CPU Bach:
An Automatic Chorale Harmonization System

Matt Hanlon mhanlon@fas

Tim Ledlie ledlie@fas

January 15, 2002

Abstract

We present an automated system for the harmonization of four-part chorales in the style of J.S. Bach. The task of harmonization is divided into two parts, namely (1) generation of a harmonic progression and (2) realization of that progression into melodic lines for the four voice parts. We model the composition of a chord progression with a Hidden Markov Model, and derive a realization of the harmony by solving a constraint satisfaction problem. Our implementation can successfully produce well-formed chorale harmonizations for a variety of melodies.

1 Introduction

The four-part chorale is a short hymn written for four voices (soprano, alto, tenor, bass). The several hundred chorales written by J.S. Bach are generally taken as the exemplars of the style and are widely used in music pedagogy. A common exercise for students of music is to produce a four-part chorale-style harmonization given a chorale melody, generally taken to be the soprano line. **CPUBach** is a system created to automate the task of creating a well-formed chorale given its soprano line.

Harmonization of a Bach-style chorale lends itself to an algorithmic approach. While a great deal of creativity would be needed to produce one of Bach's harmonizations, to be sure, the process is far from a compositional free-for-all. Rather, Bach's chorales adhere to certain compositional rules.

These rules are well studied and can be found in any textbook on elementary tonal harmony or counterpoint (see [1] for a good introduction to some rules). When we say that a harmonization is *well formed*, we mean that it conforms to these rules.

We can envision the production of a well-formed chorale as a search in both the melodic and harmonic domains, where constraints of varying strictness exist for both domains. **CPUBach** attempts to deal with these domains separately. First it uses a Harmonizer module to generate a valid harmony stochastically using a Hidden Markov Model (HMM) for chord progressions. Then the Realizer module solves a Constraint Satisfaction Problem to assign actual notes, and therefore melody, to the voice parts.

Although it is difficult to give a quantitative evaluation of the system's performance, we consider the output of the **CPUBach** system to be qualitatively quite good.

2 System Design

Any piece of tonal music can be thought of as having two aspects that contribute to its tonal quality: harmony and melody. Harmony refers to the sonority of all parts taken together at a given instant, and the way that sonority changes with time; it is often said to be the "vertical" aspect of a piece. Melody refers to the shape of individual lines, and is identified with "horizontal" motion of voices. A human composer is concerned with both of these aspects at once. For the purposes of our system, however, we find it useful to divide the compositional task into its harmonic and melodic subtasks, and solve these sequentially.

2.1 Harmonization and HMMs

A harmonization is characterized by a sequence (progression) of chords. One approach to harmonization, the approach we adopt here, is first to derive a chord sequence with which the melody is consonant and then to realize that harmony by filling in the other voice parts to be consistent with that progression. In general a single note to be harmonized can be a consonant member of nine chord types (it can be the root, third, or fifth in a major, minor, or diminished triad), although only about three will appear with any frequency in a given tonal context. Much of the creative aspect of writing a

harmonization comes in choosing a suitable progression from the combinatorially many possible consonant sequences. Of course the sequences are not arbitrary, and certain progressions sound more"musical" than others. This is especially true at the end of chorale phrases, the cadences, where only a few categories of sequences are permitted in 17th Century practice.

We suggest that the production of a chord progression can be reasonably modelled with a Hidden Markov Model where states correspond to n-tuples of chords (a "context") and the emissions correspond to successive chords. We can conceive of the HMM as starting at a cadence, which has special-case constraints, and working backwards, emitting chords at the front of the progression sequentially based on the n-chord context following it. We work backwards in this way to the beginning of the phrase, producing a complete progression.

The first major component of **CPUBach**, the Harmonizer, implements the HMM progression generation. (Note that we say the Harmonizer produces a *progression*, or the *harmony* of a chorale; we will continue to use the term *harmonization* to refer to a fully realized chorale with assigned note values.)

The Harmonizer takes as input a melodic line. For each note in the line, it first computes the several chords to which the note might belong. It then selects one sequence of chords by transitioning among them according to our HMM. As mentioned, we harmonize a chorale one phrase at a time, starting at the cadence and working backwards. We claim that there are special harmonic constraints at the cadence, but we are able to model these constraints simply by including probability information about single chords and chord pairs at the end of phrases in our database. This is equivalent to also including in our database n-tuples that include only one or two chords but are padded with special special cadential symbols. Thus we are able to deal with cadences and the rest of phrases in a unified way.

The Harmonizer includes a database of n-tuple information for chords. Associated with each chord in the database are several features that give the chord its character: its modality, inversion, and "type," a special category used to indicate whether the chord contains non-consonant tones like sevenths. Significantly, we do not store the root of chords in the database, but only the distance between adjacent chords; this means that in our representation, two transposed versions of the same progression will have the same probability. This feature provides two major benefits. First, it reduces the number of n-tuples that must be stored by a factor of 12. Second, we

eliminate the need to incorporate knowledge of the *key* of a harmonization explicitly. Rather, we expect that with a good probability model for the HMM (a problem we will address shortly) the key of a piece will arise naturally from the progression.

The nature of harmonic composition obviates two common concerns for HMMs. First, if we gather our transition probabilities empirically, we need not concern ourselves with transitions having zero occurrences in the corpus used to make our HMM. If a progression never occurs in actual chorales, then we never want it to occur in a chorale generated by **CPUBach**. Second, we never use the probability model of our HMM to determine the *most* likely chord progression for a melody, so no search through the progression space is required. Rather the intent is to be able to provide many different progressions for the same melody. Further, a rare progression is not necessarily a bad one. Indeed a rare progression might seem all the more creative or effective precisely because it is rare.

The question remains as to what HMM transition probabilities to use. We have entered many of the probabilities by hand, based on knowledge of common chord progressions. Our current database is sufficient to produce many common progressions. But entering the data is tedious. Our current implementation uses 3-tuples of chords; it is infeasible to enter probabilities for all possible 3-tuples in Bach's idiom, which we estimate to number in the tens of thousands. As a result the Harmonizer is somewhat limited in its compositional scope, and sometimes fails to produce a progression for long or atypical melodies. Ideally the probability model would come from Bach himself. We will discuss plans to gather transition probabilities empirically in the "Future Work" section.

2.2 Realization as a CSP

At the end of this randomized generation process, the Harmonizer has produced a sequence of chords, including information about the root, inversion modality and chord type. It now remains to *realize* that harmony into a fully-voiced chorale by choosing notes for each part at each chord. As stated, the harmonizations realized by Bach and his contemporaries conform to rather rigid rules of composition. For example, the "Parallel fifths rule" is one well-known rule of elementary harmony, which states that two parts a perfect fifth apart may not move the same interval in parallel. Rules like this one can be seen as constraints on the space of possible harmonizations, and thus we may

view the production of a harmony as a CSP where the variables correspond to the note values for each voice part at every chord [1].

The second major component of **CPUBach** is the Realizer, which solves the CSP. The realizer takes as input the harmonic progression produced by the Harmonizer, and realizes it by assigning actual note values to the four voice parts. We can consider the initial domain of every note in every voice part to be all the notes in the range of that voice part. Our realization must conform to the rather strict constraint that it follows the chord progression generated by the Harmonizer. Therefore we initially remove all pitch classes[1] not associated with the chord in our progression from the domain of each note. The Bass line receives special treatment, since the inversion information in the progression generated by the Harmonizer essentially constrains the bass to notes of a single pitch class.

Like the Harmonizer, the Realizer begins at a cadence and works backwards. It works voice part by voice part, starting with the base and working up. At each chord, the Realizer chooses a pitch from the domain of the voice part and assigns that pitch to the to be the note for the current voice part in he current chord. Along the way, we remove from the domains of the voice parts notes which would violate any of the compositional constraints. The constraint most essential to creating a convincing harmonization is the "Cover the chord" constraint. Chords are generally comprised of three pitch classes, and in most contexts a realization must include all three, that is, the chord must be "covered." Therefore we remove from the domain of notes any pitch whose assignment would make it impossible to cover the chord. We further remove pitches which violate any of several other doubling and movement constraints.

If we encounter a note with an empty domain, we know that there is no valid realization with the current note assignment, so we must backtrack. When all notes are successfully assigned, we return the valid realization we have created.

Ideally, the Realizer creates a harmonization that is not just any instance of a harmonic progression but which also has good voice leading. That is, the lines of the individual voice parts should sound melodious. A simple but effective method we implement is to prefer "near-by" note assignments. We order the domains for the CSP by distance to the adjacent note already assigned for the voice part. This system works, although we feel there is

[1]By *pitch class* we mean a set of notes with the same letter name like E or B♭.

some room for improvement and wish to investigate other ways of preferring good lines in the future.

2.3 Output and Implementation Details

We have included in our system a module to write output to a MIDI file for easy evaluation. **CPUBach** is implemented as a C++ program compiled under Microsoft Visual Studio. Since the MIDI writing portion of the system does not compile on UNIX, we have provided a version of **CPUBach** without the MIDI module It should be run as follows:

```
% ./cpuBach <phrase.in>
```

where `<phrase>` is the name of a text file containing a chorale phrase in the format as shown. **CPUBach** prints a text version of the harmonization to `stdout`.

We have also provided a binary for the Windows version of the system which takes an additional argument:

```
> cpuBach <phrase.in> <midifile.mid>
```

Here `<midifile.mid>` specifies the name of the file to write the MIDI version of the harmonization.

3 Results

We are generally very pleased with the results from the **CPUBach** system. The system can produce well-formed harmonizations to a wide variety of chorale phrases. Of course, the evaluation of any composition is a largely subjective matter. We include here several example runs of the system on four short chorale phrases. Also the same examples and the MIDI file output they produce are available at

`http://www.people.fas.harvard.edu/~ledlie/cs182proj/`

We ran the system several times on each input file, generating several different harmonizations for each soprano line. Results from the various runs on an input file `phrase`n`.in` can be found in the log file `phrase`n`.txt`, and the various MIDI files associated with that melody can be found in files of the form `cpuBachOut`n`-?.mid`.

For example consider the two outputs generated for the melody in `phrase1.in`. Both of the phrases sound as though they are in the same key, namely E♭ major. Notice that the progressions are different. The first ends on an E♭ major chord; this is an authentic cadence. The second harmonization ends on a c minor chord, a deceptive cadence. None of these concepts are explicitly represented in **CPUBach** but are captured by the probability model we use.

4 Future Work

We would like to expand the **CPUBach** project in several ways. The most important of these will be the empirical determination of the HMM probabilities. A nearly complete collection of Bach chorales in MIDI format can be found on the web[4]. It would be feasible (though certainly non-trivial) to parse these files and use them as a corpus to generate the transition matrix for our HMM.

Another potential area for more work is the generation of melodic lines. Currently, given a progression, the system will always produce the same realization, although usually many will be possible. Although we choose notes with a preference for good melodic shape, it is not the case that this will result in the best melodic lines for all voices. Perhaps a better system would be to generate the melodic line by choosing in some random fashion from the domain (perhaps including a melodic weighting). Finally, we may wish to implement a melodic heuristic for realizations. We could then generate several well-formed realizations and choose among them.

5 Conclusion

We consider the algorithms behind **CPUBach** to be robust and flexible, and are very pleased with our current results. The major bottleneck in producing elaborate harmonizations is the construction of the HMM database, and we look forward to implementing a system to derive transition probabilities from the Bach MIDI corpus.

6 Bibliography

[**1**] Pachet, F. and Roy, P. *Musical Harmonization with Constraints: A survey.* Constraints, 6(1):7-19 2001.

[**2**] Lewin, David. *An Interesting Global Rule for Species Counterpoint.*

[**3**] Takashi, K. et al., *Hidden Markov Model Applied to Automatic Harmonization of Given Melodies.* 99-MUS-34, pp. 56-66, Feb 2000.

[**4**] Yaskawa, T. *371 Bach Chorales Sequenced in MIDI Format.* `http://www.midiworld.com/c`