Contour Algorithms Review

Marcos da Silva Sampaio
Universidade Federal da Bahia (UFBA)
marcos@sampaio.me

Pedro Kroger
Universidade Federal da Bahia (UFBA)
kroger@pedrokroger.net

***Abstract**: In this paper, we present some problems of two Music Contour Relations Theory operations algorithms: the Refinement of Contour Reduction Algorithm, which was developed by Rob Schultz, and the Equivalence Contour Class Prime Form algorithm, which was developed by Elizabeth Marvin and Paul Laprade. We also propose two alternative algorithms to solve these problems.*

***Keywords**: Music Contour Theory, Reduction Algorithm, Algorithm, Equivalent contour classes.*

I. Introduction

Contour is the shape or format of an object and can be either bi- or multidimensional. In music, a contour is an abstraction of elements such as pitch, chord density, duration, timbre, or intensity. A melodic contour, for instance, is a map of pitches in time.

Musical contour is "a set of points in one sequential dimension ordered by any other sequential dimension" [10]. The study of contour is important because, as is the case with motifs and pitch class sets, it can contribute to musical coherence.

The Musical Contour Relations Theory provides concepts and operations with which to establish contour identity and similarity measures for analytical and compositional purposes. This theory has supported the analysis of works by composers such as W. A. Mozart [1], Luigi Dallapicolla [6], Arnold Schoenberg [4, 10], Anton Webern [3], Olivier Messiaen [12], Elliott Carter, Pierre Boulez, Iannis Xenakis, Alois Haba, Milton Babbitt, Harrison Birtwistle, and Igor Stravinsky [2], as well as video game soundtracks [8].

In this paper, we present some problems of two Contour Theory operations algorithms: the Refinement of Contour Reduction Algorithm [13] and the Equivalence Contour Class Prime Form algorithm [6, 7]. We also propose two alternative algorithms for solving them.

II. Contour Theory basic concepts and operations

A complete presentation of Contour Theory concepts and operations is outside the scope or this paper[1]. However, an understanding of contour space, contour segment, comparison matrix, class equivalence, prime form, and contour reduction is necessary to follow this paper's premise.

[1]See Beard [1], Bor [2], and Sampaio [11] for further information concerning Music Contour Theory.

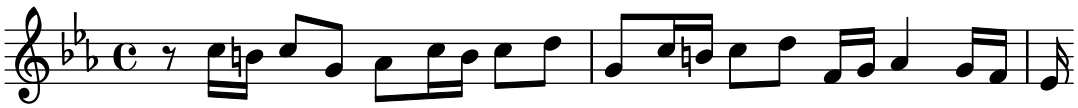


Figure 1: *Fugue 2 subject, The Well-Tempered Clavier by J.S. Bach*

Table 1: *Comparison matrix of the contour M < 5 4 5 2 3 >*

	5	4	5	2	3
5	0	-	0	-	-
4	+	0	+	-	-
5	0	-	0	-	-
2	+	+	+	0	+
3	+	+	+	-	0

Contour space[2] is a musical space composed of the contour elements (or contour points [CPs][3]). The contour space of pitches, for instance, is composed of all pitches represented as integers from low to high and numbered from 0 to $n-1$ [9], where n denotes cardinality. For instance, the contour space of pitches in the melody shown in Figure 1 is $S = \{0, 1, 2, 3, 4, 5, 6\}$, representing the pitches *Eb*, *F*, *G*, *Ab*, *B*, *C*, and *D*.

A contour segment, or, more simply, a contour, is an ordered set of contour points. For instance, the contour of the first five notes in Figure 1 is M < 5 4 5 2 3 >. Normalization (or translation) is an operation that simplifies the contour representation by re-enumerating from 0 to $n-1$. The motive M < 5 4 5 2 3 > normalization is < 3 2 3 0 1 >.

The relation of any pair of CPs is ternary: one CP is lower, equal to, or higher than the other. The contour can be analyzed and represented in a linear or combinatorial way. The linear contour regards only the relations between adjacent CPs; the combinatorial contour considers relations between adjacent and non-adjacent CPs. For instance, in the contour M, the relation between 5 and 4 is lower ($-$), whereas it is higher ($+$) between 2 and 3.

The relations among the contour points are represented in a comparison matrix (cf. Table 1).

Two contours are equivalent if they share the same comparison matrix. The set of equivalent contours forms a contour class. This class is represented by an equivalent class prime form[4]. According to Marvin and Laprade citeMarvin1987, the retrograded, inverted, and retrograded-inverted versions of a contour belong to its equivalence class. The prime form of this class can be calculated with a simple algorithm (cf. Section IV).

Any contour can be reduced to a prime form through the Contour Reduction Algorithm [10]. This algorithm, like the

Schenkerian reductive techniques,

> sets up hierarchical levels of pitch salience. High and low peaks are selected: "passing notes" and "inner voices" are pruned. That is, the algorithm provides a criterion for associating nonadjacent notes by picking out those that are presumably most obvious to the ear [10].

Robert Morris [10] proposed 25 basic contour prime forms to which any contour could be reduced. Rob Schultz [13] proposed a refinement to this algorithm because some contours such as

[2]See Lewin [5] for further information regarding musical spaces.

[3]A contour point is also known as a contour pitch, or c-pitch.

[4]Do not confuse this with prime contours, as proposed by Robert Morris [10].

A < 2 1 3 0 > are not reducible to one of the basic primes by the Morris algorithm.

The equivalence classes and prime contours are useful for checking the identity of the contours for analytical and compositional purposes. For instance, the contours A < 0 1 3 2 > and B < 1 0 2 3 > belong to the same class, as represented by their prime form < 0 1 3 2 >. The B contour is obtained by inverting and retrograding the A contour. As another example of this, the contours C < 0 2 4 1 3 > and D < 0 3 2 1 > have a common prime: P < 0 2 1 >, as obtained by the Refined Contour Reduction Algorithm.

III. Refined Contour Reduction Algorithm (by Schultz)

The Refined Contour Reduction Algorithm [13] (cf. Algorithm 1) removes intermediary CPs and preserves salient ones with the support of an auxiliary internal structure called the *max-/min-list*. This algorithm has a preliminary component (Steps 1 to 5), as well as a recurrent one (Steps 6 to 17).

CONTOUR-REDUCTION-ALGORITHM(C), where C is a contour. Let variable N:

Step 0: Set N to 0.

Step 1: Flag all maxima in C upwards; call the resulting set the max-list.

Step 2: Flag all minima in C downwards; call the resulting set the min-list.

Step 3: If all c-pitches are flagged, go to step 6.

Step 4: Delete all non-flagged c-pitches in C.

Step 5: N is incremented by 1 (i.e., N becomes N + 1).

Step 6: Flag all maxima in the max-list upward. For any string of equal and adjacent maxima in the max-list, flag all of them, unless: (1) one c-pitch in the string is the first or last c-pitch of C, then flag only it; or (2) both the first and last c-pitches of C are in the string, then flag (only) both the first and last c-pitches of C.

Step 7: Flag all minima in min-list downward. For any string of equal and adjacent minima in the min-list, flag all of them, unless: (1) one c-pitch in the string is the first or last c-pitch of C, then flag only it; or (2) both the first and last c-pitches of C are in the string, then flag (only) both the first and last c-pitches of C.

Step 8: For any string of equal and adjacent maxima in the max-list in which no minima intervene, remove the flag from all but (any) one c-pitch in the string.

Step 9: For any string of equal and adjacent minima in the min-list in which no maxima intervene, remove the flag from all but (any) one c-pitch in the strings.

Step 10: If all c-pitches are flagged, and no more than one c-pitch repetition in the max-list and min-list (combined) exists, not including the first and last c-pitches of C, proceed directly to step 17.

Step 11: If more than one c-pitch repetition in the max-list and/or min-list (combined) exists, not including the first and last c-pitches of C, remove the flags on all repeated c-pitches except those closest to the first and last c- pitches of C.

Step 12: If both flagged c-pitches remaining from step 11 are members of the max-list, flag any one (and only one) former member of the min-list whose flag was removed in step 11; if both c-pitches are members of the min-list, flag any one (and only one) former member of the max-list whose flag was removed in step 11.

Step 13: Delete all non-flagged c-pitches in C.

Step 14: If N $\neq$ 0, N is incremented by 1 (i.e., N becomes N + 1).

Step 15: If N = 0, N is incremented by 2 (i.e., N becomes N + 2).

Step 16: Go to step 6.

Step 17: End. N is the "depth" of the original contour C.

Algorithm 1 Refined Contour Reduction Algorithm (by Schultz)

A *maximum* point exists if, in a triad of CPs, the middle CP is equal to or greater than its neighbors. By default, the first and last CPs in a sequence are also maxima. The maxima-list is calculated in a linear manner with Algorithm 2 (in pseudocode). This algorithm returns a max-list from a given sequence of CPs. The *minima* and min-list are calculated in an analogous way. For instance, the maxima- and minima-list of the contour S < 1 0 3 2 > are $M = [1, 3, 2]$, and $m = [1, 0, 2]$.

Algorithm 2 *Maxima* list algorithm

MAX-LIST(S), where S is a sequence of CPs, represented as $S = \{S_0, S_1, ..., S_{n-2}, S_{n-1}\}$ and n is its cardinality.

```
Let max-list
Add S_0 to max-list
for i from 0 to n − 2 do
  if (S_{i+1} ≥ S_i) ∧ (S_{i+1} ≥ S_{i+2}) then
    Add S_i to max-list
  end if
end for
Add S_{n−1} to max-list
return max-list
```

In the Reduction Algorithm, the CPs are recurrently flagged as *maxima* and *minima*, and the CPs that are not flagged are removed. Each loop iteration increases the algorithm depth value that represents the complexity of the contour reduction. The loop finishes when all CPs are flagged.

In the first part, the contour is used as the sequence of CPs for max- and min-list calculus. In the second part, the max- and min-lists themselves are used for the calculus.

As an illustration, consider the reduction of the contour A[5] < 1 3 0 2 0 2 0 2 0 2 0 3 1 > following Algorithm 1.

Step 0. $N = 0$.

Step 1. *Maxima* flag: $\{A_0, A_1, A_3, A_5, A_7, A_9, A_{11}, A_{12}\}$ (Figure 2a).

Step 2. *Minima* flag: $\{A_0, A_2, A_4, A_6, A_8, A_{10}, A_{12}\}$ (Figure 2b).

Step 3. Conditional: all flagged, jump to Step 6.

Step 6. First part: Flag *maxima* from max-list: $\{A_0, A_1, A_5, A_7, A_{11}, A_{12}\}$ (Figure 2c).

Second part: The repeated adjacent *maxima* sequence ($\{A_5, A_7\}$) does not involve either the first or the last CP. Thus, both are flagged.

Step 7. First part: Flag *minima* from min-list: $\{A_0, A_2, A_4, A_6, A_8, A_{10}, A_{12}\}$ (Figure 2c).

Second part: The adjacent *minima* sequence ($\{A_2, A_4, A_6, A_8, A_{10}\}$) does not involve either the first or the last CP. Thus, both are flagged.

Step 8. The repeated adjacent *maxima* sequence ($\{A_5, A_7\}$) contains an intervening *minima* (A_6). Thus, the flag is kept.

Step 9. The repeated adjacent *minima* sequence ($\{A_2, A_4, A_6, A_8, A_{10}\}$) contains a slice with two intervening *maxima* ($\{A_4, A_6, A_8\}$) and two other slices without it ($\{A_2, A_4\}$ and $\{A_8, A_{10}\}$). The slices without the intervening *maxima* have the flag removed. The resulting min-list is $\{A_0, A_4, A_6, A_8, A_{12}\}$ (Figure 2d).

[5] In this analysis, we are representing the *maxima* and *minima* using small triangles above and below the CPs, with the letters M (for *maxima*) and m (for *minima*) and the contour as a sequence $A = \{A_0, A_1, ..., A_{11}, A_{12}\}$.

Step 10. There are CPs not flagged ($\{A_2, A_3, A_9, A_{10}\}$), and the combined *maxima* and *minima* repeat ($\{A_4, A_5, A_6, A_7\}$). Go to Step 11.

Step 11. Remove the flags from the combined CPs, keeping the CPs near the beginning and end of the contour (Figure 2e).

Step 12. The CPs flagged in Step 11 ($\{A_4, A_7\}$) belong to the max- and min-lists and remain unchanged.

Step 13. Remove non-flagged CPs ($\{A_2, A_3, A_5, A_6, A_9, A_{10}\}$) (Figure 2f).

Step 14. $N = 0$. Maintain the CPs as being unchanged.

Step 15. $N = 0$. Increment of 2 units: $N = 2$.

Step 16. Return to Step 6.

The second iteration begins with the CPs, *maxima* and *minima* flags illustrated in the figure 3a[6]

Step 6. First part: Flag *maxima* from max-list: ($\{A_0, A_1, A_{11}, A_{12}\}$) (Figure 3b).

Second part: The repeated adjacent *maxima* sequence ($\{A_1, A_{11}\}$) does not involve either the first or the last CP. Thus, all *maxima* are flagged.

Step 7. First part: Flag minima from min-list: ($\{A_0, A_4, A_8, A_{12}\}$).

Second part: The repeated adjacent *minima* sequence ($\{A_4, A_8\}$) does not involve either the first or the last CP. Thus, all *minima* are flagged.

Step 8. The adjacent repeated *maxima* sequence ($\{A_1, A_{11}\}$) has two *minima* intervene ($\{A_4, A_8\}$).

Step 9. The adjacent repeated *minima* sequence ($\{A_4, A_8\}$) has not yet had the *maxima* intervene. The flag of the one repeated *minima* (A_8) is removed (Figure 3c).

Step 10. There are CPs that are not flagged ($\{A_7, A_8\}$). Go to Step 11.

Step 11. There is no combined CP repetition.

Step 12. There is no combined CP repetition.

Step 13. Remove non-flagged CPs ($\{A_7, A_8\}$) (Figure 3d).

Step 14. $N = 2$. Increment of 1 unit: $N = 3$

Step 15. $N = 3$. Maintain the contour as being unchanged.

Step 16. Return to Step 6.

The third iteration begins with the CPs, *maxima* and *minima* flags illustrated in the figure 4: $\{A_0, A_1, A_{11}, A_{12}\}$ and $\{A_0, A_4, A_{12}\}$.

Step 6. First part: Flag *maxima* from max-list: ($\{A_0, A_1, A_{11}, A_{12}\}$) (Figure 4).

Second part: The repeated adjacent *maxima* sequence ($\{A_1, A_{11}\}$) does not involve either the first or the last CP. Thus, all *maxima* are flagged.

Step 7. First part: Flag *minima* from min-list: ($\{A_0, A_4, A_{12}\}$) (Figure 4).

Second part: There are no repeated adjacent *minima* sequences.

Step 8. The repeated adjacent *maxima* sequence contains an intervening *minima*. Thus, the flag is kept.

Step 9. There are no adjacent repeated *minima* sequences.

Step 10. All CPs are flagged, and there are no combined *maxima* and *minima* repetitions. Jump to Step 17.

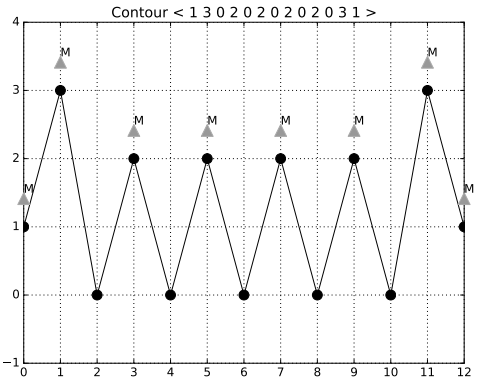
Step 17. The contour A has a depth of 3, is reduced to < 1 3 0 3 1 >, and is normalized to < 1 2 0 2 1 >.

i. Reduction algorithm review

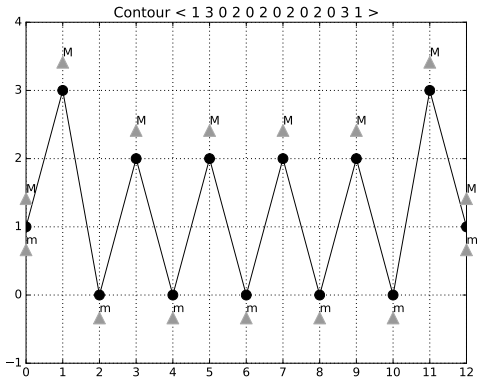
There are two problems in this algorithm:

1. Combined adjacent *maxima* and *minima* repetition is in Step 10.

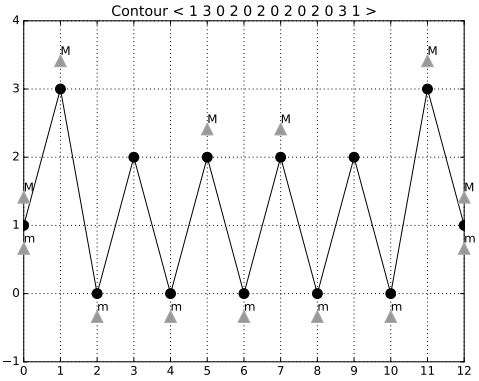
[6]For simplification reasons, we keep the original sequence index.



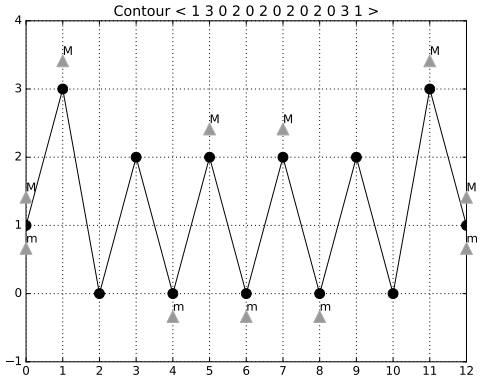
(a) *Step 1*



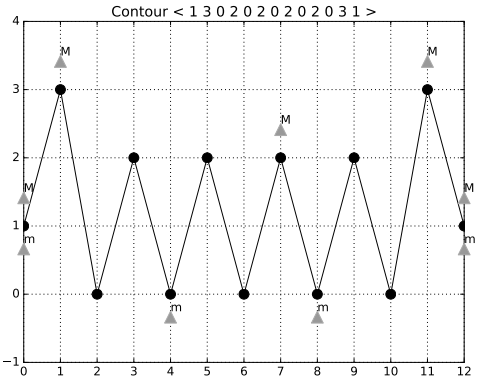
(b) *Step 2*



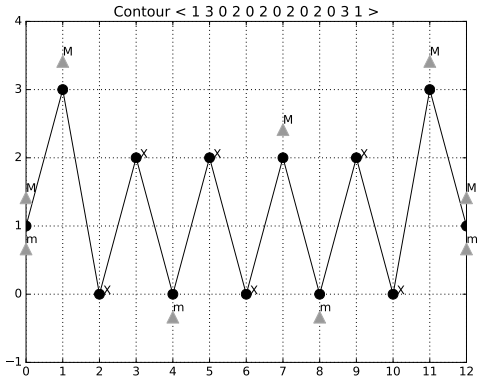
(c) *Steps 6–8*



(d) *Steps 9 and 10*

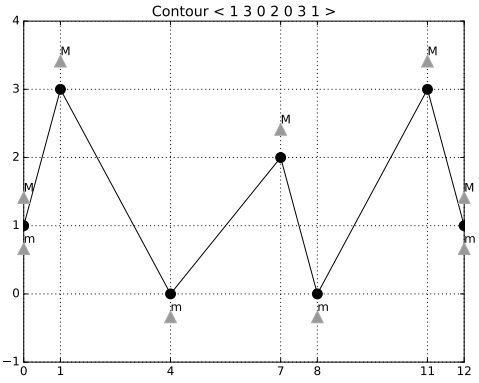


(e) *Steps 11 and 12*

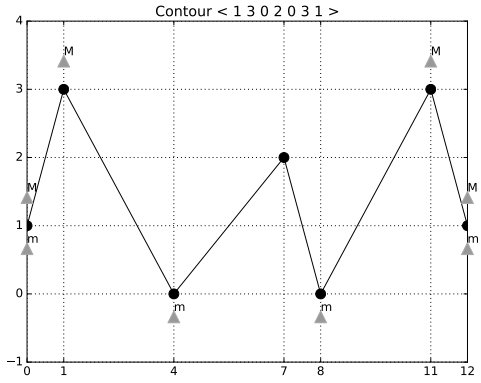


(f) *Steps 13–16*

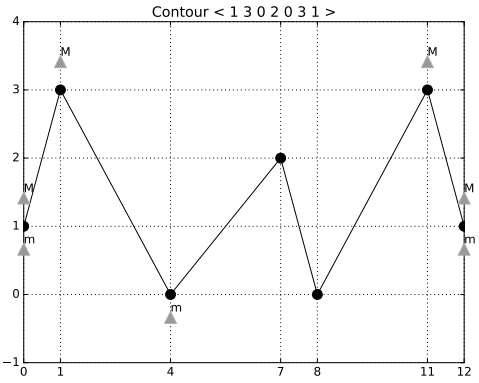
Figure 2: *Reduction algorithm flags in the first iteration*



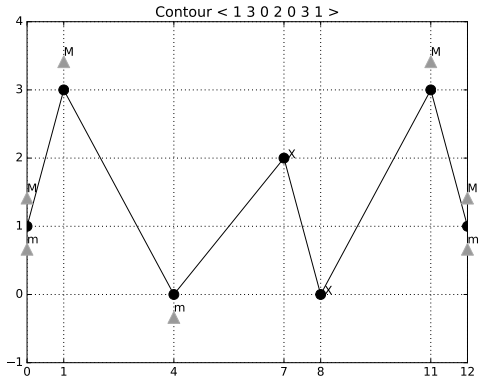
(a) *Initial state*



(b) *Steps 6–8*



(c) *Steps 9–12*



(d) *Steps 13–16*

Figure 3: *Reduction algorithm flags in the second iteration*

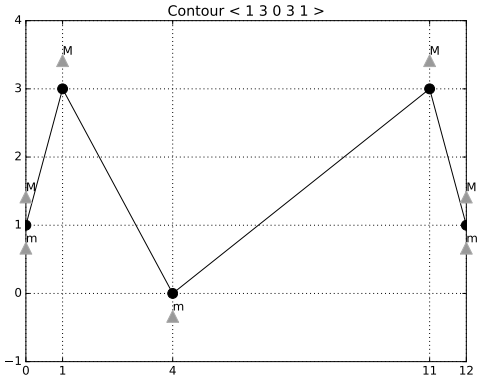


Figure 4: *Reduction algorithm flags in the third iteration*

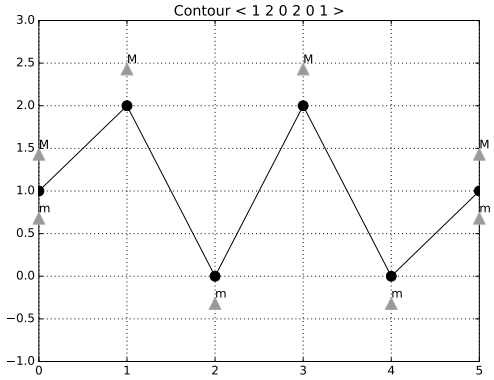


Figure 5: *Combined adjacent maxima and minima repetitions*

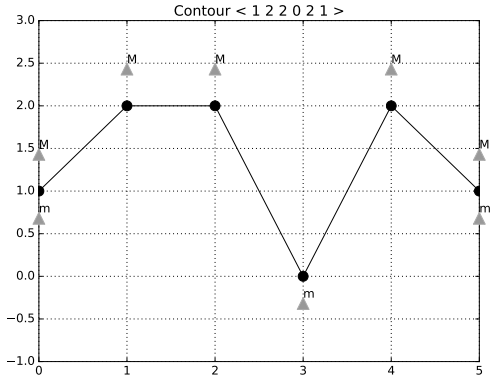


Figure 6: *Contour < 1 2 2 0 2 1 > and flagged maxima and minima.*

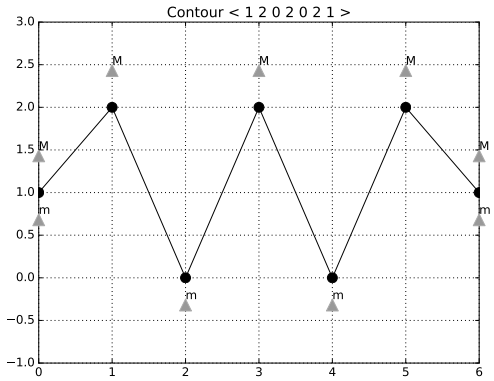
2. There is indefiniteness in the repeated *maxima* and *minima* sequences of Steps 8 and 9.

Step 10 contains two conditions with two possible results: continue with Step 11 or jump to the algorithm's end at Step 17. In the second condition, the expression "no more than one" leads to an error because prime contours cannot have combined *maxima* and *minima* repetitions, and this expression allows for at least one. Thus, the algorithm must not end while these repetitions are in the contour. For instance, the contour A < 1 2 0 2 0 1 > (Figure 5) has all CPs flagged and only one max-min combined repetition of CPs—2 and 0. Thus, in Step 10, the algorithm ends, and the repetition is not removed.

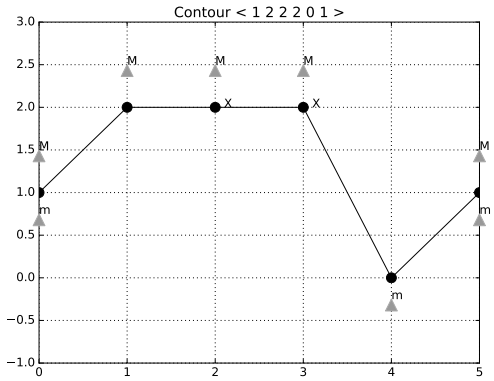
To fix this problem, this step should be rewritten as "and there are no repeated combined CPs in terms of the *maxima* and *minima*."

The expression "string of equal and adjacent *maxima*" in Step 8 is not precise concerning the length of the string. For instance, the contour A < 1 2 2 0 2 1 > (Figure 6) has three repeated *maxima*. There are two possible interpretations of "string" in this example: the unique sequence with three repeated *maxima* $\{A_1, A_2, A_4\}$, or each of the two sequences of repeated *maxima* $\{A_1, A_2\}$ and $\{A_2, A_4\}$.

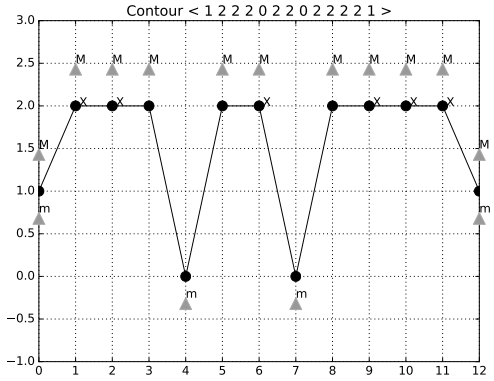
Considering the unique sequence $\{A_1, A_2, A_4\}$ as the string, there is an intervening *minima*



(a) *with minima between the maxima*



(b) *without minima between the maxima*



(c) *with and without minima between the maxima*

Figure 7: *Contour with and without minima between maxima*

(A_3), and we should move to Step 9. Considering each pair of repeated *maxima* as a string, there is no intervening *minima* in the first pair, and we should remove the flag from one of these *maxima*.

In the first case, with a unique string, the algorithm ends with the unchanged contour < 1 2 2 0 2 1 >, and with the repeated *maxima*. In the second case, with two strings, one of the *maxima* is removed, and the algorithm finishes with the correct prime contour < 1 2 0 2 1 >.

There are three particular situations for the equal and adjacent *maxima* in maxima-list:

1. with *minima* between the *maxima* (Figure 7a).
2. without *minima* between the *maxima* (Figure 7b).
3. with slices with and without *minima* between *maxima* in the same string (Figure 7c).

In the first situation, all the *maxima* flags must be kept; in the second, only the first *maxima* must keep it flag; and in the third situation, in the slice without *minima* between the *maxima*, these *maxima* must be unflag, and, in the slices with *minima* between *maxima*, the *maxima* adjacent to the intervening *minima* must be kept. In this third situation, there are cases where two equal adjacent

maxima are between two *minima*. In these cases, the second *maxima* must be unflag (See C_6 in the figure 7c).

There is a similar problem in the analogous Step 9, related to *maxima* between equal adjacent *minima*.

We propose a new version of the algorithm with these questions reviewed (Algorithm 3).

CONTOUR-REDUCTION-ALGORITHM-REVIEW(C), where C is a contour.

Let variable N.

First part

Step 0: Set N to 0.

Step 1: Flag all *maxima* in C upwards, and call the resulting sequence the max-list; flag all *minima* in C downwards, and call the resulting sequence the min-list. All modification in a max- or min-list will result in a new max-/min- list.

Step 2: If all CPs are flagged, go to Step 4.

Step 3: Delete all non-flagged CPs in C, and increment N by 1 (i.e., N becomes $N + 1$).

Second part

Step 4: Flag all *maxima* in the max-list upward, and flag all *minima* in the min-list downward.

Step 5: If there are no adjacent repeated *maxima* in max-list and *minima* in min-list, go to Step 10.

Step 6: For any string of equal and adjacent *maxima* in the max-list:

if First and last CP are present in the string **then**
  Flag them.
else if First or last CP are present in the string **then**
  Flag it.
else
  Flag all *maxima* in the string.
end if

Step 7: For any string of equal and adjacent *minima* in the max-list:

if First and last CP are present in the string **then**
  Flag them.
else if First or last CP are present in the string **then**
  Flag it.
else
  Flag all *minima* in the string.
end if

Step 8: For any string of equal and adjacent *maxima* in max-list:

if There is no *minimum* between the *maxima* **then**
  Flag the first *maxima* and unflag the others.
else if There are *minima* between all the *maxima* **then**
  Flag all the *maxima*.
else
  Flag the *maxima* that are adjacent to the *minima* and unflag the others.
  if There is two adjacent *maxima* between two *minima* **then**
    Unflag the second *maxima*.
  end if
end if

Step 9: For any string of two equal and adjacent *minima* in the min-list:

if There is no *maximum* between the *minima* **then**
  Flag the first *minima* and unflag the others.
else if There are *maxima* between all the *minima* **then**
  Flag all the *minima*.
else
  Flag the *minima* that are adjacent to the *maxima* and unflag the others.
  if There is two adjacent *minima* between two *maxima* **then**
    Unflag the second *minima*.
  end if
end if

Step 10: If there is no CP repetition in the max-list and min-list (combined), not including the first and last CPs of C, go to Step 14.

Step 11: Remove the flags on all repeated CPs except those closest to the first and last CP of C.

Step 12: If both flagged CPs remaining from Step 11 are members of the max-list, flag any one (but only one) former member of the min-list whose flag was removed in Step 11.

Step 13: If both flagged CP remaining from Step 11 are members of the min-list, flag any one (but only one) former member of the max-list whose flag was removed in Step 11.

Step 14: If all CP are flagged, go to Step 17.

Step 15: Delete all non-flagged CP in C and, if $N = 0$, increment N by 1 (i.e., N becomes $N + 1$); otherwise, increment N by 2 (i.e., N becomes $N + 2$).

Step 16: Go to Step 4.

Step 17: End. N is the "depth" of the original contour C.

Algorithm 3 Reviewed Contour Reduction Algorithm

IV. Equivalence Contour Class Prime Form

The equivalence contour class prime form is obtained with the algorithm proposed by Marvin and Laprade [7] (Algorithm 4).

Algorithm 4 Equivalence Contour Class Prime Form Algorithm

EQUIVALENCE-CONTOUR-CLASS-PRIME-FORM(cseg), where cseg is contour:
Step 1: If necessary, translate the cseg so its content consists of integers from 0 to $(n - 1)$,
Step 2: If $(n - 1)$ minus the last c-pitch is less than the first c-pitch, invert the cseg,
Step 3: If the last c-pitch is less than the first c-pitch, retrograde the cseg.

The main feature of this prime form is the compactness in the contour beginning. This feature is inspired by the Post-Tonal Theory's set class prime form. The Contour Interval Succession[7] is a useful tool for how compact the beginning of the contour is. For instance, the contour A < 0 2 1 3 4 > belongs to the same class of its retrograde, inverted, and retrograded/inverted versions: R(A) < 4 3 1 2 0 >, I(A) < 4 2 3 1 0 > and RI(A) < 0 1 3 2 4 >. Table 2 contains the interval successions of these contours. The RI (A) version starts with the lowest positive value of the Contour Interval Succession.

Marvin and Laprade's algorithm has some conditions to invert and/or retrograde the contour to find the prime form of its equivalent class.

[7] A sequence with the differences (or contour intervals) between adjacent CPs [4].

Table 2: *Contour Interval Succession of contour A < 0 2 1 3 4 >, R(A), I(A) and RI(A)*

Contour	Contour Interval Sucession
A < 0 2 1 3 4 >	$[2, -1, 2, 1]$
R(A) < 4 3 1 2 0 >	$[-1, -2, 1, -2]$
I(A) < 4 2 3 1 0 >	$[-2, 1, -2, -1]$
RI(A) < 0 1 3 2 4 >	$[1, 2, -1, 2]$

i. Algorithm review

This algorithm, however, fails in 28 of 235 equivalent classes according to the Marvin and Laprade table [7] (cf. prime form fails in Table 3). Each of these classes has two prime forms. For instance, both contours A < 0 1 3 2 4 > and RI(A) < 0 2 1 3 4> belong to the *cseg-class* 5-3, but according to the Marvin/Laprade algorithm, each one has a particular prime form. They are not changed by the algorithm steps.

Let the processing of the two contours A and RI(A) occur while comparing each step:

Step 1: Both contour A and RI(A) are normalized. Maintain the contour as being unchanged.

Step 2: Both contours have the same cardinality ($N = 5$), and the same last CP (4). Therefore, the condition $(5 - 1) - 4 < 0$ is false. Maintain both contours as being unchanged.

Step 3: Both contours have the same first (0) and last CP (4). Thus, the condition $4 < 0$ is false. Maintain both contours as being unchanged.

Hence, this algorithm fails with some classes. To fix this problem, we propose a new algorithm that is tested using the Marvin/Laprade table (Algorithm 5).

Algorithm 5 Equivalence Contour Class Prime form Algorithm Reviewed

EQUIVALENCE-CONTOUR-CLASS-PRIME-FORM-REVIEW(cseg), where cseg is the contour:
Let the array *arr*:
Step 1: Normalize *cseg*.
Step 2: Get the result of inversion (I), retrogression (R), and retrogression-inversion (RI) and add them to *arr*.
Step 3: Sort the array *arr*.
Step 4: The prime form is the first contour of the array *arr*.

V. Conclusions

In this paper, we revealed problems with the Refined Contour Reduction and the Equivalence Class Prime Form algorithms that lead them to fail with some contour inputs. We demonstrated how and why these algorithms fail, as well as how to fix the issues, and we proposed alternative algorithms.

The problems with the Refined Contour Reduction Algorithm raised here do not allow for an output of the correct reduction of some of the contours with combined maxima-minima like < 1 2 0 2 0 1 >, and with strings of repeated maxima (or minima) with intervening minima (or maxima), such as < 1 2 2 0 2 1 >. However, a review of the string definition and a small change in the algorithm's tenth step fixed these problems.

The problem with the Equivalent Class Prime Form Algorithm led to it returning two prime forms for the same class in 28 of the 235 Marvin/Laprade classes, as shown in the table showing the same [7]. The alternative algorithm proposed in this paper fixed this problem.

Table 3: *Equivalent Class with Two Prime Forms*

Cseg class	FP correta	FP incorreta
5-3	< 0 1 3 2 4 >	< 0 2 1 3 4 >
5-8	< 0 2 3 1 4 >	< 0 3 1 2 4 >
5-25	< 1 0 4 2 3 >	< 1 2 0 4 3 >
5-27	< 1 2 4 0 3 >	< 1 4 0 2 3 >
6-3	< 0 1 2 4 3 5 >	< 0 2 1 3 4 5 >
6-9	< 0 1 3 4 2 5 >	< 0 3 1 2 4 5 >
6-13	< 0 1 4 2 3 5 >	< 0 2 3 1 4 5 >
6-15	< 0 1 4 3 2 5 >	< 0 3 2 1 4 5 >
6-31	< 0 2 3 4 1 5 >	< 0 4 1 2 3 5 >
6-37	< 0 2 4 3 1 5 >	< 0 4 2 1 3 5 >
6-53	< 0 3 2 4 1 5 >	< 0 4 1 3 2 5 >
6-59	< 0 3 4 2 1 5 >	< 0 4 3 1 2 5 >
6-115	< 1 0 2 5 3 4 >	< 1 2 0 3 5 4 >
6-119	< 1 0 3 5 2 4 >	< 1 3 0 2 5 4 >
6-125	< 1 0 5 2 3 4 >	< 1 2 3 0 5 4 >
6-127	< 1 0 5 3 2 4 >	< 1 3 2 0 5 4 >
6-134	< 1 2 3 5 0 4 >	< 1 5 0 2 3 4 >
6-139	< 1 2 5 3 0 4 >	< 1 5 2 0 3 4 >
6-144	< 1 3 2 5 0 4 >	< 1 5 0 3 2 4 >
6-149	< 1 3 5 2 0 4 >	< 1 5 3 0 2 4 >
6-178	< 2 0 1 5 4 3 >	< 2 1 0 4 5 3 >
6-180	< 2 0 4 5 1 3 >	< 2 4 0 1 5 3 >
6-181	< 2 0 5 1 4 3 >	< 2 1 4 0 5 3 >
6-182	< 2 0 5 4 1 3 >	< 2 4 1 0 5 3 >
6-184	< 2 1 4 5 0 3 >	< 2 5 0 1 4 3 >
6-186	< 2 1 5 4 0 3 >	< 2 5 1 0 4 3 >
6-188	< 2 4 1 5 0 3 >	< 2 5 0 4 1 3 >
6-190	< 2 4 5 1 0 3 >	< 2 5 4 0 1 3 >

References

[1] BEARD, R. D. (2003). **Contour modeling by multiple linear regression of the nineteen piano sonatas by Mozart.** Phd Dissertation, Florida State University.

[2] BOR, M. (2009). **Contour reduction algorithms: a theory of pitch and duration hierarchies for post-tonal music.** Phd Dissertation, University of British Columbia.

[3] CLIFFORD, R. J. (1995). **Contour as a structural element in selected pre-serial works by Anton Webern.** Phd Dissertation, University of Wisconsin-Madison.

[4] FRIEDMANN, M. L. (1985). A Methodology for the Discussion of Contour: Its Application to Schoenberg's Music. **Journal of Music Theory**, vol. 29, no. 2: pp. 223-248.

[5] LEWIN, D. (2007). **Generalized musical intervals and transformations.** Oxford University Press.

[6] MARVIN, E. W. (1988). **A generalized theory of musical contour: its application to melodic and rhythmic analysis of non-tonal music and its perceptual and pedagogical implications.** Phd Dissertation, University of Rochester.

[7] MARVIN, E. W., and LAPRADE, P. A. (1987). Relating musical contours: Extensions of a Theory for Contour. **Journal of Music Theory**, vol. 31, no. 2: pp. 225-267.

[8] MORAES, T., and SAMPAIO, M. S. (2015). Relações de contornos entre elementos sonoros e visuais do jogo Super Mario Bros. In: **Proceedings of SBGame 2015**, pp. 714-717.

[9] MORRIS, R. D. (1987). **Composition with pitch-classes: A theory of compositional design.** Yale University Press.

[10] MORRIS, R. D. (1993). New Directions in the Theory and Analysis of Musical Contour. **Music Theory Spectrum**, vol. 15, no. 2: pp. 205-228.

[11] SAMPAIO, M. S. (2012). **A Teoria de Relações de Contornos Musicais: inconsistências, soluções e ferramentas.** Tese de Doutorado, Universidade Federal da Bahia.

[12] SCHULTZ, R. D. (2008). Melodic Contour and Nonretrogradable Structure in the Birdsong of Olivier Messiaen. **Music Theory Spectrum**, vol. 30, no. 1: pp. 89-137.

[13] SCHULTZ, R. D. (2009). **A diachronic-transformational theory of musical contour relations.** Phd Dissertation, University of Washington.