./chunking -m crhythm 5 4

Outputs a Christoffel word C(m,n), its ancestors in the Stern-Brocot tree, and various transformations

Example:

./chunking -m crhythm 5 4

ancestors of 5/4 are:

L: 4/3

R: 1/1

ancestors of 4/3 are:

L: 3/2

R: 1/1

ancestors of 3/2 are:

L: 2/1

R: 1/1

ancestors of 2/1 are:

L: 1/0

R: 1/1

---

length: 9, slope: 5/4, Christoffel word: abababab

abababab

1,0,1,0,1,0,1,0,0,

0,1,0,1,0,1,0,1,1,

1,1,1,1,1,1,1,1,0,

0,0,0,0,0,0,0,0,1,

BWT-word: bbbbbaaaa

Etc.

It ouputs a text file crhythm_m_n.txt with rhythm shorthand notation. Each line in this file represents a k-mer of the inverse BWT.

./chunking -m lookup <shorthand pattern>

Searches all Christoffel words, from C(1,1) to C(20,20), for a matching rhythmic pattern. The search includes bit transformations of the input pattern, and the inverse BWT matrix of all Christoffel words.

Example:

/chunking -m lookup 'II>'

Shorthand pattern to analyse: II>

1010101 7

1010101 is at pos. #4 in iBWT block #7 of C(3,4)

1010101 is at pos. #1 in iBWT block #7 of C(4,4)

./chunking -m anaphrases <input file>

Input: a text file containing rhythm patterns line by line in shorthand notation

Output: generates BWT and iBWTs of input pattern line by line; output as lilypond file (*.ly) and as shorthand notation in ascii text format (*.txt)

./chunking -m anaphrases text/a1.txt

Analyses the shorthand phrases in a1.txt. Each line corresponds to one phrase.

BWT and inverse BWT (iBWT) matrix are calculated for each phrase.

The iBWT matrix contains all cyclic rotations of all substrings of a phrase. Duplicate substrings are removed.

For each phrase, the phrase itself and the list of all its iBWT substrings (k-mers) are concatenated into a single score in lilypond format.

./lilypond.sh phrase_iBWT_0.ly

executes lilypond on the first output file generated by anaphrases to produce a pdf file.

./preview.sh phrase_iBWT_0.pdf

launches the pdf viewer to look at the printed score rendered by lilypond.

./chunking -m anaphrases <input file> <min beats> <max beats>

Optional input: min beats, max beats are positive integers denoting the minimum and maximum (exclusive) number of beats printed in the result.

./chunking -m anaphrases text/a1.txt 3 4

The two optional integer arguments denote the minimum of onsets in every substring, and the maximum of onsets (non inclusive). The example above will print all substrings for every phrase that have three onsets only.

./chunking -m printphrases text/a2.txt text/one.txt

printphrases reads in a text file containing lines of phrases in shorthand notation. The output is a single lilypond score file where each phrase is a single bar. In addition, a second text file is used to provide a list of pitches on each line corresponding to the rhythm phrase on the same line in the first text file. The pitches are provided in MIDI note number format separated by commas. If the number of pitches is smaller than the number of onsets in the rhythm, then the list of pitches is repeated in a loop. Any surplus of pitches is ignored. To transcribe the output into a score, use:

./lilypond.sh print_phrase.ly

./preview.sh print_phrase.pdf

./chunking -m sentence 60 2

The sentence algorithm takes two integers as arguments. The first one represents the number of n equidistant pulses that measure the total length of the musical sentence. The second argument, k, is the maximum number of parts that will structure the length of pulses on the next lower level. For example, if k=2, then the sentence will be divided into two phrases of unequal lengths. These phrases in turn will be divided each into two patterns of unequal length. Finally, on the level of patterns the algorithm will pick a musical form out of a catalog with seven categories that matches the length of the pattern. The musical forms are resistor, release, arch, catenary, alternating, growth, and decline. These forms are ordered sequences of 2s and 3s. Each one of the resulting sequences will be transcribed into rhythm shorthand notation (RSN). The 2s and 3s represent metric groups, or chunks. The chunks are transcribed by randomly picking form all possible onset patterns within 2 or 3 pulses. The ouputs are:

1. A lilypond input script that can be rendered by lilypond into a pdf score:

./lilypond.sh sentence_60_2.ly

./preview.sh sentence_60_2.pdf

2. The output in shorthand notation, with one line per phrase for further editing and printing via *printphrases*

3. The output of a csound score that is used to call instrument events in csound with a series of parameters and envelopes.

The ranges of the input arguments, number of pulses *n*, and number of parts *k,* are *0 < n <= 120,* and *1 < k < 6*, respectively.

./chunking -m bwt 'IXIIX'

bwt performs the Burrows-Wheeler Transform (BWT) on the standard input string and calculates the matrix of the inverse BWT (iBWT), which produces all cyclic substring rotations.

It automatically creates a text file *bwt_phrases.txt*, where each line represents one of the substrings generated by the iBWT. If the input string is a rhythm shorthand notation (RSN), then the output file can be used together with *printphrases* in order to generate a lilypond script and a csound score. The command line

./chunking -m printphrases bwt_phrases.txt text/one.txt will automatically generate the files print_phrases.ly and print_phrases.sco

bwt also works with any other ascii string. The file *bwt_phrases.txt* is always generated, but it might not necessarily be a useful input for *printphrases* anymore.

The classic BWT example is ./chunking –m bwt 'banana$'

./chunking -m bwtmel 'deafgaa' 0

bwtmel creates a matrix of cyclic substrings based on the inverse BWT process. It takes at its input a string of ascii symbols as note names, a transposition in semitones (0 = no transposition, 12 = octave up, -7 = fifth down), and a character denoting one of three possible transformations of the input string before the BWT is performed: u for inversion (Inversion of all intervals between the original pitches; *German:* Umkehrung), k for retrograde (Going backwards starting with the last pitch; *German:* Krebs), and q for the retrograde of the inversion (KU). If no character is given, then the input string is kept as it is (except for transposition). *bwtmel* automatically creates two text files: *bwt_ones.txt* and *bwt_melody.txt.* The latter contains the matrix of cyclic substrings encoded as comma-separated MIDI note values, with each line containing one element of each k-mer. The rhythms in *bwt_ones.txt* are a sequence of quarter notes. The two files can be used in order to print a lilypond score and to output a csound score with *printphrases,* for example:

./chunking -m printphrases bwt_ones.txt bwt_melody.txt

Example of zero transposition and retrograde pitch transform:

./chunking -m bwtmel 'deafga' 0 k

Example of zero transposition and inversion of pitches:

./chunking -m bwtmel 'deafga' 0 u

Example of a transposition upward by a fifth and subsequent retrograde of inversion:

./chunking -m bwtmel 'deafga' 7 q

Example of transposing the pitches an octave down and retrograde of inversion:

./chunking -m bwtmel 'deafga' -12 q

The following list gives the ascii characters used by *bwtmel* to input all chromatic pitches between C3 and B4 (MIDI note 48 to 71): {C, 1 ,D ,2 ,E ,F ,3 ,G ,4 ,A ,5 ,B ,c ,6 ,d ,7 ,e ,f ,8 ,g ,9 ,a ,0 ,b }. *bwtmel* recognizes all notes within the range of the grand piano (A0 to C7). In order to convert from a list of note names (A0 to C7) to the program's ascii codes, one can use the command line:

./chunking –m notnames <string of notenames>

Example:

./chunking -m notenames 'C3 Db3 D3 B3'

(Todo: One can also translate a file containing comma-separated MIDI notes to a file containing the corresponding ascii codes recognized by *bwtmel* using the following command)

./chunking -m partition <n> <parts-must-be-prime> <parts-not-'1'> <max prime in parts> <max int in parts> < min int in parts> <int to add to all parts> <number of distinct parts, 0 := print all partitions> <flag for p. with all parts being equal>

Example:

./chunking -m partition 90 0 1 101 90 1 0 4

This produces partitions of 90 into 4 distinct parts, all parts > 1

New:

./chunking -m partition 12 0 1 101 90 1 0 0 1

12 = 12, mean: 12 sigma: 0 Coprime pairs. Coprime sequence!

12 = 6, 6, mean: 6 sigma: 0 Coprime pairs. Coprime sequence!

12 = 4, 4, 4, mean: 4 sigma: 0 Coprime pairs. Coprime sequence!

12 = 3, 3, 3, 3, mean: 3 sigma: 0 Coprime pairs. Coprime sequence!

12 = 2, 2, 2, 2, 2, 2, mean: 2 sigma: 0 Coprime pairs. Coprime sequence!

./chunking -m propseries <chunk to divide iteratively> <number of iterations> <divisor>

Output: series of progressive subdivisions of a number (similar to reading the Fibonacci series backwards) and the differences between them. The divisor can be any positive real > 0. For golden sections one could use 1.618 as a divisor, for example.

./chunking -m intstrings <input file1> <input file2>

Input format: ascii text with lines of comma-separated rhythms in integer distance notation

Outputs various distance measurements between all pairs between file1 and file2

Example:

./chunking -m intstrings integer_rhythm1.txt integer_rhythm2.txt

./chunking -m farey2binary <Farey Sequence n> <Digestibility threshold f> <int flag activates smooth filter 2 3>

Example:

./chunking -m farey2binary 7 4 0

Filtered Farey Sequence 7 according to max. Digestibility value 4 :

0/1 1/6 1/4 1/3 1/2 2/3 3/4 5/6 1/1

Filtered Farey Sequence with common denominator:

2/12 1/12 1/12 2/12 2/12 1/12 1/12 2/12

in integer distance notation:

2 1 1 2 2 1 1 2

Note: The binary output is deactivated

./chunking -m divisors

Outputs a list of divisors for integers 1 - 499

./chunking -m divisors

...

Divisors of 4: 2

Divisors of 5:

Divisors of 6: 3 2

Divisors of 7:

Divisors of 8: 4 2

Divisors of 9: 3

Divisors of 10: 5 2

Divisors of 11:

Divisors of 12: 6 4 3 2

...

./chunking -m permutations

Outputs the Christoffel words and BWTs of ratios built by the first twenty-one 7-smooth numbers.

./chunking -m permutations2

Outputs filtered Farey Sequences. The filter is based on subdivisions taken from pairs of 7-smooth numbers. Transcriptions into integer sequences and into binary patterns are included.

Example:

F_10 filtered by subdivisions:

2 5

Filtered F_10 in integer distance notation:

2 2 1 1 2 2

and translated into binary rhythm:

1 0 1 0 1 1 1 0 1 0

1 0 5 0 5 2 5 0 5 0

./chunking -m fpoly <Farey Seq. N> <first subdivision> <second subdivision>

Example:

./chunking -m fpoly 20 5 4

F_20 filtered by subdivisions:

5 4

Filtered F_20 in integer distance notation:

4 1 3 2 2 3 1 4

and translated into binary rhythm:

1 0 0 0 1 1 0 0 1 0 1 0 1 0 0 1 1 0 0 0

1 0 0 0 5 4 0 0 5 0 2 0 5 0 0 4 5 0 0 0

./chunking -m printfarey <Farey Sequence n>

Prints all members of Farey Sequence n, with each ratio also interpreted as slope of Christoffel word, including bit pattern operations, BWT, and shorthand notations

Example:

./chunking -m printfarey 5

F(5) contains 1/2:

3 : 1/2 : aab

aab

1,1,0,

0,0,1,

1,0,1,

0,1,0,

BWT-word: baa

.

.

(.)

1

1

1

Block #1

a,a,b,

:

I

v

2

11

11

Block #2

aa,ab,ba,

X

>

+

21

111

12

Block #3

aab,aba,baa,

X

Tests:

./chunking -m propseries 89 8 1.618

./chunking -m anaphrases text/testin.txt

./chunking -m partition 30 0 1 91 9 2 0 0 0

./chunking -m intstrings text/integer_rhythm1.txt text/integer_rhythm2.txt

./chunking -m farey2binary 7 4 0

./chunking -m divisors

./chunking -m permutations

./chunking -m permutations2

./chunking -m fpoly 20 5 4

./chunking -m printfarey 5

./chunking -m crhythm 5 4

./chunking -m lookup 'I-'

./chunking -m sentence 30 2

./chunking -m anaphrases text/a1.txt

./chunking -m printphrases text/a2.txt text/mel.txt