PITCH SPELLING ALGORITHMS

David Meredith
City University, London, UK

ABSTRACT

In this paper I introduce a new algorithm called *ps13* that reliably computes the correct pitch names (e.g., C♯4, B♭5 etc.) of the notes in a passage of tonal music, when given only the onset-time and MIDI note number of each note in the passage. *ps13* correctly predicts the pitch names of 99.81% of the notes in a test corpus containing 41544 notes and consisting of all the pieces in the first book of J. S. Bach's *Das Wohltemperirte Klavier* (BWV 846--869). Three previous algorithms (those of Cambouropoulos (1996, 1998, 2002), Longuet-Higgins (1987) and Temperley (2001)) were run on the same corpus of 41544 notes. On this corpus, Cambouropoulos's algorithm made 2599 mistakes, Longuet-Higgins's algorithm made 265 mistakes and Temperley's algorithm made 122 mistakes. As *ps13* made only 81 mistakes on the same corpus, this suggests that *ps13* may be more robust than previous algorithms that attempt to perform the same task.

1. INTRODUCTION

In this paper I focus on the problem of constructing a robust *pitch spelling algorithm*—that is, an algorithm that reliably computes the correct pitch names (e.g., C♯4, B♭5 etc.) of the notes in a passage of tonal music, when given only the onset-time, MIDI note number and possibly the duration of each note in the passage.

There are good practical and scientific reasons for attempting to develop a robust pitch spelling algorithm. First, until such an algorithm is devised, it will be impossible to construct a robust *MIDI-to-notation transcription algorithm*—that is, an algorithm that reliably computes a correct score of a passage when given only a MIDI file of the passage as input. Commercial music notation programs (e.g., Sibelius (www.sibelius.com), Coda Finale (www.codamusic.com) and Nightingale (www.ngale.com)) typically use MIDI-to-notation transcription algorithms to allow the user to generate a notated score from a MIDI file encoding a performance of the passage to be notated. However, the MIDI-to-notation transcription algorithms currently used in commercial music notation programs are crude and unreliable. Also, existing audio transcription systems generate not notated scores but MIDI-like representations as output (see, for example, Walmsley, 2000). So if one wishes to produce a notated score from a digital audio recording, one typically needs a MIDI-to-notation transcription algorithm (incorporating a pitch spelling algorithm) in addition to an audio transcription system.

Knowing the letter-names of the pitch events in a passage is also indispensible in music information retrieval and musical pattern discovery. For example, the occurrence of a motive on a different degree of a scale (e.g., C-D-E-C restated as E-F-G-E) might be perceptually significant even if the corresponding chromatic intervals in the patterns differ. Such matches can be found using fast, exact-matching algorithms if the pitch names of the notes are encoded, but exact-matching algorithms cannot be used to find such matches if the pitches are represented using just MIDI note numbers. If a robust pitch spelling algorithm existed, it could be used to compute the pitch names of the notes in the tens of thousands of MIDI files of works that are freely available online, allowing these files to be searched more effectively by a music information retrieval (MIR) system.

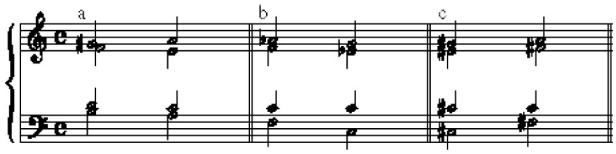

Figure 1: Examples of notes with identical MIDI note numbers being spelt differently in different tonal contexts (from Piston 1978, p. 8).

In the vast majority of cases, the correct pitch name for a note in a passage of tonal music can be determined by considering the rôles that the note is perceived to play in the perceived harmonic structure and voice-leading structure of the passage. For example, when played in isolation in an equal-tempered tuning system, the first soprano note in Figure 1a would sound the same as the first soprano note in Figure 1b. However, in Figure 1a, this note is spelt as a G♯4 because it is perceived to function as a leading note in A minor; whereas in Figure 1b, the first soprano note is spelt as an A♭4 because it functions as a submediant in C minor. Similarly, the first alto note in Figure 1b would sound the same as the first alto note in Figure 1c in an equal-tempered tuning system. However, in Figure 1b the first alto note is spelt as an F♮4 because it functions in this context as a subdominant in C minor; whereas, in Figure 1c, the first alto note functions as a leading note in F♯ minor so it is spelt as an E♯4.

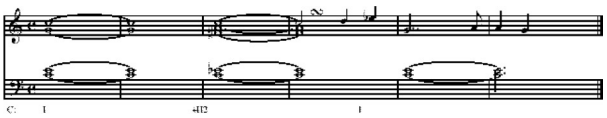

Figure 2: Should the E♭s be spelt as D♯s? (From Piston 1978, p. 390).

Nevertheless, it is not always easy to determine the correct pitch name of a note by considering the harmonic structure and voice-leading structure of its context. For example, as Piston (1978, p.390) observes, the tenor E♭4 in the third and fourth bars of Figure 2 should be spelt as a D♯4 if one perceives the harmonic progression here to be $^{+}\mathrm{II}^{2}$—I as shown. But spelling the soprano E♭5 in the fourth bar as D♯5 would result in a strange melodic line.

ISBN 3-931852-67-9
ISSN 1617-6847

Such cases where it is difficult to determine the correct pitch name of a note in a tonal work are relatively rare—particularly in Western tonal music of the so-called 'common practice' period (roughly the 18th and 19th centuries). In the vast majority of cases, those who study and perform Western tonal music agree about how a note should be spelt in a given tonal context. This poses an interesting problem for cognitive science, namely: what are the cognitive processes involved when a musically trained individual determines the correct pitch name of a note in a passage of tonal music?

A good way of trying to answer this question is to attempt to construct a robust pitch spelling algorithm that operates in a way that is consistent with what is known about the neurology, physiology, psychophysics and psychology of expert music perception and cognition. Such a pitch spelling algorithm could serve the valuable scientific purpose of furthering our understanding of the cognitive processes underlying an interesting intellectual ability exhibited by many individuals who study and perform Western tonal music.

The vast majority of notes in authoritative published editions of scores of common practice tonal works are generally agreed to be spelt correctly by those who understand Western staff notation. Therefore a pitch spelling algorithm or computational model of pitch spelling can be evaluated objectively by running it on tonal works and comparing the pitch names it predicts with those of the corresponding notes in authoritative published editions of scores of the works. However, this can only be done accurately and quickly if one has access to encodings of these authoritative scores in the form of computer files that can be compared automatically with the pitch spelling algorithm's output.

2. A COMPARISON OF THREE PITCH SPELLING ALGORITHMS

In this section, I compare the performance of three pitch spelling algorithms on a single test corpus. The algorithms compared are those of Cambouropoulos (1996, 1998, 2002), Longuet-Higgins (1987) and Temperley (2001).

2.1. The Test Corpus

The test corpus used in the comparison consists of representations of the scores of all 24 Preludes and all 24 Fugues in the first book of J. S. Bach's *Das Wohltemperirte Klavier* (BWV 846–869) encoded in what I call *OPND* format.[1] Each *OPND* representation is a set of triples, (*t*,*n*,*d*), each triple giving the onset time, *t*, the pitch name, *p* and the duration, *d* of a single note (or sequence of tied notes) in the score. The onset time and duration of each note are expressed as integer multiples of the largest common divisor of all the notated onset times and note durations in the score. Temperley's algorithm cannot deal with situations in which two or more notes with the same pitch begin at the same time. Thus, wherever two or more notes with the same pitch *p* began simultaneously at time *t*, all notes with pitch *p* and onset time *t* were removed from the *OPND* file except the one with the longest duration. This resulted in a test corpus containing 41544 notes.[2]

The "piano-roll" or MIDI-like input representations accepted by the algorithms compared in this study were derived automatically from the *OPND* representations of the pieces in the test corpus.

2.2. Longuet-Higgins's algorithm

Pitch spelling is one of the tasks performed by Longuet-Higgins's (1987) *music.p* program. Longuet-Higgins has published the full POP-11 source code of this program (Longuet-Higgins, 1987, pp. 120–126; 1993, pp. 486–492). As I am not familiar with POP-11, I translated just the pitch spelling portion of *music.p* into Lisp. This was possible because the pitch spelling portion of *music.p* operates independently of the rhythmic part.[3]

music.p accepts input in the form of a list of triples, (*p*,*t*,*t'*) each triple giving the "keyboard position" *p* together with the onset time *t* and the offset time *t'* in centiseconds of each note. The keyboard position *p* is an integer indicating the key that would have to be pressed on a normal piano keyboard in order to perform the note, with C♮3 mapping onto 0, C♯3 and D♭3 mapping onto 1, C♮4 mapping onto 12 and so on (i.e., *p* = MIDI NOTE NUMBER - 48).

The algorithm then computes a value of "sharpness" *q* for each note in the input (Longuet-Higgins, 1987, p. 111). The sharpness of a pitch name indicates the position of the pitch name on the line of fifths (Temperley, 2001, p. 117) and is therefore essentially the same as Temperley's (2001, p. 118) concept of "tonal pitch class".

In Longuet-Higgins's algorithm, if q_1 and q_2 are the sharpnesses of two notes then the interval between the notes is defined to have "degree" $\delta q = q_1 - q_2$. If $|\delta q| < 6$, the interval is defined to be "diatonic"; if $|\delta q| > 6$, it is "chromatic"; and if $|\delta q| = 6$, it is "diabolic" (Longuet-Higgins, 1987, p. 112). Longuet-Higgins's algorithm attempts to spell notes so that the degree between each note and the tonic at the point at which the note occurs is not chromatic (Longuet-Higgins, 1987, p. 113). The algorithm also incorporates various rules for dealing with chromatic passages (Longuet-Higgins, 1987, pp. 113–4).

When my implementation of Longuet-Higgins's algorithm was run on the test corpus described above, it made only 265 errors—that is, it predicted the correct pitch name for 99.36% of the notes.

2.3. Cambouropoulos's algorithm

Unlike Longuet-Higgins, Cambouropoulos has not published an implementation of his pitch spelling algorithm, nor was he able

[1] *OPND* stands for "onset, pitch-name, duration".

[2] The test corpus is available online at http://www.titanmusic.com/data.html .

[3] This Lisp implementation of the pitch spelling portion of *music.p* is available online at http://www.titanmusic.com/software.html.

ISBN 3-931852-67-9
ISSN 1617-6847

to provide me with his own implementation when I requested it. I therefore implemented my own version of his method, based on his published descriptions of it (Cambouropoulos 1996, 1998, 2002).[4]

Cambouropoulos's method involves first converting the input representation into a sequence of pitch classes in which the pitch classes are in the order in which they occur in the music (the pitch classes of notes that occur simultaneously being ordered arbitrarily). Having derived an ordered set of pitch classes from the input, Cambouropoulos's algorithm then processes the music a window at a time, each window containing a fixed number of notes (set to a value between 9 and 15). Each window is positioned so that the first third of the window overlaps the last third of the previous window. Cambouropoulos allows 'white note' pitch classes (i.e., 0, 2, 4, 5, 7, 9 and 11) to be spelt in three different ways (e.g., pitch class 0 can be spelt as B♯ C♮ or D𝄫) and 'black note' pitch classes to be spelt in two different ways (e.g., pitch class 6 can be spelt as F♯ or G♭). Given these restricted sets of possible pitch names for each pitch class, the algorithm computes all possible spellings for each window. A penalty score is then computed for each of these possible window spellings. The penalty score for a given window spelling is found by computing a penalty value for the interval between each pair of notes in the window and summing these penalty values. A given interval in a particular window spelling is penalised more heavily if it is an interval that occurs less frequently in the major and minor scales. An interval is also penalised if either of the pitch names forming the interval is a double-sharp or a double-flat. For each window, the algorithm chooses the spelling that has the lowest penalty score.

When I ran my implementation of Cambouropoulos's method on the test corpus, it made 2599 mistakes—that is, it predicted the correct pitch name for only 93.74% of the notes. When Cambouropoulos ran his own implementation of his method on the test corpus, he found that it made even more errors than my implementation. This may be due to the fact that my implementation generates three alternative transpositions of the computed spelling and chooses the one that results in the least number of errors.

2.4. Temperley's algorithm

Temperley's (2001) pitch spelling algorithm is implemented in his *harmony* program which forms one component of his and Sleator's *Melisma* system.[5] The *harmony* program accepts input in the form of a "note-list" (Temperley 2001, pp. 9–12) giving the MIDI note number of each note together with its onset time and duration in milliseconds. The *harmony* program also requires a specification of the metrical structure of the input passage which can be generated by first running the note-list through Temperley and Sleator's *meter* program (another component of the *Melisma* system).

[4] The Lisp code for this implementation of Cambouropoulos's pitch spelling algorithm is available online at http://www.titanmusic.com/software.html.

[5] The complete *Melisma* system together with documentation is available online at http://www.link.cs.cmu.edu/music-analysis

Temperley's (2001, pp. 115–136) pitch spelling algorithm searches for the spelling that best satisfies three "preference rules", the first of which stipulates that the algorithm should "prefer to label nearby events so that they are close together on the line of fifths" (Temperley 2001, p. 125). This rule is similar to the basic principle underlying Longuet-Higgins's algorithm (see above). The second rule states that if two tones are separated by a semitone and the first tone is distant from the key centre, then the interval between them should preferably be spelt as a diatonic semitone rather than a chromatic one (Temperley 2001, p. 129). The third of Temperley's preference rules steers the algorithm towards spelling the notes so that a "good harmonic representation" results (Temperley 2001, p. 131), a "good harmonic representation" being one that allows Temperley's *harmony* program to generate a correct harmonic analysis of the passage.

When Temperley's algorithm was run on the test corpus, it made only 122 mistakes—that is, it predicted the correct pitch name for 99.71% of the notes.

3. *PS13*: A NEW PITCH SPELLING ALGORITHM[6]

In this section I describe a new pitch spelling algorithm, *ps13*, which performs better than Temperley's on the test corpus described above. *ps13* reliably computes the pitch names of the notes in a passage of tonal music when given only the onset time and MIDI note number of each note in the passage. Meredith (2003) provides a detailed description of the algorithm. *ps13* is best understood to be in two parts, Part I and Part II. Part I consists of the following steps:

1. computing for each pitch class $0 \leq c \leq 11$ and each note n in the input, a pitch letter name $S(c,n) \in \{A,B,C,D,E,F,G\}$, calculated on the assumption that c is the tonic at the point in the piece where n occurs;

2. computing for each note n in the input and each pitch class $0 \leq c \leq 11$ a value $CNT(c,n)$ giving the number of times that c occurs within a context surrounding n that includes n, some specified number K_{pre} of notes immediately preceding n and some specified number K_{post} of notes immediately following n;

3. computing for each note n and each letter name l, the set of pitch classes $C(n,l) = \{c \mid S(c,n) = l\}$;

4. computing $N(l,n)=\sum_{c \in C(n,l)} CNT(c,n)$ for each note n and each pitch letter name l;

5. computing for each note n, the letter name l for which $N(l,n)$ is a maximum.

[6] Patent pending (Meredith, 2003).

ISBN 3-931852-67-9
ISSN 1617-6847

Part II of the algorithm corrects those instances in the output of Part I where a neighbour note or passing note is erroneously predicted to have the same letter name as either the note preceding it or the note following it.

In order to determine the values of K_{pre} and K_{post} that give the best results, *ps13* was run on the test corpus 2500 times, each time using a different pair of values (K_{pre},K_{post}) chosen from the set $\{(K_{pre},K_{post}) \mid 1 \leq K_{pre}, K_{post} \leq 50\}$. For each pair of values (K_{pre},K_{post}), the number of errors made by *ps13* on the test corpus was recorded.

ps13 made fewer than 122 mistakes (i.e., performed better than Temperley's algorithm) on the test corpus for 2004 of the 2500 (K_{pre},K_{post}) pairs tested (i.e., for 80.160% of the (K_{pre},K_{post}) pairs). *ps13* performed best on the test corpus when K_{pre} was set to 33 and K_{post} was set to either 23 or 25. With these parameter values, *ps13* made only 81 errors on the test corpus—that is, it correctly predicted the pitch names of 99.805% of the notes in the test corpus. The mean number of errors made by *ps13* over all 2500 (K_{pre},K_{post}) pairs was 109.082 (i.e., 99.737% of the notes were correctly spelt on average over all 2500 (K_{pre},K_{post}) pairs). Even this average value is better than the result obtained by Temperley's algorithm for this test corpus. The worst result was obtained when both K_{pre} and K_{post} were set to 1. In this case, *ps13* made 1117 errors (97.311% correct). However, provided K_{pre} is greater than about 14 and K_{post} is greater than about 21, *ps13* predicts the correct pitch name for over 99.75% of the notes in the test corpus.

4. CONCLUSIONS AND FURTHER WORK

A new pitch spelling algorithm *ps13* was shown to perform better than three previous pitch spelling algorithms on a test corpus containing 41544 notes and consisting of all the pieces in the first book of J. S. Bach's *Das Wohltemperirte Klavier*. However, more algorithms need to be compared with *ps13* and all the algorithms need to be run on a variety of stylistically different corpora before it can be claimed that the new algorithm is the most robust pitch spelling algorithm currently available. Further testing will probably show that the algorithm that works best on one style may not work best on another style.

5. REFERENCES

1. Cambouropoulos, E. (1996). A general pitch interval representation: Theory and applications. *Journal of New Music Research*, 25, 231–251.

2. Cambouropoulos, E. (1998). Towards a General Computational Theory of Musical Structure. Unpublished PhD dissertation, University of Edinburgh.

3. Cambouropoulos, E. (2002). Pitch Spelling: A Computational Model. *Music Perception,* To appear.

4. Longuet-Higgins, H. C. (1987). The Perception of Melodies. In Longuet-Higgins, H. C. (ed). *Mental Processes: Studies in Cognitive Science*. British Psychological Society/MIT Press, London. pp.105–129.

5. Meredith, D. (2003). Method of computing the pitch names of notes in MIDI-like music representations. Patent filing submitted to UK Patent Office on 11 April 2003. Available online at http://www.titanmusic.com/papers.html.

6. Piston, W. (1978). *Harmony*. Gollancz, London.

7. Temperley, D. (2001). *The Cognition of Basic Musical Structures*. MIT Press, Cambridge, MA.

8. Walmsley, P. J. (2000). Signal Separation of Musical Instruments. PhD dissertation, Engineering Department, University of Cambridge.

ISBN 3-931852-67-9
ISSN 1617-6847