DMUN at the MediaEval 2015 C@merata Task: a description of the Stravinsqi Algorithm

Andreas Katsiavalos and Tom Collins
Faculty of Technology
De Montfort University
Leicester, UK
+44 116 207 6192
tom.collins@dmu.ac.uk

ABSTRACT

This paper describes the Stravinsqi-Jun2015 algorithm, and evaluates its performance on the MediaEval 2015 C@merata task. Stravinsqi stands for STaff Representation Analysed VIa Natural language String Query Input. The algorithm parses a query string that consists of a natural language expression concerning a symbolically represented piece of music (which the algorithm parses also), and then identifies where in the music event(s) specified by the query occur. For a given query, the output is a list of time windows specifying the locations of the relevant events. Time windows output by the algorithm can be compared with time windows specified by music experts for the same query-piece combinations. Across a collection of twenty pieces and 200 questions, Stravinsqi-Jun2015 had recall .794 and precision .316 at the measure level, and recall .739 and precision .294 at the beat level. The paper undertakes a preliminary analysis of where Stravinsqi might be improved, identifies applications of the C@merata task within the contexts of music education and music listening more generally, and provides a constructive critique of some of the question categories that are new this year.

1. INTRODUCTION

The premise of the C@merata task [1] is that it is interesting and worthwhile to develop algorithms that can (1) parse a natural language query about a notated piece of music, and (2) retrieve relevant time windows from the piece where events/concepts mentioned in the query occur. The premise is strong, if we consider that each year in the U.S. alone over 200,000 freshman students declare music their intended major [2, 3], and that there is a line connecting the types of queries being set in the C@merata task and the questions these students are taught (or, by college, have already been taught) to answer. A typical question from a U.K. music appraisal exam sat at age eighteen reads "You will hear a short excerpt of music, played three times.

1. "What is the tonality of the excerpt? [E.g., major/minor]
2. "Which one of hemiola, ostinato, pedal, sequence is used in the instrumental introduction?
3. "Name the cadence at the end of the instrumental introduction. [I.e. perfect, imperfect, plagal, or interrupted]
4. "Name the interval sung to the word floating in line 6. [Answer e.g., perfect fifth]
5. "How many different chords are used in the chorus?" [Answer e.g., if the chorus chords were I, IV, I, V repeated, then three different chords are used] [4]

Appraisal of music from the Western classical tradition – that is, gaining explicit knowledge of the inner workings of pieces and their historical context – is a core part of music syllabi across Europe, North America, and Asia. The C@merata task, apart from posing an interesting research problem at the intersection of music theory, music psychology, music computing, and natural language processing (NLP), could lead to new applications that assist students, and music lovers more generally, in gaining music appraisal skills.

Other applications of research motivated by the C@merata task include sustained and enriched support of work in musicology [5], and informing solutions to various music informatics tasks, such as generation of music in an intended style [6] or expressive rendering of staff notation [7], where systems for either task may benefit from being able to automatically extract, say, cadence locations and/or changes in texture.

This paper is structured as follows. An overview of the Stravinsqi-Jun2015 algorithm is given in section 2, as well as a description of its being run on the 2015 C@merata task questions. Section 3 presents the results of this run and an analysis of where and why Stravinsqi's output and the ground truth diverge. A discussion of question categories that we found challenging or problematic appears in Section 4, followed by some concluding observations.

2. APPROACH

2.1 Overview

The Stravinsqi-Jun2015 algorithm (hereafter, Stravinsqi), which was entered into the C@merata task, is part of a Common Lisp package called MCStylistic-Jun2015 that has been under development since 2008 [8]. The MCStylistic package, free and cross-platform, supports research into music theory, music cognition, and stylistic composition, with new versions released on an approximately annual basis.[1] In addition to Stravinsqi, MCStylistic includes implementations of other algorithms from the fields of music information retrieval and music psychology, for tonal and metric analysis [e.g., 9], and for the discovery of repeated patterns (e.g., motifs, themes, sequences) [10].

A flow diagram outlining the Stravinsqi algorithm is given in Figure 1. The following aims to be a self-contained overview of Stravinsqi, while focusing on the differences between this year's (Stravinsqi-Jun2015) and last year's submission (Stravinsqi-Jun2014) [11]. As indicated in Figure 1, step 1 of Stravinsqi involves extracting the question string and divisions value from the question file. Step 2 parses the question string for mention of bar restrictions, such as "minim in measures 6-10" or "D# in bar 8", storing this information in a pair, i.e. (6, 10) or (8, 8) respectively, which will be used later to restrict imported music representations to certain time intervals. An edited version of the question string absent the restriction, such as "minim" or "D♯", is

MediaEval 2015 Workshop, September 14-15, 2015, Wurzen, Germany.

[1] http://www.tomcollinsresearch.net

passed on for subsequent NLP. If no such restriction appears in the question, then the bar restriction pair is set to `nil` (empty) and the question string is unaltered.

Steps 3 and 4 are where the NLP proper begins. One of the questions from the task description, "two quavers against a minim in the bass" [12, p. 9], prompted consideration of how synchronous ("against") and asynchronous ("followed by") commands might be combined and interpreted, since "two quavers" would be converted by Stravinsqi to "quaver followed by quaver", and so result in the synchronous/asynchronous combined query of "quaver followed by quaver against a minim in the bass". Putting the "in the bass" suffix to one side, there are two ways that this query might be interpreted:

1. (quaver followed by quaver) against (a minim);
2. (quaver) followed by (quaver against a minim).

These differing interpretations may correspond to different types of musical events. In the first instance, both quavers must sound at the same time as the minim. In the second instance, only the second quaver needs to sound against the minim. Thinking of it another way, (1) implies (2), but (2) does not imply (1). Since there was an implicit followed/against combined question of type (1) in the task description but not of type (2), Stravinsqi splits queries by synchronous commands first (step 3) and then further by asynchronous commands (step 4). The string "two quavers against a minim" would emerge as ("two quavers" "minim") from step 3 in Figure 1, and as (("quaver" "quaver") ("minim")) from step 4. As another example, "D followed by A against F followed by F" would emerge as ("D followed by A" "F followed by F") from step 3, and as (("D" "A") ("F" "F")) from step 4. An instance of this query occurs in Figure 2, b.1. In a use-case scenario, rather than assuming this synchronous/asynchronous splitting order, it may be preferable to warn a user that their synchronous/asynchronous query is ambiguous, and to ask which of the alternatives they mean.

Generalising from examples above such as "D followed by A against F followed by F", which is parsed as (("D" "A") ("F" "F")), a question string emerges from step 4 as some nested list of strings $((s_{1,1}\ s_{1,2}\ \ldots\ s_{1,n(1)})\ (s_{2,1}\ s_{2,2}\ \ldots\ s_{1,n(2)})\ \ldots\ (s_{m,1}\ s_{m,2}\ \ldots\ s_{m,n(m)}))$, where each $s_{i,j}$ is a query element. Examples of query elements include "D", "A♭4 eighth note", "perfect fifth", "melodic interval of a 2nd", etc.

In step 5, point-set representations of the relevant piece are loaded. The `xml2hum` script is used to convert each piece from its MusicXML format to kern format [13].[2] Parser functions in MCStylistic import the kern file into the Lisp environment, giving the following representations, known as *point sets*:

- Instrument/staff and clef names at the beginning of each staff;
- Time signature specifications (including any changes), expressed by the bar number where the time signature is specified or changes, together with the number of beats per bar, the type of beat, and the corresponding ontime. Ontime is the incrementing time in staff notation measured in crotchet beats, with 0 for bar 1 beat 1. A note beginning at measure 4 beat 1 in a piece with four crotchets per bar has ontime 12, for example;
- The notes of the piece, each represented by a five-dimensional point consisting of the ontime of the note, its MIDI note number (MNN), its morphetic pitch number (MPN) [10], its duration in crotchet beats, and its numeric staff number (an integer beginning with zero for the top staff). MPN is a numeric encoding of staff height, with 60 for C♭4, C♮4, C♯4, 61 for D♭4, D♮4, D♯4, etc. Morphetic pitch helps to disambiguate MNNs that refer to one of several pitches depending on the tonal context (e.g., does 61 refer to C♯4 or D♭4?). Setting MNN = 61, if MPN = 60 then the pitch must be C♯4, whereas if MPN = 61 the pitch must be D♭4. As was discussed in more detail previously [11], MPN is also useful for identifying intervals absent diminished, minor, perfect etc. qualifiers: a morphetic pitch difference of 1 always encodes some type of 2nd, a difference of 2 always encodes some type of 3rd, and so on;
- A point-set representation of the piece with three extra dimensions, one each for articulation, dynamics, and lyrics information. If, for example, a note is marked staccato, then the dynamics entry of the point will contain ".";
- The rests of the piece, each represented by a three-dimensional point consisting of the ontime of the rest, its duration, and staff number;
- The notes of the piece that are involved in ties, and the nature of those ties (e.g., tied from the previous note, tied to the following note, or both), because this information is not included in the above-mentioned note point set, but some questions refer specifically to tied notes;

[2] http://extras.humdrum.org/bin/osxintel64/

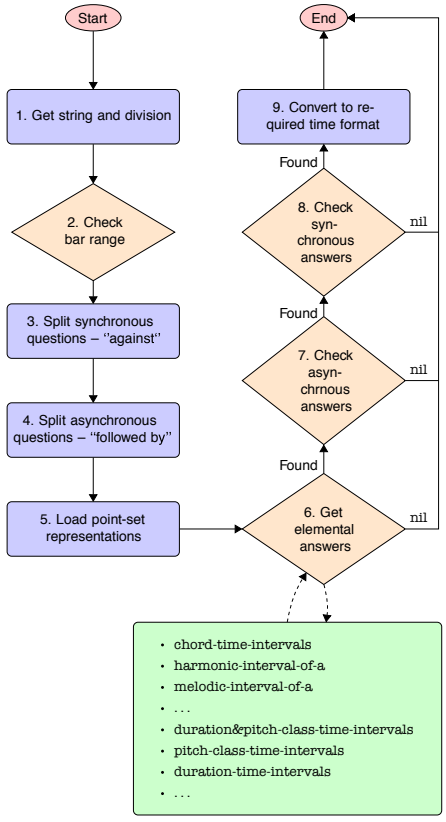

Figure 1. Flow diagram for the Stravinsqi-Jun2015 algorithm.

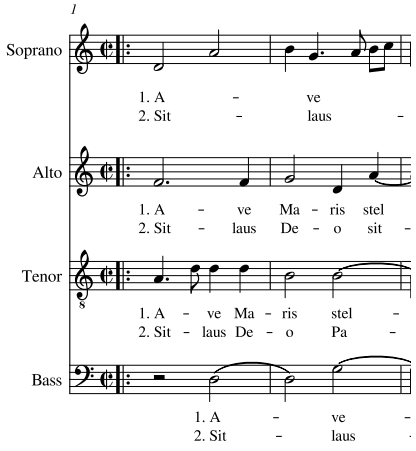

Figure 2. Bars 1-2 of 'Ave Maris Stella' from *Vespro della Beata Vergine* (1610) by Claudio Monteverdi (1567-1643).

- A texture point set, where each element consists of the time window where a texture label applies (one of "monophonic", "homophonic", "melody with accompaniment", and "polyphonic"), the texture label itself, and the confidence with which this label applies (value in [0, 1]).[3]

After the point sets are loaded, they are restricted to those points that belong to a certain bar-number range, if a range was extracted from the question string. MCStylistic utility functions `ontime-of-bar&beat-number` and `bar&beat-number-of-ontime`, as well the time signature specifications, make calculating these restrictions relatively straightforward.

Temporarily, in step 6 of Figure 1, each question element $s_{i,j}$ from step 4 is treated independently. A query element $s_{i,j}$ is passed to seventeen music-analytic sub-functions, each of which tests whether $s_{i,j}$ is a relevant query for that function, and, if so, searches for instances of the query in the piece/excerpt of music. If the query is *irrelevant* to a sub-function, that function returns `nil`. If the query is relevant but no instances of it are detected, the function returns either `nil` or a message string to the effect "relevant query but no instances detected". This message string prevents certain false-positive results begin returned (see below for an example).

One of the new music-analytic sub-functions this year is called `chord-time-intervals`. If the query element does not contain the word "chord", this function will return `nil`. If, on the other hand, the query element is something like "D major chord", "chord E G B", or "minim chord", then `chord-time-intervals` undertakes further analysis to determine which pitches and/or durations are to be sought in simultaneities. Separate functions from the set of seventeen music-analytic sub-functions (`triad-time-intervals` and `triad-inversion-time-intervals`) handle query elements such as "V7" or "tonic triad in first inversion".

Output from each of the music-analytic sub-functions (which can be time intervals, `nil`, or a message string) is stored in a list, with output from more specific functions appearing higher up in that list than output from more generic functions. For instance, three music-analytic sub-functions are `duration&pitch-class-time-intervals`, `pitch-class-time-intervals`, and `duration-time-intervals`, with the first of these being more specific than the second or third. For an element such as "dotted crotchet G" querying the excerpt in Figure 2, `duration&pitch-class-time-intervals` will return the time interval [5, 6.5), corresponding to the dotted-crotchet G in the Soprano in b.2. The function `duration-time-intervals` will return this time interval, as well as the time interval [0, 1.5), corresponding to the dotted crotchet in the Tenor in b.1. The function `duration-time-intervals` returns this extra time interval because it does not *know* that a particular pitch class is sought. Likewise, the function pitch-class-time-intervals returns [5, 6.5), as well as [4, 6) (G in Alto, b.2) and [6, 8) (G in Bass, b.2).

For a query element $s_{i,j}$, the set of time intervals $T_{i,j}$ belonging to the first (most specific) function is selected and passed on to subsequent steps – the set {[5, 6.5)} in our example. Hence, false-positive results from more generic functions (e.g., [0, 1.5), [4, 6), [6, 8) in our example) are prevented from being returned. Message strings along the lines of "relevant query but no instances detected" help to prevent another kind of false-positive result, when the excerpt in Figure 2 is queried with elements such as "B dotted crotchet". There is no dotted-crotchet B in this excerpt, so `duration&pitch-class-time-intervals` ought on first glance to return `nil`. There are, however, separate instances of the pitch-class B in Figure 2, as well as dotted crotchets, meaning both `pitch-class-time-intervals` and `duration-time-intervals` will return false-positive results. To prevent these false-positives propagating to subsequent steps (and eventually harming the algorithm's precision), the function `duration&pitch-class-time-intervals` outputs a message string rather than `nil`. The message indicates that the query was relevant to the function but that no instances were detected. The presence of a message string in more specific functions (`duration&pitch-class-time-intervals`) is used as a predicate for nullifying (removing) time intervals output by related, more generic functions (`pitch-class-time-intervals` and `duration-time-intervals`).

The output of step 6 is a nested list of time-interval sets, $((T_{1,1}\ T_{1,2}\ \ldots\ T_{1,n(1)})\ (T_{2,1}\ T_{2,2}\ \ldots\ T_{1,n(2)})\ \ldots\ (T_{m,1}\ T_{m,2}\ \ldots\ T_{m,n(m)}))$, one for each query element $s_{i,j}$, some of which may be empty. The purpose of steps 7 and 8 is to determine whether any combination(s) of these time intervals satisfy the constraints imposed by various synchronous and asynchronous parts of the question string. Stravinsqi will allow any plausible sequence of time intervals $\boldsymbol{\tau}_1 \in T_{k,1}$, $\boldsymbol{\tau}_2 \in T_{k,2},\ldots$, $\boldsymbol{\tau}_{n(k)} \in T_{k,n(k)}$ to be passed on for further processing. For time-interval sets $T_{k,1}\ T_{k,2}\ \ldots\ T_{1,n(k)}$, there may be $l(k)$ such plausible sequences. Two time intervals $[a, b]$ and $[c, d]$ are considered *plausible* if the original query contains "followed by" and they are adjacent ($b = c$), or if the original query contains, say, "followed three bars later by" and $c - b$ is the appropriate distance apart. A plausible sequence of time intervals $\boldsymbol{\tau}_1 = [a_1, b_1]$, $\boldsymbol{\tau}_2 = [a_2, b_2],\ldots$, $\boldsymbol{\tau}_{n(k)} = [a_{n(k)}, b_{n(k)}]$ is merged into one time window $U = [a_1, b_{n(k)}]$ and passed to step 8.

The input to step 8 is a nested list of merged time intervals $((U_{1,1}\ U_{1,2}\ \ldots\ U_{1,l(1)})\ (U_{2,1}\ U_{2,2}\ \ldots\ U_{1,l(2)})\ \ldots\ (U_{m,1}\ U_{m,2}\ \ldots\ U_{m,l(m)}))$, such that $U_{i,j}$ is the jth merged time interval for the ith part of a question string containing m synchronous parts. A useful way to think of the time intervals $U_{i,j}$ is arranged as a jagged two-dimensional array (table) with m rows and $l(i)$ columns in row i. If, visiting each row precisely once, there exists some column

[3] For further discussion of texture identification, as well as the identification of some other higher-level music-theoretic concepts, please see subsections 2.3 and 2.4 of [11].

index $j(i)$ such that the intersection of the corresponding intervals $U_{1,j(1)}$, $U_{2,j(2)}$,…, $U_{m,j(m)}$ is nonempty, then $v = \bigcap_{i=1,2,\ldots,m} U_{i,j(i)}$ is a time interval output by step 8. There may be none, one, or several such distinct time intervals, v_1, v_2,…, v_r.

The "Found" and "`nil`" labels attached to arrows emanating from steps 6, 7, and 8 in Figure 1 indicate how the presence or absence of time intervals emerging from these steps can propagate to a final output time interval. Supposing, for instance, that the question string is "D followed by A against F followed by F" but that no pitch-class D appears in the piece. Then there is `nil` for at least one of the sets output by step 6, which means there is `nil` for at least one of the time intervals output by step 7, which means that a nonempty intersection of time intervals does not exist in step 8, and so no answer passages are returned.

The final step of Stravinsqi, labeled Step 9 in Figure 1, comprises the conversion of the time intervals v_1, v_2,…, v_r into the XML format required by the task. Again, the MCStylistic function `bar&beat-number-of-ontime` is helpful here.

2.2 Procedure

In the overview paper for the task [1], Stravinsqi is labelled DMUN03. The differences between this run and the other submitted runs (DMUN01 and DMUN02) are not remarkable – suffice it to say we manipulated the use of the divisions value for answering synchronous questions, to investigate its effect on precision. As the `xml2kern` script is becoming increasingly out of date, the majority of the test week was spent converting the MusicXML files to kern files, if necessary re-encoding the files to correct sources of errors. The test data was opened in order to check for any knock-on effects of these error corrections, but no post-check alterations were made to the Stravinsqi algorithm. Naturally, in the last two weeks we have looked through the test questions in more detail, so as to make relevant comments here.

3. RESULTS

The evaluation is based on four metrics: measure recall (U.K., bar recall), measure precision, beat recall, and beat precision. If, for a given question, an algorithm returns k of the m correct beginning and ending bar number pairs at which the queried event occurs, then its measure recall for this question is k/m. If it returns n incorrect beginning/ending bar number pairs also, then its measure precision is $k/(m + n)$. Beat recall and precision are defined similarly, where the algorithm must be correct not only up to bar numbers, but up to the exact beat numbers as well.

Figure 3 contains a summary of results for the Stravinsqi algorithm across various question categories. The mean measure recall across all 200 questions, indicated by the black line next to the "Mean" label, is .794, and the mean measure precision, indicated by the blue line, is .316. The mean beat recall (green line) and beat precision (red line) are both slightly lower than their measure counterparts (.739 and .294 respectively), but in general it can be assumed that if Stravinsqi returned the correct beginning/ending measure number pairs for a question, then it was also able to identify the relevant beats. Stravinsqi had the highest measure and beat recall of any algorithm submitted to the 2015 C@merata task [1] and the third highest measure and beat F_1 score ($F_1 = 2PR/(P + R)$, where P is precision and R is recall).

Across eight of the eleven question categories shown in Figure 3 (Melody 1, Melody n, Harmony, Articulation, Instrument, Clef, Follow, and Synch), Stravinsqi achieves consistently high recall of approximately .75. For the remaining three categories (Time Sig., Key Sig., and Texture) it is less successful. Overall, the results suggest the need to investigate

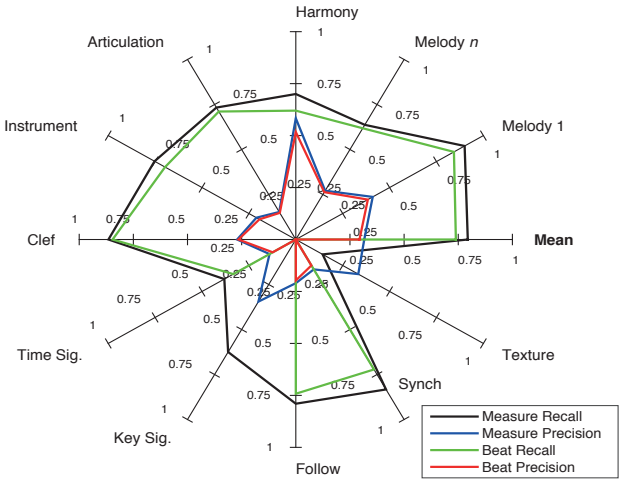

Figure 3. Results of the Stravinsqi-Jun2015 algorithm on the MediaEval 2015 C@merata task. Overall results are indicated by the mean label, and followed by results for eleven question categories.

Stravinsqi's precision being lower than its recall, so we will address and exemplify some issues now.

We have not yet incorporated in Stravinsqi restrictions to notes occurring after particular clef, time signature, or key signature changes. A query such as "G4 in the key of G major", which does not occur in Figure 2, say, would be parsed instead as "G4", which does occur in Figure 2. Sometimes, therefore, the recall of Stravinsqi remains high for such questions, but the precision will be negatively impacted.

The design of Stravinsqi is motivated more by music-perceptual than typographical concerns, based on the premise that music is primarily an auditory-cognitive phenomenon, and a visuo-cognitive phenomenon secondarily. When music perception and music theory collide, as they do occasionally in the C@merata task and beyond [14], Stravinsqi's precision can be adversely affected. According to the task description, consecutive elements "must both be on the same stave" [12, p. 7]. So if we were to query the excerpt in Figure 2 with the question "D followed by A", then the only valid answer would correspond to the D followed by A in b.1 of the Soprano. If the D appeared in some other staff (a swap with the Alto F, say), the answer would no longer be valid according to the task description, due to the typographical alteration. Even so, the altered and original excerpts may sound indistinguishable. Stravinsqi would still return an answer for this altered version, however, since being motivated by music-perceptual concerns (i.e., how the music sounds), it does not require consecutive question elements to be on the same staff. It would return a second answer also corresponding to the quaver D of the Tenor followed by the minim A of the Soprano in b.1. When music perception and music theory collide in this fashion, Stravinsqi tends to find the correct answers according to the task description, but also some extra answers that involve elements on different staves, which has a detrimental effect on its precision.

Beyond precision issues, it is possible to use Stravinsqi's performance on specific questions to make some inferences about potential inconsistencies between the task description/training collection, and test collection. Chord identification (e.g., "chord of E G B" as separate from Roman-numeral analysis) was new this year, falling under the harmony category. For Question 58, "chord of D minor in measures 109-110", for piece 6, the second movement from String Quartet in F major op.18 no.1 by Ludwig

van Beethoven (1770-1827), Stravinsqi scored maximally on all metrics. It was necessary for the chord identification algorithm to look across all four parts (Violin I, Violin II, Viola, and Cello) to retrieve notes fitting the requested D-minor chord template, and there were two such chords in b.110. For Question 127, "quarter note chord D3 A3 C4", for piece 13, the first movement from Piano Sonata in C major K545 by Wolfgang Amadeus Mozart (1756-1791), Stravinsqi scored zero on all metrics. There is a quarter-note chord of D3, A3, C4 in the left hand on the final beat of b.27, but, as with Question 58, Stravinsqi's chord identification algorithm looked across all parts (i.e., the right hand also) because the question did not specify "in the left hand" or "in the bass clef". In the right hand against the quarter-note chord are four semiquavers, C5, A4, F♯4, A4, which means that across all parts, there is no quarter-note chord but rather four chords each lasting a semiquaver, where some notes are held over from/to previous/succeeding simultaneities: (1) D3, A3, C4, C5; (2) D3, A3, C4, A4; (3) D3, A3, C4, F♯4; (4) same as (2). The task description stated, "no other notes must be sounding" apart from those pitch classes specified in the chord, but it was not clear whether this applied to pitch and octave specifications also.

4. DISCUSSION AND CONCLUSIONS

As per last year, the C@merata task description [12] was helpful and the training/test data provided a springboard for interesting research, reflecting a considerable amount of time and effort on the part of the organisers. The following comments are not intended as criticisms of the task, but as ideas for guiding the task in future years, to ensure it remains relevant as a basis for cutting-edge research at the intersection of music theory, music psychology, music computing, and NLP.

4.1 Arpeggio and Scale

For the first time this year, the C@merata task included queries relating to arpeggios and scales. These queries, like cadence, triad, and texture questions last year, refer to inherently fuzzy concepts (as opposed to queries such as "crotchet A♭4" or "fermata minim", where answers are either definitely right or definitely wrong). We devote some space to discussing definitions and examples of arpeggios and scales, therefore, since future iterations of the C@merata task might involve additional instances of similarly complex concepts, and these need to be incorporated in a music-theoretically appropriate manner.

The generally accepted definition of *arpeggio* is the "sounding of the notes of a chord in succession rather than simultaneously" [15]. The definition of arpeggio given in the task description is a sequence of at least three notes, all ascending or all descending, and "for simplicity, each note is defined to be separated from the previous one by a third (major or minor) or a fourth (perfect)" [12, p. 4]. The definition includes further, auxiliary remarks about accidentals, which are inherited from a preceding definition of scale. It is not clear whether other parts of the scale definition should be inherited also (e.g., the sequence of notes should "not contain any rests"). There are numerous examples of scales and arpeggios that contain rests: for instance, Figure 4 b.47 contains a C-major arpeggio in the piano left hand, C2, E2, G2, C3, followed by an F-major arpeggio, where notes are separated by rests. Specifying that the sequence of notes in a scale or arpeggio should not contain rests might not lead to an appropriate definition in general.

The most significant way in which the accepted and C@merata definitions of arpeggio diverge is in the former's mention of "notes of a chord" and the latter's reliance on melodic intervals of a third or fourth. One realisation of a C7 arpeggio is

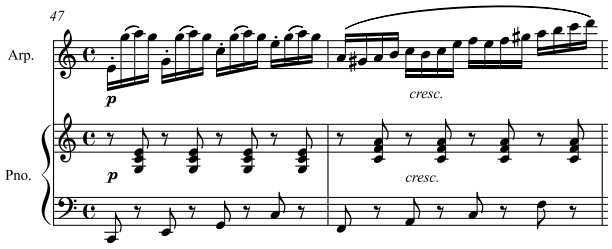

Figure 4. Bars 47-48 of the first movement from Arpeggione Sonata in A minor D821 by Franz Schubert (1797-1828).

C2, E2, G2, B♭3, C3. The interval between the final two notes here is a second, so it does not conform to the C@merata definition, although it *is* an arpeggio. According to the accepted definition, a serviceable arpeggio-identifying algorithm needs to build on a successful chord identification algorithm, rather than trying to identify arpeggios in terms of melodic intervals. Future C@merata questions might aim to reward the development of successful chord and triad identification algorithms, in the presence and absence of non-chord tones, as this would fill a gap in existing research. The current approach, of sidestepping this research gap and defining a complex concept in terms of other properties, might not provide the strongest basis for future research.

The generally accepted definition of *scale* is a sequence of notes, all ascending or all descending, "long enough to define unambiguously a mode, tonality, or some special linear construction, and that begins and ends (where appropriate) on the fundamental note of the tonality or mode" [16]. The definition of scale given in the task description also began by stipulating an all-ascending or all-descending sequence of notes, adding that it must "consist of contiguous notes of the scale and not contain any rests. No note is repeated unless we reach the octave. However, a scale does not need to be a complete octave. For simplicity we will say that each note in the scale will not carry an accidental except from the key signature. This will effectively make all scales either major or natural minor. So C D E F G is a scale (C major) and A B C D is a scale (A minor)" [12, p. 4]. An issue with the C@merata definition is that the scale does not have to begin on the fundamental note, so while the pitch-classes C, D, E, F, G belong to the C major scale, they belong also to A natural minor, F major, and D natural minor. Disqualifying note sequences that contain accidentals might also lead to an inappropriate definition. For example, the opening four pitch classes in Figure 5, C, B♭, A♭, G♮, would be disqualified since G♮ does not appear in the key signature, but otherwise they begin on C and proceed to define C natural minor. According to the accepted definition, a serviceable scale-identifying algorithm needs to build on an algorithm for identifying segments that unambiguously define a mode, tonality, or some special linear construction, rather than trying to identify scales in terms of contiguous, non-accidental notes. Again, future C@merata questions might aim to reward the development of such supporting algorithms, as this would fill a gap in existing research.

To add weight to our observation above that some concepts are inherently fuzzy, the pitch classes of b.351-352.1 in Figure 5 unambiguously define the scale of E♭ major or C natural minor. This ten-pitch-class sequence begins but does not end on C, however, so according to the accepted definition ("begins *and* ends on the fundamental note"), it can be neither E♭ major nor C

natural minor. Furthermore, only seven of the nine intervals descend, with one upward inflection between G♮ and A♭ and one repeated note, which contradicts both definitions' specifications of all ascending or all descending notes. Many listeners would still say that b.351 in Figure 5 sounds *scalic*, however, even if it does not conform to all criteria. Bars 352-353, while also scalic, are more complicated still, with Tchaikovsky delineating a set of nine pitch classes that are not modal, tonal, or linear in construction.

Although we did not include it in this year's submission, we implemented a proof-of-concept algorithm with the aim of identifying both arpeggios and scales. Our motivation was to scrutinise the task definitions of arpeggio and scale more thoroughly, and to lay the groundwork for future C@merata question categories that also comprise more complex musical concepts. One algorithm, with one small alteration, seemed to capture arpeggio and scale concepts, which is perhaps not surprising since both can be considered sequential musical structures. Although their identification might be achieved via the use of composite queries (i.e. "C4 followed by E4 followed by G4" is one realisation of a C-major arpeggio), we formed a hierarchical set of rules according to the task description [12], leading to the definition of a more generic results space.

The proof-of-concept algorithm does not incorporate NLP, but instead searches for note sequences that adhere to specific rules, which at the moment are not available for parameterisation. In addition to the C@merata definitions for these concepts, some additional assumptions were made to simplify the process. During note import, only single notes (i.e. neither rests nor chords) from each staff are taken into consideration. This assumption, that a sequential structure is monophonic, simplifies the identification processes, as otherwise voice segregation procedures have to be considered [17]. The left-hand red notes in bar 17 of Figure 6 illustrate such a case arising from the repetition of a chord, where if the chords were to be segregated into their constituent notes, this would result in many identifiable – but nonetheless perceptually false or very difficult to hear – arpeggios. In addition, the ascending movement of inversions of the same chord in the right hand would result in four different arpeggios (indicated by four different colours in Figure 6), where a more precise answer here indicating only one arpeggio might suffice.

The algorithm, which is implemented in music21 [5], considers each note only once, using a dynamic data structure that we refer to as a *trail list*. Notes that comply with the rules for a sequence type (arpeggio or scale) are aggregated into the trail list. For a given note $\mathbf{n}_i$, and $N = (\mathbf{n}_{i-j}\ \mathbf{n}_{i-j+1} \ldots \mathbf{n}_{i-1})$ an existing trail list, the algorithm goes through the series of rules that build up the sequence type. Failure at any point of this rule series causes the existing trail list N to be output, if j is sufficiently large. The dynamic trail list is emptied, $N' = ()$, since the sequence cannot be continued, and if the current note $\mathbf{n}_i$ that breached a rule is scale-valid then it is added as the first element of the new trail list, $N' = (\mathbf{n}_i)$. The rule series is the same for both arpeggios and scales – the only difference being in the values associated with permissible intervals between adjacent notes. This is a preliminary approach to the identification of sequential structures, which helped us to interrogate some aspects of the arpeggio and scale definitions, and we will aim to incorporate it into the Stravinsqi algorithm for next year.

4.2 Conclusion

This paper has provided an overview of the Stravinsqi-Jun2015 algorithm, and described its performance on the 2015 C@merata task. Stravinsqi achieved high recall (approximately .75) in eight of the eleven question categories, and had the highest

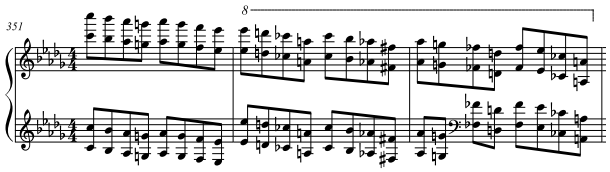

Figure 5. Bars 351-353 of the first movement from Piano Concerto no.1 in B♭ major op.23 by Pyotr Ilyich Tchaikovsky (1840-1893).

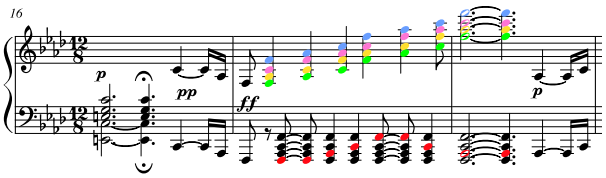

Figure 6. Bars 16-18 of the first movement from Piano Sonata no.23 in F minor op.57 by Beethoven. The highlighted notes in the right hand indicate four arpeggios; in the left hand, one arpeggio among the repeated chord.

measure and beat recall of any algorithm submitted to the task [1]. Its measure and beat F_1 score was third highest overall, inviting investigation of potential issues with the algorithm's precision. Some of these issues were addressed and exemplified in section 3, but further analysis of the results is required to determine whether Stravinqi's precision can be improved while adhering to our general design principle of favouring music-perceptual over typographical concerns. That is, music is heard as well as seen, so if "D followed by A" occurs in terms of sonic events but not in terms of the staff notation (e.g., these pitch classes do follow one another but not in the same staff), then this passage should still be amongst the answers.

In the introduction, it was remarked that there is a line connecting the types of queries being set in the C@merata task and the examination questions that students of the Western classical tradition are taught to answer. Five examples of such examination questions were provided. This year, C@merata queries happened to overlap with part of examination question 2 (pedal) and fully with examination question 4 (lyric plus melodic interval). Unlike last year, however, this year's C@merata test set was lacking cadence and functional harmony queries, which would have overlapped with examination questions 3 and 5 respectively. We perceived a general tendency to replace musically interesting questions (e.g., concerning cadence, triad, hemiola, ostinato, sequence, etc.) with questions that were linguistically challenging to parse but of less musical relevance (e.g., Question 130, "fourteen sixteenth notes against a whole note chord in the bass"). Next year, we would welcome the reintroduction of more musically interesting (if complex) question categories, to reestablish and strengthen the line that connects C@merata queries with concepts that are relevant for music students and enthusiasts.

5. REFERENCES

[1] Sutcliffe, R. F. E., Fox, C., Root, D. L., Hovy, E, and Lewis, R. 2015. The C@merata Task at MediaEval 2015: Natural language queries on classical music scores. In *MediaEval 2015 Workshop, Wurzen, Germany, September 14-15, 2015*. http://ceur-ws.org.

[2] Kena, G, et al. 2015. *The Condition of Education 2015*. U.S. Department of Education, Washington, DC. Retrieved June 4, 2015 from http://nces.ed.gov/pubsearch.

[3] Eagan, K, et al. 2014. *The American Freshman: National Norms Fall 2014*. Higher Education Research Institute, UCLA, Los Angeles, CA. Retrieved March 15, 2015 from http://www.heri.ucla.edu.

[4] AQA. 2012. *Music in Context: Past Paper for A-level Music Unit 4*. AQA, Manchester, UK. Retrieved January 13, 2014 from http://www.aqa.org.uk.

[5] Cuthbert, M. S., and Ariza, C. 2010. music21: a toolkit for computer-aided musicology and symbolic music data. In *Proceedings of the International Symposium on Music Information Retrieval* (Utrecht, The Nethlerands, August 09 - 13, 2010). 637-642.

[6] Collins, T., Laney, R., Willis, A, and Garthwaite, P. H. 2015. Developing and evaluating computational models of musical style. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*. DOI: 10.1017/S0890060414000687

[7] van Herwaarden, S., Grachten, M, and de Haas, W. B. 2014. Predicting expressive dynamics in piano performances using neural networks. In *Proceedings of the International Symposium on Music Information Retrieval* (Taipei, Taiwan, October 27 - 31, 2014). 47-52.

[8] Collins, T. 2011. *Improved Methods for Pattern Discovery in Music, with Applications in Automated Stylistic Composition*. Doctoral Thesis. Faculty of Mathematics, Computing and Technology, The Open University.

[9] Volk, A. 2008. The study of syncopation using inner metric analysis: linking theoretical and experimental analysis of metre in music. *J. New Music Res.* 37, 4, 259-273.

[10] Meredith, D., Lemström, K., and Wiggins, G. A. 2002. Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music. *J. New Music Res.* 31, 4, 321-345.

[11] Collins, T. 2014. Stravinsqi/De Monfort University at the C@merata 2014 task. *Proceedings of the C@merata Task at MediaEval 2014*.

[12] Sutcliffe, R., Fox, C., Root, D. L., Hovy, E, and Lewis, R. 2015. *Task Description v2: C@merata 15: Question Answering on Classical Music Scores*. Retrieved June 4, 2015 from http://csee.essex.ac. uk/camerata.

[13] Sapp, C. S. 2013. *Humdum Extras*. Retrieved March 3, 2014 from http://wiki.ccarh.org/wiki/Humdrum_Extras.

[14] Cook, N. 1994. Perception: a perspective from music theory. In *Musical perceptions*, R. Aiello and J. A. Sloboda, Eds. Oxford University Press, Oxford, UK, 64-95.

[15] Franklin, T. 2015. Arpeggio/arpeggiation. In *Oxford Music Online*. Retrieved June 4, 2015 from www.oxfordmusiconline.com.

[16] Drabkin, W. 2015. Scale. In *Oxford Music Online*. Retrieved June 4, 2015 from www.oxfordmusiconline.com.

[17] Cambouropoulos, E. 2008. Voice and stream: perceptual and computational modeling of voice separation. *Music Perception.* 26, 1, 75-94.