synthesis and signal processing tools developed at CCRMA, Stanford University [8]. CLM/R is a realtime extension of CLM. It compiles LISP codes to binary executables to realize realtime sound processing using graphical user interface.

The system runs on Silicon Graphics workstation O2 (R5000/180MHz, Irix 6.3) and on the Intel PC platform (Pentium II/333MHz, RedHat Linux 5.2). The system layer is shown in Fig. 3.

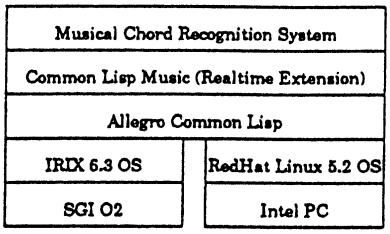


Figure 3: System Layer

Running on CLM/R, this system takes in audio signals from a line-in_or a sound file, inferes the chords, and displays the result in realtime, simultaneous with the audio playback.

4 Experiments and Results

The parameters I chose for the experiments were as follows: the sampling frequency f_s = 8.0kHz, monaural, the DFT length N = 2048 without an overlap, and the reference frequency f_{ref} = 65.4Hz (C2). With these settings, the system produced fine PCP's without mainlobe interference on pure tone inputs coming from a YAMAHA SY-77 music synthesizer.

4.1 Responses to Keyboard Sound

I evaluated the capability of the two matching methods using sounds from a YAMAHA PSR-520 electronic keyboard. The line-in was the input source. I chose three timbres (`Occalina`, `Strings1`, and `GrandPno`).

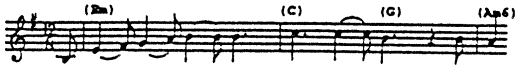
In the nearest neighbor method, initial CTT's worked well in segregating all the 27 chords played in any of the pure tone, `Occalina`, `Strings1`, and `GrandPno`.

In the weighted sum method, I tuned up the table by try-and-error to have the following accuracy: 100% (27/27) at the pure tone and `Occalina`, 93% (25/27) at `GrandPno` and `Strings1`.

4.2 Responses to Actual Music

I tested the system using musical sounds taken from CD recordings. For repeatability, sound files were used as the input source. The system made a series of guesses on the chord type and root, which I scored to give the accuracy by (# correct guesses)/(# all guesses). A chord guess was counted as correct when its all pitch classes were included in the actual chord.

In the following examples the system is demonstrated using a 50" audio excerpt from the opening theme of Smetana's "Moldau" (below).



Output Example

Fig. 4 is the first 8 seconds of resulting PCP's and chords by the weighted sum method. Black squares in the figure represent values of PCP elements by their size.

At 4.10, the system heard both the preceding `Em` and the starting `C`, to output a `CM7`.

The system's output after 7.43, `F#dim`, is actually `Am6`, or `F#dim7`.

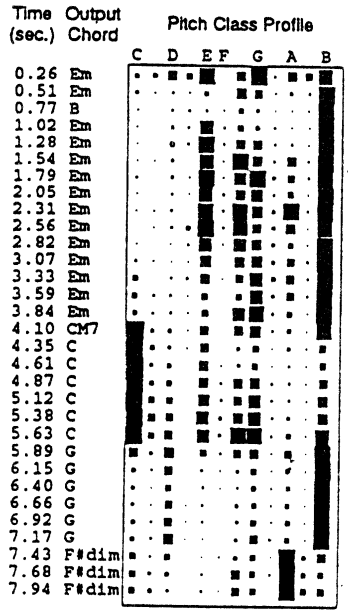


Figure 4: Output Example

This example resulted from the best heuristics parameters (described later) and a table which was fully tuned up for actual music. This was in order to estimate the potential capability of the framework. The final accuracy reached 94% (196/208).

Through the experiment I found that the tables tuned for keyboard input did not give meaningful results in actual music, and vice versa.

Effects of Heuristics

I tested the effects of the heuristics and tuned the heuristics parameters as follows.

Fig. 5 shows how the heuristics affects the PCPs for three cases. In each graph, the time goes from left to right.

a) Without the heuristics, PCP's are quite noisy.

b) "Smoothing over Time" reduces noise, but it oversmoothes, blurring chord change points.

c) "Chord Change Sensing" helps preserve the chord change points.

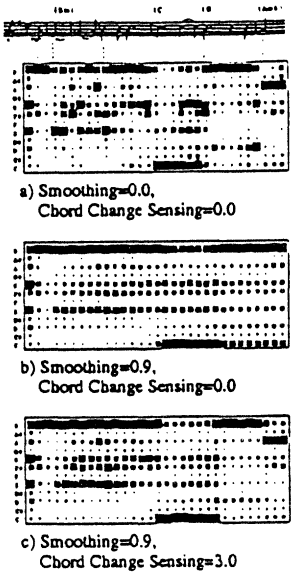


Figure. 5: Effects of Heuristics

Fig. 6 is an accuracy plot for various heuristics parameters.

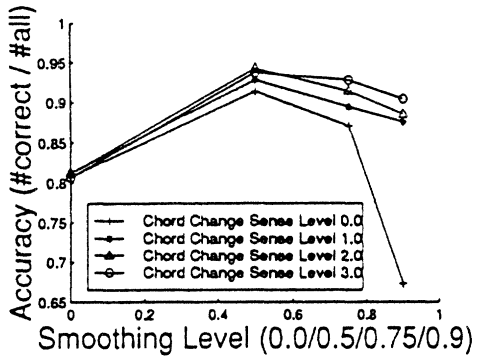


Figure 6: Accuracy v.s. Heuristics Parameters

Fig. 6 shows that the best result occurs at smoothing level = 0.5, chord change sensing = 2.0 which were actually used in the example in Fig. 4.

5 Discussion

The experiments show that this realtime system can successfully recognize musical chords at the signal level. It assigns correct chord names even in complex multi-timbral situations like orchestral music. Given the size of the system (1371 lines in LISP source code), this is a remarkable performance.

The PCP proves to be a reasonable internal representation. It conveys information sufficient to determine the chord type and root, though it discards the bass note information. The chords which cause "aliasing" in a PCP (S2 in Table 1) can be recognized correctly.

Both the weighted sum method and the nearest neighbor method can correctly segregate 93% through 100% of the chords played with the pure tones or with synthesized sounds. On actual music, the weighted sum method can recognize chords with a high accuracy of 94%, while the nearest neighbor method results in insufficient accuracy. Therefore, these classifiers are sufficient at least in the ideal input situation, but better classifiers may be necessary to process actual music input.

The two heuristics for edge-preserving smoothing improve the overall accuracy by about 10%.

Future work will include a better classifier, the table tuning technique, bass note tracking, use of musical context for the symbol level error correction.

6 Acknowledgement

This research was done under direction of Professor Jonathan Berger at CCRMA, Stanford University. I appreciate all of his supports and fruitful discussions. I would also express my gratitude to Mr. Masatada Wachi, the director of board in YAMAHA Corporation, who gave me the opportunity of doing this research.
U.S. Patent Pending: Serial No. 09-281526

References

[1] Chris Chafe, David Jaffe, Kyle Kashima, Bernard Mont-Reynaud, Julius Smith, *Techniques for Note Identification in Polyphonic Music*, STAN-M-29, CCRMA, Stanford University Oct. 1985.

[2] Chris Chafe and David Jaffe, *SOURCE SEPARATION AND NOTE IDENTIFICATION IN POLYPHONIC MUSIC*, STAN-M-34, CCRMA, Stanford University Apr. 1986.

[3] Kunio Kashino, Kazuhiro Nakadai, Tomoyoshi Kinoshita and Hidehiko Tanaka, *Application of Bayesian Probability Network to Music Scene Analysis*, Proceedings of IJCAI 1995, 1995.

[4] Yushi Aono, Haruhiro Katayose, and Seiji Inokuchi, *A Real-time Session Composer with Acoustic Polyphonic Instruments*, Proceedings of ICMC 1998, pp. 236-239, 1998.

[5] Marc. Leman, *Music and Schema Theory*, Springer-Verlag.

[6] Donald R. Tveter, *The Pattern Recognition: BASIS OF ARTIFICIAL INTELLIGENCE*, IEEE COMPUTER SOCIETY.

[7] Michele Biasutti, *Sharp low- and high-frequency limits on musical chord recognition*, Hearing Research, No.105, pp. 77-84, 1997.

[8] Bill Schottstaedt, *Machine Tongues XVII. CLM: Music V Meets Common Lisp.*, Computer Music Journal, Vol.18, No.2, pp.30-38, 1994.