David Meredith
Centre for Cognition, Computation, and Culture
Department of Computing
Goldsmiths College, University of London
New Cross, London SE14 6NW UK
dave@titanmusic.com

Optimizing Chew and Chen's Pitch-Spelling Algorithm

Pitch-spelling algorithms attempt to compute the correct pitch names (e.g., C#4, B♭5) of the notes in a passage of tonal music, when given only the onset time, MIDI note number, and possibly the duration and voice of each note. This article reports on a study in which Chew and Chen's (2003a, 2003b, 2005) pitch-spelling algorithm was re-implemented and then optimized by running it with a range of different parameter value combinations on a test corpus containing 195,972 notes and consisting of 216 movements from works by eight Baroque and Classical composers. The results of this evaluation cast doubt upon some of the claims made by Chew and Chen that were based on results obtained by running their algorithm on a much smaller test corpus containing only 4,462 notes and consisting of just two movements from sonatas by Beethoven and You-Di Huang's *Song of Ali-Shan.* The results presented here suggest that Chew and Chen's algorithm could be simplified in various ways without compromising its performance.

Background

There are good practical and scientific reasons for attempting to develop a reliable pitch-spelling algorithm. For example, such an algorithm must be incorporated into any system for transcribing music from audio or MIDI to staff notation. Furthermore, encoding the pitch names of the notes in a collection of MIDI files can make certain music information retrieval tasks more effective (Meredith 2006). Developing a reliable pitch-spelling algorithm can also further our understanding of the cognitive mechanisms that underlie the perception and cognition of tonal music. For example, Temperley (2001, p. 122) claims that "recognizing spelling distinctions" (i.e., identifying the pitch names of the notes in a piece) is "of direct experiential importance, for pitches, chords, and keys" and "provides useful input in harmonic and key analysis."

Pitch-spelling algorithms have been developed by a number of researchers aside from Chew and Chen, including Longuet-Higgins (1976, 1987a, 1993), Cambouropoulos (1996, 1998, 2001, 2003), Temperley (2001), Honingh (2006), Stoddard et al. (2004), and Meredith (2003, 2005, 2006, 2007). My dissertation (Meredith 2007) provides a detailed analysis and evaluation of my *ps13* algorithm together with the algorithms proposed by Longuet-Higgins, Cambouropoulos, Chew and Chen, and Temperley. To set the current discussion in context, brief descriptions of Temperley and Sleator's *Melisma* system and my *ps13* algorithm will now be given.

Using Temperley and Sleator's *Melisma* System for Pitch Spelling

Temperley's (2001) theory of music cognition consists of preference rule systems for six aspects of musical structure: meter, phrasing, counterpoint, harmony, key, and pitch spelling. Most of this theory has been implemented by Daniel Sleator in a suite of computer programs called *Melisma* (available on-line at www.link.cs.cmu.edu/music-analysis). These programs take "note list" representations as input in which the pitch of each note (or sequence of tied notes) is represented by its MIDI note number, and its onset-time and offset-time are given in milliseconds. In Temperley's theory, the tonal pitch class (TPC) of a note is an integer that indicates the position of the pitch name class of the note on the line of fifths.

Temperley's theory of pitch spelling consists of three "tonal-pitch-class preference rules" (TPRs). He claims that TPR 1, which states that notes should be spelled so that they are as close together as possible on the line of fifths, is "the most important" TPR and that "in many cases, this rule is sufficient to ensure the correct spelling of passages" (Temperley 2001, p. 125). TPR 2 is designed to ac-

Computer Music Journal, 31:2, pp. 54–72, Summer 2007
© 2007 Massachusetts Institute of Technology.

count for the way that notes are typically spelled in chromatic scale segments. TPR 3 states that the system should "prefer TPC representations which result in good harmonic representations" (Temperley 2001, p. 131). He formally defines the concept of a "good harmonic representation" in the first rule in his theory of harmony, HPR 1 (Temperley 2001, p. 149), which states that, in choosing the roots for chords, certain specified TPC-root relationships should be preferred over others. His theories of pitch spelling and harmony are therefore interdependent, and this is reflected in the fact that they are both implemented in the `harmony` program in *Melisma.*

The complexity of Temperley's pitch-spelling algorithm is increased still further by the fact that his theory of harmony depends on his theory of metrical structure. For example, the second harmonic preference rule states that the system should "prefer chord spans that start on strong beats of the meter" (Temperley 2001, pp. 151, 359). The `harmony` program therefore requires as input both a "note list" and a representation of the metrical structure of the passage in the form of a "beat list" of the type generated by the *Melisma* `meter` program.

Consequently, if one wishes to use Temperley's theory to determine the pitch names of the notes in a passage, one must first use the `meter` program to generate a beat list from a note-list and then use the `harmony` program to compute the pitch names of the notes in the passage from the note list and the beat list. In an attempt to take harmonic rhythm into account when computing metrical structure, Temperley and Sleator also experimented with a "two-pass" method (Temperley 2001) in which the `meter` and `harmony` programs are both run twice, once in a special "prechord" mode and then again in "normal" mode.

Meredith's *ps13* algorithm

In other publications (Meredith 2006, 2007), I have described a two-stage algorithm called *ps13.* In this algorithm, the pitch name of a note is assumed to depend on the local key and voice leading. In the first stage of *ps13,* the pitch name implied for a note by a tonic is assumed to be the one that lies closest to that tonic on the line of fifths. The strength with which a pitch name is implied for a note is assumed to be proportional to the sum of the frequencies of occurrence, within a context around the note, of the tonics that imply that pitch name. In the second stage of *ps13,* certain neighbor-note and passing-note errors in the output of the first stage are corrected. However, I found that omitting the second stage of the algorithm improved its performance on both "clean" test data and data containing temporal perturbations (Meredith 2006, 2007).

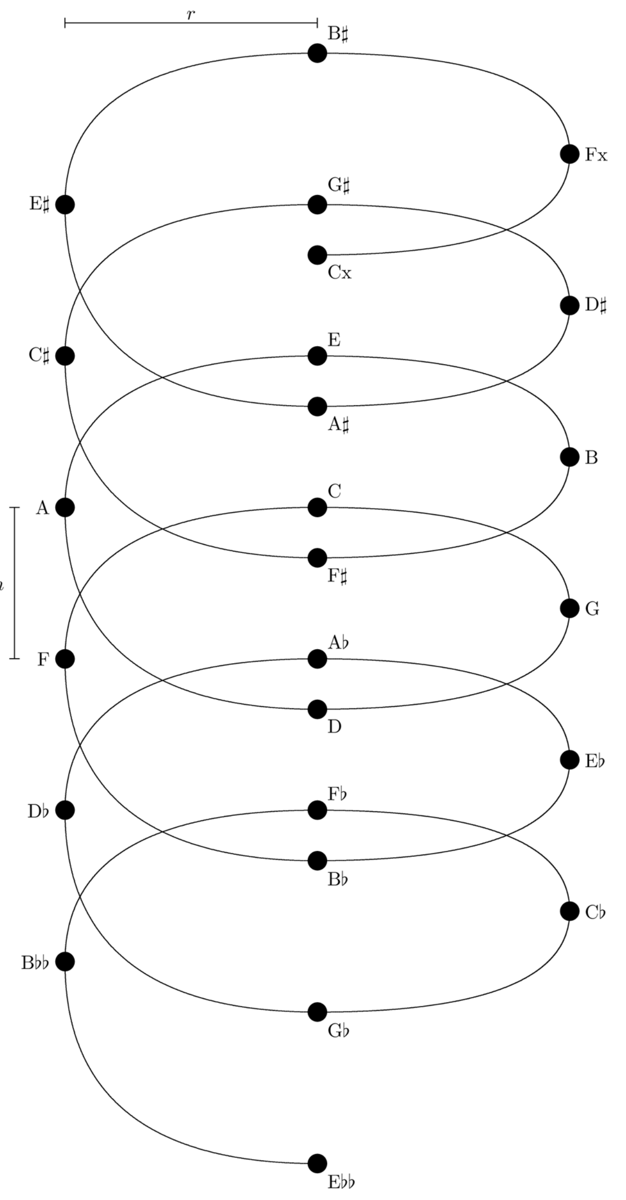
Chew and Chen's Pitch-Spelling Algorithm

Chew and Chen (2003a, 2003b, 2005) describe several variants of a real-time pitch spelling algorithm based on Chew's (2000) "Spiral Array Model," which is a geometric model of tonal pitch relations. In the spiral array, pitch classes are arranged on a helix so that adjacent pitch classes along this helix are a perfect fifth apart and adjacent pitch classes along the length of the cylinder in which the helix is embedded are a major third apart (see Figure 1). As Chew and Chen (2005, p. 67) point out, the spiral array is "a spiral configuration of the line of fifths." That is, the spiral array can be constructed by "coiling up" the line of fifths. Like Longuet-Higgins's two-dimensional "map of notes" (1987b) and his three-dimensional "tonal space" (Longuet-Higgins 1987a), Chew's spiral array is a geometric representation of the idea that, in tonal music, pitches a major third and a perfect fifth apart are perceived to be particularly closely related. Chew and Chen (2005, p. 66) claim that the "depth added by going from one to three dimensions [i.e., from the line of fifths to the spiral array] allows the modeling of more complex hierarchical relations."

Chew and Chen (2005) define the *index* of a note to be an integer that indicates the position of the pitch class of the note on the line of fifths. The index of C-natural is defined to be 0, and the index increases by 1 for each step in the "sharp" direction; thus, the index of G is 1, that of F is –1, and so on.

The *aspect ratio* of the spiral array is a measure of how "tightly coiled" it is. The aspect ratio is de-

Figure 1. Part of Chew's (2000) spiral-array model representing the perceived tonal relations between pitch name classes. Here, r *is the radius of the cylinder in which the helix is embedded, and* h *is the distance parallel to the axis of this cylinder between two points one step apart along the helix. The ratio* r/h *is the aspect ratio of the spiral array (Chew and Chen 2005).*



fined to be r/h, where r is the radius of the cylinder in which the pitch-class helix is embedded, and h is the distance parallel to the axis of the helix corresponding to one step along the spiral array (i.e., two pitch name classes a perfect fifth apart) (see Figure 1). The value of the aspect ratio determines the ratio of the distances in the spiral array between notes a major third apart and notes a perfect fifth apart.

Chew and Chen set r/h to $\sqrt{(15/2)}$ to ensure that "the major thirds and [perfect] fifths are equidistant" (Chew 2004).

Let's suppose that S is a set of notes in a piece of tonal music, that $\mathbf{p}(n)$ denotes the vector representing the three-dimensional position in the spiral array of the pitch class of the note n, and that $d(n)$ is the duration of note n. Chew and Chen (2005) define the *center of effect* (CE) of S, denoted by $CE(S)$, to be

$$CE(S) = \frac{\sum_{n \in s} d(n) \cdot \mathbf{p}(n)}{\sum_{n \in s} d(n)}. \tag{1}$$

That is, the CE of a set of notes is the weighted centroid of the position vectors of the pitch classes of the notes in the spiral array, each note being weighted by its duration. Given r and h, the spiral-array position vector of a note, $\mathbf{p}(k)$, can be computed from its index k using the following formula (Chew and Chen 2005):

$$\mathbf{p}(k) = \left\langle r\sin(k\pi/2), r\cos(k\pi/2), kh \right\rangle. \tag{2}$$

In Chew and Chen's (2005) algorithm, the CE for a set of notes is used to represent (or "act as a proxy for") the key. The basic principle underlying the algorithm is that each note should be spelled so that it is as close as possible in the spiral array to the CE of the notes that precede it. Considering only the notes that precede the one to be spelled allows the algorithm to process music in real time. This principle is almost identical to that expressed in Temperley's (2001) first "tonal-pitch-class preference rule" (TPR 1), which states that nearby events in a passage of tonal music should be assigned pitch names "so that they are close together on the line of fifths." The two theories differ in that Chew and Chen use the spiral array, whereas Temperley uses the line of fifths. Also, the windowing system used in Temperley and Sleator's implementation of Temperley's theory is quite different from that used by Chew and Chen.

In Chew and Chen's (2005) algorithm, it is assumed that the input data gives the MIDI note number, together with the onset and duration in milliseconds of each note. The data is then divided into "equal time slices" called *chunks,* and the algorithm spells the notes one chunk at a time. Let $W_{\text{sound}}(i, j)$ denote the set of notes that are sounding in a window consisting of chunks i to j, and let $W_{\text{start}}(i, j)$ denote the set of notes that start in a window consisting of chunks i to j. Chew (2004) has confirmed that, in her and Chen's implementation, the CE for a window is calculated by considering the notes that sound in the window, not the notes that start in it. Also, in this implementation, each note is weighted not by its total duration but by the duration for which it sounds within the window over which the CE is being calculated.

Suppose that the algorithm is about to spell the notes in chunk j. According to Chew and Chen (2005), the steps shown in Appendix A of the present article are executed. [Editor's note: The algorithm is essential to the discussion; it was moved to an appendix for technical reasons.] Steps 1 and 2 constitute what they call "Phase I" of the algorithm and steps 3 to 5 make up "Phase II." The procedure for spelling the notes in the first chunk of the music to be processed is different from that described in Appendix A. The notes in the first chunk are first spelled so that their indices are as close as possible to some specified *initial index* which Chew and Chen (2005) set to 2 (corresponding to D-natural), on the grounds that this biases "the notation towards fewer sharps and flats." Note that this strategy for initializing the algorithm is essentially identical to that used by Temperley (2001). Temperley initializes the "center of gravity" (COG) on the line of fifths to 4 (i.e., D-natural), because "the COG is generally about two steps in the sharp direction from the tonic." It is therefore not surprising that, when Chew and Chen (2003a, 2003b) set the initial index to 0 (corresponding to C), they found that "there was a bias toward the flatted keys" (Chew 2004). Having assigned pitch names to the notes in the first chunk, a CE for this first chunk is then computed, and the notes in the first chunk are respelled so that they are as close as possible on the spiral array to this newly computed CE.

Versions of the Algorithm Described in Chew and Chen's Publications

As Chew and Chen (2005) point out, most of the variants of their algorithm described in their publications are either specific instances or classes of instances of the algorithm described in Appendix A, in which some of the parameters w_s, w_r, f, r/h, and the initial index are set to particular values. Table 1 gives the combinations of parameter values in the algorithm described in Appendix A that correspond to the versions of the algorithm described by Chew and Chen in their publications.

The first row in Table 1 gives the parameter values for the "two-phase boot-strapping algorithm" (Chew 2004; Chew and Chen 2005). This version is simply the algorithm described in Appendix A, with r/h set to $\sqrt{(15/2)}$. The second and fifth rows of Table 1 indicate that Chew and Chen's (2003a) "Algorithm 1" and the algorithm described by Chew and Chen (2003b), which are essentially the same in that both use just the cumulative CE to assign pitch names, can be implemented by setting both w_s and f to 0 (Chew and Chen 2005). If $w_s = 0$, then, from Equation 3 (in Appendix A), $CE_{\text{global},j} = CE(W_{\text{sound}}(j, j-1))$. Chew and Chen (2005, p. 71) state that, when the window over which a CE is to be computed is empty, "a CE is not generated, and the algorithm defaults to a do-nothing step." Therefore, setting w_s to 0 causes steps 1 and 2 of the algorithm described in Appendix A to be skipped. Because $f = 0$, there is also no point in computing $CE_{\text{local},j}$ in step 3, since this value will be multiplied by 0 when computing $CE_{\text{hybrid},j}$ in step 5. When $f = 0$ we should therefore assume that the term $f.CE_{\text{local},j}$ in Equation 6 (in Appendix A) is ignored. To summarize, when $f = 0$ and $w_s = 0$, only steps 4 and 5 of the algorithm in Appendix A are executed, and, in step 5, $CE_{\text{hybrid},j}$ is set to equal $CE_{\text{cum},j}$.

Chew and Chen (2005, p. 71) also state that their sliding-window algorithm (2003a) can be implemented using the algorithm in Appendix A by "setting w_s to the desired window size, w_r to 0 and f to 1." This is indicated in the third row of Table 1. Setting w_r to 0 implies that the window over which the local CE would be calculated will be empty, which, in turn, implies that "a CE is not generated, and the algorithm defaults to a do-nothing step" (Chew and Chen 2005, p. 71). Setting w_r to 0 therefore causes step 3 of the algorithm in Appendix A to be skipped. It also causes the term $f.\ CE_{\text{local},j}$ in Equation 6 to be ignored. Setting f to 1 means that the cumulative CE is multiplied by 0 in step 5 of the algorithm;

Table 1. The Various Versions of Chew and Chen's Pitch-Spelling Algorithm Described in Their Publications

Algorithm	*Publication*	w_s	w_r	f	r/h	*Init. index*
Two-phase boot-strapping algorithm	Chew and Chen (2005); Chew (2004)	Any	$< w_s$	$0 \le f \le 1$	$\sqrt{(15/2)}$	2
Algorithm 1: Cumulative CE	Chew and Chen (2003a)	0	0	0	?	0
Algorithm 2: Sliding window	Chew and Chen (2003a)	Any	0	1	?	0
Algorithm 3: Two-phase assignment	Chew and Chen (2003a)	Any	$< w_s$	$0 \le f \le 1$	?	0
Cumulative CE	Chew and Chen (2003b)	0	0	0	?	0

therefore there is no point in calculating the cumulative CE in step 4, and the term $(1 - f).CE_{\text{cum},j}$ in step 5 is also ignored. To summarize, setting w_r to 0 and f to 1 causes only steps 1 and 2 of the algorithm in Appendix A to be executed. Finally, as indicated in the fourth row of Table 1, the "two-phase assignment" algorithm (Chew and Chen 2003a) is essentially the same as the algorithm in Appendix A with the initial index set to 0.

The ChewChen Implementation of Chew and Chen's Algorithm

In this study, Meredith's (2007) implementation of the algorithm, called ChewChen, was used. I provide full pseudo-code for this implementation elsewhere (Meredith 2007). The ChewChen algorithm has eleven parameters: *CCNoteList*, w_s, w_r, f, *AspectRatio, ChunkSize, InitialIndex, MinSAIndex, MaxSAIndex, StartOrSound,* and *SAOrLOF.* The w_s, w_r, and f parameters have the same meanings in ChewChen as they do in Equations 3, 4, and 6 in Appendix A; *AspectRatio* gives the spiral array aspect ratio, r/h, as defined earlier; and the *ChunkSize* parameter specifies the duration of each chunk in milliseconds. In Chew and Chen's own implementations of their algorithm, the chunk duration was defined to be a metrical unit in the score-based test corpus. However, chunk size can only be defined in this way if the metrical structure of the input passage is known. In the ChewChen implementation, it is assumed that the input data does not contain metrical information (as is often the case in real-world transcription situations), so the chunk size is set to be a fixed number of milliseconds in duration. Every onset time and duration in the test corpus used here was strictly proportional to its notated value, and every input encoding was set to be at a reasonable tempo. However, the algorithm was also run on a "noisy" version of this test corpus in which the durations and onset times of the corresponding movements in the clean version were multiplied and shifted respectively by small random amounts in order to simulate (very crudely) asynchronous chord onsets and expressive local tempo changes.

The CCNoteList parameter is an ordered set of records representing the passage of music to be spelled. Each record in CCNoteList represents a single note or sequence of tied notes in the music. As previously explained, the notes in the first chunk are initially spelled so that their indices are as close as possible to some specified initial index that Chew and Chen (2005) set to 2. The InitialIndex parameter of the ChewChen algorithm is used to specify this initial index.

Chew and Chen (2003b, 2005) explain that their algorithm is actually restricted to assigning pitch name classes within a specified range on the spiral array. Specifically, their own implementation can only assign pitch name classes whose indices are between –15 (F-double-flat) and 19 (B-double-sharp), inclusive. In ChewChen, this range is specified by providing the minimum and maximum permitted spiral array indices in the parameters *MinSAIndex* and *MaxSAIndex,* respectively.

As previously explained, in their own implementation, Chew and Chen calculate the CE for a window by considering the notes that sound in it, not the notes that start in it. They also weight each note by the duration for which it sounds within the window over which the CE is being calculated. However, it seems possible that good results might be obtainable by adopting the simpler strategy of con-

Table 2. Combination of Values for w_s, w_r, and f Required for Switching Off Each Combination of the CE Types (Global, Local, and Cumulative)

CEs to switch off	*Parameter values*	*Operational CEs*
Global	$w_s = 0$	Cumulative
Local	$w_r = 0$ or $f = 0$	Cumulative
Cumulative	$f = 1$	Global and local
Global and Local	$w_s = 0$	Cumulative
Global and Cumulative	$w_s = 0$ and $f = 1$	None
Local and Cumulative	$w_r = 0$ and $f = 1$	Global
Global, Local, and Cumulative	$w_s = 0$ and $f = 1$	None

sidering the notes that start in a window and weighting each note by its entire duration. In the ChewChen algorithm, it is possible to adopt this latter strategy instead of that used by Chew and Chen by setting the *StartOrSound* parameter to "Starting" instead of "Sounding."

Finally, as previously pointed out, Chew and Chen (2005) acknowledge that the spiral array is just "a spiral configuration of the line of fifths." However, they claim (2005, p. 66) that the "depth added by going from one to three dimensions [i.e., from the line of fifths to the spiral array] allows the modeling of more complex hierarchical relations." Nevertheless, it is not obvious that "the modeling of more complex hierarchical relations" necessarily leads to better pitch-spelling performance. In the ChewChen algorithm, the user can choose to use the line of fifths instead of the spiral array by setting the *SAOrLOF* parameter to "LOF" instead of "SA." This allows the algorithm's performance when using the line of fifths to be directly compared with its performance when using the spiral array.

Switching On and Off the Local, Global, and Cumulative CEs

As should by now be apparent, the parameters w_s, w_r, and f can be used to "switch off" one or more of the three types of CE (global, local, or cumulative) that Chew and Chen's algorithm uses to spell the notes. Table 2 gives the parameter values required to turn off each combination of these three types of CE as well as the CEs that are left operational after doing so.

Thus, if the global CE is switched off by setting w_s to 0, no names are assigned in Phase I, so no local CE can be computed in Phase II. Consequently, switching off the global CE has the net effect of causing the notes to be spelled as close as possible to the cumulative CE. (See the first row in Table 2.)

The local CE can be switched off by setting either w_r or f to 0. Either way, the notes are spelled as close as possible to the cumulative CE in Phase II, overriding the global CE spellings assigned in Phase I. (See the second row in Table 2.) The cumulative CE can be switched off by setting f to 1. This causes the global CE spellings assigned in Phase I to be modified in Phase II using the local CE. (See the third row of Table 2.)

Both the global and local CEs are effectively switched off by setting w_s to 0, because switching off the global CE in Phase I means that the local CE cannot be computed in Phase II. In this case, therefore, the notes are spelled so that they are as close as possible to the cumulative CE in Phase II. (See the fourth row in Table 2.) The fact that the global CE cannot be switched off without also switching off the local CE means that if both the global and cumulative CEs are switched off, no notes are spelled whatsoever. (See the fifth row in Table 2.) To switch off both the local and cumulative CEs (i.e., omit Phase II), w_r must be set to 0 and f must be set to 1. (See the sixth row in Table 2.) The net result is that the notes are spelled so that they are as close as possible to the global CE. Finally, all the CEs can be switched off by setting w_s to 0 and f to 1. Setting w_s to 0 switches off the global CE and therefore the local CE; setting f to 1 switches off the cumulative CE. (See the seventh row in Table 2.)

There are therefore just four possible CE combinations in practice: (1) all three CEs have an effect ($w_s \neq 0$, $w_r \neq 0$, and $0 < f < 1$); (2) just the cumulative CE is used ($f = 0 \vee (0 < f < 1 \wedge (w_r = 0 \vee w_s = 0))$); (3) just the global CE is used ($f = 1$, $w_r = 0$, and $w_s > 0$); or (4) only the global and local CEs have an effect ($f = 1$, $w_r > 0$, and $w_s > 0$).

Chew and Chen's Own Evaluations of Their Algorithms

The test corpus used by Chew and Chen (2003a, 2003b) consisted of just two movements from Beethoven's piano sonatas: the third movement of the *Sonata in G major, Op. 79* (1375 notes) and the first movement of the *Sonata in E major, Op. 109* (1516 notes). In a later evaluation (2005), they ran the algorithm on the *Song of Ali-Shan,* a set of variations on a Taiwanese folksong by You-Di Huang (1,571 notes), in addition to the two Beethoven piano sonata movements. It is difficult to see how one could justify assuming that the results obtained on such a small corpus generalize to some interesting larger population of tonal works.

Chew and Chen (2005) claim that, over all three pieces, the algorithm makes 28 errors when $\langle w_s, w_r, f \rangle$ is either $\langle 4, 3, 0.8 \rangle$ or $\langle 8, 6, 0.9 \rangle$. However, it seems that only the cumulative CE version of their algorithm was run on the third movement of Beethoven's *Sonata in G major, Op. 79.* If this is indeed the case, then the results for this movement cannot meaningfully be combined with the results obtained when the two-phase version of the algorithm was run on the other two movements in the test corpus. Therefore, their claim that their algorithm made 28 errors over all three movements may be invalid. Chew and Chen (2005, p. 71) state that they "set the chunk size to one beat" and that "the beat size was read from the MIDI files." It is therefore also possible that the chunks did not all have the same duration in milliseconds in all three of the movements in their test corpus.

In Chew and Chen's experiments, the least number of errors on the first movement of Beethoven's Op. 109 was made when the two-phase algorithm was used and $\langle w_s, w_r, f \rangle$ was set to $\langle 4, 3, 0.8 \rangle$, $\langle 4, 3, 0.7 \rangle$, or $\langle 8, 6, 0.9 \rangle$. With these settings, the algorithm made 27 errors on this movement. They state (2005, p. 74) that "the next best result had 30 errors using the parameters $\langle 8, 6, 0.8 \rangle$ and $\langle 16, 6, 0.8 \rangle$" and claim that "since the best results were achieved with high values for *f,* we can deduce that the local context is more important than the global [i.e., cumulative] context." However, the second-lowest note-error count on the first movement of Beethoven's Op. 109 was actually 28 errors and was made when the parameters $\langle w_s, w_r, f \rangle$ were set to $\langle 4, 2, 0.6 \rangle$—that is, with *f* set to the lowest value tested in their experiments. This clearly casts some doubt on their claim that the local context is more important than the cumulative context.

However, the main point to be made here is that, owing to the small size of the test corpus and the small range of parameter values tested in Chew and Chen's experiments, any conclusions drawn from their results must be considered tentative at best.

A More Thorough Evaluation of Chew and Chen's Algorithm

A more thorough evaluation of Chew and Chen's algorithm was therefore carried out in which the ChewChen implementation of the algorithm (Meredith 2007) was run 1,296 times on a test corpus *C* consisting of 216 movements by eight Baroque and Classical composers (J. S. Bach, Beethoven, Corelli, Handel, Haydn, Mozart, Telemann, and Vivaldi). This test corpus contained 195,972 notes, almost exactly equally shared between the eight composers represented. On each of the 1,296 runs of ChewChen over *C,* the algorithm was run with a different combination of parameter values. Table 3 shows, for each parameter, the set of values used in the evaluation.

First, all parameter value combinations in the cross product of the sets in the right-hand column of Table 3 were generated to give a set *P.* Then a reduced set *P′* of parameter value combinations was generated by removing all elements from *P* except: (1) those in which the global, local, and cumulative CEs all have an effect (i.e., those for which $w_s \neq 0$, $w_r \neq 0$, and $0 < f < 1$); (2) those that have one

Table 3. The Sets of Parameter Values Used to Evaluate Chew and Chen's Algorithm

Parameter	*Set of values used*
w_s	{0, 4, 8, 16}
w_r	{0, 2, 4, 6}
F	{0, 0.25, 0.5, 0.75, 1}
AspectRatio	{√(2/15), √(15/2)}
ChunkSize (in msec)	{500, 1000, 2000}
StartOrSound	{"Sounding", "Starting"}
SAOrLOF	{"SA", "LOF"}
⟨*MinSAIndex, MaxSAIndex*⟩	{⟨–15, 19⟩, ⟨–22, 26⟩}

particular combination of values for w_s, w_r, and *f* that causes only the cumulative CE to be used (the combination $f = 0$, $w_r = 2$, and $w_s = 4$ was used, as the results for this combination had already been obtained in an earlier experiment); (3) those in which only the global CE is used (i.e., those in which $f = 1$, $w_r = 0$, and $w_s > 0$); and (4) those in which only the global and local CEs have an effect ($f = 1$, $w_r > 0$, and $w_s > 0$). (See the discussion above concerning switching the different types of CE on and off.)

A subset of P', denoted by P'', was then generated, containing all the parameter value combinations in P' except (1) those in which $w_r > w_s + 1$ (apart from the chunk currently being spelled, the local context window should not contain chunks that are not in the global context window); and (2) those in which *SAOrLOF* is set to "LOF" and *AspectRatio* is √(15/2). (Note that when *SAOrLOF* is "LOF," the *AspectRatio* parameter has no effect, so there is no point in using more than one "dummy" value of *AspectRatio* in this case.) The ChewChen algorithm was run on the test corpus *C* with each of the 1,296 parameter value combinations in P''.

Results and Discussion

In this study, the spelling accuracy of an algorithm is measured in terms of *note error count* and *note accuracy.* The note error count of an algorithm *A* over a set of movements *S* is the total number of notes in *S* spelled incorrectly by *A*. The *note accuracy* of *A* over *S* is the proportion of notes in *S* spelled correctly by *A*, expressed as a percentage. In this study, the standard deviation of the note accuracies achieved by an algorithm over the eight composers in *C* was used to measure the extent to which the note accuracy of an algorithm is dependent on the style of the music being processed.

Parameter Value Combinations Achieving Highest Note Accuracy

The highest overall note accuracy achieved by the ChewChen algorithm in this experiment on the test corpus *C* was 99.15%. The algorithm achieved this note accuracy with 12 of the 1,296 parameter value combinations in P''. The twelve parameter-value combinations with which ChewChen achieved this highest note accuracy were those in which $w_s = 8$, $w_r = 2$, $f = 0.5$, and *ChunkSize* was set to 500 msec. That is, ChewChen performed best when (1) the local, global, and cumulative CEs all had an effect; (2) the local context window was as small as possible; (3) the global context window was a moderate size; (4) the local and cumulative CEs were given equal weighting in Phase II; and (5) the chunks were small, leading to a frequent updating of the CEs.

The parameters that were critical for achieving the highest note accuracy were therefore those controlling the duration of the windows used (determined by w_r, w_s, and ChunkSize) and the relative weighting given to the local and cumulative CEs in Phase II (determined by *f*). Note that, for these twelve best versions of the algorithm, it did not matter whether the spiral array or the line of fifths was used; nor did the aspect ratio of the spiral array make any difference to the results. Note also that, for these twelve best parameter value combinations, changing the range of permitted indices from ⟨*MinSAIndex, MaxSAIndex*⟩ = ⟨–15, 19⟩ to ⟨*MinSAIndex, MaxSAIndex*⟩ = ⟨–22, 26⟩ made no difference to the results.

Table 4 gives the parameter values for the twelve most accurate versions of the ChewChen algorithm tested, together with an identification code, CCOP01–CCOP12, for each. The results obtained using these twelve most-accurate versions of the ChewChen algorithm are summarized in Tables 5 and 6. As can be seen in Table 5, the standard devia-

Table 4. Parameter Values for the Best Performing Versions of the ChewChen Algorithm Tested

Code	w_s	w_r	*f*	*AspectRatio*	*ChunkSize*	*StartOrSound*	*SAOrLOF*	*MinSAIndex*	*MaxSAIndex*
CCOP01	8	2	0.5	√(15/2)	500	"Sounding"	"SA"	–22	26
CCOP02	8	2	0.5	√(15/2)	500	"Sounding"	"SA"	–15	19
CCOP03	8	2	0.5	√(2/15)	500	"Sounding"	"LOF"	–22	26
CCOP04	8	2	0.5	√(2/15)	500	"Sounding"	"LOF"	–15	19
CCOP05	8	2	0.5	√(2/15)	500	"Sounding"	"SA"	–22	26
CCOP06	8	2	0.5	√(2/15)	500	"Sounding"	"SA"	–15	19
CCOP07	8	2	0.5	√(15/2)	500	"Starting"	"SA"	–22	26
CCOP08	8	2	0.5	√(15/2)	500	"Starting"	"SA"	–15	19
CCOP09	8	2	0.5	√(2/15)	500	"Starting"	"LOF"	–22	26
CCOP10	8	2	0.5	√(2/15)	500	"Starting"	"LOF"	–15	19
CCOP11	8	2	0.5	√(2/15)	500	"Starting"	"SA"	–22	26
CCOP12	8	2	0.5	√(2/15)	500	"Starting"	"SA"	–15	19

The first column gives an identification code for each parameter value combination.

Table 5. Note Accuracies Expressed as Percentages for Each Set of Algorithms on the Complete Test Corpus (*Comp*), and for Each Subset of the Test Corpus Containing Movements by One of the Eight Composers (*Bach* to *Viva*)

Algorithm	*Bach*	*Beet*	*Core*	*Hand*	*Hayd*	*Moza*	*Tele*	*Viva*	*Comp*	*Mean*	*SD*
CCOP01–06	99.29	98.73	99.38	99.44	98.51	99.06	99.39	99.40	99.15	99.15	0.35
CCOP07–12	99.39	98.80	99.44	99.46	98.28	99.10	99.37	99.37	99.15	99.15	0.42

The columns labeled *Mean* and *SD* give the mean and standard deviation, respectively, of the values in columns *Bach* to *Viva*. CCOP01–06 differ from CCOP07–12 only in that *StartOrSound* is set to "Sounding" in the former and "Starting" in the latter.

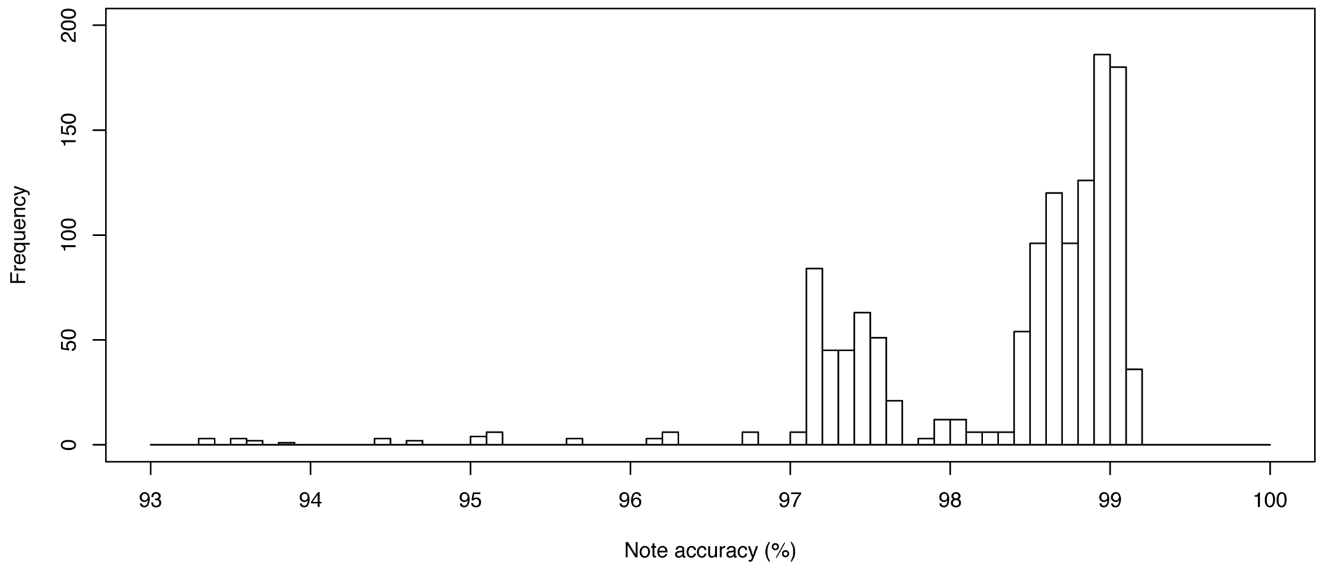
Table 6. Note Error Counts for Sets of Algorithms on the Complete Test Corpus (*Complete*), and for Each Subset of the Test Corpus Containing Movements by One of the Eight Composers (*Bach* to *Vivaldi*)

Algorithm	*Bach*	*Beethoven*	*Corelli*	*Handel*	*Haydn*	*Mozart*	*Telemann*	*Vivaldi*	*Complete*
	(24505)	*(24493)*	*(24493)*	*(24500)*	*(24490)*	*(24494)*	*(24500)*	*(24497)*	*(195972)*
CCOP01–06	175	311	152	136	365	230	149	147	1665
CCOP07–12	150	295	138	132	421	220	155	155	1666

The number in parentheses underneath each column heading gives the number of notes in that subset of the test corpus. CCOP01–06 differ from CCOP07–12 only in that *StartOrSound* is set to "Sounding" in the former and "Starting" in the latter.

tion (*SD*) of the note accuracies achieved by algorithms CCOP01–06 over the eight composers was lower than that for algorithms CCOP07–12. This indicates that, of these twelve best versions of the algorithm, the six which took into account all the notes sounding within each window when calculating the CEs (i.e., CCOP01–06) were less dependent on style than those which only took into account the notes starting in each window. It is worth noting, however, that all twelve of the algorithms CCOP01–12 were less dependent on style over the corpus *C* than any of the other algorithms that I considered (Meredith 2007). Note also, from Table 6, that, although CCOP07–12 only made one more error overall than CCOP01–06, the errors made by CCOP01–06 were different in general from those

Figure 2. Histogram showing the distribution of note accuracies achieved by ChewChen *over C for all the 1,296 parameter value combinations in* P″.



made by CCOP07–12, as is evident from the differing note error counts for the individual composers.

Frequency Distribution of Note Accuracies Over *P*″

The set of parameter value combinations *P*″ can be partitioned into four classes according to the combination of CEs that is used (i.e., cumulative, global alone, global and local, or all three). If *p* is a parameter value combination in *P*″, then let GLC(*p*) denote the combination of CEs employed when the ChewChen algorithm is run with the parameter value combination *p*, where GLC(*p*) = "GLC" if and only if the global, local, and cumulative CEs are all used; GLC(*p*) = "GL" if and only if the global and local CEs are exclusively used; GLC(*p*) = "G" if and only if the global CE is used exclusively; and GLC(*p*) = "C" if and only if the cumulative CE is used exclusively.

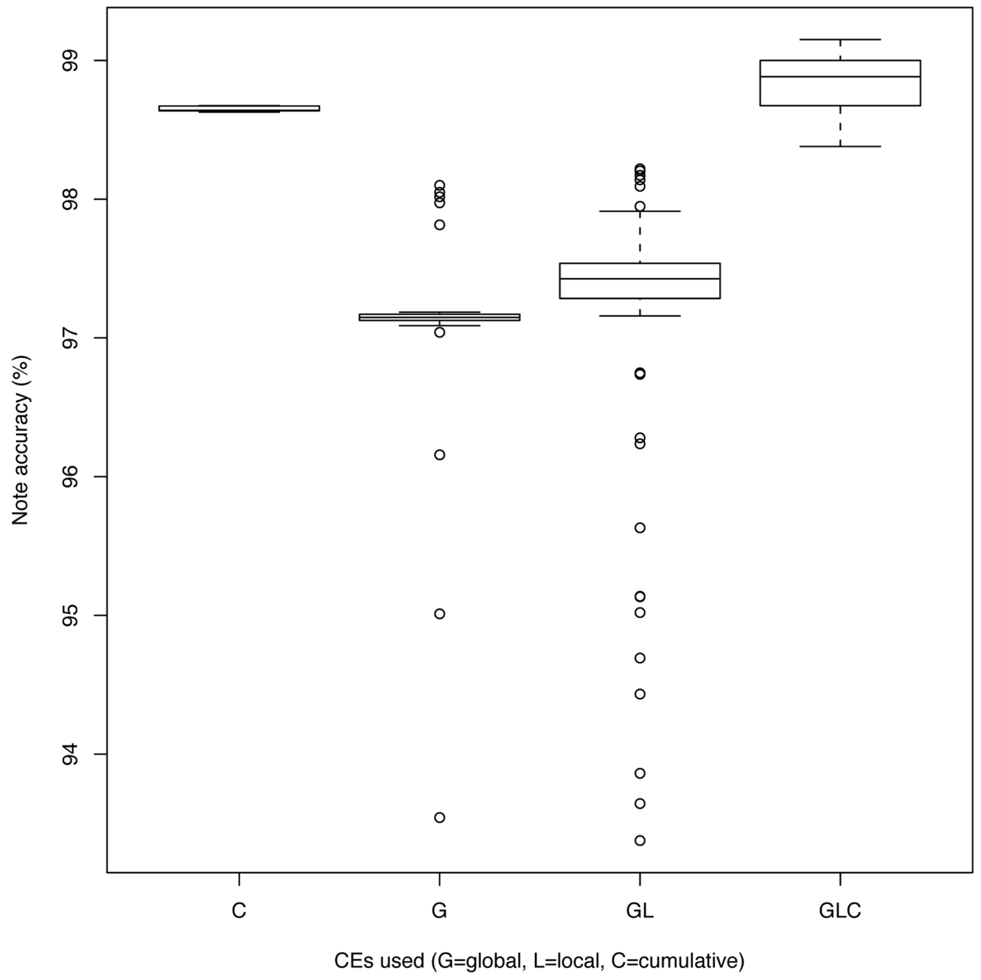
If [*x*] denotes the set $\{p | p \in P'' \wedge \mathrm{GLC}(p) = x\}$, then *P*″ can be partitioned into the four equivalence classes: [GLC], [GL], [G], and [C]. The histogram in Figure 2 has two peaks, suggesting that the note accuracies over the whole of *P*″ can be derived from two separate populations, one with a mode at just above 97% and the other with a mode at around 99%.

The box plot in Figure 3 shows the distributions of note accuracies over each of these four equivalence classes. From this box plot, it is clear that the distribution in Figure 2 has two peaks, because the note accuracies for [GL] and [G] are mostly between 97 percent and 98 percent, whereas those for [GLC] and [C] lie above 98 percent.

As indicated in Figure 3, all the versions of the algorithm that used the cumulative CE were more accurate than those that did not, with those that used the cumulative CE achieving a median note accuracy of 98.86 percent and those that did not achieving a median note accuracy of 97.35 percent. Figure 3 also shows that the note accuracies achieved by the versions in which the cumulative CE was not used are much more widely dispersed than those achieved by the versions in which the cumulative CE was used: the standard deviation of the note accuracies for [G] ∪ [GL] was 0.8145, whereas it was only 0.1990 for [C] ∪ [GLC]. This suggests that using the cumulative CE makes the algorithm much less sensitive to changes in its parameter values. Indeed, by using the cumulative CE alone, it was possible to achieve almost as high a note accuracy as it was by combining it with the local and global CEs.

These results cast doubt on Chew and Chen's suggestion that "a purely sliding-window method should perform better than a purely cumulative-window method" (2005, p. 74) and suggest that the

*Figure 3. Box plot of note accuracy against GLC(*p*) for all* p *in* P".



opposite may, in fact, be the case. Note that the results reported here were obtained on a relatively large collection of Baroque and Classical works of which several contained highly chromatic passages, whereas Chew and Chen's claims were based on results obtained on just two Beethoven sonata movements and *The Song of Ali-Shan* by You-Di Huang, which is a highly stable, tonal work (indeed, it is almost entirely diatonic).

The non-parametric, distribution-free Wilcoxon rank sum test for two independent samples (which is equivalent to the Mann–Whitney test) was used to determine whether there were any statistically significant differences with respect to note accuracy between the four equivalence classes [C], [G], [GL], and [GLC]. According to this test, all four classes were significantly different from each other with respect to note accuracy ($p < 10^{-7}$). Also, the note accuracies achieved when the cumulative CE was used (i.e., $[C] \cup [GLC]$) were very significantly different from those achieved when it was not ($[G] \cup [GL]$) ($p < 2.2 \times 10^{-16}$).

Effect of *SAOrLOF* and *AspectRatio*

The set of parameter value combinations P'' tested in this evaluation can be partitioned into 432 equivalence classes E_i such that each class E_i contains all and only those elements in P'' that have a particular combination of values for the parameters w_s, w_r, *f, ChunkSize, StartOrSound, MinSAIndex,* and *MaxSAIndex.* In other words, within each class E_i, the parameter value combinations differ only with respect to *SAOrLOF* and *AspectRatio.* Each of the 432 classes E_i contains three parameter value combinations: one with *SAOrLOF* set to "LOF" and *AspectRatio* = √(2/15); another with *SAOrLOF* set to "SA" and *AspectRatio* = √(2/15); and a third with an *SAOrLOF* of "SA" and *AspectRatio* = √(15/2).

Out of the 432 equivalence classes E_i, it was found that there were only two classes in which the three parameter value combinations did not have the same note-error count. Moreover, both of these equivalence classes contained only poorly performing parameter value combinations that achieved note accuracies between 93.64 percent and 95.02 percent over *C.* In other words, for 99.54 percent of the parameter value combinations tested (including the best-performing ones), it did not matter whether the spiral array or the line of fifths was used, and the aspect ratio of the spiral array made no difference to the results.

Effect of *StartOrSound*

The set of parameter value combinations *P″* can be partitioned into 648 equivalence classes F_i such that each F_i contains two parameter value combinations that are identical, except that, in one, *StartOrSound* = "Starting," whereas, in the other, *StartOrSound* = "Sounding." The effect of the *StartOrSound* parameter can be studied by comparing the note accuracies of the two parameter value combinations in each of the equivalence classes F_i.

It was found that changing the value of *StartOrSound* from "Sounding" to "Starting," without changing any of the other parameter values, increased the overall note accuracy over *C* in 510 (78.7 percent) of the 648 cases tested and decreased note accuracy in the remaining 138 cases. However, in 621 (95.83 percent) of the 648 cases tested, the absolute change in percentage note accuracy over *C* caused by changing the value of *StartOrSound* was less than 0.3. In the remaining 27 cases (4.17 percent), changing the value of *StartOrSound* from "Sounding" to "Starting" reduced the note accuracy expressed as a percentage by 1.6 or more. It is worth noting that the parameter value combinations in which *StartOrSound* = "Starting" in the 27 most affected F_i were precisely the 27 worst performing (i.e., least accurate) combinations of the 1,296 tested.

From a practical point of view, this result suggests that not much is gained by using Chew and Chen's more complex method of calculating the centers of effect for the notes (corresponding to *StartOrSound* = "Sounding"), instead of the simpler method suggested above (i.e., with *StartOrSound* = "Starting") in which only the notes starting in each window are taken into account when calculating the CEs, and each note is weighted by its whole duration. In particular, for the twelve most-accurate versions tested, using the simpler method in CCOP07–12 resulted in just one more error over the entire test corpus (Table 6).

Effect of w_s

Recall that w_s is the size of the global context window. The effect of w_s on the performance of the ChewChen algorithm can be studied by examining the effect that changing the value of w_s has on the overall note accuracy over *C* when the values of all the other parameters are kept constant. To do this, each of the equivalence classes [G], [GL], and [GLC], previously defined, can themselves be partitioned into smaller equivalence classes G_i such that the parameter value combinations in each G_i are identical except for the values of w_s. Because w_s must be greater than or equal to $w_r - 1$, each G_i will contain three parameter value combinations when $w_r < 6$: one for $w_s = 4$, one for $w_s = 8$, and one for $w_s = 16$. However, an equivalence class G_i in which $w_r = 6$ will contain only two parameter value combinations: one in which $w_s = 8$ and another in which $w_s = 16$.

It was found that increasing w_s from 4 to 8 significantly increased the percentage note accuracy (according to the Wilcoxon test) when the local window was used, but the size of the increase was greater (0.29) when the cumulative window was not used than when it was (0.02). Increasing w_s from 8 to 16 significantly increased the percentage note accuracy by around 0.10 when only the global and local windows were used. Increasing w_s from 4 to 16 significantly increased the percentage note accuracy when the cumulative window was not used, but the increase was greater (0.48) when the local window was used than when it was not (0.03). When all the tested parameter value combinations were considered together, increasing the global window size from 4 to either 8 or 16 significantly increased the percentage note accuracy by around 0.05, whereas increasing it from 8 to 16 significantly increased the percentage note accuracy, but only by about 0.01.

Effect of w_r

Recall that w_r is the size of the local context window. The effect that w_r has on the performance of the ChewChen algorithm can be studied by examining the effect that changing the value of w_r has on the overall note accuracy when the values of all the other parameters are kept constant. To do this, each of the equivalence classes [GL] and [GLC] can themselves be partitioned into smaller equivalence classes H_i such that the parameter value combinations in each H_i are identical except for the values of w_r. Because w_s must be greater than or equal to $w_r - 1$, each H_i will contain three parameter value combinations when $w_s \geq 6$: one for $w_r = 2$, one for $w_r = 4$, and one for $w_r = 6$ (see Table 3). However, an equivalence class H_i in which $w_s = 4$ will contain only two parameter value combinations: one in which $w_r = 2$, and another in which $w_r = 4$.

It was found that increasing the size of the local window generally led to a reduction in overall note accuracy. Under most conditions, increasing the local window size by 2 or 4 chunks reduced the percentage note accuracy by less than 0.2, with slightly larger decreases occurring when the cumulative CE was not used.

Effect of f

The effect that f has on the performance of the ChewChen algorithm can be studied by measuring the change in overall note accuracy that results when f is changed and the values of the other parameters are held constant. When $w_r = 0$ and $f = 1$, only the global CE calculated in Phase I of the algorithm has any effect. When $w_r = 0$ and $f < 1$, only the cumulative CE has any effect, regardless of the specific value of f. Therefore, for each case in which $w_r = 0$, we only need to compare the result obtained when $f = 1$ with that obtained when $f = 0$. Furthermore, when $f = 0$, the values of w_r and w_s are immaterial. Therefore, to investigate the effect on note accuracy of changing f when $w_r = 0$, we must compare the result for each parameter value combination p in which $w_r = 0$ and $f = 1$ with that for the parameter value combination in which $f = 0$ and all other parameters (except possibly w_s and w_r) have the same values as in p. The total set of parameter value combinations over which this comparison is carried out will be denoted by $S_{w_r=0}$. $S_{w_r=0}$ can be partitioned into 288 pairs of parameter value combinations I_i such that each equivalence class I_i contains one parameter value combination p in which $w_r = 0$ and $f = 1$, and a second combination in which $f = 0$ and the other parameter values (apart from possibly w_r and w_s) have the same values as in p.

It was found that changing f from 0 to 1 when $w_r = 0$ resulted in a highly significant pseudomedian reduction in the percentage note accuracy of 1.495. Note that when $w_r = 0$, changing f from 0 to 1 is equivalent to changing from using only the cumulative CE to using only the global CE (see previous discussion).

For $w_r \neq 0$, then (1) when $f = 1$, only the global and local CEs have an effect; (2) when $f = 0.75$, all three CEs have an effect, but in Phase II, the local CE has a greater influence than the cumulative CE; (3) when $f = 0.5$, all three CEs have an effect, and in Phase II, the local and cumulative CEs have the same degree of influence; (4) when $f = 0.25$, all three CEs have an effect, but in Phase II, the cumulative CE has a greater influence than the local CE; and (5) when $f = 0$, only the cumulative CE has an effect.

To investigate the effect of changing f when $w_r \neq 0$, the set $S_{w_r \neq 0}$ containing all parameter value combinations in P'' in which $w_r \neq 0$, can be partitioned into 288 equivalence classes J_i such that each J_i contains parameter value combinations that are identical except for their values of f. Each J_i therefore contains five parameter value combinations, one for each of the five values of *f* used in the evaluation.

It was found that changing between any of the five values of *f* resulted in a highly significant change in overall note accuracy. The value of *f* that led to the highest accuracy was 0.5, followed in order by 0.25, 0, 0.75, and 1. The best accuracy was therefore achieved when the cumulative and local CEs made equal contributions in Phase II. Changing *f* from 0.5 to 0.25 (i.e., making the contribution of the cumulative CE greater than that of the local CE) reduced the overall percentage note accuracy by about 0.12. Changing *f* from 0.25 to 0 (i.e., eliminating the local CE altogether) reduced the overall percentage note accuracy by a further 0.25. However, omitting the local CE altogether was marginally better than weighting it more heavily than the cumulative CE in Phase II: changing *f* from 0 to 0.75 reduced the percentage note accuracy by a small, but apparently significant, 0.05. Finally, eliminating the cumulative CE altogether produced the worst results: changing *f* from 0.75 to 1 caused a relatively large drop of around 1.2 in the percentage note accuracy. (For more details, see Meredith 2007.)

These results clearly cast doubt on Chew and Chen's claims that the local context is more important than the cumulative context and that "a purely sliding-window method should perform better than a purely cumulative-window method" (2005, p. 74). In fact, the results obtained on the relatively large and varied test corpus used here seem to suggest, first, that a "purely cumulative-window method" is more accurate than one that only uses the sliding windows; and, second, that the cumulative context is just as important as the local context in Phase II of the algorithm.

Effect of *ChunkSize*

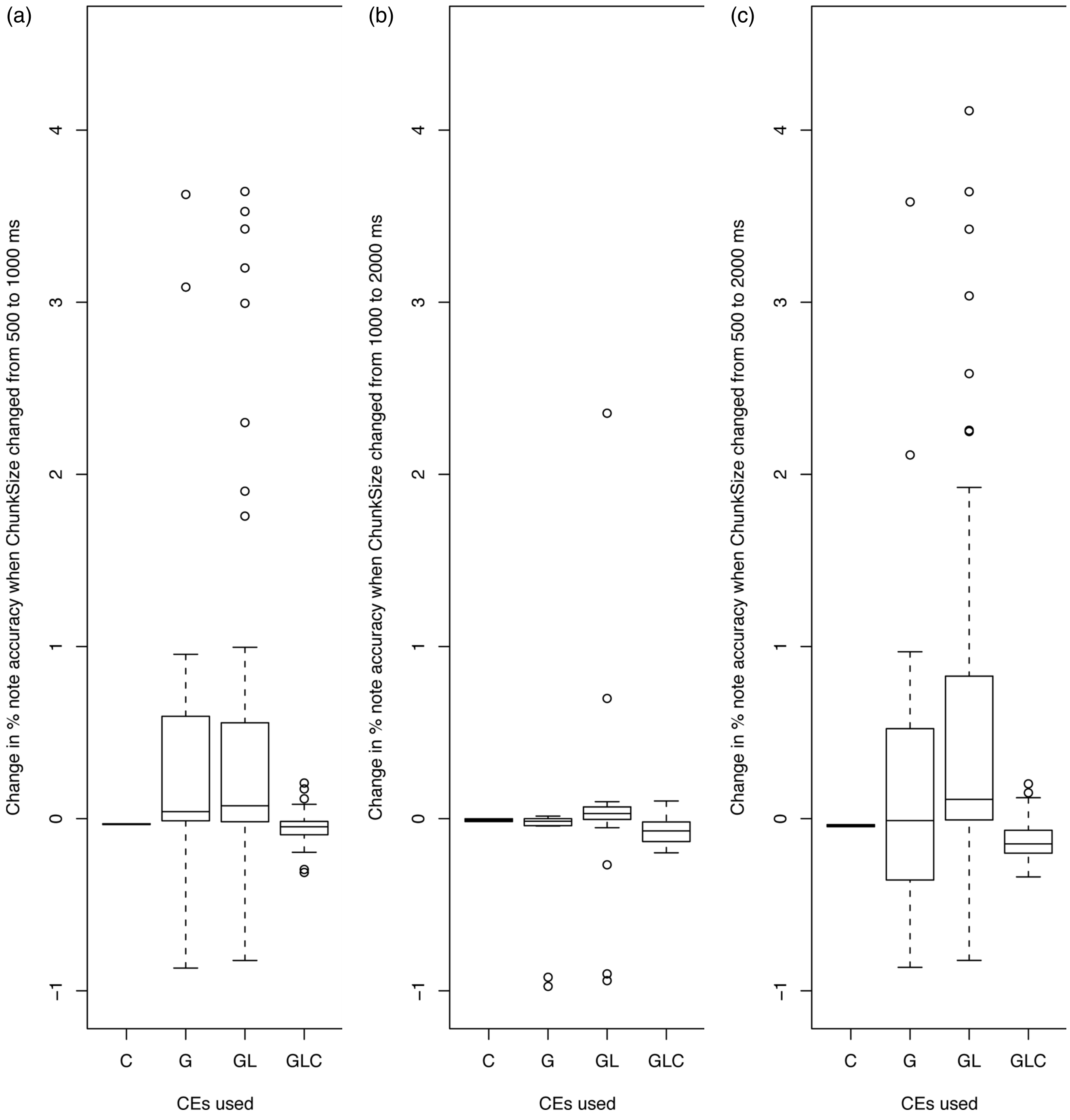
In Chew and Chen's own implementations of their algorithm, the chunk duration was defined to be a metrical unit in the score-based test corpus. However, chunk size can obviously only be defined in this way if the metrical structure of the input passage is known. In the study reported here, it was assumed that the input data did not contain metrical information (as is usually the case in real-world transcription situations), so the chunk size was set to three different absolute time periods. The effect that *ChunkSize* has on the performance of the ChewChen algorithm can be studied by examining the effect that changing the value of *ChunkSize* has on the overall note accuracy when the values of all the other parameters are kept constant. To do this, each of the equivalence classes [GLC], [GL], [G], and [C] can themselves be partitioned into smaller equivalence classes K_i such that the parameter value combinations in each K_i are identical except for the values of *ChunkSize.*

The three box plots in Figure 4 give a general impression of the way that the note accuracy is affected over each of the sets [GLC], [GL], [G], and [C] when *ChunkSize* is changed from 500 to 1,000 msec (Figure 4a), 1,000 to 2,000 msec (Figure 4b), and 500 to 2,000 msec (Figure 4c). These box plots suggest that, when the cumulative CE is not used, then (1) increasing *ChunkSize* from 500 to either 1,000 or 2,000 msec often causes an increase in note accuracy; (2) increasing *ChunkSize* from 500 to either 1,000 or 2,000 msec has a much more varied effect than when it is increased from 1,000 to 2,000 msec; and (3) increasing *ChunkSize* from 1,000 to 2,000 msec generally has only a very small effect on note accuracy.

When the cumulative CE is used, (1) increasing *ChunkSize* generally decreases the overall note accuracy; and (2) increasing *ChunkSize* generally has a larger and more varied effect when the global and local CEs are also used. A more detailed statistical analysis has shown these effects to be significant (Meredith 2007).

As just observed, increasing *ChunkSize* generally reduces note accuracy when the cumulative CE is used. Increasing *ChunkSize* reduces the frequency with which the CEs are updated but also increases the size of the local and global windows. As discussed above, all the parameter value combinations in which the cumulative CE is used achieved higher note accuracies than those that did not (Figure 3). Using the cumulative CE also seems to reduce the sensitivity of the algorithm to changes in the pa-

Figure 4. Box plots showing the effect for each of the sets [GLC], [GL], [G], and [C] on percentage note accuracy of changing ChunkSize *from (a) 500 to 1,000 msec; (b) 1,000 to 2,000 msec; and (c) 500 to 2,000 msec.*



rameter value combinations. It therefore seems that the cumulative CE dominates over the other two CEs when it is used. In those cases where the cumulative CE is used, one might therefore expect the overall effect of changing *ChunkSize* to be dominated by the effect that this has on the cumulative CE. Because the cumulative CE is always calculated over the entire segment of music that has already passed, the only effect that increasing *ChunkSize* has on the cumulative CE is to reduce the frequency with which it is updated, which one would expect to lead to lower note accuracy, as observed.

On the other hand, when the cumulative CE is not used, increasing *ChunkSize* increases the sizes of the global and local windows as well as reducing the frequency with which the CEs are updated. Intuitively, one would expect there to be some optimal window sizes for calculating the global and local CEs: if these windows are too big, the algorithm will not be sufficiently sensitive to changes in key; if they are too small, the context may not contain enough information to determine the key accurately. This intuition is in agreement with the results obtained here, which suggest that over the set [G], increasing *ChunkSize* from 500 to 1,000 msec generally improves note accuracy, but that increasing it from 1,000 to 2,000 msec slightly reduces note accuracy. This seems to imply that, when the global CE is used alone, there is an optimal value for *ChunkSize* that is somewhere between 500 and 2,000 msec. Similarly, over the set [GL], increasing *ChunkSize* from 500 to 1,000 msec increases the percentage note accuracy by a pseudomedian value of around 0.15, but increasing *ChunkSize* by a further 1,000 to 2,000 msec only increases the percentage note accuracy by around 0.03.

Note that the twelve best-performing parameter value combinations (Table 4) are in [GLC] and that, in these cases, *ChunkSize* is set to 500 msec.

Effect of *MinSAIndex* and *MaxSAIndex*

To investigate the effect of changing the value of ⟨*MinSAIndex, MaxSAIndex*⟩, the complete set of parameter value combinations tested, P'', was partitioned into 648 equivalence classes L_i such that each L_i contained two parameter value combinations that were the same, except that, in one, ⟨*MinSAIndex, MaxSAIndex*⟩ was set to ⟨–15, 19⟩, whereas in the other, it was set to ⟨–22, 26⟩. Recall that, when ⟨*MinSAIndex, MaxSAIndex*⟩ = ⟨–15, 19⟩, the algorithm is restricted to choosing pitch classes that lie between F-double-flat and B-double-sharp (inclusive) on the line of fifths. However, when ⟨*MinSAIndex, MaxSAIndex*⟩ = ⟨–22, 26⟩, the algorithm can choose from a wider range of pitch classes, extending from F-triple-flat to B-triple-sharp, inclusive.

It was found that, for over 99 percent of the parameter value combinations tested, changing ⟨*MinSAIndex, MaxSAIndex*⟩ from ⟨–15, 19⟩ to ⟨–22, 26⟩ had no effect when the cumulative CE was used and changed the overall note accuracy when the cumulative CE was *not* used. When the cumulative CE was not used, changing ⟨*MinSAIndex, MaxSAIndex*⟩ from ⟨–15, 19⟩ to ⟨–22, 26⟩ increased the percentage note accuracy by an average of 0.01 in about half of the cases and reduced it by an average of 0.8 in the other half of the cases. Changing ⟨*MinSAIndex, MaxSAIndex*⟩ had no effect on the best-performing parameter value combinations.

Comparing Chew and Chen's Algorithm with Other Pitch-Spelling Algorithms

As previously mentioned, I compared the performance of a number of pitch spelling algorithms, including those of Chew and Chen, on the test corpus *C* used here (Meredith 2007). I found that CCOP01–12 were the least dependent on style of the algorithms that were tested over this corpus. The optimized versions of Chew and Chen's algorithm (i.e., CCOP01–12) achieved a note accuracy of 99.15 percent over *C*, and the only algorithms that achieved a higher note accuracy than CCOP01–12 over *C* were the best versions of my *ps13* algorithm and the best versions of Temperley and Sleator's *Melisma* programs. However, the best versions of the *Melisma* programs were much more dependent on style than CCOP01–12 and also highly sensitive to tempo. For most practical applications, therefore, CCOP01–12 would be preferable to the *Melisma* programs for pitch spelling.

Nevertheless, the optimized versions of Chew and Chen's algorithm made over 50 percent more errors over the test corpus *C* than my PS13s1 algorithm, which achieved an overall note accuracy of 99.44 percent and a low standard deviation in note accuracy of 0.49 over the eight composers. Moreover, a real-time version of PS13s1 spelled 99.19 percent of the notes in *C* correctly, thus making over 4 percent fewer errors than the best versions of Chew and Chen's algorithm. Also, PS13s1 is somewhat simpler to implement than Chew and Chen's algorithm as it uses a much simpler windowing process.

I also ran the best versions of the ChewChen algorithm (i.e., CCOP01–12) on a temporally "noisy" version of *C,* which is denoted here by *C′* (Meredith 2007). *C′* was derived from *C* by artificially introducing temporal deviations of the type typically found in MIDI files derived from human performances. Specifically, the onset times of the notes were randomly shifted by up to 50 msec, and the durations of notes were randomly multiplied by a factor between 0.5 and 1.5. This provided a very crude simulation of the spreading of chords and inaccuracies in synchonicity typical in performances (Gabrielsson 1999).

Interestingly, the six best versions of the ChewChen algorithm in which only the notes starting in each window were considered when calculating the CEs (i.e., CCOP07–12) performed better over the noisy corpus *C′* than the six best versions in which the notes sounding in each window were considered (i.e., CCOP01–06). CCOP07–12 spelled 99.12 percent of the notes in *C′* correctly with a standard deviation over the eight composers of 0.47, whereas CCOP01–06 spelled 99.01 percent of the notes in *C′* correctly with a standard deviation over the eight composers of 0.55.

By comparison, the best-performing algorithm over *C′* in my study was the best (non-real-time) version of my PS13s1 algorithm, which spelled 99.41 percent of the notes in *C′* correctly with a standard deviation over the eight composers of 0.50. The real-time version of this algorithm achieved an overall note accuracy over *C′* of 99.16 percent with a standard deviation over the eight composers of 0.53.

Conclusions

The results of this study suggest that Chew and Chen's pitch-spelling algorithm (as described in Appendix A) works best when (1) the local, global, and cumulative CEs all have an effect; (2) the local context window is relatively small (two chunks); (3) the global context window is a moderate size (eight chunks); (4) the local and cumulative CEs are given equal weighting; and (5) the chunks are relatively small (500 msec), leading to a frequent updating of the CEs. This result, obtained on a test corpus containing 195,972 notes and 216 movements from works composed by eight different composers between 1681 and 1808, casts doubt on Chew and Chen's claim (based on results obtained on just two Beethoven sonata movements and a highly stable tonal work by You-Di Huang) that the local context is more important than the cumulative context. Indeed, of the 1,296 parameter value combinations tested, all those in which the cumulative CE was used achieved higher note accuracies than those in which it was not, with those that used the cumulative CE achieving a mean note accuracy of 98.83 percent and those that did not achieving a mean note accuracy of 97.19 percent. Moreover, using the cumulative CE alone worked almost as well as using it in combination with the global and local CEs. The results obtained on the relatively large and varied test corpus employed here cast doubt on Chew and Chen's claim that, in general, "a purely sliding-window method should perform better than a purely cumulative-window method" and suggest that the opposite may, in fact, be the case. Nevertheless, the algorithm worked best when the local and cumulative contexts were given equal weight in Phase II of the algorithm.

It was also shown that Chew and Chen's algorithm performs just as well when the spiral array is replaced with the simpler line of fifths, strongly suggesting that—at least for pitch spelling—the spiral array offers no advantages over the simpler line of fifths. The performance of the algorithm is also largely unaffected when Chew and Chen's method of calculating the CEs, which involves considering all the notes that sound in a window, is replaced with a simpler method in which only the notes starting in a window are considered. Moreover, the latter, simpler method worked better than the former when the data contained temporal deviations of the type that typically occurs in MIDI files derived from human performances.

When compared with other algorithms Chew and Chen's algorithm performed relatively well but was outperformed on both "clean" and "temporally noisy" data by my PS13s1 algorithm (Meredith 2006, 2007), which can be made to work in real time and uses a much simpler windowing process than Chew and Chen's algorithm.

Acknowledgments

The research reported here was funded by EPSRC Grant No. GR/S17253/02. I would like to thank Elaine Chew for kindly providing further details of her and Mr. Chen's implementation of their pitch-spelling algorithm. I would also like to thank Geraint Wiggins, Doug Keislar, and an anonymous reviewer for their helpful comments on an earlier version of this article.

References

Cambouropoulos, E. 1996. "A General Pitch Interval Representation: Theory and Applications." *Journal of New Music Research* 25(3):231–251.
Cambouropoulos, E. 1998. "Toward a General Computational Theory of Musical Structure." PhD dissertation, University of Edinburgh.
Cambouropoulos, E. 2001. "Automatic Pitch Spelling: From Numbers to Sharps and Flats." Paper presented at the VIII Brazilian Symposium on Computer Music, 8 January, Fortaleza, Brazil.
Cambouropoulos, E. 2003. "Pitch Spelling: A Computational Model." *Music Perception* 20(4):411–429.
Chew, E. 2000. "Toward a Mathematical Model of Tonality." PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts.
Chew, E. 2004. Personal correspondence, 29 December.
Chew, E., and Y.-C. Chen. 2003a. "Determining Context-Defining Windows: Pitch Spelling Using the Spiral Array." Paper presented at the Fourth International Conference on Music Information Retrieval, 26–30 October, Baltimore, Maryland.
Chew, E., and Y.-C. Chen. 2003b. "Mapping MIDI to the Spiral Array: Disambiguating Pitch Spellings." *Proceedings of the 8th INFORMS Computer Society Conference.* Norwell, Massachusetts: Kluwer, pp. 259–275.
Chew, E., and Y.-C. Chen. 2005. "Real-Time Pitch Spelling Using the Spiral Array." *Computer Music Journal* 29(2):61–76.
Gabrielsson, A. 1999. "The Performance of Music." In D. Deutsch, ed. *The Psychology of Music,* 2nd ed. San Diego, California: Academic Press, pp. 501–602.
Honingh, A. 2006. "Pitch Spelling Using Compactness." Paper presented at the *Ninth International Conference on Music Perception and Cognition,* 22–26 August, Bologna.
Longuet-Higgins, H. C. 1976. "The Perception of Melodies." *Nature* 263(5579):646–653.
Longuet-Higgins, H. C. 1987a. "The Perception of Melodies." In H. C. Longuet-Higgins, ed. *Mental Processes: Studies in Cognitive Science.* London: British Psychological Society/MIT Press, pp. 105–129.
Longuet-Higgins, H. C. 1987b. "Two Letters to a Musical Friend." In H. C. Longuet-Higgins, ed. *Mental Processes: Studies in Cognitive Science.* London: British Psychological Society/MIT Press, pp. 64–81.
Longuet-Higgins, H. C. 1993. "The Perception of Melodies." In S. M. Schwanauer and D. A. Levitt, eds. *Machine Models of Music.* Cambridge, Massachusetts: MIT Press, pp. 471–495.
Meredith, D. 2003. "Pitch-Spelling Algorithms." *Proceedings of the Fifth Triennial ESCOM Conference.* Hanover, Germany: ESCOM, pp. 204–207.
Meredith, D. 2005. "Comparing Pitch-Spelling Algorithms on a Large Corpus of Tonal Music." In U. K. Wiil, ed. *Computer Music Modeling and Retrieval, Second International Symposium.* Berlin: Springer, pp. 173–192.
Meredith, D. 2006. "The *ps13* Pitch-Spelling Algorithm." *Journal of New Music Research* 35(2):121–159.
Meredith, D. 2007. "Computing Pitch Names in Tonal Music: A Comparative Analysis of Pitch-Spelling Algorithms." D.Phil. dissertation, University of Oxford.
Stoddard, J., C. Raphael, and P. E. Utgoff. 2004. "Well-Tempered Spelling: A Key-Invariant Pitch Spelling Algorithm." Paper presented at the Fifth International Conference on Music Information Retrieval, 10–14 October, Barcelona.
Temperley, D. 2001. *The Cognition of Basic Musical Structures.* Cambridge, Massachusetts: MIT Press.

Appendix A

The following provides a brief overview of Chew and Chen's pitch-spelling algorithm.

1. First, the algorithm computes the *global CE, $CE_{\text{global},j}$*, which is the CE of the set of notes in a sliding *global context window* that consists of the w_s chunks immediately preceding the jth chunk. In other words, the algorithm computes the value

   $$CE_{\text{global},j} = CE(W_{\text{sound}}(j - w_s, j - 1)) \tag{3}$$

   where w_s, the number of chunks in the sliding global context window, is a parameter of the algorithm.
2. Next, the algorithm names each note in chunk j so that its pitch name is as close as possible to $CE_{\text{global},j}$ in the spiral array.
3. Then the algorithm computes a *local CE, $CE_{\text{local},j}$*, which is the CE of the set of notes in a *local context window* that consists of the chunk j that has just been spelled, together with the $(w_r - 1)$ chunks immediately preceding the jth chunk. That is, it computes the value of

   $$CE_{\text{local},j} = CE(W_{\text{sound}}(j - w_r + 1, j)) \tag{4}$$

   where w_r, the number of chunks in the local context window, is another parameter of the algorithm.
4. Next, the algorithm computes the *cumulative CE, $CE_{\text{cum},j}$*, which is the CE of the notes in a *cumulative window* that consists of all the chunks preceding the jth chunk. That is, it computes the value of

   $$CE_{\text{cum},j} = CE(W_{\text{sound}}(1, j - 1)) \tag{5}$$

5. Finally, the notes in chunk j are respelled so that their pitch names are as close as possible to the *hybrid CE,*

   $$CE_{\text{hybrid},j} = f.CE_{\text{local},j} + (1 - f).CE_{\text{cum},j} \tag{6}$$

   where f is a parameter with a value between 0 and 1 which determines the relative weight given to the local and cumulative CEs.