ALGORITHMIC HARMONIZATION OF TONAL MELODIES USING WEIGHTED PITCH CONTEXT VECTORS

Peter van Kranenburg
Meertens Institute, Utrecht University
peter.van.kranenburg@meertens.knaw.nl

Eoin Kearns
Meertens Institute
eoin.kearns@meertens.knaw.nl

ABSTRACT

Most melodies from the Western common practice period have a harmonic background, i.e., a succession of chords that fit the melody. In this paper we provide a novel approach to infer this harmonic background from the score notation of a melody. We first construct a pitch context vector for each note in the melody. This vector summarises the pitches that are in the preceding and following contexts of the note. Next, we use these pitch context vectors to generate a list of candidate chords for each note. The candidate chords fit the pitch context of a given note each with a computed strength. Finally, we find an optimal path through the chord candidates, employing a score function for the fitness of a given candidate chord. The algorithm chooses one chord for each note, optimizing the total score. A set of heuristics is incorporated in the score function. The system is heavily parameterised, extremely flexible, and does not need training. This creates a framework to experiment with harmonization of melodies. The output is evaluated by an expert survey, which yields convincing and positive results.

1. INTRODUCTION

One of the essential aspects of Western folk music is that it is in oral circulation among practitioners regardless of formal music training. As such, the transmitted music is expected to conform to melodic patterns which belong to Western music traditions. This is most tangible in the perception of rules of tonality, including the perception of stable scale tones, modes, and key centres [1]. These factors dictate the implied harmonic movement within the melody. Detecting this implied harmony is an integral part of the accompaniment of folk music. With this knowledge, it is possible to create musically meaningful harmonic progressions, using symbolic chord representations to accompany a melody.

In this paper, our aim is to explicitly design a model of how to generate a sequence of accompanying chords for a given melody, such that e.g., a guitarist could play along. Most of the recent work on this task involves machine learning in which a model is trained on a set of examples. In contrast, an essential aspect of our approach is to explicitly incorporate musical expert knowledge into the model. Our model is heavily parameterized. This has the advantage of allowing the user to have full control over the process. A disadvantage could be that the resulting model lacks the flexibility to handle various situations, which often is a reason to train a neural network instead of handcrafting a model. Our results show, however, that our current model is capable of generating convincing chord sequences for a given melody.

The model can be employed in a wide range of applications. It allows a musician to quickly obtain a suitable accompaniment for a given melody. This can be accomplished by using the default parameter settings that are established in this paper, but the model also allows to tune the parameters to get a certain desired effect. In Section 5 of this paper we provide an example in which the number of generated chords greatly varies, while each generated chord sequence is acceptable to accompany the melody. Thus, the generated harmony can be adjusted to various levels of mastering an instrument.

From a music theory perspective, our model can be considered an experimental framework to explore general principles of harmonization. In this approach, the model is used to better understand these principles. It is extremely instructive to add a heuristic to the model, or to adjust a parameter, and to examine the cases in which this leads to strong chord sequences, but even more so to examine the cases that are not acceptable. These are conditions under which the general rule apparently fails. In the current paper, we do not elaborate on this use of our model, but it is an important affordance that we do not want to be left unmentioned.

We also can imagine the system being used in an artistic way, rather than to just generate an accompaniment for practical use. In the current implementation, we incorporate well-established principles of harmonization, but it is very well possible to include other heuristics that generate chord sequences that, although not adhering to the general principles of Western tonality, could be considered an artistic contribution, or an inspiration for a new composition.

Finally, we mention the possible educational use of the model. By exploring the generated chord sequences, students can get ideas to improve or enrich their own compositions or improvisation.

2. RELATED WORK

Multiple approaches have been taken for the task of automatic harmonization [2]. Early applications of formal grammars and rule-based algorithms for automatic harmonization [3, 4] mainly sought to compose chorales in the style of Bach. This was achieved by harmonizing the soprano melody line using a set of rules to ascertain harmonic choices. These rules and heuristics are informed by observation of chorales, and enhanced by rules found in treatises. The resulting systems output successful harmonizations of existing melodies, as well as new compositions.

Context free grammars have found use for this task. Koops [5] adopts this approach in his HarmTrace and FHarm models to derive the harmonic function of a chord in its tonal context according to a set of predefined rules.

Temperley [6] proposed a rule-based algorithm to harmonize a melody by dividing the piece temporally into segments (chord spans). All possible roots are then assigned a score according to a set of four rules. The model prefers root relations which best conform to the circle of fifths. The model also predominantly chooses chord spans which begin on the metrical downbeat, and identifies and prefers ornamental dissonances which can be resolved in the subsequent chord span. While approach is related to Temperley's algorithm, it is more flexible as it does not hard-code one musical model, but instead allows basically any kind of musical preference by redefining the chord transition scoring function. Our approach is also more practical since it not only generates a sequence of root notes, but also the chord qualities.

Most of the more recent approaches are based on some form of machine learning, sometimes explicitly stating the aim to include a "minimal use of music knowledge" [7]. These approaches include Statistical Grammar learning [7], Hidden Markov Models [8–10], and neural networks [11, 12]. [13] presents a hybrid approach based on Markov chains, combining a music theoretic framework with learning from data. Our approach is distinct in that it does not require learning at all, and thus allows for full control over the process of generating the sequences.

3. DATA

The algorithm is evaluated using MTC-FS-INST-2.0, which forms part of the Meertens Tune Collections [14]. The data set consists of c. eighteen thousand melodies, both vocal and instrumental, collected from Dutch sources. The melodies have a variety of time signatures and modes. We use the pre-computed features as distributed in MTCFeatures. [1] Since our model relies on notated meter, we only use the melodies with a meter.

3.1 Music Representation

We represent the melodies as sequences of feature values, one value per note. In this paper, we use three features as provided by MTCFeatures, namely `pitch40`, `beatstrength`, and `beatinsong`, which gives onset times in units of the beat. The base-40 representation of pitch preserves the pitch spelling [15]. It includes 40 values per octave representing 40 possible pitches starting with C𝄫 and ending with B×. We map all pitch values into one octave. We use the encoding as designed by Hewlett with one adaptation: we give the first pitch (C𝄫) index 0 instead of index 1, which has a practical advantage when doing the implementation in Python.

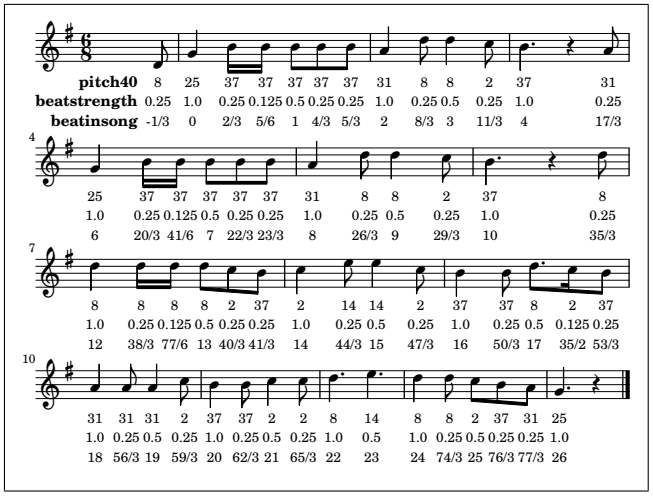

Figure 1. Example melody with the values for `pitch40`, `beatstrength`, and `beatinsong` per note.

We use the beatstrength as computed by the music21 meter model [16, 17]. Music21 is a Python library for processing symbolic musical scores. We heavily use this library. In the meter model of music21, a beatstrength is computed for each note, which indicates the metric weight at the moment of onset of the note. The main accent in the measure gets value 1.0, secondary accents get value 0.5, lower metric positions get 0.25, 0.125, etc. Figure 1 shows an example.

4. METHOD

Our approach to generate a sequence of chords for a given melody consists of three stages: First, we construct for each note a vector summarising the pitch context of that note. Second, we generate for each note a list of potential chords from the pitch context vector. Each chord gets a score indicating the extent to which it fits the pitch context. Finally, for each note, we choose one of the candidate chords, based on its score, and on a chord transition score, such that the sum of all transition scores across the sequence of chords is maximized.

The evaluation also consists of several steps. First, we tune the various parameters on a randomly chosen set of melodies. Next, we use the best parameter setting to generate chord sequences for an independent, disjoint set of melodies. We then provide six music experts with the results and to provide us with a rating of each harmonization on a five-level rating scale. Finally, we use statistics to explore and summarise the responses.

[1] `https://zenodo.org/record/3551003`

In the following of this section, we will explain each of these steps in detail.

4.1 Weighted Pitch Context Vectors

For a given note, which we indicate as the *focus note*, we consider both a preceding and a following context. These consist of the sequences of notes that are preceding, and respectively following the focus note. For both the preceding and the following context we construct a *weighted pitch context vector*. Each of these vectors has 40 elements corresponding to the 40 pitches in base-40 representation. The value of each of the elements represents the "amount" of the corresponding pitch that is present in the context of the focus note. The full context vector is a 80-dimensional vector which is the concatenation of the preceding and following context vectors.

4.1.1 Length of Contexts

Choosing the length of the contexts is not straightforward. It is, in fact, an important parameter in our model. The music21 meter model provides metric information for each note, notably concerning the *beat* and the *beatstrength* of a note (as explained in Section 3.1). This allows us to express the length of the context as a number of beats. This seems a good approach since the beat is a perceptually meaningful unit. Alternatives would be a fixed number of notes or a certain amount of score time. We did not explore these for the current study.

We experimented with different values for the context length, as well as with a variable context length based on the beatstrengths of the surrounding notes. We found that the latter approach, with variable length, yielded the best results in terms of acceptable chord sequences. In our resulting implementation the context length is computed as follows. We start with the focus note. For the preceding context, we consider the notes before the focus note in reversed order, starting with the note directly before the focus note, and we keep adding the notes to the context until (and including) we reach a note with beatstrength 1.0. For the following context, the same procedure is followed, except that the first encountered note with beatstrength of 1.0 is not included in the context. In our implementation, there is also a parameter whether to include the focus note itself into the context or not. Since the current aim is to generate a chord for the focus note, we always add the focus note to the contexts.

The consequence of this procedure is that for a note on the main accent of the measure (i.e., the first note), the preceding context is the entire previous measure, and the following context is the remainder of the measure of the focus note. In contrast, for notes that are not on the main accent, the preceding context includes all the previous notes in the same measure, while the following context includes the remaining notes in the measure. To a certain extent, this accounts for harmonic progression at different metric levels.

4.1.2 Weighting of Context Notes

The contribution of each context note to the value of the corresponding pitch in the pitch context vector is determined by two components: the metric weight (beatstrength) of the context note, and the distance to the focus note.

Intuitively, the duration of a context note has an impact on its importance in the context of the corresponding focus note. Therefore, we do not simply take the beatstrength of the moment of onset of the context note as weighting factor. Instead, we compute a metric grid, which is a succession of evenly spaced moments in score time. The basic unit of the grid, i.e., the distance between two subsequent positions in the grid, is the greatest common divisor of all note durations. Therefore, each note of the melody starts at a position in the grid, and the "span" of the note mostly includes several grid positions. The metric weighting factor of a context note is the sum of metric weights (beatstrengths) of all positions of the metric grid that are in the "span" of the note. Thus, the duration of the note, as well as the metric importance of the note are incorporated in the weighting. This approach also accounts for syncopation. During the span of a syncopated note, a grid-position with higher metric weight than the metric weight at the start of the note occurs. This is included in the sum.

Also intuitively, the further a note is away from the focus note, the lower the importance in the context of the focus note. In our model, we use a linearly decreasing windowing function. The metric weighting factor of a given context note is multiplied by the value of this window function at the position of the onset of the context note. The value of the window function during the span of the focus note is 1.0, and is linearly decreasing towards the end of the context. The value at the end of the context is a parameter in our model. We set this to a value slightly higher than 0.0 in order to have some influence from the notes that are at the outer boundaries of the contexts.

4.2 Generating Candidate Chords

Once we have computed a pitch context vector consisting of a preceding and following pitch context for each of the notes in a melody, we use these vectors to generate a set of candidate chords for each of the notes in a melody.

In our current implementation, we consider four types of chords: diminished triad, minor triad, major triad, and dominant seventh chord, and we consider three types of context: preceding context, following context, and full context. The full context just is a superposition, i.e., an element-wise sum, of the preceding and following contexts. Discerning these three types of contexts is a crucial element in our model. It allows the method to determine the position of a chord change. If the set of chords that is implied by the preceding context is sufficiently different from the set of chords that is implied by the following context, a chord change is likely, while the presence of a chord that sufficiently fits the full context likely results in a continuation.

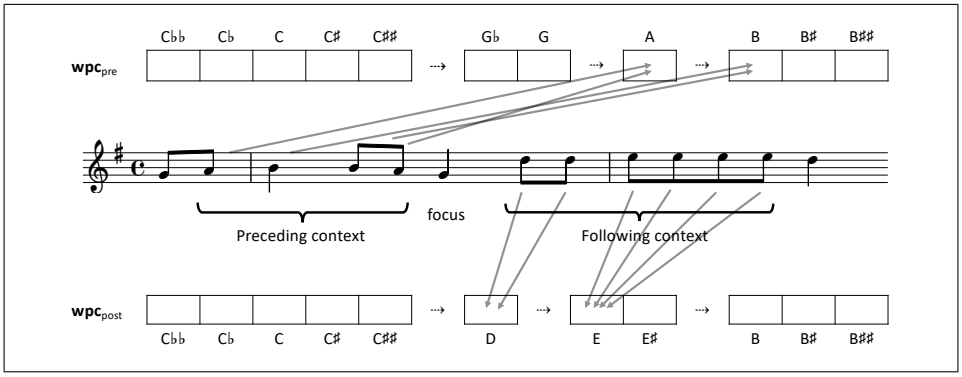

Figure 2. Example of a Pitch Context Vector. The vector consists of two parts, $\mathbf{wpc}_{pre}$ and $\mathbf{wpc}_{post}$, which represent respectively the preceding and the following context. The full Pitch Context Vector is a concatenation of the two parts. Each element in the vector gets a value representing the 'amount' of the corresponding pitch that is present in the context.

Thus, considering 40 possible root notes, we have 160 (40*4) possible chords for each of the preceding, following, and full contexts.

4.2.1 Candidate Chord Score and Strength

For each focus note, we construct a 120*4 matrix, containing a score for each possible chord in each possible context.

The score of a chord with respect to a context vector is determined by two factors: first, the extent to which the chord pitches match the pitches in the context vector, and, second, whether the root note of the chord is present in the local scale. We will explain these two factors in the following.

For each chord quality (diminished, minor, major, and dominant), a chord mask is defined. This is a 40-dimensional binary vector with ones at the positions of the corresponding chord tones. E.g., for a major chord on C𝄫 the positions 0, 12, and 23, corresponding with C𝄫, E𝄫 and G𝄫 are assigned value 1, while other positions get value 0. To compute the score of this chord for a given context, we multiply the mask element-wise with the pitch context vector, and we sum the resulting values. The resulting value represents the overlap between the context and the chord.

To compute the scores for all possible root notes, we subsequently rotate the mask over all possible 40 shifts, and compute for each shift the sum of products. We do this for the preceding context vector, the following context vector, and the full context vector. For the repertoire we have, we do not perform all 40 shifts, we only take into account natural root notes, root notes with one flat, and root notes with one sharp.

Next to these scores, we also compute a *strength* value for each of the possible chords. The strength takes a value between 0 and 1, and is computed as the ratio of the sum of the pitch context values for the chord tones (as determined by the mask) and the sum of all pitch context values. E.g, if a pitch context vector has some weight for C, E, G, and A, a C major chord would get a high score, but a strength lower than 1.0, because there is also weight for the A, which is not a chord tone.

To obtain a single score for each chord candidate, we simply multiply the score with the corresponding strength. This implies a penalty for non-chord tones within the pitch context.

We normalize the score matrix for a given focus note by dividing all scores by the highest score. Thus, the best fitting candidate always has a score of 1.0.

4.2.2 Local Scale

A second factor that determines the possible candidate chords is the local scale. As with the chord mask, we define a scale as a 40-dimensional binary vector. The elements with value 1 are the scale tones. For each note in the melody we derive a local scale vector. This records the alterations of the stemtones that are 'in use' at that position in the melody. For each stemtone $\in \{A, B, C, D, E, F, G\}$, we look for the occurrence closest to the focus note, accepting all possible alterations, and we record the alteration in the scale vector. This accounts for modulations. E.g., if in a melody in D major a G♯ occurs, which is eventually cancelled back to a G, the notes that are closer to the G♯ have a 1 at position 26 in the local scale vector (the base40 representation of G♯) while the notes closer to the G have a 1 at position 25.

One problem is posed if a stemtone is missing altogether in a melody. For example, the melody in Figure 1 lacks the note F. The key signature suggests a F♯, but that is not available to our algorithm. In these cases, we add the tone with the most likely alteration to the scale vector. For sharps, we find this by following the circle of fifths upwards from the missing tone and check the alteration of the next tone. For example, if a C♯ is present in the local scale, we infer that the scale should have a F♯, and not a F natural. For flats, we do the same, but we inspect the circle of fifths in reversed order. For edge cases, we include both the natural and altered tone in the scale. E.g., if stemtone G is missing throughout the melody, and the scale does have a C♯ and a D natural, we include both the G natural and the G♯ as possible scale tones in the local scale vector.

We use the local scale for a given focus note to eliminate those chord candidates that have a root which is not in the scale, by setting its score to 0.0. E.g., a C♯ diminished chord fits a context vector with weight for pitches E and

G, but we eliminate this candidate for a melody in C major because C♯ is not in the scale (except when sometime during the melody the C is temporarily raised).

4.3 The Chord Transition Score

The result of the procedure as described in the previous section is a sequence of matrices, one for each melody note, containing a score for each possible chord for that melody note. The next challenge is to choose one chord for each melody note out of these 120*4 possibilities. For that, we employ a chord transition scoring function (TRS), which computes a score for a given succession of chords, c_1 and c_2 for two subsequent melody notes, n_1 and n_2.

This transition scoring function can be considered a model of what would be a good chord transition. We implement this function as a series of heuristics, each penalizing the score if an aspect of the transition is undesired. We discern two kinds of penalty which could be described as a "total ban" and a "discouragement" respectively. For a total ban, we assign a very low score (-10 in our implementation), which forbids the transition in almost all cases. For a discouragement, we multiply the score by multiplier $\in [0, 1]$. The lower the multiplier, the higher the penalty for the undesired aspect of the chord transition. In our model we include the following heuristics.

- The initial transition score is the candidate score of c_2 for note n_2, as computed in the previous step.
- If the root note of c_2 differs from the root of c_1, multiply with 0.8. This stimulates continuation of a chord.
- If the root notes of both chords are the same, but the chord qualities differ, multiply with 0.1. Except for a change from major to dominant.
- For a root movement other than a prime, a fourth, or a fifth, multiply by 0.75. Root movements of fourths and fifths generally account for good harmonic progression.
- If c_1 is a dominant chord and the root of c_2 is not a fourth higher, multiply by 0.1. We strongly want a V-I relation after a dominant chord.
- If the root of c_2 is a fourth up, and c_1 is not major or dominant, multiply by 0.8.
- If c_1 is diminished, and the root of c_2 is not a semitone up, multiply by 0.1. We strongly want a VII-I relation after a diminished chord.
- If the beatstrength of n_2 is below a threshold, do not allow a chord change (score -10), except for a transition from major to dominant with the same root. The threshold is determined by the meter. For 2/4, 2/8, and 2/2 meter we take 0.25, for all other meters 0.5.
- Do not allow a chord change (score -10) if n_2 is not a chord tone in c_2, and if the beatstrength of n_2 is 0.5 or higher. The seventh of a dominant chord is not considered a chord tone. On strong metric positions, we want chord tones in the melody.
- As an exception to the previous rule, do always allow a chord change to c_2 if the next note after n_2 is a chord tone of c_2, and has a lower beatstrength than n_2. This allows for appoggiaturas.
- Do not allow (score -10) a chord that starts at a low beatstrength (<1.0) to continue past a note with higher beatstrength. Except for a chord that starts on an upbeat. This prevents chord syncopation.
- If the final root change is not a fourth up, or a fifth down, multiply with 0.1.
- If the final root change is a fifth up (a plagal cadence), multiply with 0.8.
- Only allow the root or the third of c_2 as melody note if n_2 is the final note of the melody. If this is not the case assign score -10. If the final note is the third, multiply with 0.75.

4.4 Finding the Optimal Sequence

We designed an algorithm that optimizes the score for a sequence of chord transitions. It takes the sequence of chord score matrices as input and uses the chord transition scoring function. Algorithm 1 shows the pseudo code of our algorithm. We fill a matrix, Score, which contains for each note, and for each possible chord, the total score of the chord sequence up until that note and that chord. In parallel, we fill a traceback matrix, Trace, which for each note, and for each chord, points to the chord of the previous note which is the previous chord in the sequence (i.e., maximises the total score of the chord sequence). After both the Score and Trace matrices are filled, we find the chord sequence by finding the chord with the maximal score for the final note, and following the trace back according to the pointers in the Trace matrix.

Algorithm 1 Algorithm to find the optimal sequence of chord transitions, in which l is the length of the melody in number of notes, Cand is the sequence of matrices with scores for the chord candidates, and TRS is the Chord Transition Scoring Function as defined in Section 4.3.

Require: Cand : ARRAY[l][120][4] of float
  function HARMSCORE(Cand)
    declare Score : ARRAY[l][120][4] of float
    declare Trace : ARRAY[l][120][4][2] of int
    Score[0] ← Cand[0]
    for n in $\{1, 2, \ldots, l-1\}$ **do**
      ixs1 ← indices of cells in Cand[$n-1$] > 0
      ixs2 ← indices of cells in Cand[n] >0
      for (p_2, c_2) in ixs2 **do**
        declare S : ARRAY[120][4] of float
        for (p_1, c_1) in ixs1 **do**
          $trs \leftarrow$ TRS(Cand, p_1, c_1, p_2, c_2)
          S[p_1][c_1] ← Score[$n-1$][p_1][c_1] $+trs$
        end for
        $(p_m, c_m) \leftarrow$ **argmax**(S)
        Score[n][p_2][c_2] ← **max**(S)
        Trace[n][p_2][c_2] ← (p_m, c_m)
      end for
    end for
    return Score, Trace
  end function

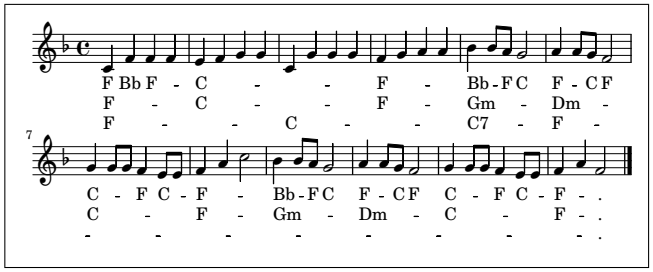

Figure 3. Example of harmonic progressions at various levels of abstraction.

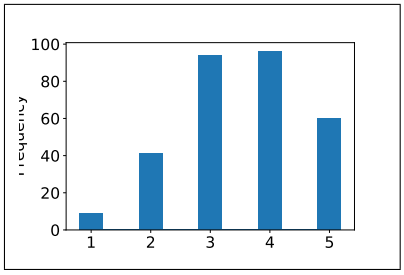

Figure 4. Distribution of ratings.

4.5 Evaluation

To evaluate our algorithm, we first tuned the various parameters ourselves by inspecting the parameter space and chose settings which seemed to yield good results. The values as reported in Section 4.3 are the result of this process.

Next, we randomly chose another unrelated set of 50 melodies and computed chord sequences for these using the parameter values from the previous step. We then asked six music experts to evaluate each harmonization. All evaluators are practicing musicians on a professional level, and have extensive experience in musical analysis. They were given a five level scale and a set of directions in order to rate the harmonizations:

1. Bad. Numerous basic mistakes.
2. Somethings are good but contains a number of incorrect chord choices.
3. Largely okay, small number of incorrect chord choices.
4. Acceptable harmonization.
5. Excellent harmonization. No improvements to be made.

Evaluators were also given a set of directions on how to rate the harmonizations. They were asked to judge to what extent the chords fit the melody, with an emphasis on the correctness of chords with regards to the local context, as opposed to creativity. They were not to take voice leading into consideration for the chord correctness, as the bass line is not modelled in this version of the algorithm.

We then use these ratings to compute inter-rater agreement and explore the extremes.

5. RESULTS

5.1 Parameter Exploration

Exploring the parameter space of our model is an interesting endeavor which appears meaningful in itself. It allows a better understanding of general textbook rules for harmonizing melodies. By implementing and manipulating these principles in our scoring function we can observe the impact of the rigorous application of these principles.

As an example, there are various ways to influence the change rate of chords. Figure 3 shows three sequences of chords at different levels of abstraction. For the middle sequence we used the default parameters as established in the previous sections. The other sequences have been obtained by changing the following parameters with respect to the defaults. The top sequence is generated by tolerating chord changes at every metric level, and by not penalizing root changes. For the bottom sequence we set the context lengths to the length of the entire melody, i.e., all preceding notes are in the preceding context and all following notes are in the following context of a given note. These three sequences show which harmonies are implied by the same melody at different time scales. This could be employed in a hierarchic strategy of harmonization, by e.g. first generating a sequence on a high level to find modulations and extended harmonic sections, and subsequently using that high level sequence as a background for the selection of more fine-grained chord sequences.

5.2 Expert Ratings

Figure 4 shows the distribution of the ratings of the experts. The average over all ratings is 3.52 and the standard deviation is 1.05. It can be observed that only a minority of the harmonizations got a rating lower than 3. Only one harmonization (no. 48) has a highest rating of 2 across the raters, and only six have a highest rating of 3. All 45 others got a 4 or 5 as highest rating. 22 sequences got a 1 or 2 as lowest rating, and 28 sequences 3 or higher. It appears that our algorithm produces an acceptable output, but there are still some issues to address. Some problems we observed are related to tonality, e.g., starting and ending in a different key (mostly the parallel), or including a leading tone at inappropriate places. Also, a low harmonic movement might be unsatisfactory.

6. CONCLUDING REMARKS

We presented a successful approach to generate a sequence of chords to accompany a folk-like melody by leveraging musical expert knowledge and a dynamic programming algorithm to find an optimal trace through the chord space. [2]

There are many directions to further build on the current model. We plan to address the observed shortcomings in a next version. Our framework can be used to explore theory on harmonization or to model implied harmony. It also can serve as tool in educational settings, and of course to generate a accompaniment for a performance.

[2] The full code of our implementation as well as the test set, the expert ratings, and a demo are available at: `https://github.com/pvankranenburg/ismir2023`.

7. ACKNOWLEDGEMENTS

This work has been enabled by the H2020 Project *Polifonia: a digital harmoniser for musical heritage knowledge* funded by the European Commission Grant number 101004746.

8. REFERENCES

[1] J. Bharucha, "Anchoring effects in music: The resolution of dissonance," *Cognitive Psychology*, vol. 16, no. 4, pp. 485–518, 1984.

[2] D. Makris, I. Karydis, and S. Sioutas, "Automatic melodic harmonization: An overview, challenges and future directions," in *Trends in Music Information Seeking, Behavior, and Retrieval for Creativity*. IGI Global, 06 2016, pp. 146–165.

[3] M. Baroni and C. Jacoboni, "Computer generation of melodies: Further proposals," *Computers and the Humanities*, pp. 1–18, 1983.

[4] K. Ebcioğlu, "An expert system for harmonizing four-part chorales," *Computer Music Journal*, vol. 12, no. 3, pp. 43–51, 1988.

[5] H. V. Koops, J. P. Magalhães, and W. B. de Haas, "A functional approach to automatic melody harmonisation," in *Proceedings of the First ACM SIGPLAN Workshop on Functional Art, Music, Modeling amp; Design*. New York, NY, USA: Association for Computing Machinery, 2013, pp. 47–58.

[6] D. Temperley, "An algorithm for harmonic analysis," *Music Perception: An Interdisciplinary Journal*, vol. 15, no. 1, pp. 31–68, 1997.

[7] D. Ponsford, G. Wiggins, and C. Mellish, "Statistical learning of harmonic movement," *Journal of New Music Research*, vol. 28, no. 2, pp. 150–177, 1999.

[8] J.-F. Paiement, D. Eck, and S. Bengio, "Probabilistic melodic harmonization," in *Canadian Conference on AI*, 2006.

[9] I. Simon, D. Morris, and S. Basu, "Mysong: automatic accompaniment generation for vocal melodies," in *CHI '08: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2008, pp. 735–734.

[10] S. A. Raczyński, S. Fukayama, and E. Vincent, "Melody harmonization with interpolated probabilistic models," *Journal of New Music Research*, vol. 42, no. 3, pp. 223–235, 2013.

[11] H. Lim, S. Ryu, and K. Lee, "Chord generation from symbolic melody using blstm networks," in *18th International Society for Music Information Retrieval Conference*, 2017, pp. 621–627.

[12] Y.-C. Yeh, W.-Y. Hsiao, S. Fukayama, T. Kitahara, B. Genchel, H.-M. Liu, H.-W. Dong, Y. Chen, T. Leong, and Y.-H. Yang, "Automatic melody harmonization with triad chords: A comparative study," *Journal of New Music Research*, vol. 50, no. 1, pp. 37–51, 2021.

[13] C.-H. Chuan and E. Chew, "Generating and evaluating musical harmonizations that emulate style," *Computer Music Journal*, vol. 35, no. 4, pp. 64–82, 2011.

[14] P. Van Kranenburg and M. De Bruin, "The meertens tune collections: Mtc-fs-inst 2.0," Meertens Institute, Amsterdam, Meertens Online Reports 2019-1, 2019.

[15] W. B. Hewlett, "A base-40 number-line representation of musical pitch," *Musikometrika*, vol. 4, pp. 1–14, 1992.

[16] M. S. Cuthbert and C. Ariza, "Music21: A toolkit for computer-aided musicology and symbolic music data," in *Proceedings of the 11th International Conference on Music Information Retrieval (ISMIR 2010)*, 2010, pp. 637–642.

[17] C. Ariza and M. S. Cuthbert, "Modeling beats, accents, beams, and time signatures hierarchically with music21 meter objects," in *Proceedings of the International Computer Music Conference*, New York, 2010, pp. 216–223. [Online]. Available: http://mit.edu/music21/papers/2010MeterObjects.pdf